

# **3D Visualization of Data from Groundwater Flow and Transport Models**

by

MEINTJES BEKKER

**DISSERTATION  
PRESENTED FOR THE DEGREE OF  
MAGISTER SCIENTIAE**

**IN THE FACULTY OF SCIENCE  
IN THE DEPARTMENT OF COMPUTER SCIENCE AND  
INFORMATICS**

**AT THE  
UNIVERSITY OF THE FREE STATE  
BLOEMFONTEIN  
SOUTH AFRICA**

**SUPERVISORS: PROF. H.J. MESSERSCHMIDT  
PROF. W.H. CHIANG**

**NOVEMBER 2000**



# *Acknowledgements*

---

A number of people have contributed significantly towards finishing this thesis. I take this opportunity to list these contributors and thank them:

Prof. H.J. Messerschmidt and Prof. W.H. Chiang my supervisors for their guidance and encouragement.

Prof. G.J. van Tonder for giving me the opportunity to work on the visualization project.

The Water Research Commission (WRC) for financial support.

Jenny Lake for her loving support, encouragement and linguistic help.

Riaan Grobbelaar for making use of his printer and CD-Writer.

Jinhui Zhang for his ideas and suggestions.

Catherine Bitzer for her linguistic help.

My family and friends for their support and encouragement.

My God who guided me through my studies.



# *Contents*

---

Acknowledgements .....	i
List of Figures.....	vii
List of Tables .....	ix
List of Listings.....	xi
<b>Chapter 1 Introduction.....</b>	<b>1</b>
1.1 Problem Description.....	1
1.2 Hypothesis and Research Questions .....	2
1.3 Research Design .....	3
1.4 Structure of the Following Chapters .....	4
<b>Chapter 2 Literature Overview .....</b>	<b>5</b>
2.1 Introduction .....	5
2.2 Visualization.....	5
2.3 Types of Data for Visualization.....	7
2.3.1 Entities .....	8
2.3.2 Relationships.....	8
2.3.3 Attributes of Entities or Relationships.....	8
2.3.4 Operations Considered as Data.....	9
2.4 Visualization Paradigms.....	10
2.4.1 Scalar Visualization .....	10
2.4.2 Vector Visualization .....	15
2.4.3 Modeling.....	19
2.4.4 Quantification .....	20
2.5 Model of Perceptual Processing.....	20
2.6 Applications.....	21
2.6.1 3D Medical Imaging .....	21
2.6.2 Computational Fluid Dynamics .....	22
2.6.3 Finite Element Analysis.....	23
2.6.4 Financial Visualization .....	24
2.6.5 Algorithm Visualization .....	25
2.6.6 Entertainment.....	26
2.6.7 Visualization on the Web.....	27
2.7 Software .....	27
2.8 Advantages and Disadvantages.....	28
2.9 Previous and Related Work .....	29
<b>Chapter 3 Visualization Tool's Environment .....</b>	<b>31</b>
3.1 Introduction .....	31

3.2	MODFLOW .....	32
3.3	MODFLOW File Output.....	33
3.4	MT3DMS.....	34
3.5	MT3DMS File Output .....	34
3.6	PMWIN .....	35
3.7	PMWIN File Output.....	35
3.8	XYZ File .....	37
3.9	Visualization Tool.....	38
<b>Chapter 4</b>	<b>The Visualization Toolkit Class Library.....</b>	<b>41</b>
4.1	Introduction .....	41
4.2	Goals .....	42
4.3	Visualization Pipeline .....	43
4.3.1	Data Objects.....	43
4.3.2	Process Objects .....	49
4.4	Imaging Pipeline.....	60
4.5	Graphics Black Box .....	60
4.5.1	Render Window .....	60
4.5.2	Render Window Interactor.....	60
4.5.3	Renderer .....	61
4.5.4	Light.....	61
4.5.5	Camera .....	61
4.5.6	Actor .....	61
4.5.7	Property.....	62
4.5.8	Mapper .....	62
4.5.9	Lookup Table .....	62
4.5.10	Transform.....	62
4.6	Putting It Together.....	62
<b>Chapter 5</b>	<b>3D Visualization Tool.....</b>	<b>65</b>
5.1	Introduction .....	65
5.2	Model Outline .....	67
5.3	Geospatial Model .....	69
5.4	Potentiometric Surface .....	78
5.5	XYZ Surface .....	82
5.6	Isosurface .....	86
5.7	Hydraulic Components .....	89
5.8	Parameter .....	92
5.9	Summary.....	95
<b>Chapter 6</b>	<b>Evaluation, Future Research and Conclusion.....</b>	<b>97</b>
6.1	Introduction .....	97
6.2	Evaluation .....	97
6.3	Future Research.....	99
6.4	Conclusion.....	100

**Appendix A Program Code**

**Appendix B Object Structure**

**Appendix C User's Guide**

**References**

**Abstract**

**Opsomming**





# List of Figures

---

Figure 2.1:	Visualization Process (Adapted from Ware, 2000 and Schroeder <i>et al.</i> , 1998b).	6
Figure 2.2:	Flow density colored mapped.	11
Figure 2.3:	Contour lines of X-ray intensity.	12
Figure 2.4:	Isosurface of flow density.	12
Figure 2.5:	Carpet plot of $F(x, y) = e^{-x} \cos(10y)$ .	13
Figure 2.6:	Different representations of a volume rendered foot (Mahoney, 2000).	14
Figure 2.7:	3D vectors (using oriented and scaled lines).	15
Figure 2.8:	3D vectors (using oriented and scaled glyphs).	16
Figure 2.9:	Vibration of beam.	16
Figure 2.10:	Streamlines for flow velocities.	17
Figure 2.11:	Tensor ellipsoids.	18
Figure 2.12:	Tensor streamlines.	19
Figure 2.13:	Visual Information Processing (Adapted from Ware.).	20
Figure 2.14:	A CT slice through a human head.	22
Figure 2.15:	Skin and bone isosurfaces.	22
Figure 2.16:	CFD visualization.	23
Figure 2.17:	Blow modeling finite element analysis.	24
Figure 2.18:	Visualization of multidimensional financial data.	25
Figure 2.19:	Towers of Hanoi.	26
Figure 2.20:	The Perfect Storm (Robertson, 2000).	26
Figure 2.21:	Visualization in the oil industry (Moltenbrey, 1999).	30
Figure 3.1:	Visualization tool's environment.	32
Figure 3.2:	Discretized aquifer system.	33
Figure 3.3:	PMWIN model.	35
Figure 3.4:	An example of an XYZ file edited in WordPad.	37
Figure 3.5:	Discretized aquifer system visualized in the visualization tool.	38
Figure 3.6:	Visualization tool.	39
Figure 4.1:	The Visualization Toolkit (vtk).	42
Figure 4.2:	Application built with toolkits (Schroeder <i>et al.</i> , 1998a).	42
Figure 4.3:	Dataset objects.	44
Figure 4.4:	Vtk cell types.	45
Figure 4.5:	Structured points.	46
Figure 4.6:	Rectilinear grid.	47
Figure 4.7:	Structured grid.	47
Figure 4.8:	Unstructured points.	48
Figure 4.9:	Polygonal data.	48
Figure 4.10:	Unstructured grid.	49
Figure 4.11:	Example of append polygonal data filter being used.	50
Figure 4.12:	Example of clip polygonal data filter being used.	51
Figure 4.13:	Example of contour filter being used.	52
Figure 4.14:	Example of decimation filter being used.	53
Figure 4.15:	Wireframe example of decimation filter being used.	53
Figure 4.16:	Example of delaunay filter being used.	54
Figure 4.17:	Example of outline filter being used.	55
Figure 4.18:	Direction of normals shown by cones.	56
Figure 4.19:	Example of poly data normals not used.	56

Figure 4.20:	Example of poly data normals used.....	57
Figure 4.21:	Cylinder not smoothed.....	58
Figure 4.22:	Cylinder smoothed.....	58
Figure 4.23:	Polygons not triangulated.....	59
Figure 4.24:	Polygons triangulated.....	59
Figure 4.25:	Cone.....	63
Figure 4.26:	Cone visualization pipeline diagram.....	64
Figure 5.1:	Notation used with visualization pipeline diagrams.....	66
Figure 5.2:	Viewpoints in 3D space.....	66
Figure 5.3:	Notation used with geometry and topology diagrams.....	67
Figure 5.4:	Model Outline visualization object.....	68
Figure 5.5:	Model Outline visualization pipeline.....	68
Figure 5.6:	Geospatial Model visualization object.....	69
Figure 5.7:	Geospatial Model visualization pipeline.....	70
Figure 5.8:	Left-top view of a one-layer Geospatial Model visualization object (a) with the average option being used. (b) The same layer with the average option not being used.....	71
Figure 5.9:	(a) Normal (b) and average geometry for Geospatial Model visualization object.....	72
Figure 5.10:	(a) Top, (b) sides and (c) bottom of a Geospatial Model visualization object for a layer.....	74
Figure 5.11:	Topology for Geospatial Model visualization object.....	75
Figure 5.12:	Potentiometric Surface visualization object.....	78
Figure 5.13:	Potentiometric Surface visualization pipeline.....	79
Figure 5.14:	(a) Normal (b) and average geometry for Potentiometric Surface visualization object.....	80
Figure 5.15:	(a) Normal (b) and average topology for Potentiometric Surface visualization object.....	82
Figure 5.16:	XYZ Surface visualization object.....	83
Figure 5.17:	XYZ Surface visualization pipeline.....	84
Figure 5.18:	Geometry for XYZ Surface visualization object.....	85
Figure 5.19:	Topology for XYZ Surface visualization object.....	85
Figure 5.20:	Concentration Isosurface (MT3DMS) visualization object.....	86
Figure 5.21:	Isosurface visualization pipeline.....	87
Figure 5.22:	Geometry for Isosurface visualization object.....	87
Figure 5.23:	Topology for Isosurface visualization object.....	88
Figure 5.24:	Hydraulic Components visualization object.....	89
Figure 5.25:	Hydraulic Components visualization pipeline.....	91
Figure 5.26:	Parameter visualization object.....	92
Figure 5.27:	Parameter visualization pipeline.....	93
Figure 5.28:	(a) Normal and (b) average topology for Parameter visualization object.....	94
Figure 6.1:	Fresh-, saltwater interface.....	98

# *List of Tables*

---

Table 3.1:	PMWIN main file. ....	36
Table 3.2:	PMWIN geometry and boundary conditions files. ....	36
Table 3.3:	PMWIN hydraulic components files. ....	36
Table 3.4:	PMWIN parameter files. ....	37
Table 5.1:	Abbreviations used with Decision Node. ....	66



# *List of Listings*

---

Listing 4.1:	Program code for Cone example.....	63
Listing 5.1:	Pseudo code, creating normal geometry for Poly Data dataset of the Geospatial Model visualization object. ....	72
Listing 5.2:	Pseudo code, creating average geometry for Poly Data dataset of the Geospatial Model visualization object. ....	72
Listing 5.3:	Pseudo code, creating normal top topology for Poly Data dataset of the Geospatial Model visualization object. ....	75
Listing 5.4:	Pseudo code, creating average top topology for Poly Data dataset of the Geospatial Model visualization object. ....	76
Listing 5.5:	Pseudo code, creating normal sides topology for Poly Data dataset of the Geospatial Model visualization object. ....	76
Listing 5.6:	Pseudo code, creating normal geometry and the normal scalar dataset attribute for Poly Data dataset of the Potentiometric Surface visualization object. ....	81
Listing 5.7:	Pseudo code, creating average geometry and the average scalar dataset attribute for Poly Data dataset of the Potentiometric Surface visualization object. ....	81
Listing 5.8:	Pseudo code, creating geometry and the scalar dataset attribute for Poly Data polygonal dataset of the XYZ Surface visualization object. ....	85
Listing 5.9:	Pseudo code, creating geometry for Structured Grid dataset of the Isosurface visualization object. ....	88
Listing 5.10:	Pseudo code, creating the scalar dataset attribute for Structured Grid dataset of the Isosurface visualization object. ....	88
Listing 5.11:	Pseudo code, creating normal topology and the normal scalar dataset attribute for Unstructured Grid dataset of the Parameter visualization object....	94



# *Chapter 1*

---

## **Introduction**

The purpose of scientific computing is insight, not numbers (Bowie, 1995).

### **1.1 Problem Description**

As is well known, fresh water is essential to human life. However, a mere 0.33% of the total volume of fresh water on earth is contained in rivers and lakes, where it is fairly accessible to man. According to Botha (1998), the remaining volume of fresh water on earth is found in glaciers (an estimated 75%) and as groundwater (24.67%).

To date no practical or economical means for making use of glacial water has been implemented, rendering groundwater the largest fresh water resource available to man. However, it remains a difficult task locating points of abstraction and abstracting groundwater from its environment. It is imperative for these points of abstraction to be well managed in order to guard against dewatering and to prevent the pollution of groundwater as a whole. Groundwater flow and transport models constitute an important tool for the management of these points and ultimately, the conservation and management of this valuable fresh water resource.

Prior to the construction of an actual model of a situation in the model domain, the modeler creates his own conceptual model of the particular situation; a mental picture of the workings of the groundwater system and its geospatial properties. Unfortunately though, in the not too distant past, scientists, academics, students and consultants have been limited to the use of words for the communication of these mental pictures.

Groundwater flow and transport models make use of large amounts of data for input and generate even larger amounts of output data. Existing models are problematic in the sense that expensive field experiments are conducted for the measuring and calculation of model

input. Costly errors are made when data are entered incorrectly. Furthermore, large amounts of output data need to be interpreted and understood in order to derive conclusions and make recommendations. The volume of output data generated is so large that it cannot be grasped by the human brain. This makes interpretation and comprehension of the data a difficult, if not an impossible task and most currently available model tools for aiding comprehension are restricted to two dimensions.

For the purposes of management and conservation of groundwater, the conclusions and recommendations derived from the groundwater models are communicated and explained to the responsible managers. However, managers are often laymen regarding geohydrological concepts, rendering the communication of this information a difficult task. The same problem presents itself in motivating the public. Much of the work done with models is aimed at the preservation of aquifers, rivers, streams, lakes and wetlands. In cases where the public is responsible for funding a particular project, it stands to reason that costs need to be motivated and explained to the persons in question. This involves an adequate description of the problem and proposed solutions; a task which often necessitates the use of certain geohydrological concepts unfamiliar to members of the public. This is also the case in situations where money donations for conservation issues are motivated or where these issues are voiced to members of the government and other involved parties.

## **1.2 Hypothesis and Research Questions**

Intelligence amplification (IA) is a term that has been created to describe the idea behind the development of tools which aid human thought, as opposed to tools thinking on behalf of their users (Brooks, 1996). This thesis has aimed to develop such a tool for use in the field of groundwater studies to aid the thinking of scientists, lecturers, students, consultants, managers and members of the public. This has been done by taking into account the characteristics and workings of human visual perception, making three-dimensional (3D) visualization an ideal tool for understanding and communicating conceptual models, verifying model input, understanding model output, explaining and communicating conclusions and recommendations, and motivating expenses. Due to the fact that humans not only have strong two-dimensional (2D) visual abilities, but are also able to integrate the different perspectives of a 3D object and other visual clues into a



mental picture, we are capable of interactive visualization. Therefore we are able to manipulate our points of view and rotate the visual objects, which aid us in achieving a better understanding of it. Furthermore, we have the talent for recognizing changes in an image. By creating scenes at different time intervals we are able to recognize trends and spot areas of rapid change.

As a consequence of its many advantages, 3D Visualization is widely used in research and design as well as the industry and entertainment arenas, such as medical research, aircraft and automobile design, and the world of cinema. The application of visualization in the field of geohydrology is, however, far behind the standard of technological advancement in other fields of study. This is particularly due to the fact that the importance of groundwater resources has been ignored for a long time. It is also partially due to computerized numerical groundwater modeling being a relatively new field of research.

The visualization tool discussed in this and the following chapters has been developed for use on a personal computer (PC) and not the traditional and powerful, but extremely expensive, graphic workstations. The 1960s saw useable display systems sold in the region of \$100 000 (Machover, 2000), but tremendous advances in PC hardware technology have made PC-based tools a possibility. Unfortunately though, the power of personal computers is limited and one of the key issues involved in the development of this tool was maintaining the rendering and interacting speed on a level which would not frustrate the user.

The advantages of 3D Visualization are put to good use in the development of this visualization tool, but one should bear in mind that it can be purposely misused by understating or exaggerating model results.

### **1.3 Research Design**

The operating systems targeted for the tool are Windows 95, Windows 98 and Windows NT. Microsoft Visual C++<sup>®</sup> with MFC and The Visualization Toolkit (vtk) are the respective programming language and visualization class library used.

The visualization tool has been developed as a sponsored project for the Water Research Commission (WRC). The tool has been developed to a point of completion which satisfies the WRC and the purposes of this thesis. The research and development have been recorded up until this point.

## **1.4 Structure of the Following Chapters**

The thesis has been structured in the following way: Chapter 2 is a literature overview of visualization and work related to this project. Chapters 3 to 5 encompass the design and development process related to the tool; Chapter 3 describes the tool's environment and input data and gives a short description, together with a diagram of the tool's components, which are used as an explanation of the structure of Chapters 4 and 5. Chapter 4 provides an explanation of The Visualization Toolkit (vtk), the visualization class library used for development of the tool and Chapter 5 outlines the decisions of design taken for each tool, component. Chapter 6 is an evaluation of the results obtained with the tool, discusses future work and contains a conclusion to the thesis.

Furthermore, there are three appendices to the study. Appendix A contains the code and pseudo code discussed in the thesis, and Appendix B consists of object structures for the program modules. Appendix C constitutes the user's guide to the tool which also contains a step-by-step tutorial of the manner in which the tool can be used.

The accompanying CD-ROM contains the installation program for the visualization tool.

# *Chapter 2*

---

## Literature Overview

### 2.1 Introduction

This chapter, taken as a whole, is a definition of visualization. It starts by introducing visualization terms and defining the visualization processes. Then the different types of data which are generally visualized are discussed. After the data discussion examples of techniques often used on the data are introduced to give an idea of how to visualize. The way humans perceive visualizations and then process that information is an important aspect to keep in mind when developing visualization software. This is discussed before the definition of visualization ends, with examples of fields where visualization has been successfully applied.

The chapter ends with a look at related fields of study to the visualization of groundwater flow and transport models.

### 2.2 Visualization

McCormick *et al.* officially introduced the term *scientific visualization* in 1987 (Schroeder *et al.*, 1998b). Although the term visualization only existed from 1987, the principle of visualization has existed long before that, for thousands of years, with cave paintings as proof. The difference between the visualization used in caves and scientific visualization is the medium on which it is created. While the canvas has changed over the years, the essence of visualization communicating information has stayed the same. Today software applications increasingly rely on 3D graphics and visualization to display information and to communicate complex, voluminous data in a simple and efficient way.

The field of visualization does not only cover a wide time span, but also covers a broad scientific field and includes elements of computer graphics, imaging, computer science, computational geometry, numerical analysis, statistical methods, data analysis and human perception (Schroeder *et al.*, 1998b).

This broad field of visualization and the many fields in which it is applied, leads to different terminology used for visualization. *Scientific visualization* being the first, was defined as the field in computer science that encompasses user interface, data representation and processing algorithms, visual representations and other sensory presentation such as sound or touch. *Data visualization* is not only confined to the sciences and engineering and therefore includes data such as those from financial, marketing or business institutions. The term data visualization is even broad enough to include the application of statistical methods and other standard data analysis techniques. A more recent term is *information visualization*. This field undertakes to visualize abstract information such as directory and file structures on a computer or abstract data structures (Schroeder *et al.*, 1998b). Yet another term, *software visualization*, is used to define the use of computer graphics and animation to help illustrate and present computer programs, processes, and algorithms. Software visualization systems can be used in teaching to help students understand how algorithms work. They can also be used in program development as a way to help programmers understand their code better (GVU Center, [www.cc.gatech.edu/gvu/](http://www.cc.gatech.edu/gvu/)).

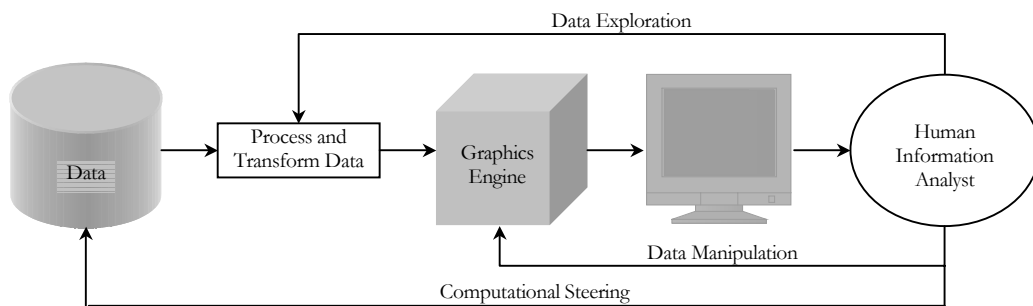


Figure 2.1: Visualization Process (Adapted from Ware, 2000 and Schroeder *et al.*, 1998b.).

The terminology used to define visualization in the previous paragraph is rather definitions of sub fields in visualization than a definition of visualization itself. A more general way of defining visualization is to look at the processes involved. According to Ware the four basic steps in the process of visualization, with a number of feedback loops as seen in Figure 2.1, are:

- The collection and storage of data. Data can result from computational methods like, finite element, finite difference, boundary element and numerical analysis. It can also result from measured data such as Computed Tomography (CT), Magnetic Resonance Imaging (MRI), ultrasound, satellite, laser digitize, stock and other financial data (Schroeder *et al.*, 1998b). Chapter 3 is an example of data from groundwater flow and transport models that are visualized with the visualization tool developed for this thesis. Section 2.3 “*Types of Data for Visualization*” discusses the different types of data that are generally visualized.
- The preprocessing of the data entails transforming the data so that it can be understood and used by the graphics engine. Chapter 5 discusses, among other things, how the data from the groundwater flow and transport models are processed and transformed for visualization with the visualization tool.
- The next step of the visualization process is for the hardware and the graphics algorithms to produce an image on the screen. Chapter 4 gives a view of a few of the aspects of a graphics engine used in the development of the visualization tool.
- The last step just before everything starts over again or before changes are made, is for the human to perceive and to understand. Acting on understanding can lead to *computational steering* that enhances the interactivity of the overall process. Computational steering directly controls the generation of the data (Johnson *et al.*, 1999). *Data exploration*, by processing and transforming the data, or *data manipulation*, that changes properties within the graphics engine, are two other possible reactions to understanding the data and are done to improve understanding even further.

## 2.3 Types of Data for Visualization

The different types of visualization data or also known as information, are classified to give structure to an introduction of commonly used visualization techniques in section 2.4: “*Visualization Paradigms*”. The structure is made possible because specific visualization techniques are best applied on specific types of data. What is to follow is only an informal classification of the data.

Data consists of two fundamental types, entities and relationships or also referred to as relations. Entities are the objects we wish to visualize, and relationships define the structures and patterns that relate entities to one another. Sometimes the relationships are

provided explicitly, and sometimes discovering the relationships is the purpose of visualization. Another form of data is called attributes and can exist for entities or relationships. A car, for example, can have its color as an attribute (Ware, 2000).

### **2.3.1      *Entities***

Entities are the objects of interest. People can be entities, for example, as well as fish and fishponds. Single things can be an entity like a fish, or a group of things like a school of fish can be an entity (Ware, 2000).

### **2.3.2      *Relationships***

Relationships form the structures that relate entities. Relationships can be structural and physical (defines how a house is made of its component parts) or conceptual (relationship between a store and its customers). Relationships can also be causal (one event causes another) and temporal (defining an interval between to events) (Ware, 2000).

### **2.3.3      *Attributes of Entities or Relationships***

Entities and relationships can have attributes. An attribute is defined as the property of an entity and cannot be thought of independently. For example the temperature of water is an attribute of the water (Ware, 2000).

Ware classifies attribute data that are generally used for visualization into three classes:

- *Category data* which work as a labeling function. Children for example, can be classified into boys and girls.
- *Integer data* are discrete numbers used to order things. Movies can be ordered according to preference, for example.
- *Real-number data* have interval and ratio properties. For interval data the value between data values can be derived. Think of the time of departure and arrival of an aircraft. Ratio on the other hand makes full use of the power of real numbers, where zero is used as reference. A for example is twice as large as B.

Attribute data are also classified according to the dimensional property of the data itself. *Scalar* data are one-dimensional (1D) quantities and are used as single values at each location. Examples of scalar data are temperature, pressure, density, elevation, stock prices and the weight of a person. Scalar data are the simplest and most common form of visualization data (Schroeder *et al.*, 1998b and Ware, 2000).

*Vector* data are data with magnitude and direction. Examples of vector data include flow velocity, particle trajectory, wind motion, gradient function and the direction a person is traveling (Schroeder *et al.*, 1998b and Ware, 2000). *Tensors* are complex mathematical generalizations of vectors and matrices. This higher order form of attribute data can for example describe both the direction and shear forces, such as occur in materials that are being stressed (Ware, 2000). A special type of vectors are *normals* which are direction vectors with a magnitude  $|n| = 1$ . Normals are often used by the graphics engine to control the shading of objects (Schroeder *et al.*, 1998b).

*Texture coordinates* are used to map a point from Cartesian space into one-, two- or 3D texture space. The texture space is referred to as texture map. Texture maps are regular arrays of color, intensity and/or transparency values that provide extra detail to rendered objects. The most common use of texture mapping is to project a photograph onto one or more polygons that will then yield a more detailed image without using a large number of graphics primitives (Schroeder *et al.*, 1998b).

A *field* of scalars, vectors or tensors can exist. The gravitational field of the earth is for example a 3D vector field, data attribute, but the strength of the gravity at the earth's surface is a 2D scalar attribute (Ware, 2000).

### 2.3.4 *Operations Considered as Data*

Entities and relationships describe most kinds of data, but it doesn't capture the operations that may be performed on entities and relationships. Some of these operations are easy to visualize while others are not. Operations listed by Ware are:

- Mathematical operations on numbers such as multiplication and division.
- Merging of two lists.
- Inverting a value to its opposite.

- Bringing an entity and relationship into existence.
- Deleting an entity or relationship.
- Transforming an entity.
- Forming a new object out of other objects.
- Splitting a single entity into its component parts.

Theoretical entities (for example correlations between variables or cluster of variables, or certain underlying mechanisms that are not immediately visible) result from data analysis and are called *metadata*, or more generally in the database modeling community, *derived data*. Meta data can also consist of new entities, relationships or rules and should be visualized as primary data, having entities, relationships and their attributes to visualize, although some are more abstract than others (Ware, 2000).

## 2.4 Visualization Paradigms

The definition of visualization in the following sections is from the viewpoint of visualization techniques applied to the data. This section aims to give an overview of some visualization techniques used in visualization applications.

### 2.4.1 *Scalar Visualization*

Scalars are commonly found with many different algorithms to visualize them (Schroeder *et al.*, 1998b).

#### 2.4.1.1 *Color Mapping*

Color mapping, as seen in Figure 2.2, is a scalar visualization technique that maps scalar values to colors and is a simple way to increase the information content of visualizations. It works on the principle that the scalar values are used as indices into a color lookup table (Schroeder *et al.*, 1998b).



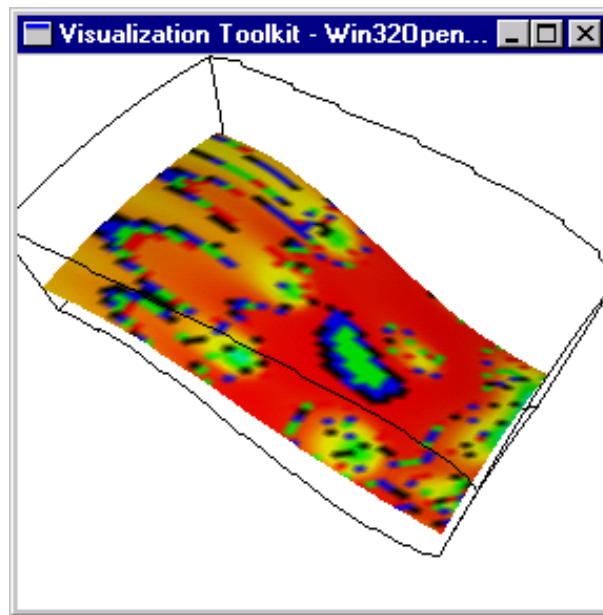


Figure 2.2: Flow density colored mapped.

#### 2.4.1.2 Contouring (*Isolines and Isosurfaces*)

Contouring is an extension of color mapping. With color mapping the eye separates similarly colored areas into distinct regions and with contouring the boundaries between these regions are constructed. The resulting boundaries are contour lines, also referred to as isolines, for 2D scalar data and surfaces, called isosurfaces for 3D scalar data. 2D contour line examples are topological maps with contour lines and isotherms for constant temperatures. Figure 2.3 shows contour lines of X-ray intensities for a cross slice of a human head (Schroeder *et al.*, 1998b).

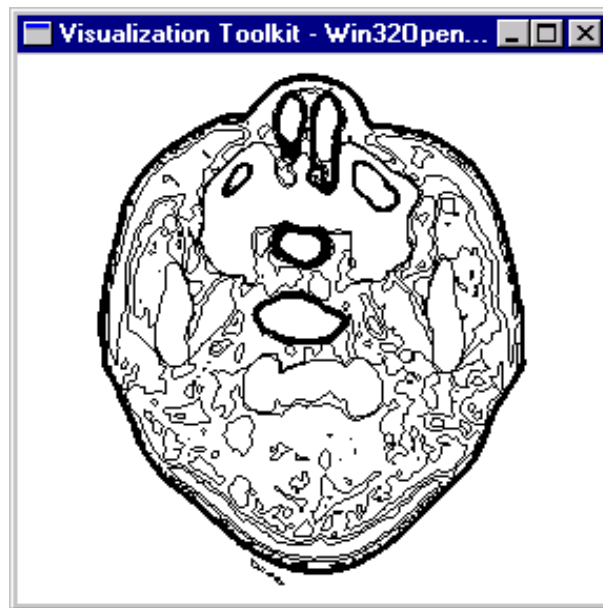


Figure 2.3: Contour lines of X-ray intensity.

Examples of isosurfaces include constant medical image intensities corresponding to body tissues such as skin, bone or other organs, as shown in section 2.6.1 “*3D Medical Imaging*” or more abstract examples of isosurfaces are constant pressure, temperature in fluid flow or fluid density, as seen in Figure 2.4 (Schroeder *et al.*, 1998b).

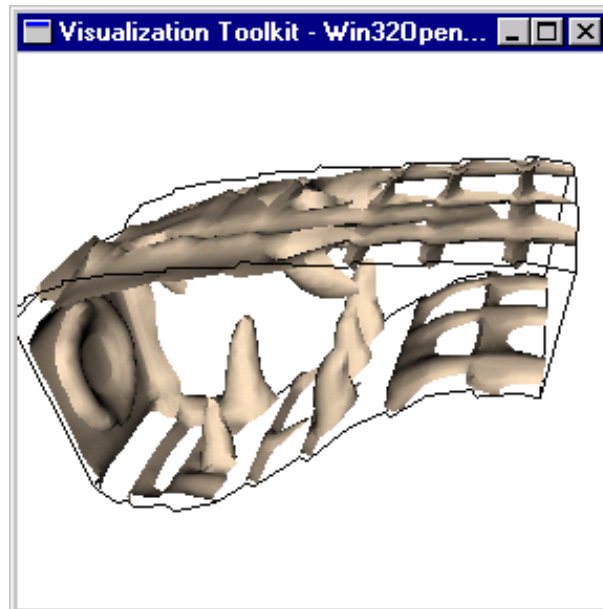


Figure 2.4: Isosurface of flow density.

### 2.4.1.3 Carpet Plots

Carpet plots are typically used to visualize functions of the form  $F(x, y) = z$ . The x-y plane is distorted according to the z function value. Color mapping is often used to introduce another variable into the visualization. Figure 2.5 shows the carpet plot of the function  $F(x, y) = e^{-r} \cos(10r)$  and the colors indicate derivative values (Schroeder *et al.*, 1998b).

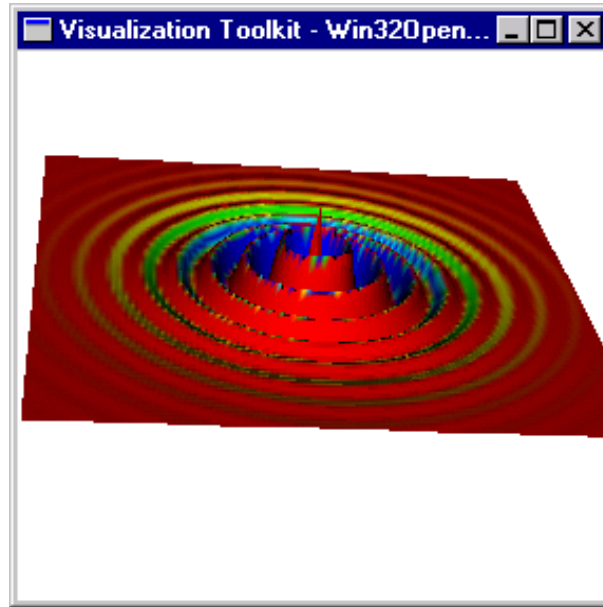


Figure 2.5: Carpet plot of  $F(x, y) = e^{-r} \cos(10r)$ .

### 2.4.1.4 Volume Rendering

Users of visualization most generally make use of conventional polygon-based 3D rendering techniques. These techniques are primarily concerned with representing surfaces of objects by placing polygons on wireframe models that can be twisted and rotated, but without being able to simulate the interiors of the object (Mahoney, 2000).

Volume rendering on the other hand is a projection technique that produces image displays of 3D volumetric data without creating intermediate geometric representations. Its main characteristic is the production of view-dependent snapshots of volumetric data, rather than the extraction of geometric information such as with isocontouring (Bajaj, 2000).

Volume rendering has the advantage of more natural looking visualization results. The volume-rendered feet in Figure 2.6 for example, provide significantly more insight than could be achieved by using surface rendering. This does not only apply to the medical

field, but also to non-destructive testing, rapid prototyping, reverse engineering, oil and gas exploration, physics, astronomy, fluid dynamics, meteorology, molecular modeling, and furthermore the ability to easily and accurately represent atmospheric effects such as water, fire, clouds and explosions, makes it an excellent technique to apply in entertainment applications such as films and interactive games. Unfortunately volume rendering has the disadvantage of not always being real-time interactive because of the volumes of data and the computational intensity of the technique itself (Mahony, 2000).

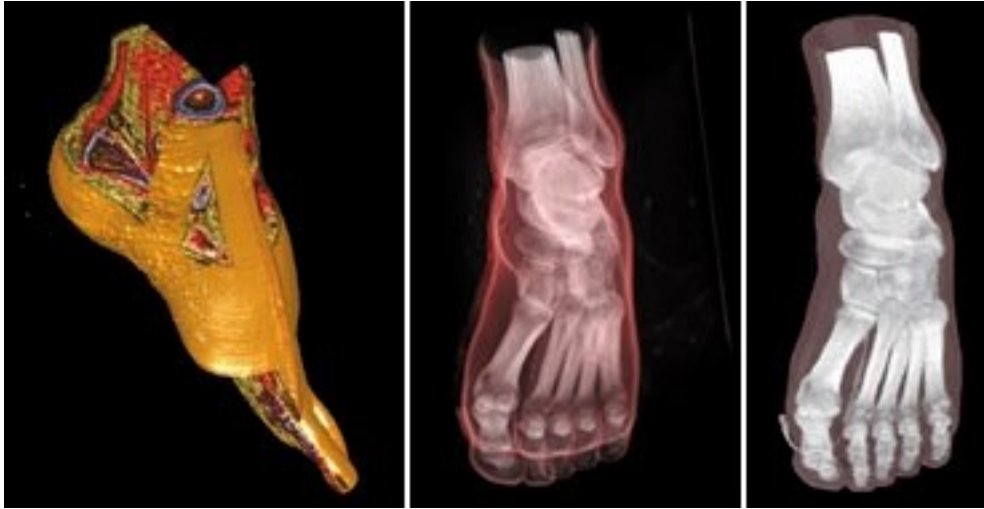


Figure 2.6: Different representations of a volume rendered foot (Mahoney, 2000).

#### 2.4.1.5 *Scalar Field Topology*

Field topology in general refers to the analysis and classification of critical points and computation of relationships between the critical points of field data. Computation and display of field topology provides a way to summarize very large sets of data. Techniques such as volume rendering and line integral convolution provide qualitative global views of field topology (Bajaj, 2000).

Scalar field topology is where a vector field is calculated as the gradient of the scalar function. Critical points in scalar fields are defined as zero gradient and can be classified into maxima, minima, saddle points and degenerate cases (Bajaj, 2000).

## 2.4.2 Vector Visualization

Vector data has direction and magnitude and source examples are fluid flow or derivatives (rate of change) for some quantity (Schroeder *et al.*, 1998b).

### 2.4.2.1 Hedgehogs and Glyphs

One of the easiest visualization techniques for vector visualization is to use an oriented, scaled line for each vector. The result is often called hedgehog because of the bristly result. Figure 2.7 shows a hedgehog for blood flow in the region of the human carotid arteries.

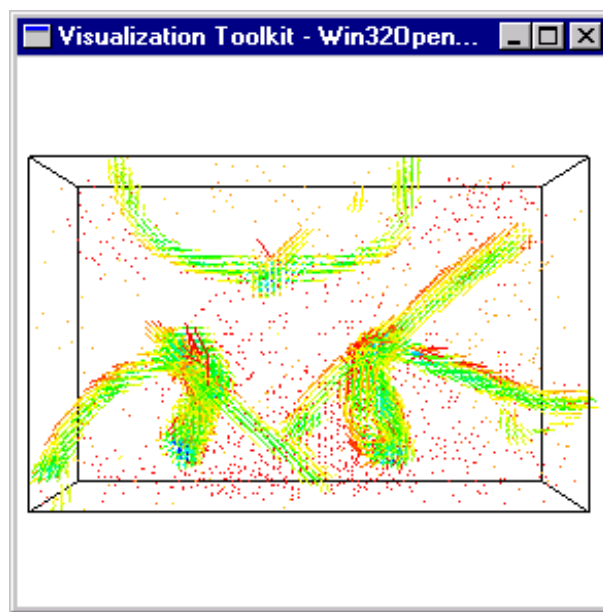


Figure 2.7: 3D vectors (using oriented and scaled lines).

A variation of the hedgehog technique is to use arrows to show the direction of the lines. The lines can also be replaced with 2D or 3D geometric representations such as triangles or cones, called glyphs. Figure 2.10 shows a close-up view of blood flow in the region of the human carotid arteries, using scaled glyphs (Schroeder *et al.*, 1998b).

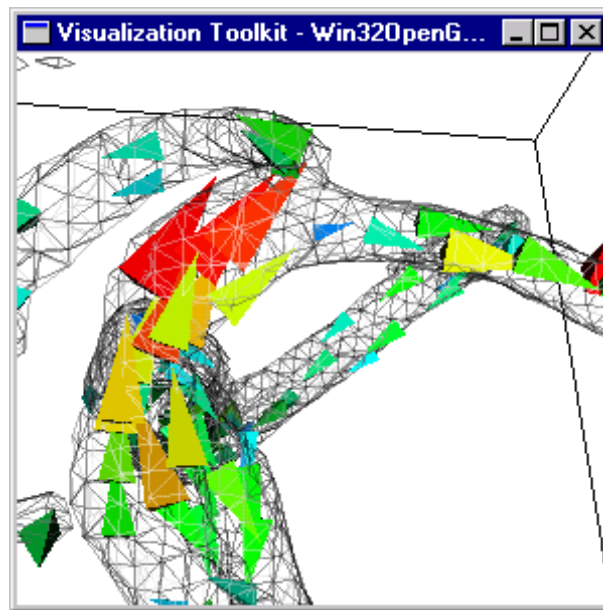


Figure 2.8: 3D vectors (using oriented and scaled glyphs).

#### 2.4.2.2 *Warping Geometry*

A good technique to visualize vector data associated with motion in the form of velocity or displacement, is to warp or deform geometry according to the vector field. Figure 2.9 shows the motion of a vibrating beam. The original undeformed outline is shown in wireframe (Schroeder *et al.*, 1998b).

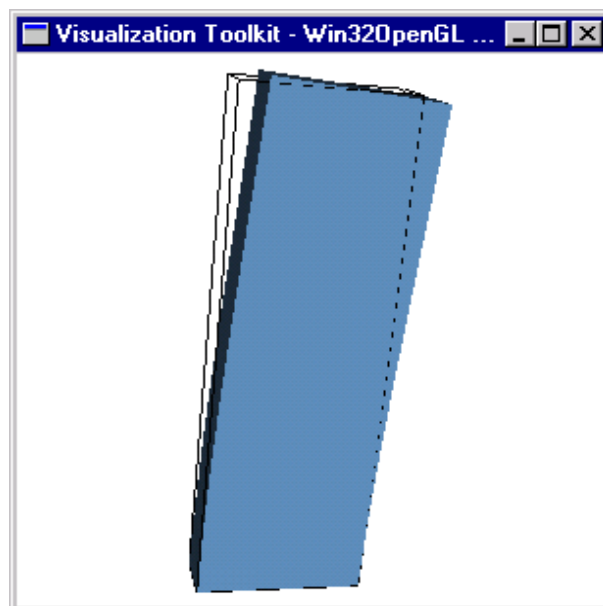


Figure 2.9: Vibration of beam.

### 2.4.2.3 Streamlines and Streamtubes

Point positions can be connected over many time steps with the result being a numerical approximation to a particle trace, represented as a line. Figure 2.10 shows computed flow velocities as streamlines for a small kitchen (Schroeder *et al.*, 1998b).

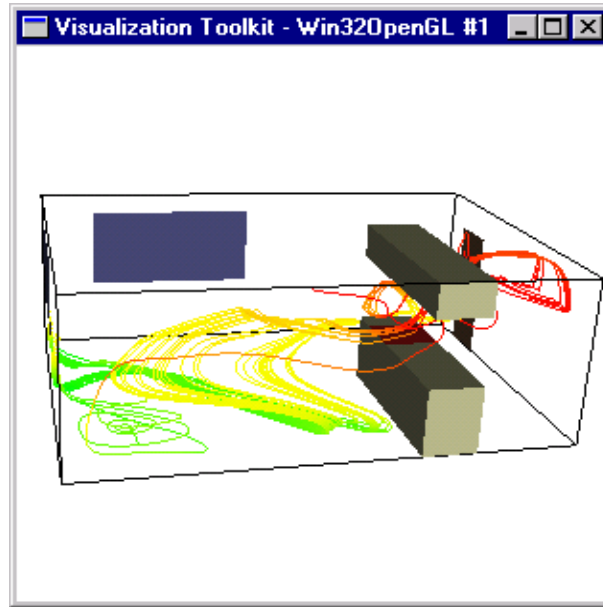


Figure 2.10: Streamlines for flow velocities.

### 2.4.2.4 Vector Topology

The general idea of field topology is described in section 2.4.1.5 “Scalar Field Topology”. For vector topology, given a continuous vector field, the locations at which the vector becomes zero are called critical points (Bajaj, 2000).

### 2.4.2.5 Tensor Visualization

Tensors are used to describe the state of displacement or stress in 3D material (Ware, 2000).

### 2.4.2.6 Tensor Ellipsoids

Eigenvectors and eigenvalues characterize a  $3 \times 3$ , real symmetric matrix, which is the tensor. Ellipsoids are used to represent the size of the eigenvalues and the orientation of the eigenvectors. Figure 2.11 is the visualization result of a single point load, applied to

elastic material, resulting in singular stress and strain values. At the surface of the material the ellipsoids flatten because there is no stress perpendicular to the surface (Schroeder *et al.*, 1998b).

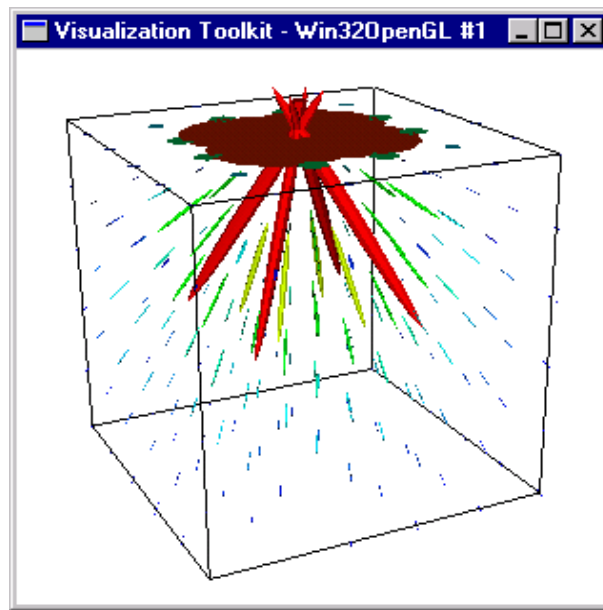


Figure 2.11: Tensor ellipsoids.

#### 2.4.2.7 *Hyperstreamlines*

The tensor eigenvectors define three perpendicular vector fields corresponding to the major, medium and minor eigenvalues. One of these fields is used to generate streamlines while the other two fields control the cross-section of an ellipse that is swept along the streamline, as seen in Figure 2.12. The hyperstreamlines flare as they approach the stress singularity. A plane at the bottom also shows the symmetric nature of the stress field, and is colored by effective stress, using a different color mapping from the hyperstreamlines (Schroeder *et al.*, 1998b).



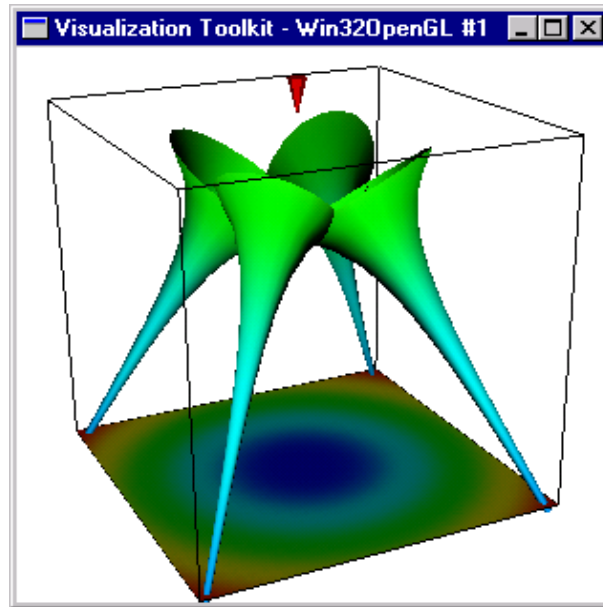


Figure 2.12: Tensor streamlines.

### 2.4.3 Modeling

Visualization techniques that do not fall into the groupings for scalar, vector and tensor visualization techniques are grouped under modeling techniques by Schroeder *et al.*. Modeling techniques can be used to create simple, geometry objects such as spheres, cones and cubes. These objects can be used alone or in combination to model geometry. During the visualization process, modeling techniques may also be used to create supporting geometry such coordinate axis. Data attribute creation is also a result of these techniques. Textures, texture coordinates and scalar values are created over a uniform grid. If the scalar values are created using a mathematical function, it belongs to a specific aspect of modeling, called implicit functions. Implicit functions are of the form  $F(x, y, z) = c$  and are used for geometric modeling, selecting data and visualizing complex mathematical descriptions. Implicit functions or the boolean combination of implicit functions can be used to model geometric objects. The approach is to evaluate these functions on a regular array of points or volume and to then generate scalar values at each point, which are visualized using volume rendering or isosurface techniques (Schroeder *et al.*, 1998b).

Implicit Modeling is an extension of the implicit function, modeling technique. The difference is that a distance function instead of an implicit function is used to create the

scalars. The distance function is computed as a Euclidean distance to a set of generating primitives such as points, lines or polygons (Schroeder *et al.*, 1998b).

Other modeling techniques are used to cut objects with a surface (Schroeder *et al.*, 1998b). Modeling techniques are also used for data extraction and data reduction, of which the height field and geometry reduction techniques are examples (Bajaj, 2000).

#### 2.4.4 Quantification

The idea of visualization is not only to see data, but also to navigate and query data for better understanding. *Contours* are for example a volumetric quantification technique that is applied on contour surfaces, which are created through the isocontouring of scalar data. Given an isovalue, one can compute the surface area of the corresponding isosurface, the volume of the inside region, or any other metric property (called signature) function for the specific isovalue (Bajaj, 2000).

## 2.5 Model of Perceptual Processing

The way in which humans perceive and then process that information is an important issue to keep in mind while developing visualization software. According to Ware as seen in Figure 2.13, a generalized information-processing model of human visualization consists of two stages.

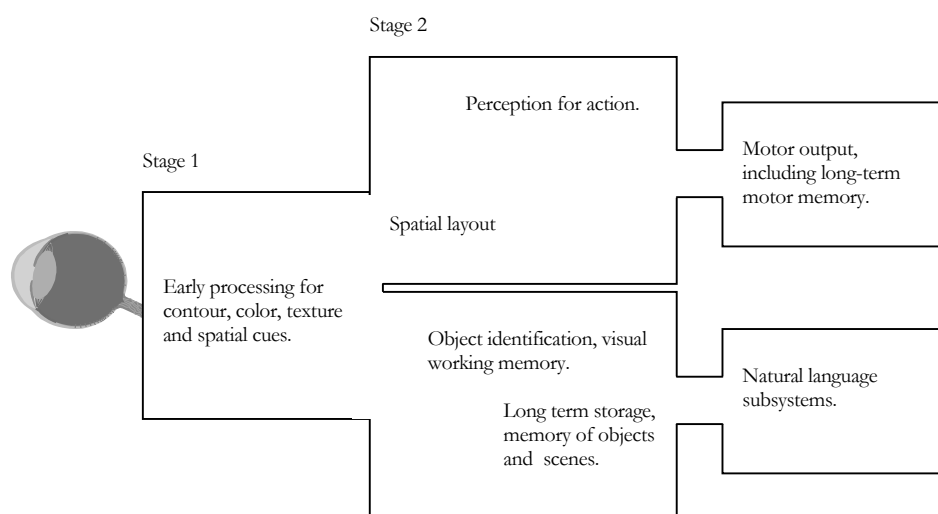


Figure 2.13: Visual Information Processing (Adapted from Ware.).

During the first stage parallel processing is done to extract low-level properties, such as contour, color, texture and spatial cues (orientation and movement patterns) from the visual scene. Rapid parallel processing and the transitory nature of information, which is briefly held in an iconic store, are also characteristics of the first stage (Ware, 2000).

The second stage is subdivided into a subsystem for interacting with the environment, and a subsystem specialized for object recognition that makes use of working and long-term memory. Compared to the rapid parallel processing of the first stage, the processing of the second stage is slow, and in serial. More emphasis is placed on the arbitrary aspects of symbols (Ware, 2000).

## 2.6 Applications

Visualization is used in a variety of applications. Medical visualization systems make use of volume rendering and imaging to reconstruct anatomical structures of living patients from Computed Tomography (CT), Magnetic Resonance Imaging (MRI) and ultrasound scans. Engineering visualization systems have been developed to enable engineers to understand and communicate design intent and to perform maintenance and assembly analysis. Engineers also make use of visualization systems for analysis, such as structural and computational fluid dynamics. Visualization is not, however, only restricted to the pure sciences, but is also applied to financial data for example (Schroeder *et al.*, 1998b).

### 2.6.1 3D Medical Imaging

Radiology is a medical discipline that deals with images of human anatomy. The images result from imaging modalities (each one having their diagnostic strengths) such as: X-ray, X-ray CT, MRI and ultrasound. The radiologist and the physician decide on what modality to use, but for the most part the radiologist with his or her special training, deals with the complex anatomical relationships in these 2D representations. Problems sometimes arise when communicating this understanding to physicians and surgeons. Figure 2.14 is an image of one of ninety-three, CT slices spaced 1.5 mm apart. Each slice consists of  $256^2$  pixels spaced 0.8 mm apart with twelve bits of gray level (Schroeder *et al.*, 1998b).

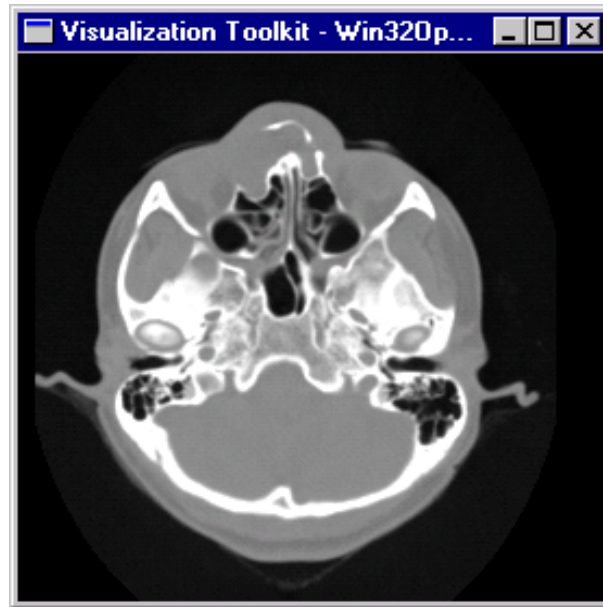


Figure 2.14: A CT slice through a human head.

Figure 2.15 shows two isosurfaces corresponding to the human skin and bone. The data of the CT slices, as mentioned for Figure 2.14, was used for the isosurface contouring (Schroeder *et al.*, 1998b).

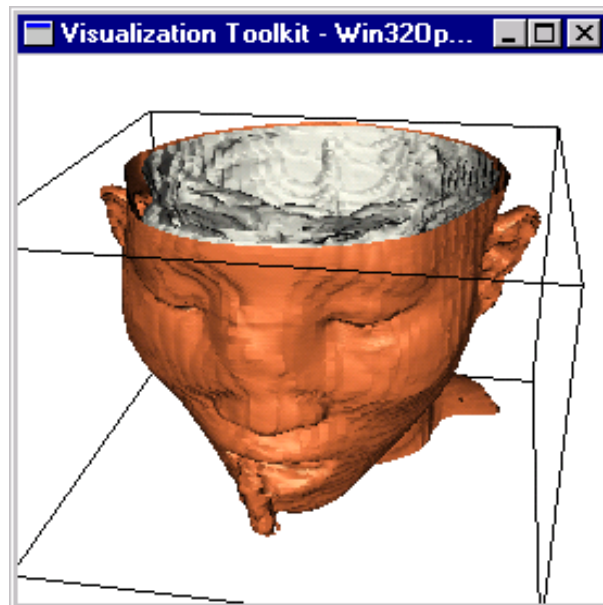


Figure 2.15: Skin and bone isosurfaces.

## 2.6.2 Computational Fluid Dynamics

Computational Fluid Dynamics (CFD) studies the flow of fluids in and around complex structures. CFD analyses often employed finite difference grids, that discretize the

problem domain into small computational cells. The grid allows the analyst to create large systems of equations that can be solved by the computer. The result of the equations is large amounts of scalar and vector data in the flow field. Figure 2.16 is an example of CFD data being visualized. It consists of a finite difference grid with a higher density in the regions where rapid changes occur in the flow variables. The grid is displayed in wireframe so that the computational cells can be seen. The resulting vector field is visualized by streamtubes created by using the computational grid just in front of the post as a source for seeds. The dataset for this visualization resulted from the flow simulation of liquid oxygen in a rocket engine. The post perpendicular to flow promotes mixing of liquid oxygen (Schroeder *et al.*, 1998b).

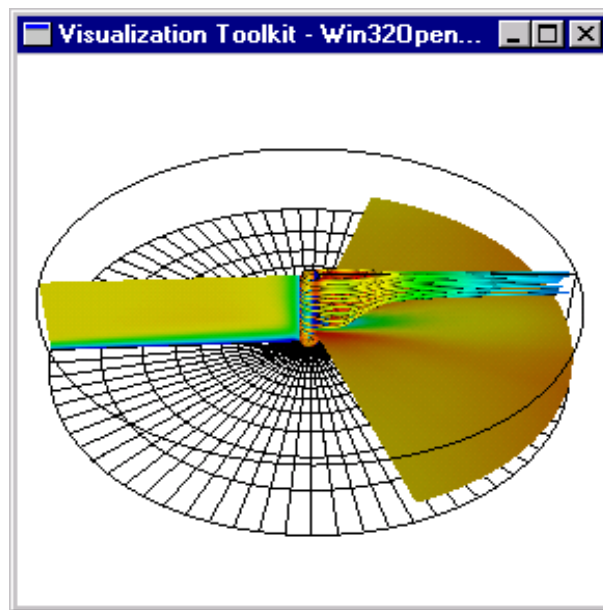


Figure 2.16: CFD visualization.

### 2.6.3 *Finite Element Analysis*

Finite element analysis is a numeric technique for finding solutions of partial differential equations. Applications of finite element analysis are linear and nonlinear, structural, thermal, dynamic, electromagnetic and flow analysis (Schroeder *et al.*, 1998b).

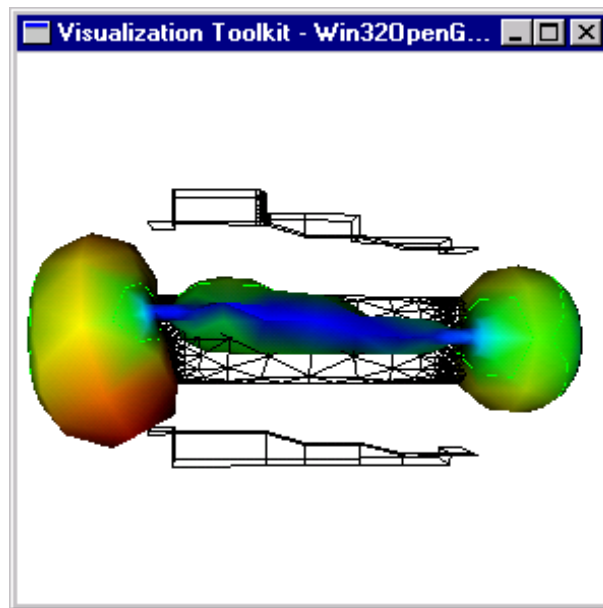


Figure 2.17: Blow modeling finite element analysis.

Figure 2.17 is the visualization of the extrusion blow modeling process where material is extruded through an annular die to form a hollow cylinder. The cylinder is called the parison. The mold halves are then closed on the parison, while at the same time the parison is inflated with air. Some of the parison material remains within the mold while some become waste material. Plastic models are often manufactured using a blow modeling process. Designing the parison die and molds is not easy and improper design results in large variations in the wall thickness. In some cases the part may fail in thin walled regions. As a result analysis tools, based on finite element techniques have been developed to assist the design of molds and dies. A visualization example of the data from these tools is shown in Figure 2.17. The color of the parison indicates its thickness. This kind of visualization clearly shows problem areas where the walls are too thin (Schroeder *et al.*, 1998b).

#### 2.6.4 Financial Visualization

Historical 2D plotting techniques such as line, scatter plots, bar charts and pie charts can now be supplemented with 3D visualization techniques, especially for vast volumes of information in the financial sector. Interactive visualization techniques make day-to-day processing possible, which leads to a better understanding of complex financial data and other timely decisions (Schroeder *et al.*, 1998b).

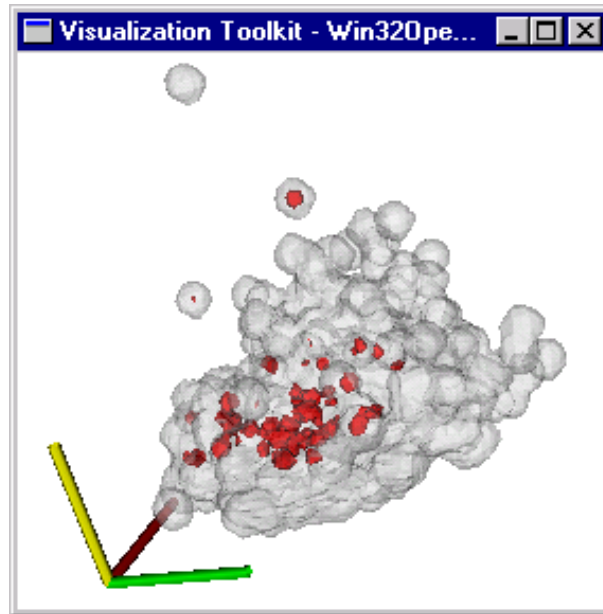


Figure 2.18: Visualization of multidimensional financial data.

Figure 2.18 is the visualization result of financial data from 3000 loan accounts with each account having six different variables. The axes are monthly payment, interest rate and loan amount. The outer, grayish surface shows the total population and the darker, red surfaces indicate accounts that are delinquent on loan payments. This information can be used to characterize bad credit risks (Schroeder *et al.*, 1998b).

### 2.6.5 Algorithm Visualization

Visualization is also a teaching tool and can be used to display algorithm and data structures. Figure 2.19 is an image of the visualization of the recursive operation of the Tower of Hanoi puzzle. In this puzzle there are three pegs. In the initial position there are one or more disks of varying diameter on the pegs. The disks are sorted according to disk diameter, so that the largest disk is on the bottom, followed by next largest, and so on. The goal of the puzzle is to move the disks from one peg to another, moving the disks one at a time and never placing a larger disk on top of a smaller disk (Schroeder *et al.*, 1998b).

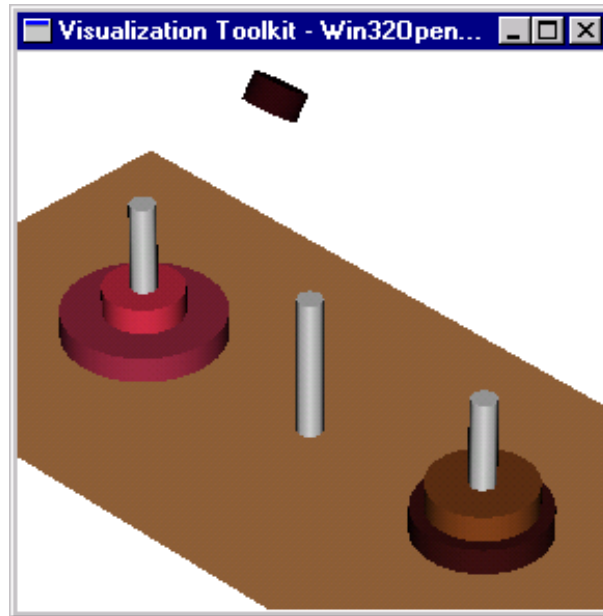


Figure 2.19: Towers of Hanoi.

### 2.6.6 *Entertainment*

During late October 1991 in the North Atlantic, a low-pressure system filled with cold air collided with a hurricane filled with warm air, with the devastating effect of waves 30 meters high. The movie about this storm, titled with the name meteorologists give to such a rare event, is “Perfect Storm”. The problem with making the movie was creating such a storm on film. Actors could be filmed on a full-size model of a boat, and the boat could fit in a tank of water, but a 30-meter wave certainly could not. The answer was to use visualization to create a digital storm with water that looked and moved realistically. Figure 2.20 is a screenshot from the movie, created entirely with computers. Fluid simulation was used to shape and move the ocean, particle dynamics for the crest of the wave and cloth simulation for the sail (Robertson, 2000).



Figure 2.20: The Perfect Storm (Robertson, 2000).



### 2.6.7 *Visualization on the Web*

Visualization on the web is used to enable people throughout the world to interact and share information quickly and efficiently compared to other methods. The web has the advantage that researchers can access research reports and interact directly with the data, including visualizing the results. It also has the added advantages that reports can be published immediately so that anyone with web access can view them, and results can be modified as work progresses (Schroeder *et al.*, 1998b).

## 2.7 **Software**

Creating and reproducing pictures for visualization was hindered throughout history by technical problems that stood in the way of widespread use. The Chinese proverb “a picture is worth ten thousand words” became only a cliché after the advent of inexpensive and simple technology for producing pictures (Foley *et al.*, 1996). Today computers have evolved sufficiently to supplement the printing press for visualization.

As with computer hardware advances, graphics software standards have also advanced significantly. Better graphics software has been used to make dramatic improvements in the look and feel of user interfaces and the increasing use of 3D effects are expected, both for aesthetic reasons and for providing new metaphors for organizing, presenting and navigating through information. Perhaps the most important new movement in visualization is the increasing concern for modeling objects and not just for creating the pictures. Interest in describing time-varying geometry and behavior of 3D objects is also growing. Thus visualization is increasingly concerned with simulation, animation and a “physics” movement for both modeling and rendering in order to create objects that look and behave as realistically as possible. As the tools and capabilities available become more and more sophisticated and complex, the need arises to be able to apply them effectively. Rendering is no longer the bottleneck and therefore researchers are beginning to apply artificial intelligence techniques to assist in the design of object models, as for example in motion planning and in the layout of effective 2D and 3D graphical presentations (Foley *et al.*, 1996).

It is clear that 3D graphics and visualization are entering mainstream use. As evidence of this we cite the use of 3D graphics in the entertainment and gaming industries and especially its support on the PC. For example there are now several 3D graphics software API's on the PC, including OpenGL, and hardware boards ranging in cost from hundreds to thousands of US dollars (Schroeder *et al.*, 1998b).

A recent added advantage to visualization software is the object-oriented approach for 3D graphics and visualization. It influenced commercial systems such as AVS, IBM Data Explorer and Iris Explorer (Schroeder *et al.*, 1998b).

The visualization toolkit (vtk), used to develop the visualization tool, is one of several visualization systems today. "AVS was one of the first commercial systems available. IBM's Data Explorer (DX), originally a commercial product, is now open source and is known as OpenDX. NAG Explorer and Template Graphics Amira are other well known commercial systems." (Schroeder, 2000).

## 2.8 Advantages and Disadvantages

Visualization is one of the most *natural* means of communicating with a computer, because of humans' highly developed 2D and 3D pattern-recognition abilities, which enable pictorial data to be rapidly and efficiently perceived and processed. The information that pictures give in the processes of design, implementation and construction are indispensable. With scientific visualization, scientists and engineers realized that they could not interpret the *huge amounts* of data without comprehending the data and highlighting trends and phenomena in various kinds of graphical presentations (Foley *et al.*, 1996 and Ware 2000).

Visualization on a computer is better than the printing press or photographs, because it has the added advantage that with the computer not only pictures of concrete, "real-world" objects can be made, but also of abstract, synthetic objects such as mathematical surfaces in four-dimensions and of data that have no inherent geometry. Furthermore, with a computer visualization is not confined to static images. Static pictures are a good means of communicating information but dynamically varying pictures are most often even better, which is especially true for both real and abstract time-varying phenomena. The use of

dynamics is especially effective when the user can control the animation by adjusting the speed, the portion of the total scene in view, the amount of detail shown, and the geometric relationship of the objects in the scene to one another (Foley *et al.*, 1996).

Interactive visualization in the form of *motion dynamics* and *update dynamics* is another advantage of visualization, because it permits extensive, high-bandwidth user-computer interaction, which significantly enhances the ability to understand data, to perceive trends and to visualize real imaginary objects. Motion dynamics is where the objects can be moved and tumbled with respect to a stationary observer. The objects can also remain stationary and the viewer can move around them, pan to select the portion in a view and zoom in and out for more or less detail. Update dynamics, on the other hand, is the actual change of shape, color or other properties of the objects being viewed (Foley *et al.*, 1996).

Designers of 3D objects such as automobiles, airplanes and buildings want to see how their preliminary results look. Creating realistic computer-generated images is an often easier, less expensive and more effective way to see preliminary results rather than building models and prototypes. It allows more alternative designs to be considered (Foley *et al.*, 1996).

Visualization in general often results in the perception of emergent properties of data that were not anticipated and helps to highlight problems which are not immediately apparent. Visualization does not only reveal things about the data itself, but also about the way it is collected. Appropriate visualization errors and artifacts of data are easily identified, which makes visualization an invaluable quality control tool, and very often the discovery of errors and artifacts leads to hypothesis formation (Ware, 2000).

## 2.9 Previous and Related Work

The science applied to the oil industry and geohydrology are well related but if the role that visualization plays in the two fields is compared, the difference moves off the scale. This is not too surprising if one keeps in mind the money involved in oil, which makes everyone go around, compared to potable groundwater which is often taken for granted. Black gold buy state of the art technology and the latest advancement in software or expertise to

address their visualization needs, and are an upper limit of what can be done, using visualization.

Figure 2.21 shows how visualization is a valuable technique used in the oil industry to locate the nooks and crannies that often contain hidden oil deposits. Potentially oil-rich fault areas appear as steeply dipping surfaces. Visualization has saved oil companies millions of dollars helping the experts select the best sites to drill. It has also saved a lot more money reducing the time it takes to analyze voluminous heaps of data (Moltenbrey, 1999).

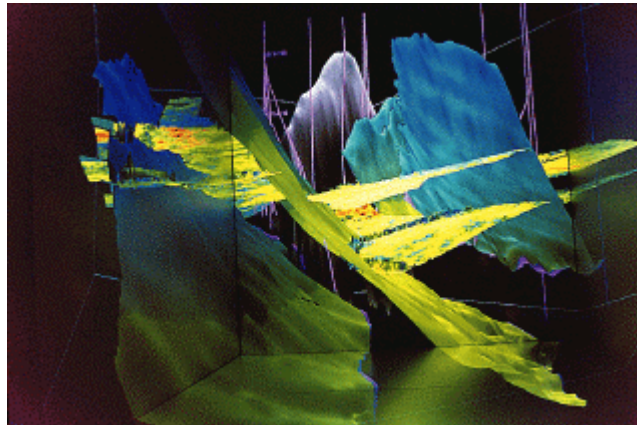


Figure 2.21: Visualization in the oil industry (Moltenbrey, 1999).

# *Chapter 3*

---

## **Visualization Tool's Environment**

### **3.1 Introduction**

MODFLOW (McDonald and Harbaugh, 1998) is one the first groundwater models developed and grew much in popularity because of its good results. Because of this growth, the number of application fields for the description and prediction of the behavior of groundwater systems increased significantly over the past few years. Numerous investigators have developed various add-ons to MODFLOW in the form of packages, models and programs to model more real-world system scenarios. MT3DMS (Zheng and Wang, 1998) is one of the models developed to add transport-modeling functionality to MODFLOW.

MODFLOW, with only a textual user interface also left a void for a graphical user interface (GUI), which would make it easier to use. The various add-ons also needed to be grouped for easier accessibility and use. PMWIN (Chiang and Kinzelbach, 2000) is a simulation system that gives MODFLOW the needed GUI and groups the add-ons conveniently together. Modeling is made a lot easier in this way. While PMWIN solves the interface and distributed character of the add-ons, the technical advancements in computer hardware has created an opportunity for a tool that will improve the understanding of groundwater modeling. The visualization tool developed for this thesis aims to do that.

Figure 3.1 depicts the visualization tool's environment. The modeler inputs data into PMWIN and interacts with it to get the modeled result. PMWIN uses, among other programs, MODFLOW and MT3DMS for groundwater flow and transport calculations. PMWIN, MODFLOW and MT3DMS produce output files and the visualization tool uses the information in these files for 3D visualization. The visualization tool can also make use of output from a file, containing x, y and z coordinates. The following sections give a

more detailed discussion on the different components of the visualization tool's environment.

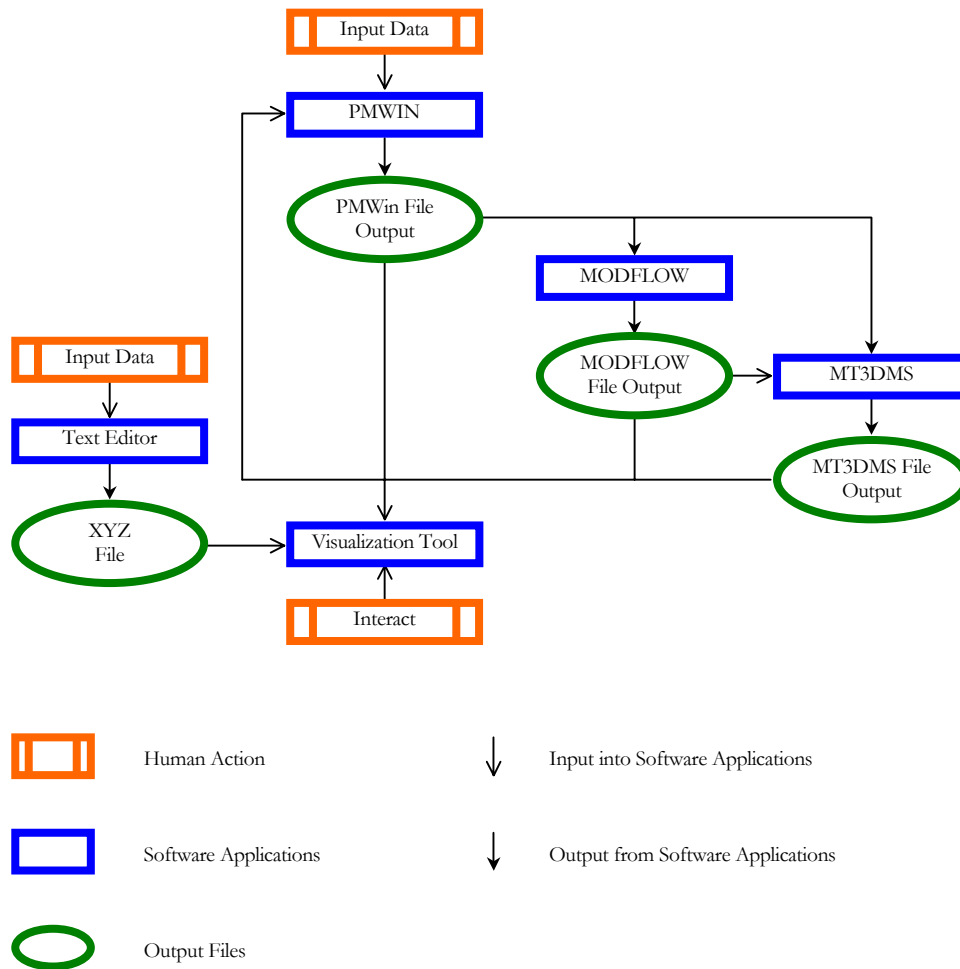


Figure 3.1: Visualization tool's environment.

## 3.2 MODFLOW

MODFLOW is a modular, 3D, finite-difference, groundwater model developed by the U.S. Geological Survey. With the first version of MODFLOW, it was possible to simulate the effect of wells, rivers, drains, head-dependent boundaries, recharge and evapotranspiration. Since the first release numerous investigators have developed various add-ons. These add-ons take the form of packages, models or sometimes simply programs. Packages are integrated into MODFLOW when it deals with a specific feature of the hydrologic system to be simulated, such as wells, recharge or a river. Models or programs can also be integrated or they can be used as stand-alone components with MODFLOW. A stand-alone model or program communicates with MODFLOW through data files (Chiang and Kinzelbach, 2000). MT3DMS uses such an approach.

MODFLOW, being a 3D groundwater model, and because it simulates flow in three dimensions, uses a block-centered, finite-difference approach to simulate groundwater flow within in an aquifer (McDonald and Harbaugh, 1998).

A real-world aquifer system is replaced in MODFLOW by a discretized domain consisting of finite difference blocks (cells). Figure 3.2 shows a spatial discretization of an aquifer system as a mesh of cells for which hydraulic heads are calculated. The grid forms the framework of the numerical model and one or more model layers can represent hydrostratigraphic units. The thickness of each model cell and the width of each column and row may be variable. The locations of cells are described in terms of columns, rows and layers. MODFLOW and PMWIN use an index notation  $[j, i, k]$  for locating cells. For example the cell located in the 5<sup>th</sup> column, 2<sup>nd</sup> row and the first layer is donated by  $[5, 2, 1]$  (McDonald and Harbaugh, 1998).

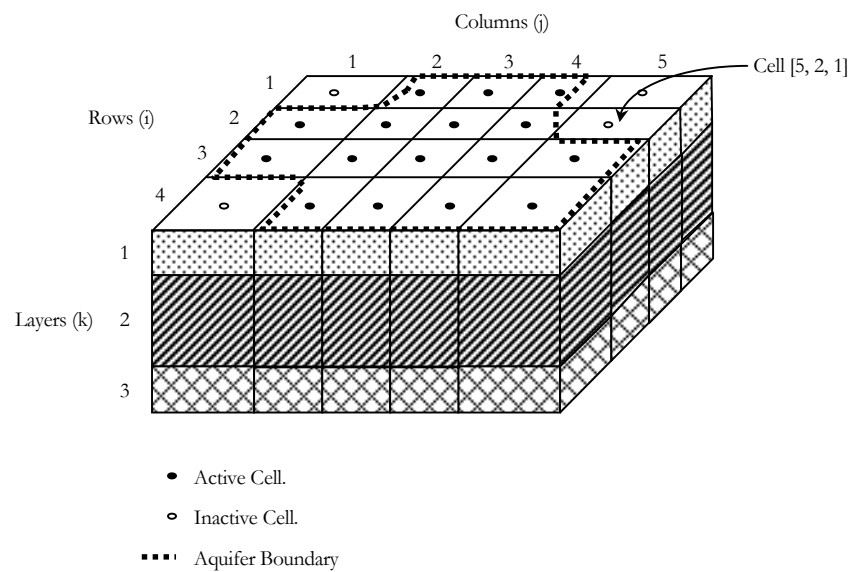


Figure 3.2: Discretized aquifer system.

MODFLOW solves the equations that describe 3D groundwater flow. The calculated result is groundwater heads for each cell in the finite difference grid. The heads are output to a file as discussed in the next section.

### 3.3 MODFLOW File Output

MODFLOW file output used by the visualization tool is the *heads.dat* file, which contains the calculated groundwater heads for different stress periods and time steps.

### 3.4 MT3DMS

MT3DMS is a **M**odular **3D** **M**ulti**S**pecies **T**ransport model for the simulation of advection, dispersion and chemical reactions of contaminants in groundwater systems and was developed for use with any block-centered, finite-difference flow model such as MODFLOW. Because MT3DMS makes use of the assumption that changes in the concentration field will not affect the flow field significantly, a flow model is developed and calibrated. The information needed by the transport model is saved to disk files and is only retrieved by the transport model when needed. In short, it allows the user to construct and calibrate a flow model independently. After the flow simulation is complete, MT3DMS simulates solute transport by using the calculated heads and various flow terms saved by MODFLOW. Since most potential users of a transport model are likely to have been familiar with one or more flow models, MT3DMS provides an opportunity to simulate contaminant transport without the need to learn a new flow model or to modify an existing flow model to fit the transport model. In addition, separate flow simulation and calibration outside the transport model result in substantial savings in computer memory (Zheng and Wang, 1998).

### 3.5 MT3DMS File Output

The result of transport simulation is a concentration for each discretized cell as discussed in section 3.2 “MODFLOW”. Different species concentrations are outputted for specified elapsed times. The output of MT3DMS are *mt3d???ucn* files. The question marks designate a species number from 001 to 030. A \*.ucn file therefore contains the cell concentrations for a specific species at specified elapsed times.



### 3.6 PMWIN

Processing Modflow for Windows (PMWIN) is a simulation system for modeling groundwater flow and pollution with MODFLOW-88, MODFLOW-96, PMPATH, MT3D, MT3DMS, MOC3D, PEST and UCODE. The PMWIN system has a graphical user interface that supports models and programs and other useful modeling tools (Chiang and Kinzelbach, 2000).

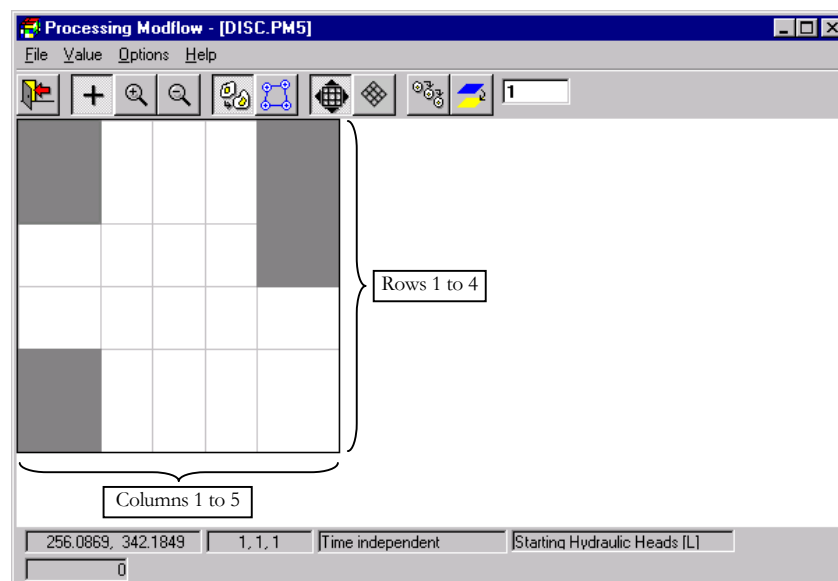


Figure 3.3: PMWIN model.

Figure 3.3 shows how the discretized aquifer system of MODFLOW in Figure 3.2 will look in PMWIN. The gray cells are the inactive cells. Note that only the top view of the first layer 1 is visible.

### 3.7 PMWIN File Output

PMWIN saves most of the user-specified data in binary files, by using the model name as the filename and the extension given in the following tables (Chiang and Kinzelbach, 2000). Only the files that the visualization tool uses are listed.

Table 3.1: PMWIN main file.

File Extension	Description
<i>pm5</i>	Main file of PMWIN, used to save model data.

Table 3.2: PMWIN geometry and boundary conditions files.

File Extension	Description
<i>top</i>	Top elevation of layers.
<i>bot</i>	Bottom elevation of layers.
<i>ibd</i>	IBOUND matrix used by MODFLOW.
<i>tic</i>	ICBOUND matrix used by MT3DMS.

Table 3.3: PMWIN hydraulic components files.

File Extension	Description
<i>ibd</i>	Fixed head cell, when value < 0.
<i>tic</i>	Fixed concentration cell, when value < 0.
<i>ghb</i>	General head boundary.
<i>wel</i>	Discharge and Recharge wells.
<i>drc</i>	Drain.
<i>ric</i>	River.
<i>Wac</i>	Horizontal flow barrier.
<i>c85</i>	Reservoir.
<i>cb1</i>	Time variant specified head.
<i>c55</i>	Time variant specified concentration.

Table 3.4: PMWIN parameter files.

File Extension	Description
<i>con</i>	Horizontal Hydraulic Conductivity.
<i>lea</i>	Vertical Hydraulic Conductivity.
<i>sto</i>	Specific Storage.
<i>btc</i>	Transmissivity.
<i>lkn</i>	Vertical Leakance.
<i>scv</i>	Storage Coefficient.
<i>por</i>	Effective Porosity.
<i>yld</i>	Specific Yield.

### 3.8 XYZ File

The xyz file contains real-world x, y and z coordinates from which a polygonal mesh is to be created in the visualization tool. The x, y and z coordinates are inputted into a text editor as seen in Figure 3.4. The first column contains the x coordinates, the second the y coordinates and the third the z coordinates.

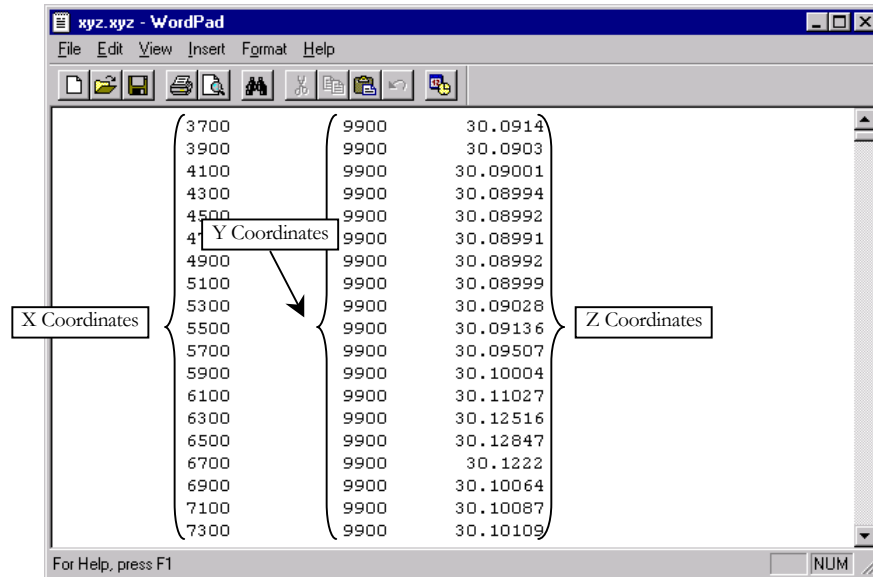


Figure 3.4: An example of an XYZ file edited in WordPad.

### 3.9 Visualization Tool

The visualization tool uses the file output of PMWIN, MODFLOW, MT3DMS and the XYZ file, for visualization. Chapter 5 discusses how the information contained in these files is visualized. Figure 3.5 shows what the discretized aquifer system displayed in Figure 3.2 can look like, using the visualization tool. The visualization tool and its different components are shown in Figure 3.6.

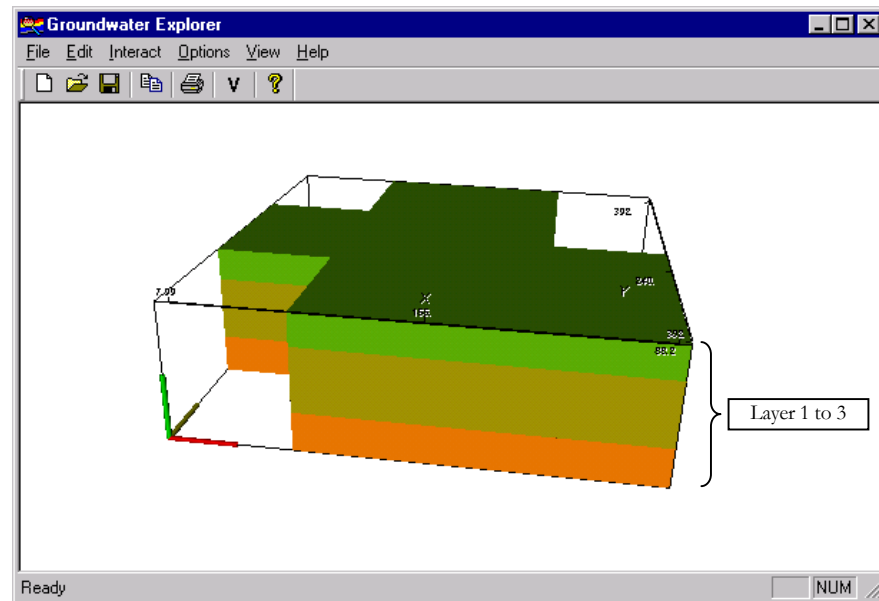


Figure 3.5: Discretized aquifer system visualized in the visualization tool.

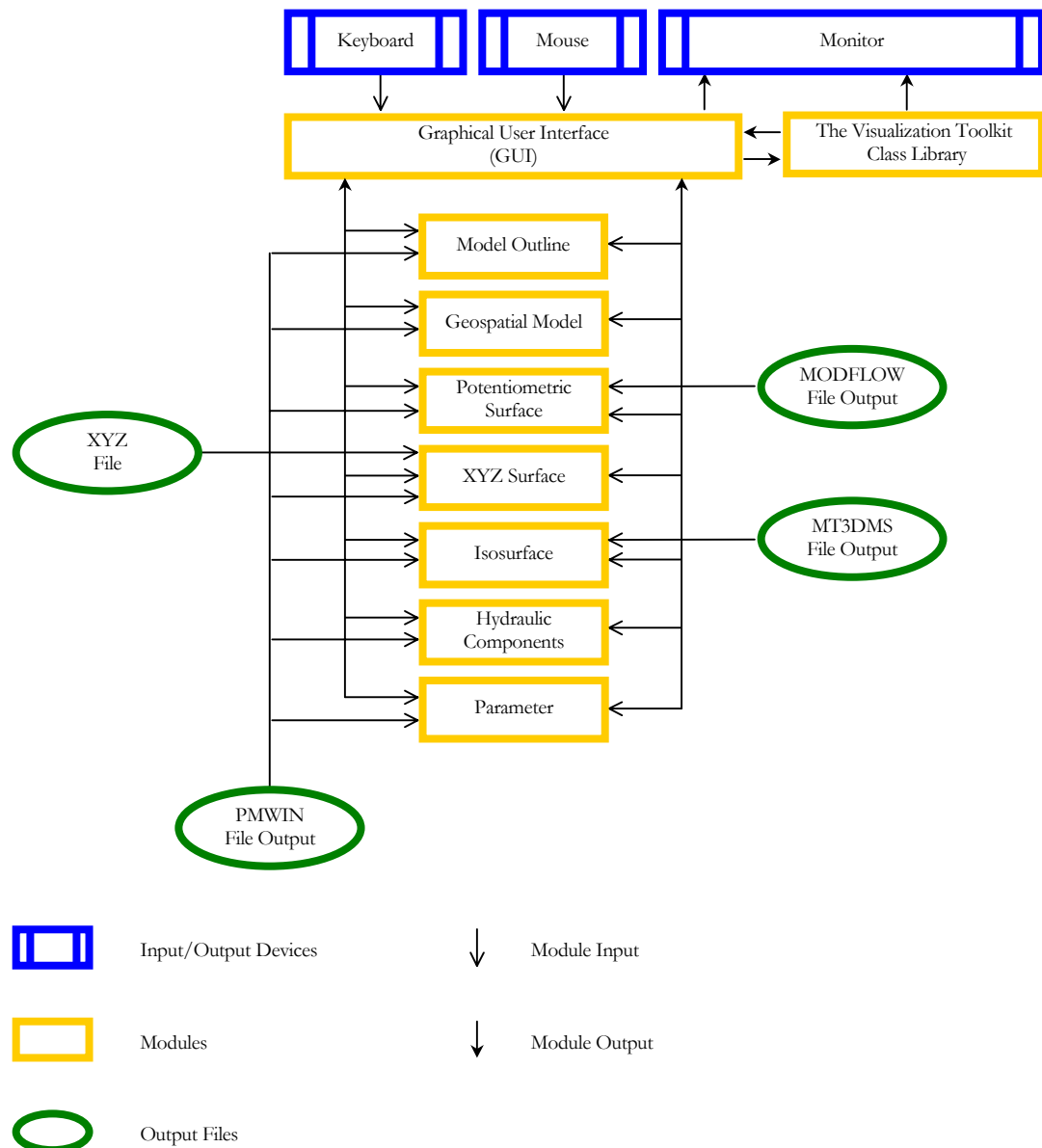


Figure 3.6: Visualization tool.

From Figure 3.6 it can be seen that the user of the visualization tool interacts with it by means of the keyboard, mouse and monitor. The GUI is windows based. The visualization tool makes use of a graphics library called the Visualization Toolkit (vtk) and is discussed in the following chapter. The visualization tool consists of the Model Outline, Geospatial Model, Potentiometric Surface, XYZ Surface, Isosurface, Hydraulic Components and Parameter program modules. These program modules, their design and the visualization objects that result from them, are discussed in Chapter 5.



# Chapter 4

---

## The Visualization Toolkit Class Library

### 4.1 Introduction

The Visualization Toolkit (vtk) was used in the development of the visualization tool, the software result of this thesis. Vtk is a free, open-source, portable (WinTel/Unix), object-oriented C++ class library for visualization, image processing and 3D computer graphics (Schroeder *et al.*, 1998a; Schroeder *et al.*, 1998c and Schroeder *et al.*, 2000).

For the development of the visualization tool a good understanding of how vtk works was necessary in order to put it to effective use. This brings us to the goal of this chapter, which is to describe and explain vtk and its underlying objects; more specifically the objects that were used in the visualization tool. The chapter starts with important design goals set out for vtk by its creators. The remaining part of the chapter is structured as seen in Figure 4.1, in which vtk's working is summarized. *Information* is used as input into the *visualization* or *imaging pipeline* or both pipelines, and undergoes changes as it flows through the pipeline. The word pipeline is used to describe a data-flow process that can take two or more distinct stages or steps. The result that flows from the other side of the pipeline is *graphical data* and it is used as input to a *graphics block box*, which almost magically produces a picture on the computer monitor.

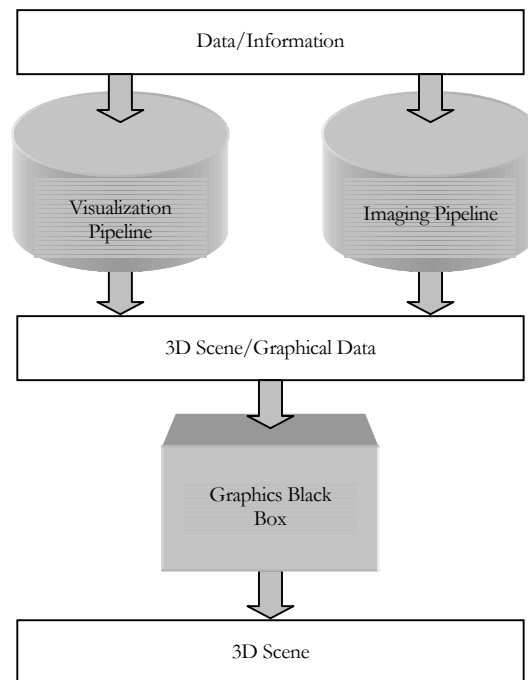


Figure 4.1: The Visualization Toolkit (vtk).

## 4.2 Goals

The creators of vtk used their experience from previous work and their dealings with users of visualization systems in setting out design goals for vtk. One of the design goals was to make it easy for applications that use vtk to remain flexible. This goal, by keeping in mind that large monolithic systems tend to be detrimental to software flexibility, was met by making use of a toolkit philosophy. Toolkits enable complex applications to be built from small pieces. Figure 4.2 illustrates the idea. Vtk was therefore developed as a sharply focused object library that can easily be embedded and distributed into applications.

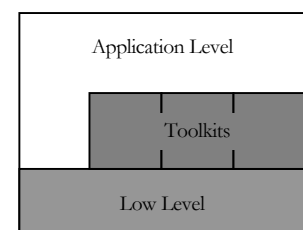


Figure 4.2: Application built with toolkits (Schroeder *et al.*, 1998a).

Another design goal implemented was to keep vtk portable. Vtk's creators have been skeptical that any given graphics library will become a standard. The idea was a high-level abstraction for 3D graphics that would be independent from new graphic libraries. This idea makes it easy for applications using vtk to be ported as new standards become



available. Going along with this was the goal to keep the core system independent of differing windowing systems on Unix, PC and other platforms (Schroeder *et al.*, 1998a).

The purpose of making vtk's source code freely available was to get vtk widely used and supported. This has the benefit of an open-source software development (the bazaar model) which is inherently more scalable than a closed-team development (the cathedral model). Bugs can be discovered and fixed faster, algorithms are improved, and valuable suggestions are received from users of vtk (Schroeder *et al.*, 1998a and Schroeder *et al.*, 2000).

All of these goals have helped vtk to develop credibility with its users in the graphics and visualization fields. The added benefit of it being free makes it an excellent tool for educational and research purposes (Schroeder *et al.*, 1998a).

## 4.3 Visualization Pipeline

We've seen from Figure 4.1 that the role of the visualization pipeline is to transform information into graphical data. In the pipeline modules are connected into a network. The modules perform algorithmic operations on information as it flows through the visualization network. The execution of the network is controlled in response to demands for data or in response to user input (Schroeder *et al.*, 1998a).

The visualization pipeline consists of two basic types of objects: *data objects* and *process objects*. Data objects represent and enable operations on the information that flows through the network and process objects are the modules or algorithmic portions of the visualization network. They are further classified into one of three types: sources, filters and mappers (Schroeder *et al.*, 1998a).

### 4.3.1 Data Objects

Many types of information or data are produced by the variety of fields that apply visualization. This brings us to the purpose of data objects which are to represent information in the visualization pipeline. Data objects in the visualization pipeline are called datasets (Schroeder *et al.*, 1998b).

Figure 4.3 shows that a dataset consists of an organizing structure and attributes. The organizing structure itself consists of two components, namely topology and geometry. Topology or cells of a dataset can be defined as a set of properties that are invariant to certain geometric transformations such as rotation, translation and nonuniform scaling. Put more simply, cells can be thought of as the building blocks of datasets. Geometry or points is the instantiation of topology, the specification of position in 3D space. For example, saying that a polygon is a voxel, specifies topology. Providing the point coordinates of the voxel specifies geometry (Schroeder *et al.*, 1998b). From this it is easy to understand that the definition of the organizing structure of a dataset has resulted from the discrete nature of data that are used for visualization. Geometry or points are located where data is known and topology or cells allow interpolation between points (Schroeder *et al.*, 1998b).

Attributes, the other component of a dataset, are supplemental information associated with geometry and/or topology. Typical attribute data are scalars, vectors, normals, texture coordinates, tensors and user-defined attributes (Schroeder *et al.*, 1998b).

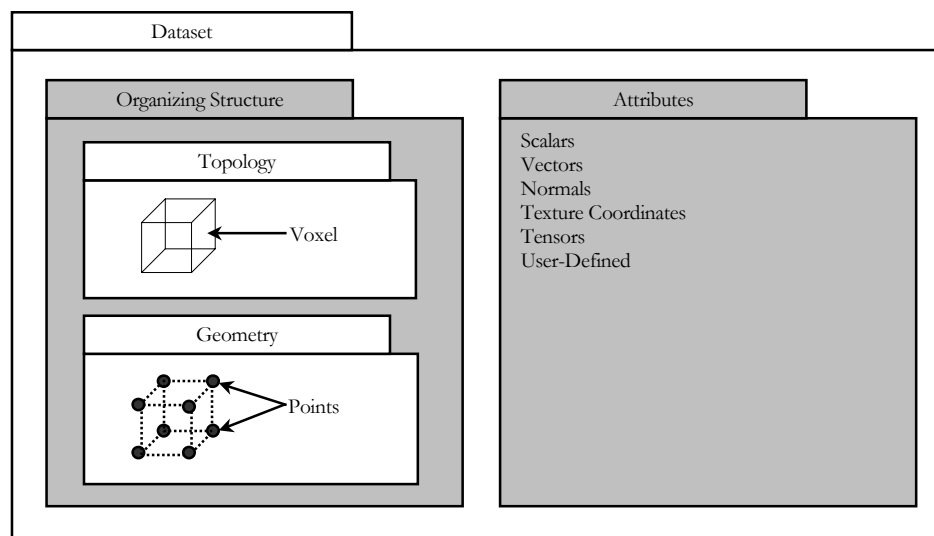


Figure 4.3: Dataset objects.

Vtk defines twelve types of cells, but sometimes, because of interpolation problems, the need arises to define new cell types. Figure 4.4 shows vtk's twelve cell types.

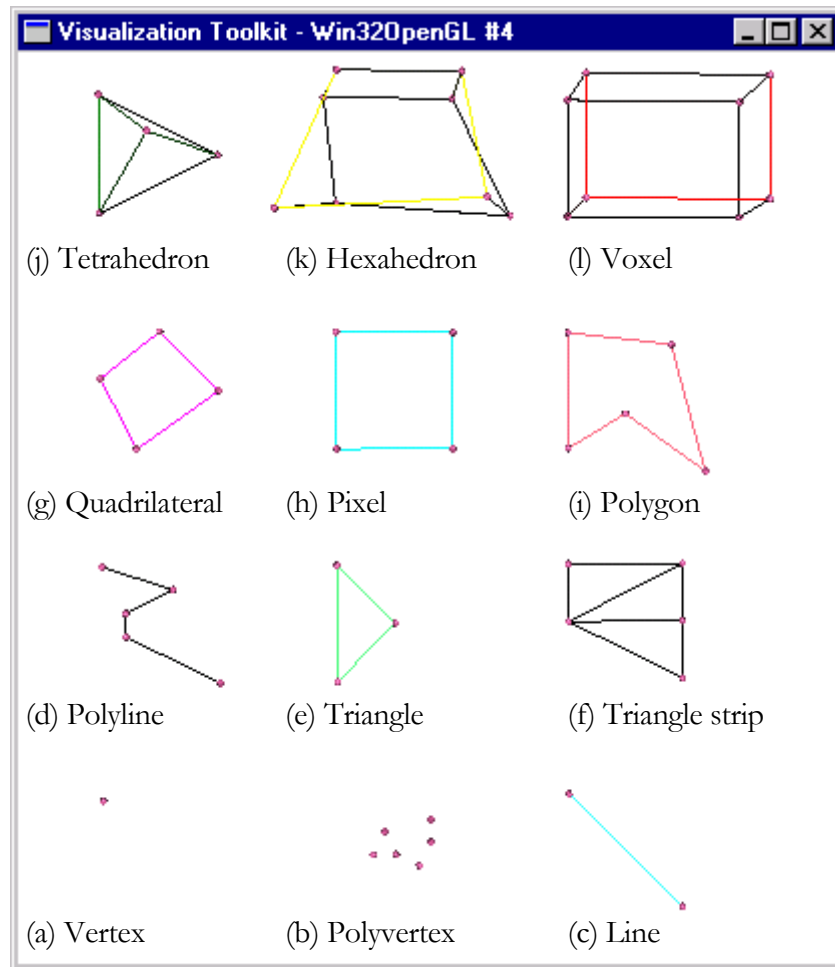


Figure 4.4: Vtk cell types.

A vertex is a primary zero-dimensional (0D) cell and a polyvertex is a composite 0D cell, consisting of more than one vertex (Schroeder *et al.*, 1998b).

The 1D cells consist of a line and a polyline cell. This is a composite 1D cell, consisting of one or more connected lines (Schroeder *et al.*, 1998b).

A triangle is a primary 2D cell, and a triangle strip is a composite 2D, consisting of one or more triangles. A quadrilateral is also a primary 2D cell which is convex. Its edges do not intersect. Another primary 2D cell is a pixel, with all its edges perpendicular to its adjacent edges. The edges are perpendicular to one of the coordinate axes  $x$ ,  $y$  or  $z$ . A polygon, the last of the 2D cells, is also a primary 2D cell that can be nonconvex but it is without internal loops and it cannot self-intersect (Schroeder *et al.*, 1998b).

Tetrahedron, a primary 3D cell, has six edges and four triangular faces. A hexahedron, also a primary 3D cell, consists of six quadrilateral faces, twelve edges and eight vertices. The faces and edges should not intersect with any other faces and edges, and the hexahedron must be convex. The primary 3D cell voxel has topology equivalent to that of a hexahedron, but with additional geometric constraints. Each face of the voxel is perpendicular to one of the coordinate axes (Schroeder *et al.*, 1998b).

The datasets that are built from these cells are: structured points, rectilinear grid, structured grid, unstructured points, polygonal data and unstructured grid datasets.

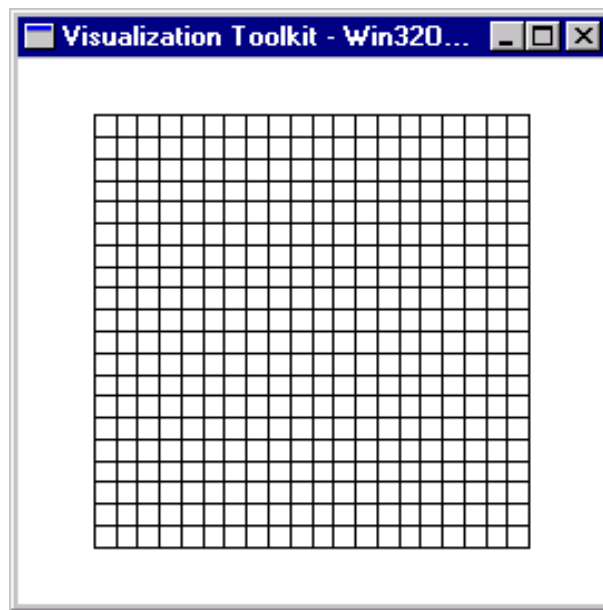


Figure 4.5: Structured points.

A structured points dataset (Figure 4.5) “is a collection of points and cells arranged on a regular, rectangular lattice. The rows, columns and planes of the lattice are parallel to the global x-y-z coordinate system.” (Schroeder *et al.*, 1998b).

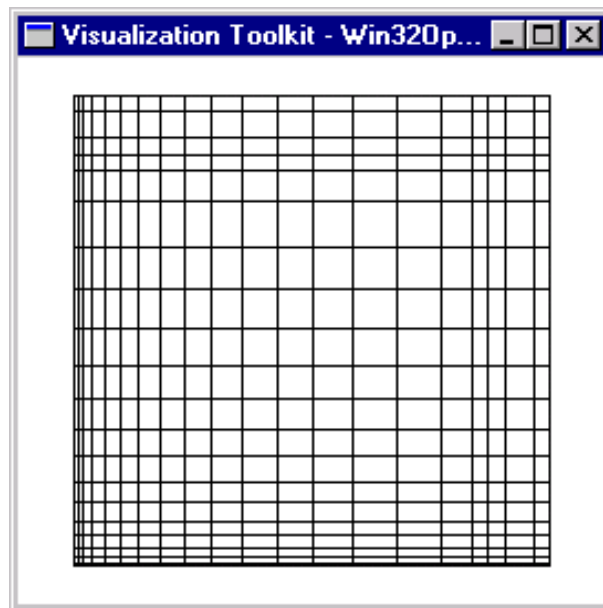


Figure 4.6: Rectilinear grid.

A rectilinear grid dataset (Figure 4.6) “is a collection of points and cells arranged on a regular lattice. The rows, columns and planes of the lattice are parallel to the global x-y-z coordinate system. While the topology of the dataset is regular, the geometry is only partially regular. That is, the points are aligned along the coordinate axis, but the spacing between the points may vary.” (Schroeder *et al.*, 1998b).

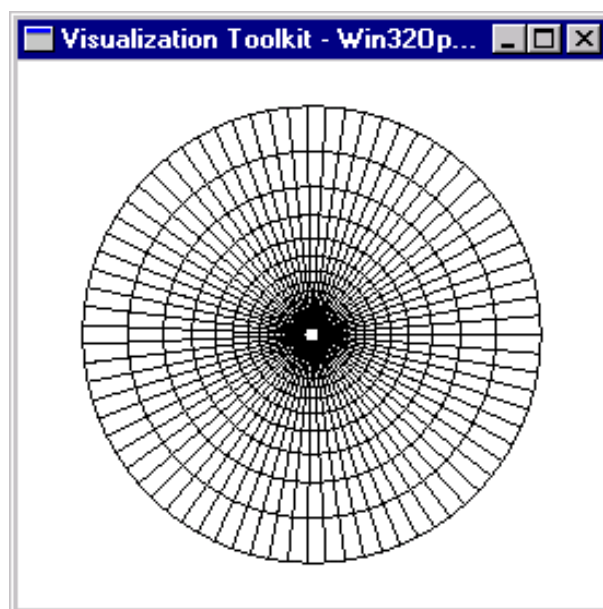


Figure 4.7: Structured grid.

A structured grid dataset (Figure 4.7) “is a dataset with regular topology and irregular geometry. The grid may be warped into any configuration in which the cells do not overlap or self-intersect.” (Schroeder *et al.*, 1998b).

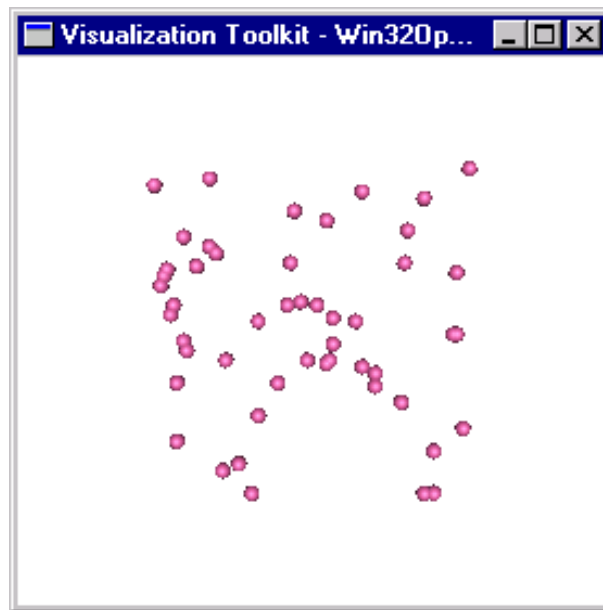


Figure 4.8: Unstructured points.

Unstructured points (Figure 4.6) “are points irregularly located in space. There is no topology in an unstructured point dataset and the geometry is completely unstructured. The vertex and polyvertex cells are used to represent unstructured points.” (Schroeder *et al.*, 1998b).

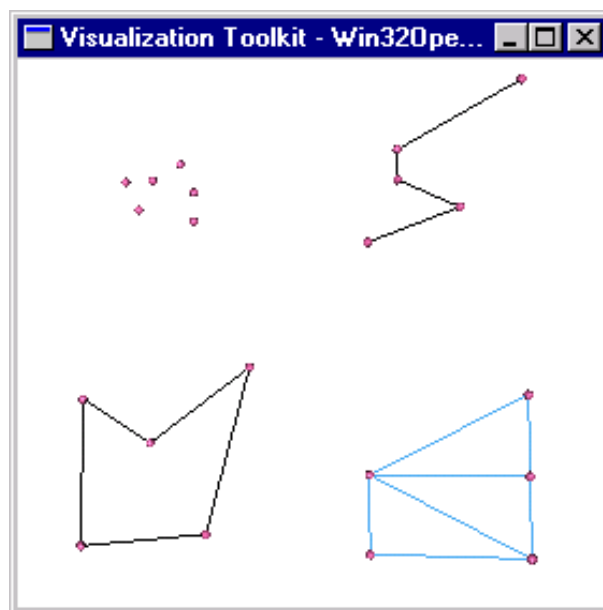


Figure 4.9: Polygonal data.

Polygonal data (Figure 4.9) “consists of vertices, polyvertices, lines, polylines, polygons and triangle strips.” (Schroeder *et al.*, 1998b).

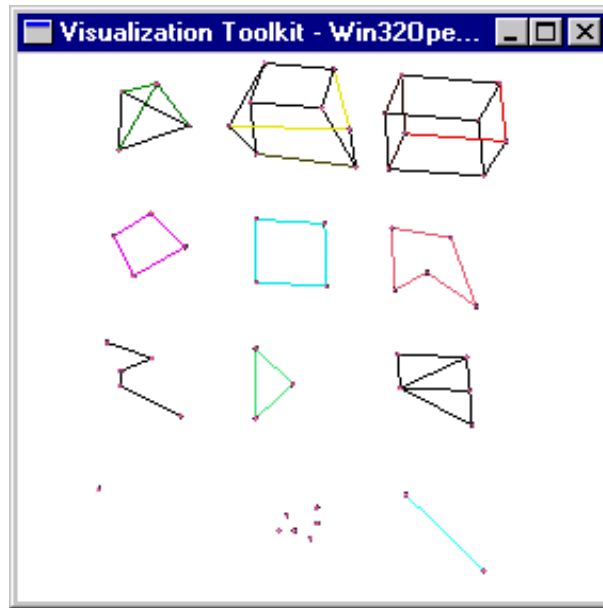


Figure 4.10: Unstructured grid.

The last of the datasets, unstructured grid (Figure 4.10), is a dataset of which “the topology and geometry are completely unstructured. Any cell type can be combined in arbitrary combinations in an unstructured grid.” (Schroeder *et al.*, 1998b).

### 4.3.2 *Process Objects*

As mentioned in section 4.3, “*Visualization Pipeline*”, the process objects are the modules or algorithmic portions of the visualization network and are further classified into one of three types: sources, filters and mappers (Schroeder *et al.*, 1998a).

#### 4.3.2.1 *Sources*

Source objects initiate the visualization network and generate one or more output datasets (Schroeder *et al.*, 1998a).

#### 4.3.2.2 *Filters*

“Filters require one or more inputs and generate one or more outputs.” (Schroeder *et al.*, 1998a). The different kinds of filters are not explained in detail for the purpose of this

thesis, only a short description is given of what they do. If possible, the visual effect of applying them are shown in figures.

#### **4.3.2.2.1 Append Polygonal Data**

Append polygonal data is a “filter that appends one or more polygonal datasets into a single polygonal dataset. All geometry is extracted and appended, but point attributes (i.e. scalars, vectors, normals) are extracted and appended only if all datasets have the same point attributes available. (For example, if one dataset has scalars but another does not, scalars will not be appended.)” (Stifter, 2000).

The v, t and k characters in Figure 4.11 were polygonal datasets that were appended into a single polygonal dataset.



Figure 4.11: Example of append polygonal data filter being used.

#### **4.3.2.2.2 Clean Polygonal Data**

Clean polygonal data is a “filter that takes polygonal data as input and generates polygonal data as output.” Clean Polygonal Data also “merges duplicate points (within specified tolerance) and transforms degenerated topology into appropriate form (for example, triangle is converted into line if two points of triangle are merged).” (Stifter, 2000).



#### 4.3.2.2.3 *Clip Polygonal Data*

Clip polygonal data is a filter that clips polygonal data using either an implicit function or input scalar data. “Clipping means that it actually "cuts" through the cells of the dataset, returning everything inside of the specified implicit function (or greater than the scalar value) including "pieces" of a cell.” (Stifter, 2000).

The front, top part of the cow in Figure 4.12 is clipped away using a plane. The part of the cow which is clipped away is represented as wireframe.

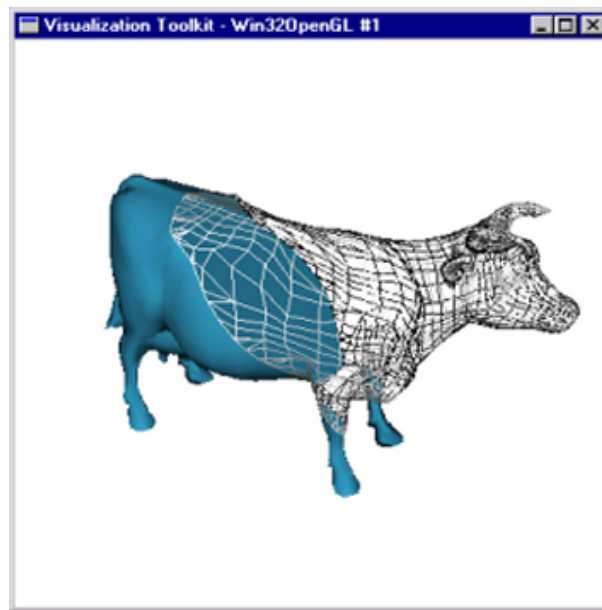


Figure 4.12: Example of clip polygonal data filter being used.

#### 4.3.2.2.4 *Contouring*

Contouring is a “filter that takes as input any dataset and generates on output isosurfaces and/or isolines. The exact form of the output depends upon the dimensionality of the input data. Data consisting of 3D cells will generate isosurfaces; data consisting of 2D cells will generate isolines; and data with 1D or 0D cells will generate isopoints. Combinations of output type are possible if the input dimension is mixed.” (Stifter, 2000).

In Figure 4.13 isosurfaces at  $F(x, y, z) = c$  for different  $c$  values are created from the function  $F(x, y, z) = x^2 + 2y^2 + 3z^2 + yz$ .

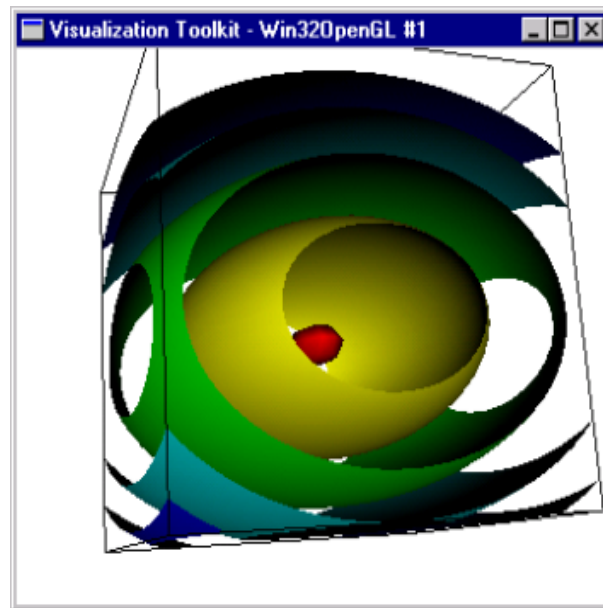


Figure 4.13: Example of contour filter being used.

#### 4.3.2.2.5 Decimation

Decimation is a “filter to reduce the number of triangles in a triangle mesh, forming a good approximation to the original geometry.” The input is polygonal data and only triangles are treated. If polygonal meshes are to be decimated, they first have to be triangulated with the Triangulation filter (Stifter, 2000).

Figure 4.14 shows the effect of decimation on the man’s face. Remember that the idea is to reduce the number of triangles and to keep a good approximation of the original geometry. The required reduction increases from the face in the top right-hand corner, in an anti-clockwise direction, to the face in the bottom right-hand corner. The wireframe representations in Figure 4.15 of the faces in Figure 4.14, clearly show the reduction in triangles.

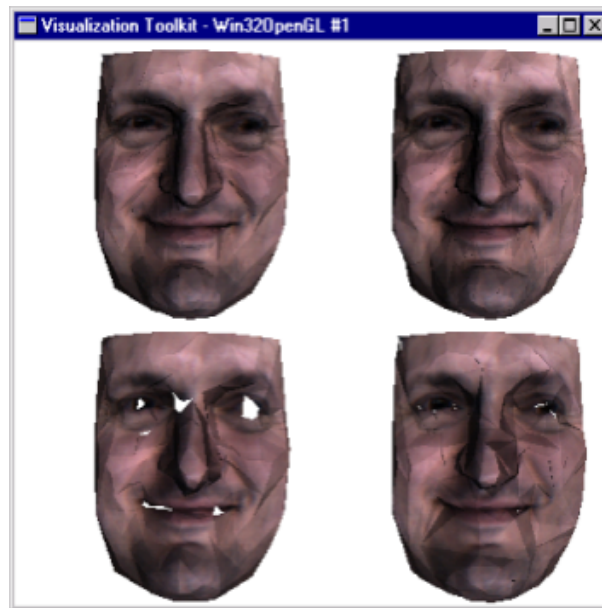


Figure 4.14: Example of decimation filter being used.

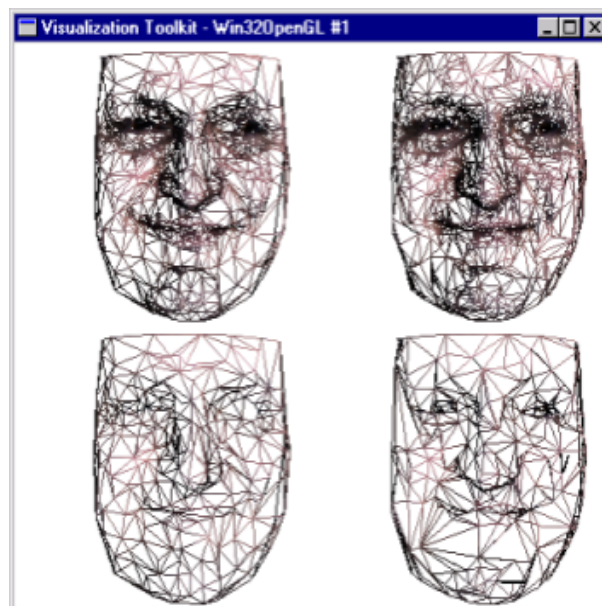


Figure 4.15: Wireframe example of decimation filter being used.

#### 4.3.2.2.6 *Delaunay*

Delaunay is a filter that constructs a 2D triangle mesh from a list of input points, as seen in Figure 4.16 (Stifter, 2000).

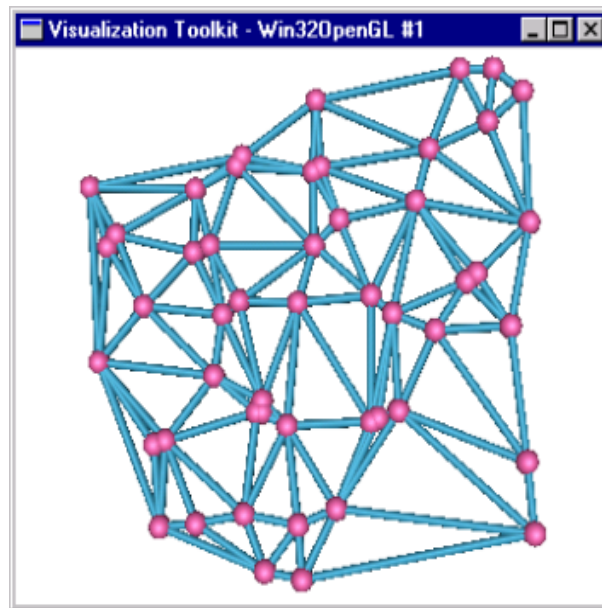


Figure 4.16: Example of delaunay filter being used.

#### **4.3.2.2.7 Geometry**

Geometry is a “general-purpose filter to extract geometry (and associated data) from any type of dataset.” The geometry filter can also be used to convert any type of data to polygonal data (Stifter, 2000).

#### **4.3.2.2.8 Outline**

Outline is a “filter that generates a wireframe outline of any dataset. The outline consists of the twelve edges of the dataset bounding box.” (Stifter, 2000). Figure 4.17 shows a sphere and its outline.

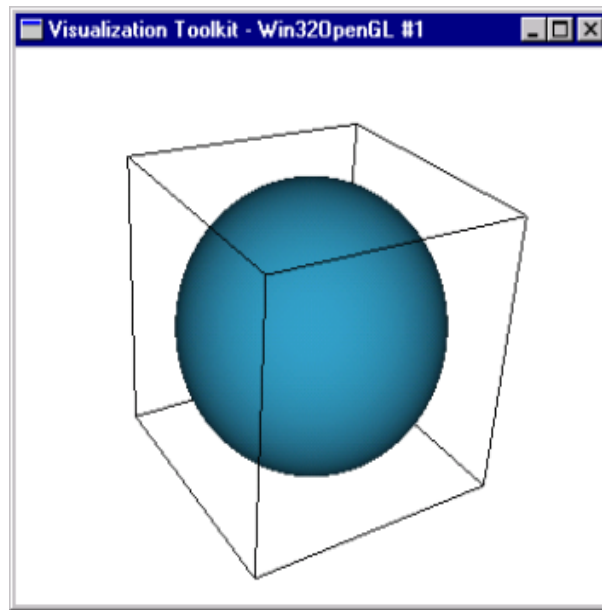


Figure 4.17: Example of outline filter being used.

#### **4.3.2.2.9 Polygonal Data Normals**

Polygonal data normals is a “filter that computes point normals for a polygonal mesh. The filter can reorder polygons to insure consistent orientation across polygon neighbors. Sharp edges can be split and points duplicated with separate normals to give crisp (rendered) surface definition. It is also possible to globally flip the normal orientation.” (Stifter, 2000).

The cones on the man’s face in Figure 4.18 shows the direction of the polygonal data normals at certain positions on the man’s face. These normals are used to produce a smoother-looking mesh. Figure 4.19 shows an object rendered without using polygonal normals and Figure 4.20 shows the smoother-looking result.

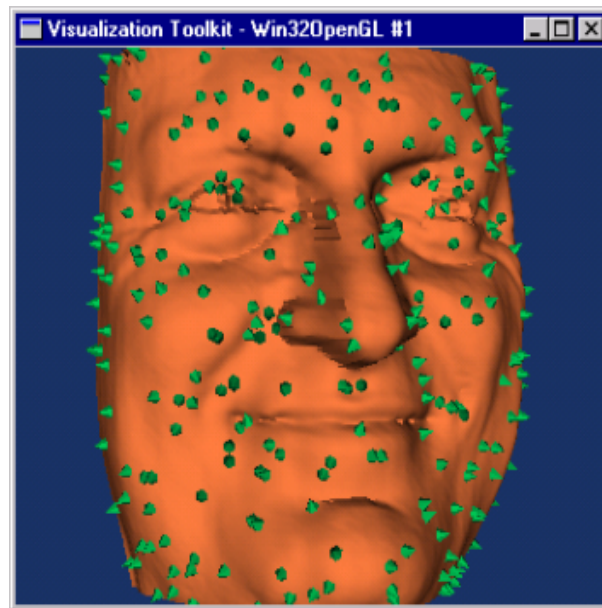


Figure 4.18: Direction of normals shown by cones.

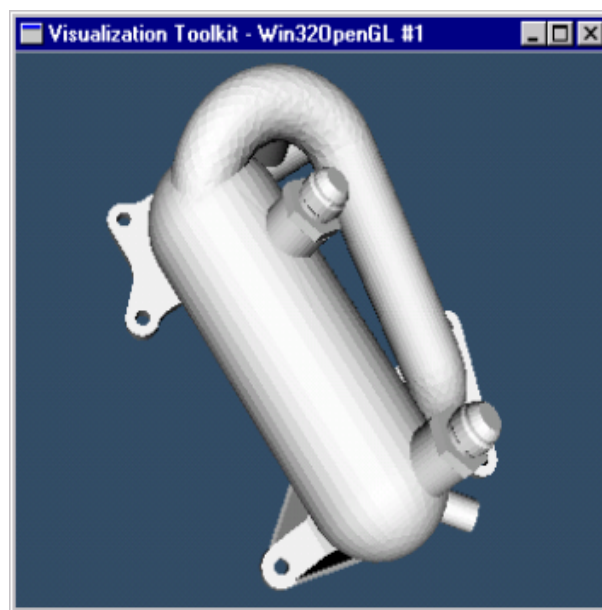


Figure 4.19: Example of poly data normals not used.

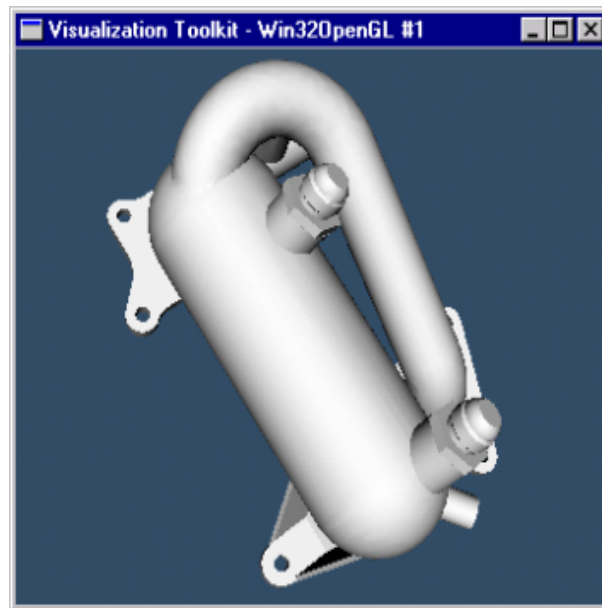


Figure 4.20: Example of poly data normals used.

#### ***4.3.2.2.10 Smooth Polygonal Data***

Smooth polygonal data is a “filter that adjusts point coordinates using Laplacian smoothing. The effect is to “relax” the polygonal mesh, making the cells better shaped and the vertices more evenly distributed.” (Stifter, 2000).

Figure 4.21 shows part of a cylinder shape object with an irregular surface. Figure 4.22 is the same cylinder shape object, after the smooth polygonal data filter has been used on it.

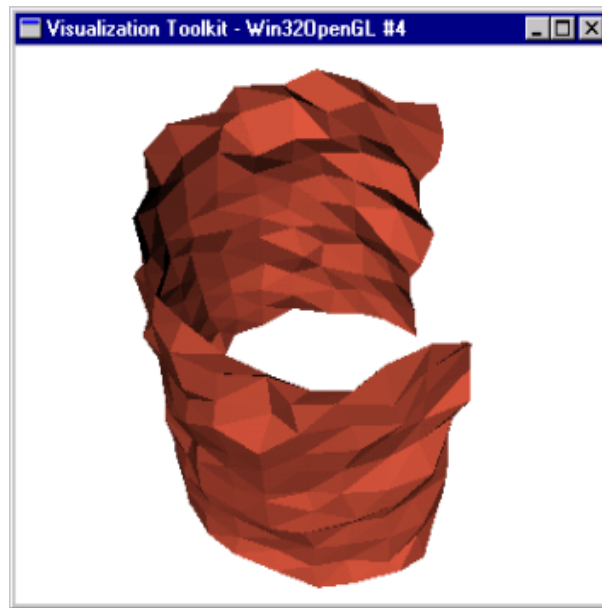


Figure 4.21: Cylinder not smoothed.

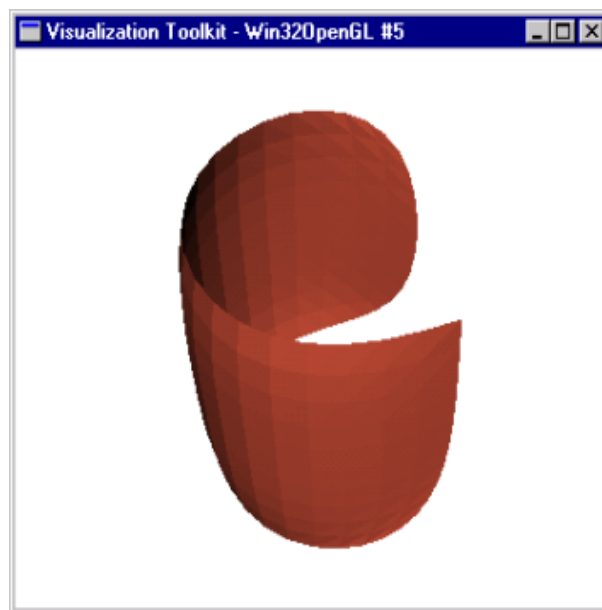


Figure 4.22: Cylinder smoothed.

#### 4.3.2.2.11 *Triangulation*

Triangulation “generates triangles from input polygons and triangle strips.” (Stifter, 2000).

Figure 4.24 shows the triangulated result of the polygons in Figure 4.23.



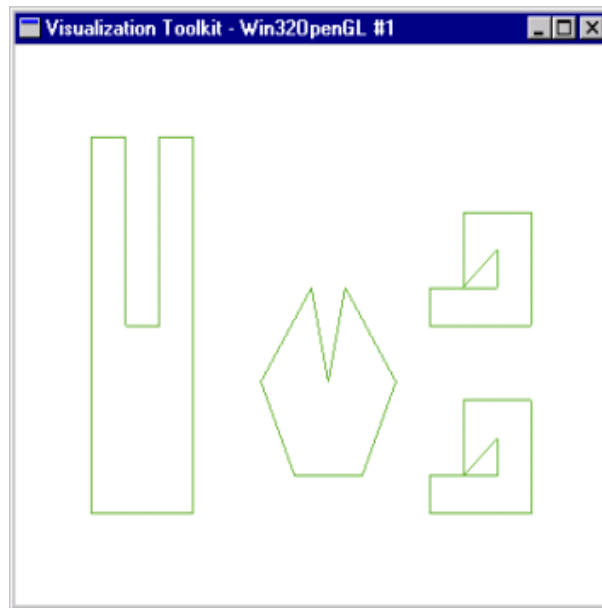


Figure 4.23: Polygons not triangulated.

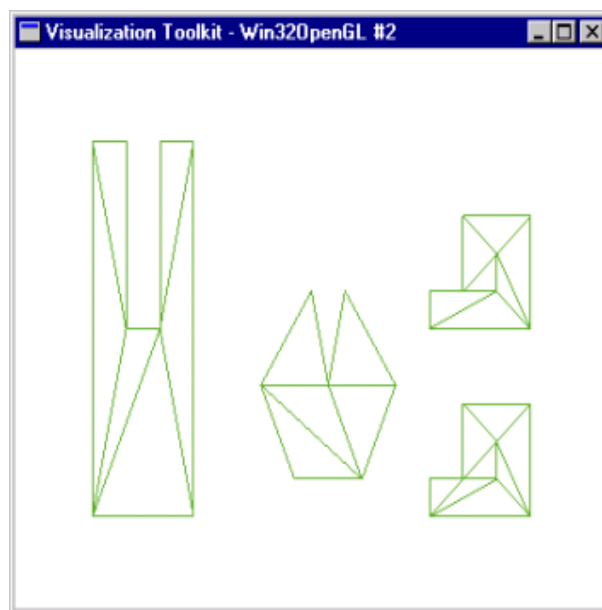


Figure 4.24: Polygons triangulated.

#### 4.3.2.3 Mappers

Mappers which require one or more inputs, terminate the visualization network (Schroeder *et al.*, 1998a)

## 4.4 Imaging Pipeline

The imaging pipeline, as seen in Figure 4.1, is a specialized implementation of the visualization pipeline designed to support specific types of data, especially structured point datasets that are used to represent imaging information or data.

## 4.5 Graphics Black Box

We have seen from Figure 4.3 that the role of the graphics black box is to transform graphical data into pictures (Schroeder *et al.*, 1998c). The graphics black box is an abstract model for 3D computer graphics, with the abstraction based on the movie-making industry, and with influences from windowing based graphical user interfaces (Schroeder *et al.*, 1998a). This movie-making abstraction for 3D computer graphics makes it easy to use and understand.

The graphics black box does not create the picture by making use of magic alone, but also uses the objects that it is built from. The render window, render window interactor, render, light, camera, actor, property, mapper, lookup table and transform are the core objects that make up the graphics black box (Schroeder *et al.*, 1998c). In short an actor represents graphical data or objects, lights illuminate the actors and camera constructs a picture by projecting the actors onto a view plane. We call the combination of lights, camera and actors the scene and refer to the rendering process as rendering the scene (Schroeder *et al.*, 1998b).

### 4.5.1 *Render Window*

The render window is a window on the display window that has one or more regions, called viewports, into which one or more renderers draw to create a scene. Multiple render windows can be created (Schroeder *et al.*, 1998a and Schroeder *et al.*, 1998c).

### 4.5.2 *Render Window Interactor*

Once a scene is created, you want to interact with it. Vtk offers two approaches. The first is to create your own interactor by specifying event bindings. The second is to use the

render window interactor, which is an object for manipulating the camera, picking objects, invoking user-defined methods, entering and exiting stereo viewing, and changing some of the properties of actors (Schroeder *et al.*, 1998c).

### **4.5.3      *Renderer***

The renderer draws into a render window and together they manage the interface between vtk and the computer's windowing system (Schroeder *et al.*, 1998c). The renderer also coordinates the rendering of lights, cameras and actors (Schroeder *et al.*, 1998a).

### **4.5.4      *Light***

A light or lights are used to illuminate actors in a 3D scene. A scene will be dark and uninformative if not illuminated. Lights are also used to manipulate lighting properties such as brightness and color. The color of actor which we perceive is the composite result of the rays of light emitted by the surface properties of the actor (Schroeder *et al.*, 1998a; Schroeder *et al.*, 1998b and Schroeder *et al.*, 1998c).

### **4.5.5      *Camera***

The camera object controls how a 3D scene is projected onto a plane, to form a 2D image during the rendering process. The camera has several methods for positioning, pointing and orienting it. In addition, the camera controls perspective projection and stereo viewing (if enabled). Cameras are not needed for 2D imaging data (Schroeder *et al.*, 1998c).

### **4.5.6      *Actor***

An actor is an object rendered by a renderer in the scene. "Actors are defined in terms of mapper, property and a transform objects." (Schroeder *et al.*, 1998a). Vtk makes use of two types of actors. One actor is always visible and the other, which stores multiple levels of detail, can automatically switch between them. It selects which level of detail to use, based on how much time has been allocated to render (Schroeder *et al.*, 1998c).

### **4.5.7     *Property***

The property object represents the rendered attributes of an actor such as object color, lighting, texture map, and drawing and shading styles (Schroeder *et al.*, 1998a).

### **4.5.8     *Mapper***

The mapper provides the interface between the visualization pipeline and the graphics black box. It is also used to color actors by mapping information through a lookup table (Schroeder *et al.*, 1998c).

### **4.5.9     *Lookup Table***

An important technique used in visualization is color mapping. The idea behind this technique is to map a data value through a lookup table to obtain a color. The color is applied during rendering to points or cells, which are the building blocks of an actor (Schroeder *et al.*, 1998c).

### **4.5.10    *Transform***

A transform object consists of  $4 \times 4$  transformation matrices and methods to manipulate translation, scale and rotation. The transform object is also used to specify the position and orientation of actors, cameras and lights (Schroeder *et al.*, 1998a and Stifter, 2000).

## **4.6        Putting It Together**

The goal of this chapter was to introduce and explain how vtk, the visualization class library which was used in the development of the visualization tool, works. This chapter is also necessary for explaining the different modules of the visualization tool in the next chapter. The final idea is to put the whole chapter together with a program example that produces the picture as seen in Figure 4.25.

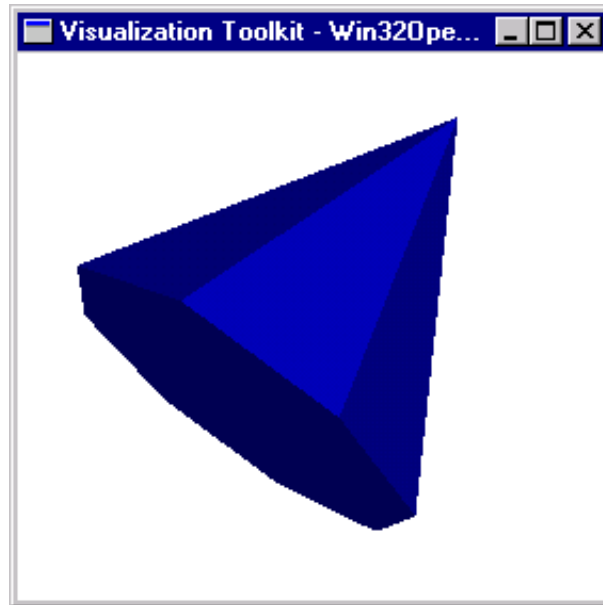


Figure 4.25: Cone.

Listing 4.1: Program code for Cone example.

```

001 #include "vtkActor.h"
002 #include "vtkConeSource.h"
003 #include "vtkPolyDataMapper.h"
004 #include "vtkRenderer.h"
005 #include "vtkRenderWindow.h"
006 #include "vtkRenderWindowInteractor.h"
007
008 void main(int iArgument, char* pcArgument[])
009 {
010     // visualization pipeline
011     vtkConeSource* pConeSource = vtkConeSource::New();
012     pConeSource->SetResolution(8);
013
014     vtkPolyDataMapper* pPolyDataMapper = vtkPolyDataMapper::New();
015     pPolyDataMapper->SetInput(pConeSource->GetOutput());
016
017     // create actor of graphics black box
018     vtkActor* pActor = vtkActor::New();
019     pActor->SetMapper(pPolyDataMapper);
020     pActor->GetProperty()->SetColor(0, 0, 1);
021
022     // initialize the graphics black box
023     vtkRenderer* pRenderer = vtkRenderer::New();
024     pRenderer->SetBackground(1, 1, 1);
025     pRenderer->AddActor(pActor);
026
027     vtkRenderWindow* pRenderWindow = vtkRenderWindow::New();
028     pRenderWindow->AddRenderer(pRenderer);
029     pRenderWindow->SetSize(300, 300);
030
031     vtkRenderWindowInteractor* pRenderWindowInteractor =
032     vtkRenderWindowInteractor::New();
033     pRenderWindowInteractor->SetRenderWindow(pRenderWindow);
034
035     // do the magic
036     pRenderWindow->Render();
037     pRenderWindowInteractor->Start();
038
039     // clean up
040     pConeSource->Delete();
041     pPolyDataMapper->Delete();
042     pActor->Delete();

```

```
042  pRenderer->Delete();  
043  pRenderWindow->Delete();  
044  pRenderWindowInteractor->Delete();  
045 }
```

From the code listing we see the different processes or stages of the visualization pipeline in lines 010 to 015. A cone (`pConeSource`) is created and the output is used as input to a mapper object (`pPolyDataMapper`). The mapper object is then used as input to an actor object (`pActor`), which is a graphics black box object. The visualization pipeline diagram, as seen in Figure 4.26, represents these stages and processes of the visualization pipeline, and is based on the functional model of Rumbaugh's Object-Oriented Modeling and Design (Rumbaugh, 1991).

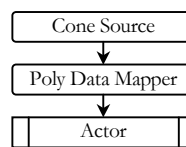


Figure 4.26: Cone visualization pipeline diagram.

Visualization pipeline diagrams as seen in Figure 4.26 are used to explain the visualization tool's modules in the next chapter. A detailed notation for the visualization pipeline diagrams is defined in Chapter 5.

Lines 022 to 032 initialize the graphics black box and lines 034 and 036 produce the magic, as seen in Figure 4.25.

# *Chapter 5*

---

## 3D Visualization Tool

### 5.1 Introduction

The software result of this thesis is a 3D visualization tool, developed for intelligence amplification (IA) on groundwater flow and transport models. In Chapter 3 the visualization tool's program modules were shown in Figure 3.6. The effort in the development of each of these modules was to design the most effective (concerning interaction response time) visualization pipeline and to create geometry, topology and scalar attribute data for the datasets. A few additional pipeline processes had to be developed: color tables, processes for basic statistics such as minimum and maximum and a lookup table class, was derived from vtk's to obtain specific functionality. The results of this effort are the program modules which produce the different visualization objects. The objects that can be added to the visualization tool's scene are: Model Outline, Geospatial Model, Potentiometric Surface, XYZ Surface, Isosurface, Hydraulic Components and Parameter.

This chapter discusses each of the visualization tool's program modules. Visualization pipeline diagrams will be used to explain the design of a module and Figure 5.1 is the notation used with the diagrams. In the following paragraphs and sections a **bold** typeface is used to refer to text that are used within the diagrams.

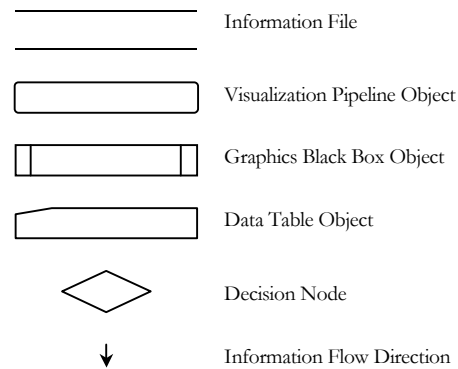


Figure 5.1: Notation used with visualization pipeline diagrams.

The **Information Files** have been discussed in Chapter 3 and contain the data that the visualization tool uses to create visualization objects. Refer to Chapter 4 for a discussion on the **Visualization Pipeline Objects** and **Graphics Black Box Objects**. **Data Table Objects** contain tabulated data which consist of scalar value ranges with assigned colors. Abbreviations used with the **Decision Node** are listed in Table 5.1.

Table 5.1: Abbreviations used with **Decision Node**.

AV	Average
DC	Decimate
SM	Smooth
CP	Clip
LOD	Level of Detail

The 3D result of the visualization tool is discussed for design issues, from different viewpoints. The viewpoints as used in the following discussions are defined in Figure 5.2.

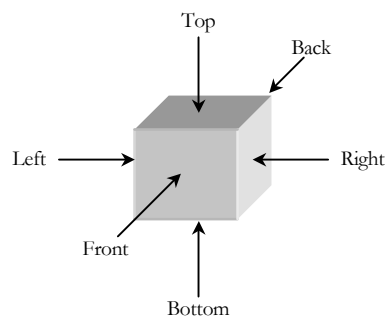


Figure 5.2: Viewpoints in 3D space.



Geometry and topology for datasets are design issues discussed for all the visualization tool's modules. The notation used with the geometry and topology diagrams are shown in Figure 5.3. A **Groundwater Model Cell** is one of the MODFLOW and PMWIN model cells that discretizes an aquifer system, as in Figure 3.2. **Points** show where geometry is defined and **Point Indexes** are the indexes of the points into the arrays that contain them. A point has x, y and z coordinates that define its position in the 3D space. Cells define topology and are themselves built from **cell edges**. For example a pixel consists of four cell edges, structured to the definition of a pixel.

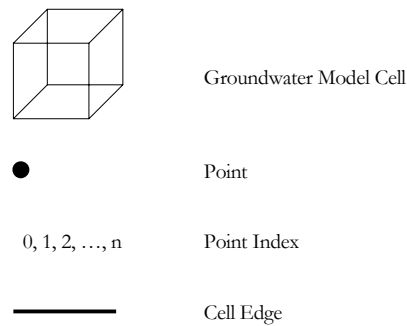


Figure 5.3: Notation used with geometry and topology diagrams.

## 5.2 Model Outline

The Model Outline visualization object is a wire frame, bounding box of the model domain. The bounding box is defined by specifying the bounds of the model domain. Minimum and maximum x, y and z real-world, corner coordinates define the bounds. Figure 5.4 shows a Model Outline visualization object with its minimum and maximum corners for which real-world, bound coordinates are specified.

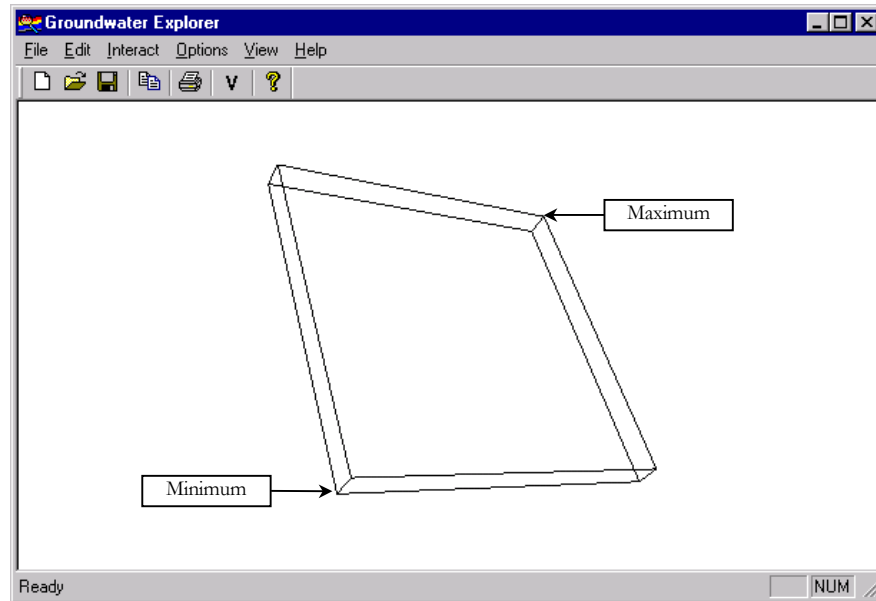


Figure 5.4: Model Outline visualization object.

The visualization pipeline diagram, as seen in Figure 5.5, summarizes the design of the Model Outline visualization module.

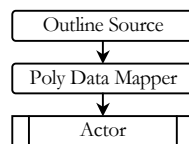


Figure 5.5: Model Outline visualization pipeline.

To display the Model Outline visualization object in the visualization tool's scene, the user has to specify the minimum and maximum real-world, corner coordinates of the model outline box. The minimum and maximum real-world corner coordinates are used as input into the visualization pipeline source object, **Outline Source**. The source object creates the wire frame, bounding box and the polygonal dataset that results from the **Outline Source** object is used as input into the polygonal mapper object, **Poly Data Mapper**. **Poly Data Mapper** in turn is used as input into a graphics black box actor object, **Actor**. The graphics black box then produces the magic, which is a picture of the Model Outline visualization object as seen in Figure 5.4. The reason why an actor and not a LOD actor is used is to have at least one visualization object visible with interaction so that the user does not get disoriented. The Model Outline visualization object was the obvious choice, consisting of only a few polylines and having an unnoticeable influence on response time with interaction.

### 5.3 Geospatial Model

The Geospatial Model visualization object is an outline polygonal mesh of the active model cells for each layer. Inactive model cells are not shown. Different Geospatial Model properties can be set for each layer. Figure 5.6 shows a Geospatial Model visualization object for a two-layer model. The outline polygonal mesh is built from polygonal quadrilaterals that can be triangulated for decimation and smoothing.

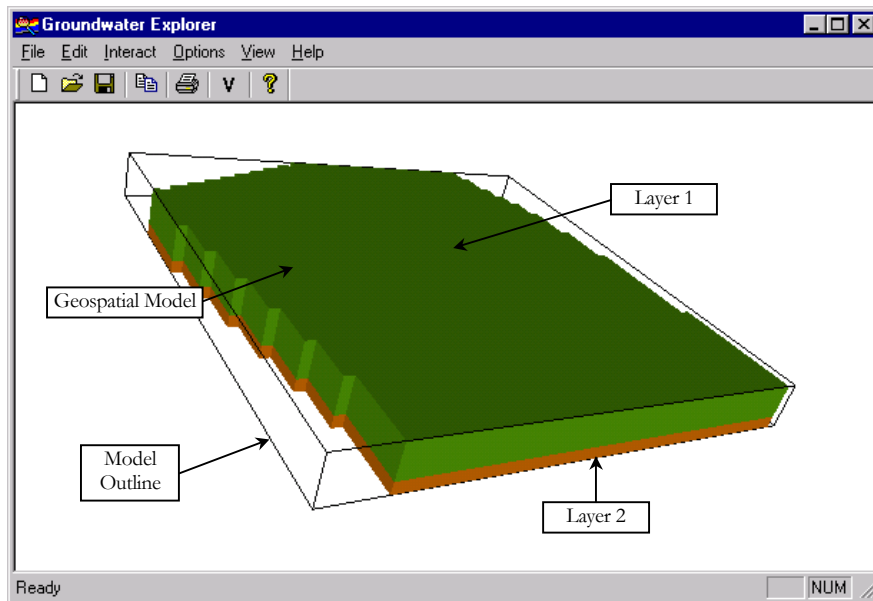


Figure 5.6: Geospatial Model visualization object.

The visualization pipeline diagram, as seen in Figure 5.7, summarizes the design of the Geospatial Model program module.

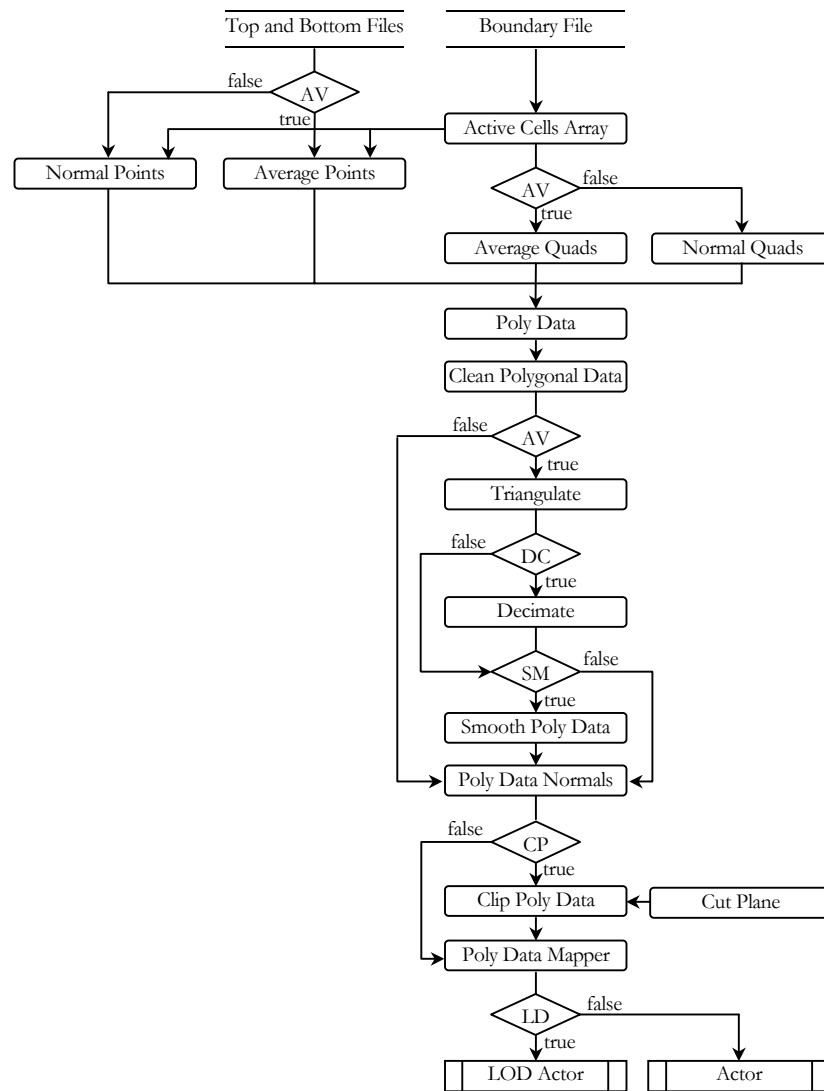


Figure 5.7: Geospatial Model visualization pipeline.

The **Top and Bottom Files** with *top* and *bot* file extensions, and the **Boundary File** with an *ibd* file extension, are output files of PMWIN as discussed in Chapter 3, “*PMWIN File Output*”. Geometry for a polygonal dataset **Poly Data**, is created from the **Top and Bottom Files**. One of two kinds of geometry can be created, namely normal points or average points. The kind of geometry that is created depends on the **AV** (average) user option and it has a visual smoothing effect as seen in Figure 5.8.

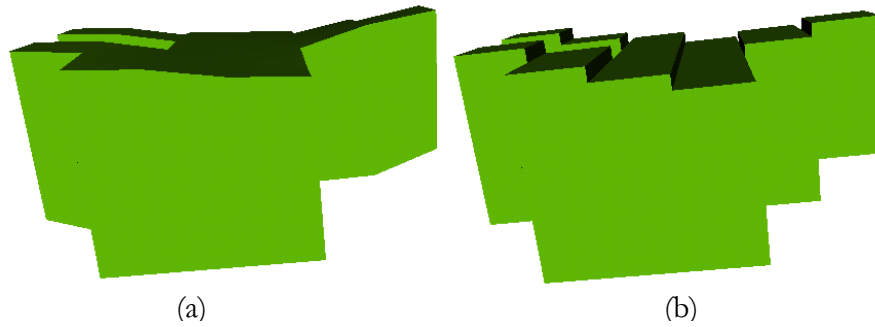


Figure 5.8: Left-top view of a one-layer Geospatial Model visualization object (a) with the average option being used. (b) The same layer with the average option not being used.

The average option is useful for groundwater models where the difference in elevation for adjacent model cells is small, because it results in fewer quadrilaterals used to create the polygonal mesh of the Geospatial Model visualization object. Another advantage of the average option is that it results in a smoother, polygonal mesh that has a more natural look. This is important if one keeps in mind that the first layer of the Geospatial Model often represents surface topology. When taking computer-processing time into account, fewer quadrilaterals also have the advantage of faster rendering and interaction reaction times. Figure 5.9 shows an example of normal and average geometry for two model cells. For normal geometry, coordinates at cell corners are used to define the points for geometry and for average geometry an average  $z$  coordinate value is calculated for points that are shared by more than one active model cell. For example the point with index 1 in Figure 5.9 (b) is calculated as follows:

$$z \text{ coordinate}(1b) = [z \text{ coordinate}(1a) + z \text{ coordinate}(2a)]/2,$$

Where 1b is the point with index 1 in Figure 5.9 (b) and 1a and 2a are the points with indexes 1 and 2 in Figure 5.9 (a).

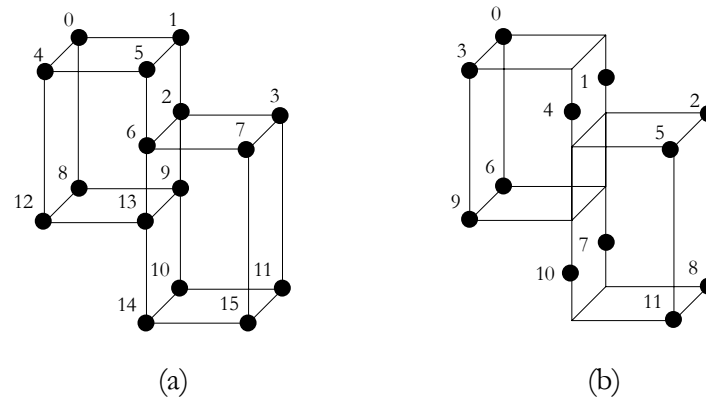


Figure 5.9: (a) Normal (b) and average geometry for Geospatial Model visualization object.

The pseudo code creating normal and average geometry is listed in Listing 5.1 and Listing 5.2, respectively.

Listing 5.1: Pseudo code, creating normal geometry for **Poly Data** dataset of the Geospatial Model visualization object.

```

001 Create a point array.
002 Set the size of the point array =  $2 \times 2 \times \text{number of rows} \times 2 \times \text{number of columns}$ .
003 Read the top elevation for all cells into a top elevation array.
004 Read the bottom elevation for all cells into a bottom elevation array.
005 For number of rows.
006     For number of columns.
007     {
008         If it is an active cell.
009         {
010             Get top elevation of cell from top elevation array for current
011             row and column.
012             Get bottom elevation of cell from bottom elevation array for
013             current row and column.
014         }
015         Else
016         {
017             Use minimum elevation of all bottom and top elevations as top
018             elevation.
019             Use minimum elevation of all bottom and top elevations as bottom
020             elevation.
021         }
022         Create 2 column coordinates for current row and column.
023         Create 2 row coordinates for current row and column.
024         Calculate 4 top point indexes into point array for 4 top points of
025         cell.
026         Insert 4 points into the point array, using the 4 top point indexes,
027         row and column coordinates and top elevation.
028         Calculate 4 bottom point indexes into the point array for 4 bottom
029         points of cell.
030         Insert 4 points into point array, using the 4 bottom point indexes,
031         row and column coordinates and bottom elevation.
032     }
033 }
```

Listing 5.2: Pseudo code, creating average geometry for **Poly Data** dataset of the Geospatial Model visualization object.

```

001 Create an average point array.
002 Set the size of the average point array =  $2 \times (\text{number of rows} + 1) \times (\text{number of}$ 
003  $\text{columns} + 1)$ .
004 Initialize the average point array.
```

```

004 Read the top elevation for all cells into a top elevation array.
005 Read the bottom elevation for all cells into a bottom elevation array.
006 For number of rows.
007     For number of columns.
008         If it is an active cell.
009             {
010                 Calculate 4 top average point indexes into average point array
                    for cell.
011                 Add the top elevation to the average point array, using the 4
                    top average point indexes.
012                 Calculate 4 bottom average point indexes into average point
                    array for cell.
013                 Add the bottom elevation to the average point array, using the 4
                    bottom average point indexes.
014             }
015 Create a point array.
016 Set the size of the point array = 2 × (number of rows + 1) × (number of columns +
    1).
017 For indexes of point array.
018 {
019     Calculate row and column indexes using the point array index.
020     Insert a point into the point array where:
021         point.x = Get the column coordinate for column index.
022         point.y = Get the row coordinate for row index.
023         point.z = Get the average elevation from average point array.
024 }

```

Figure 5.7 shows that the **Boundary File** is used to create an **Active Cells Array** that contains a boolean flag for each model cell. If a model cell is active, the flag is set to true and if a cell is inactive, the flag is set to false. The **Active Cells Array** is used in creating geometry and topology. Only active cells are used to create normal and average geometry as seen from line 008 in Listing 5.1 and line 008 in Listing 5.2.

The result of the **Normal Points** or **Average Points** processes, is geometry contained in a point array that is used as input into the **Poly Data** polygonal dataset. Up to now no topology has been created. The **Normal Quads** and **Average Quads** processes do this. Again, whether normal or average topology is created, depends on the user. The **Normal Quads** and **Average Quads** processes consist of sub-processes, creating the normal and average top, sides and bottom topology parts of the Geospatial Model visualization object. The reason for using sub-processes is that the user of the visualization tool has the option of displaying the top, sides or bottom of a Geospatial Model visualization object for each model layer, as seen in Figure 5.10.

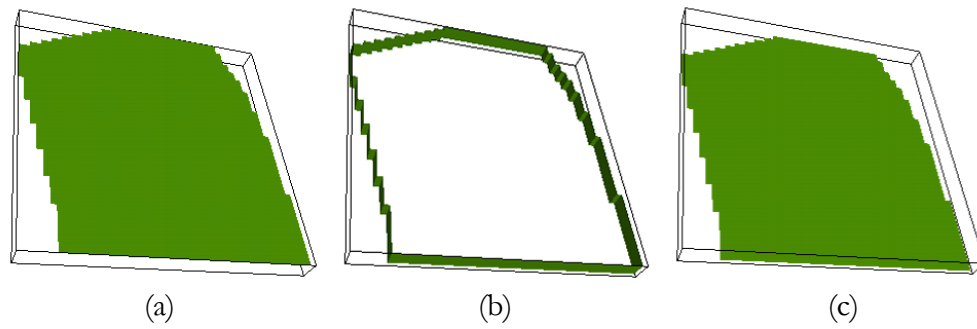


Figure 5.10: (a) Top, (b) sides and (c) bottom of a Geospatial Model visualization object for a layer.

Figure 5.11 shows an example of normal and average topology for two model cells of the Geospatial Model visualization object. Normal top and normal bottom topology are shown in Figure 5.11 (a) and Figure 5.11 (e) respectively, and are stretched sideways to show underlying points that will otherwise be on top of each other, if viewed from the top and bottom.



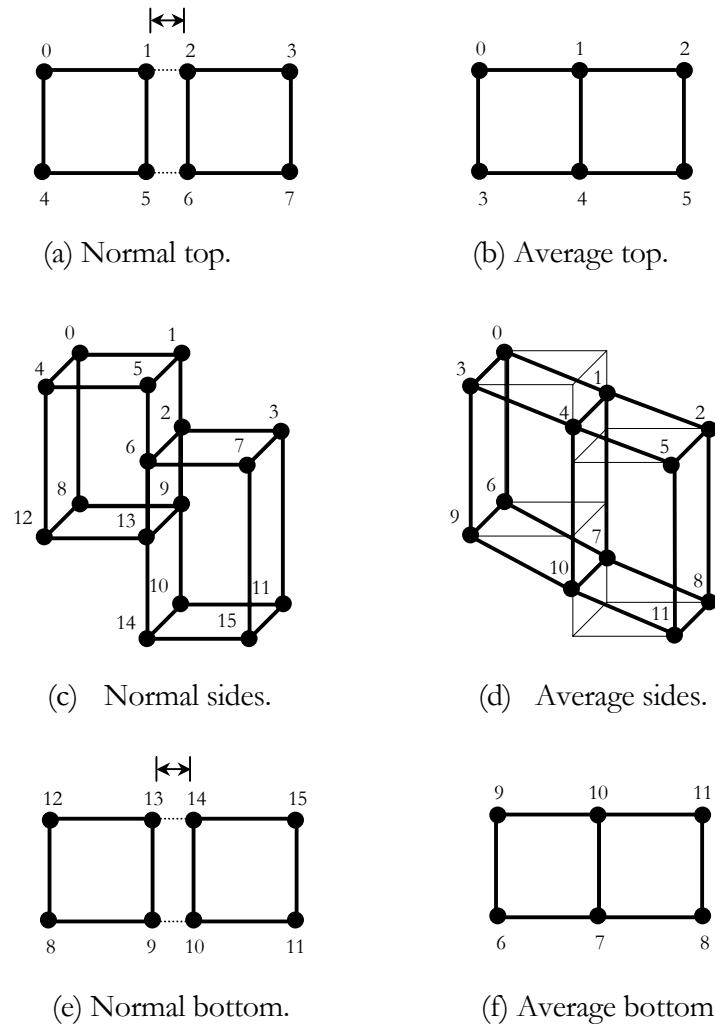


Figure 5.11: Topology for Geospatial Model visualization object.

Listing 5.3 lists the pseudo code that creates normal top topology, as seen in Figure 5.11 (a) and Listing 5.4 lists the pseudo code that creates average top topology, as seen in Figure 5.11 (b).

Listing 5.3: Pseudo code, creating normal top topology for **Poly Data** dataset of the Geospatial Model visualization object.

```

001 Create a cell array.
002
003 // Horizontal top.
004 For number of rows.
005     For number of columns.
006         If it is an active cell.
007             {
008                 Calculate 4 horizontal top, point array indexes for
quadrilateral.
009                 Insert the next quadrilateral into the cell array, using the 4
horizontal top vertical, point array indexes.
010             }
011
012 // Right vertical.

```

```

013   For number of rows.
014       For number of columns - 1.
015           If it is an active cell and cell to right is an active cell.
016               {
017                   Calculate 4 right vertical, point array indexes for
quadrilateral.
018                   Insert the next quadrilateral into the cell array, using the 4
right vertical, point array indexes.
019               }
020
021   // Front vertical.
022   For number of rows - 1.
023       For number of columns.
024           If it is an active cell and cell in front is an active cell.
025               {
026                   Calculate 4 front vertical, point array indexes for
quadrilateral.
027                   Insert the next quadrilateral into the cell array, using the 4
front vertical, point array indexes.
028               }

```

The code creating normal bottom topology is the same as for creating normal top topology for a Geospatial Model visualization object, with the only difference in the calculation of the point indexes. Figure 5.11 (e) shows the normal bottom topology.

Listing 5.4: Pseudo code, creating average top topology for **Poly Data** dataset of the Geospatial Model visualization object.

```

001   Create a cell array.
002
003   // Horizontal top.
004   For number of rows.
005       For number of columns.
006           If it is an active cell.
007               {
008                   Calculate 4 top, point array indexes for quadrilateral.
009                   Insert the next quadrilateral into the cell array, using the 4,
point array indexes.
010               }

```

As with normal bottom topology, the code creating average bottom topology is the same as for calculating average top topology of a Geospatial Model visualization object, with the only difference in the calculation of the point indexes.

The remaining topology for the **Poly Data** dataset to be created is the normal or average sides topology, as listed in Listing 5.5.

Listing 5.5: Pseudo code, creating normal sides topology for **Poly Data** dataset of the Geospatial Model visualization object.

```

001   For number of rows.
002       For number of columns.
003           If it is an active cell.
004               {
005
006                   // left
007                   If it is the first column or if the cell to the left of cell is
inactive.
008                   {

```

```

009             Calculate 4, point array indexes for quadrilateral.
010             Insert the next quadrilateral into the cell array, using
                the 4, point array indexes.
011         }
012
013         // right
014         If it is the last column or if the cell to the right of cell is
                inactive.
015         {
016             Calculate 4, point array indexes for quadrilateral.
017             Insert the next quadrilateral into the cell array, using
                the 4, point array indexes.
018         }
019
020         // back
021         If it is the first row or if the cell to the back of cell is
                inactive.
022         {
023             Calculate 4, point array indexes for quadrilateral.
024             Insert the next quadrilateral into the cell array, using
                the 4, point array indexes.
025         }
026
027         // front
028         If it is the last row or if the cell to front of the cell is
                inactive.
029         {
030             Calculate 4, point array indexes for quadrilateral.
031             Insert the next quadrilateral into the cell array, using
                the 4, point array indexes.
032         }
033     }

```

The code for creating average sides topology is the same as for creating normal sides topology of a Geospatial Model visualization object, with the only difference in the calculation of the point indexes.

The result of the **Normal Quads** and **Average Quads** processes is topology which is used as input into the **Poly Data** polygonal dataset. All input into the polygonal dataset has been created up to this stage.

For the discussion on the functionality of the visualization pipeline filters that follow, refer to Chapter 4, “*Filters*”. After the dataset has been created the filters can start their work. The **Clean Polygonal Data** filter cleans the **Poly Data** polygonal dataset of unnecessary points. If the user has selected the **AV** (average) option, the polygonal dataset’s topology, which consists of quadrilaterals, is triangulated into triangles using the **Triangulate** triangulation filter. Triangles are the correct topology type to use with the **Decimate** and **Smooth Poly Data** filters. As with the average, the use of the **Decimate** and **Smooth Poly Data** filters depends on the user. The output of one or the other is used as input into the **Poly Data Normals** filter, which computes point normals, and the result is a smoother-looking visualization object.

Views of a cross-section often help to obtain a better understanding of a groundwater model and its information. When the **CP** (clip) user option is selected, the **Clip Poly Data** filter cuts the Geospatial Model visualization object using the **Cut Plane** source object as a knife. The direction in which is cut, is specified by the user as a normal vector on the **Cut Plane**. Depending on the **CP** user option, the uncut or cut result is used as input into the **Poly Data Mapper** object, the interface between the visualization pipeline and graphics black box. The final step is to use the **Poly Data Mapper** as input into a graphics black box object, **LOD Actor** or **Actor**. The graphics black box uses the actor object with its other objects to do magic, as seen in Figure 5.6.

## 5.4 Potentiometric Surface

Potentiometric Surface is a visualization object, as seen in Figure 5.12. It visualizes groundwater heads in the highest active cells of all layers or the groundwater heads in active cells for a specified layer. The groundwater heads are also visualized for a specified stress period and time step.

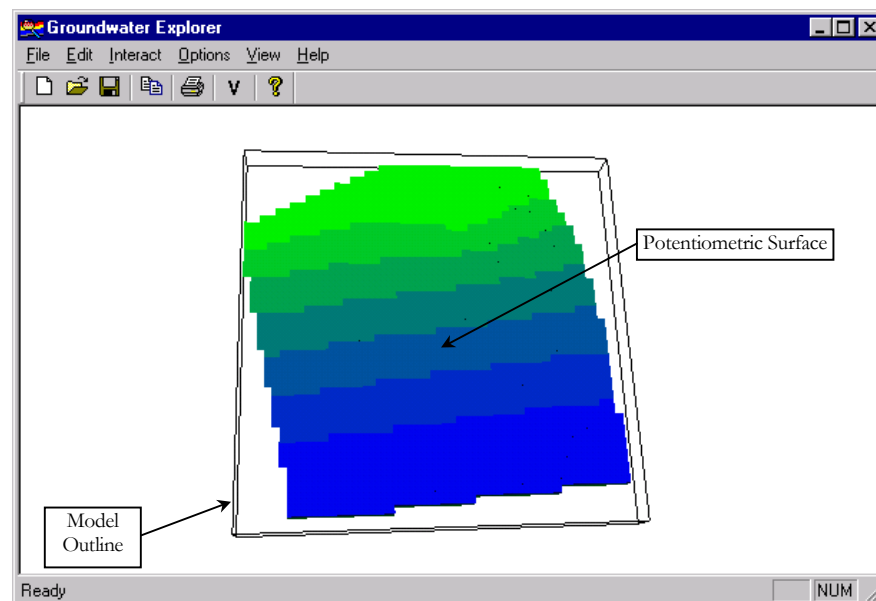


Figure 5.12: Potentiometric Surface visualization object.

The visualization pipeline diagram in Figure 5.13 summarizes the design of the Potentiometric Surface program module.

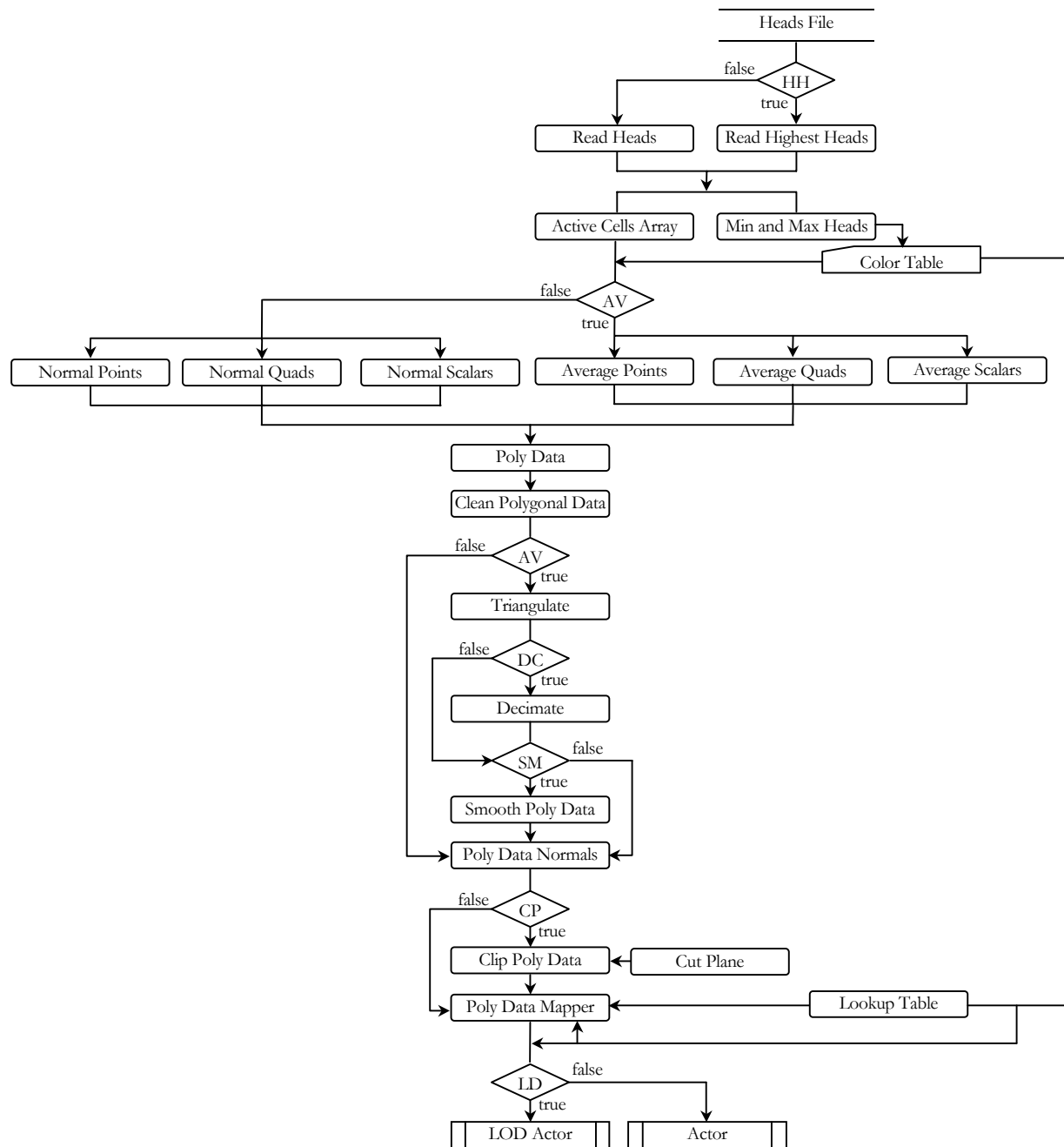


Figure 5.13: Potentiometric Surface visualization pipeline.

The **Heads File**, *heads.dat* is an output file of MODFLOW, as discussed in Chapter 3, “MODFLOW File Output”. Geometry, topology and the scalar dataset attribute for a **Poly Data** polygonal dataset are created from the **Heads File**. A user option, **HH** (highest heads), allows the user to visualize groundwater heads in the highest active cells or the groundwater heads in a specified layer. The **Read Heads** process reads model calculated groundwater heads from the **Heads File** for a specified layer, and the **Read Highest Heads** process reads the model calculated groundwater heads in the highest active cells of

all layers from the **Heads File**. From the heads that were read, the **Min and Max Heads** process obtains the minimum and maximum elevation of the calculated groundwater heads. These minimum and maximum groundwater head values are used as the range of groundwater head values to be visualized in the **Color Table** data table object. The **Active Cells Array** process creates an array that contains a boolean flag, which is set to true for active cells and to false for inactive cells. The **Active Cell Array** is used to create geometry, topology and the scalar dataset attribute. Geometry, topology and the scalar dataset attribute are only created for active cells.

As with the Geospatial Model one of two kinds of geometry can be created: normal points or average points. **AV** (average) is a user option that has a visual, smoothing effect as seen in Figure 5.8. The groundwater heads that were read flows through to the **Normal Points**, **Normal Quads** and **Normal Scalar** processes, if the user does not want to visualize with average. The **Normal Points** and **Normal Scalar** processes are contained in one function in the visualization tool where the z coordinate of a point is assigned as a scalar value to the same point. The scalar dataset attribute is used for color mapping through the **Lookup Table**.

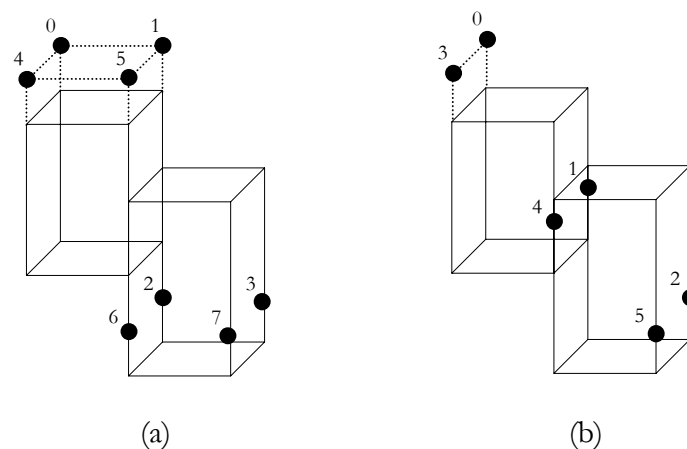


Figure 5.14: (a) Normal (b) and average geometry for Potentiometric Surface visualization object.

The pseudo code that creates normal geometry and the normal scalar dataset attribute for the **Poly Data** polygonal dataset, is listed in Listing 5.6. Listing 5.7 lists the pseudo code that creates average geometry and the average, scalar dataset attribute, using the **Average Points**, **Average Quads** and **Average Scalars** processes. Figure 5.14 is an example of geometry for two active model cells containing groundwater heads. Figure 5.14 (a) shows

normal geometry and Figure 5.14 (b) shows average geometry; note that groundwater head values for a cell can be outside the domain of the cell.

Listing 5.6: Pseudo code, creating normal geometry and the normal scalar dataset attribute for **Poly Data** dataset of the Potentiometric Surface visualization object.

```

001  Read the groundwater head for all cells into groundwater head array.
002  Create a point array.
003  Set the size of the point array = 2 × number of rows × 2 × number of columns.
004  Create a scalar array.
005  Set the size of the scalar array = 2 × number of rows × 2 × number of columns.
006  For number of rows.
007      For number of columns.
008          {
009              If it is an active cell and head value is in Color Table range.
010                  Get groundwater head of cell from groundwater head array for
                      current row and column.
011              Else
012                  Use minimum groundwater head of all groundwater heads as
                      groundwater head.
013              Create 2 column coordinates for current row and column.
014              Create 2 row coordinates for current row and column.
015              Calculate 4 point indexes into point array for 4 points of cell.
016              Insert 4 points into point array, using the 4, point indexes, row and
                      column coordinates and head value.
017              Insert 4 scalars into scalar array, using the 4, point indexes and
                      head value.
018          }

```

Listing 5.7: Pseudo code, creating average geometry and the average scalar dataset attribute for **Poly Data** dataset of the Potentiometric Surface visualization object.

```

001  Read the groundwater heads for all cells into groundwater head array.
002  Create an average point array.
003  Set the size of the average point array = (number of rows + 1) × (number of
      columns + 1).
004  Initialize the average point array.
005  For number of rows.
006      For number of columns.
007          If it is an active cell and head value is in Color Table range.
008              {
009                  Calculate 4 point indexes into average point array for cell.
010                  Insert 4 points into average point array, using the point
                      indexes and head.
011              }
012  Create a point array.
013  Set the size of the point array = (number of rows + 1) × (number of columns + 1).
014  Create the scalar array.
015  Set the size of the scalar array = (number of rows + 1) × (number of columns + 1).
016  For indexes in point array.
017      {
018          Calculate row and column indexes using the point array index.
019          Insert a point into the point array where:
020              point.x = Get column coordinate for column index.
021              point.y = Get row coordinate for row index.
022              point.z = Get average groundwater head from average point array.
023          Insert average groundwater head into the scalar array.
024      }

```

Creating topology for the Potentiometric Surface visualization object also depends on the average user option, as with geometry.

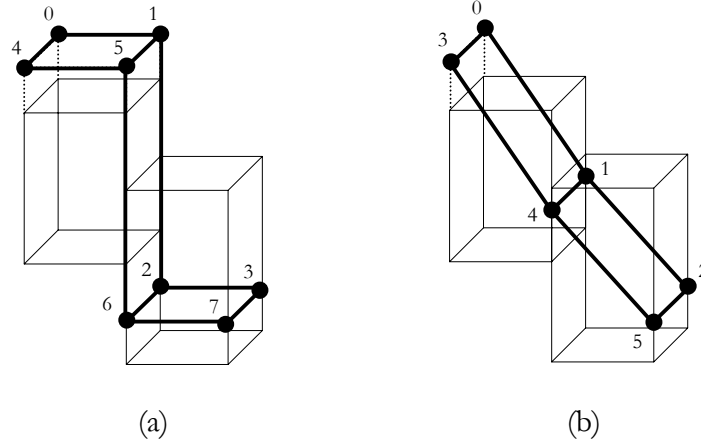


Figure 5.15: (a) Normal (b) and average topology for Potentiometric Surface visualization object.

If the user does not use the average option, normal topology is created in the **Normal Quads** process. The **Average Quads** process is used to create average topology. The code, which creates the normal and average topology, is the same as the code that creates top normal and top average topology for the Geospatial Model visualization object as in Listing 5.3 and Listing 5.4. The output of the **Normal Quads** or **Average Quads** processes is used as input into the **Poly Data** dataset. As with the Geospatial Model visualization object the same visualization pipeline filters are applied on the **Poly Data** dataset of the Potentiometric Surface visualization object. See the “*Geospatial Model*” section for a discussion on data flow through the filters to the actor objects of the graphics black box. The only difference is the addition of another graphics black box object, **Lookup Table**. The **Lookup Table** uses the **Color Table** and is specified as input into **Poly Data Mapper**, where it is used to color map the scalar attribute data. The reuse of code can clearly be seen from the object structures for the modules, as in Appendix B.

## 5.5 XYZ Surface

The XYZ Surface visualization object is created from x, y and z real-world coordinates. Figure 5.16 shows a XYZ Surface visualization object.



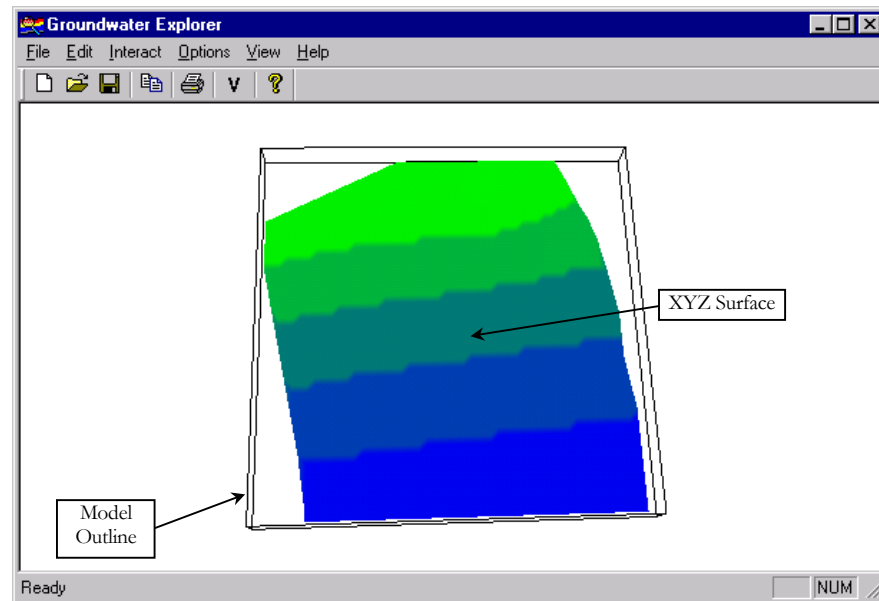


Figure 5.16: XYZ Surface visualization object.

The visualization pipeline diagram, as seen in Figure 5.12, summarizes the design of the XYZ Surface program module.

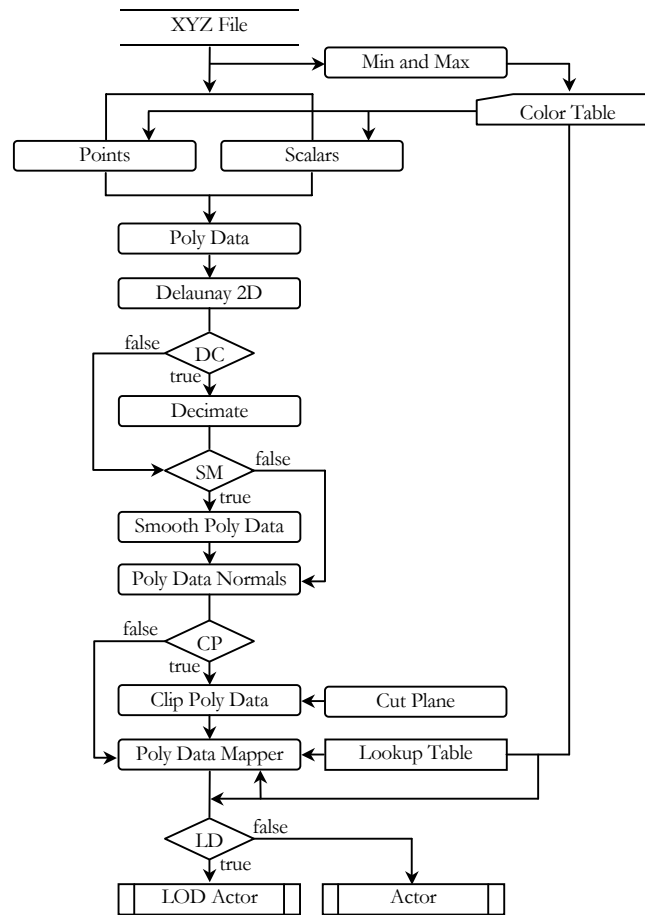


Figure 5.17: XYZ Surface visualization pipeline.

Vtk's visualization pipeline objects make it easy to visualize **XYZ File** information. The **XYZ File**, with an *xyz* file extension is an output file of a text editor, as discussed in Chapter 3, “*XYZ File Output*”. The XYZ Surface visualization pipeline starts by reading the x, y and z real-world coordinates from the **XYZ File** into a point array, using the **Points** process. At the same time, in the same program function the z coordinates are used in the **Scalars** process as the scalar dataset attribute for the polygonal dataset, **Poly Data**. Only points with scalar data which fall in the **Color Table** scalar value range, are used for visualization. As always, the points define the geometry and are used as input into the polygonal dataset, **Poly Data**. The scalar dataset attribute is also specified as input into **Poly Data**.

Listing 5.8 lists the pseudo code that creates geometry and the scalar dataset attribute for the **Poly Data** dataset.

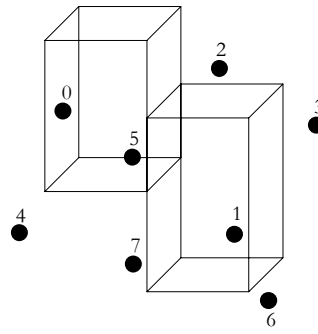


Figure 5.18: Geometry for XYZ Surface visualization object.

Listing 5.8: Pseudo code, creating geometry and the scalar dataset attribute for **Poly Data** polygonal dataset of the XYZ Surface visualization object.

```

001 Create a point array.
002 Create a scalar array.
003 While not end of file.
004 {
005     Read x, y and z coordinate from file.
006     If z coordinate is in Color Table range.
007     {
008         Insert x, y and z coordinate as a point into point array.
009         Insert z coordinate as a scalar into scalar array.
010     }
011 }

```

Up to this point the polygonal dataset, **Poly Data** is without topology. This is where the **Delaunay 2D** filter comes into action. **Delaunay 2D** is a triangulation filter, as discussed in Chapter 4, and it creates topology for the polygonal dataset.

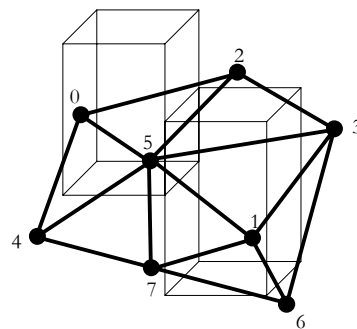


Figure 5.19: Topology for XYZ Surface visualization object.

The dataset is complete when the **Delaunay 2D** filter has created the topology, and the other filters can take over the data flow. The filters used and the graphics black box visualization object **Lookup Table** is the same as discussed in the “*Geospatial Model*” and “*Potentiometric Surface*” sections.

## 5.6 Isosurface

Isosurfaces are created from concentrations in model cells, calculated by MT3DMS. Figure 5.20 shows two isosurfaces for two different concentrations.

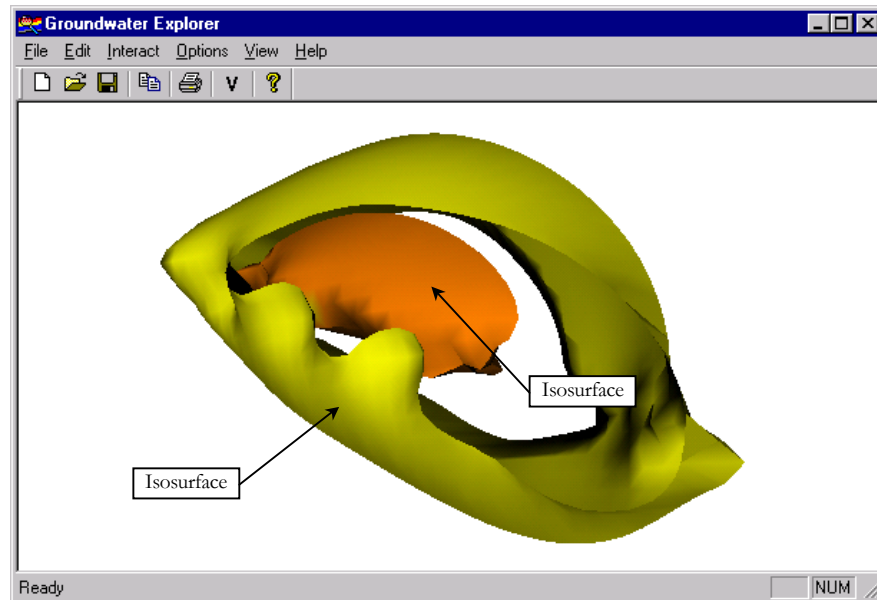


Figure 5.20: Concentration Isosurface (MT3DMS) visualization object.

The visualization pipeline diagram as seen in Figure 5.17 summarizes the design of the Isosurface program module.

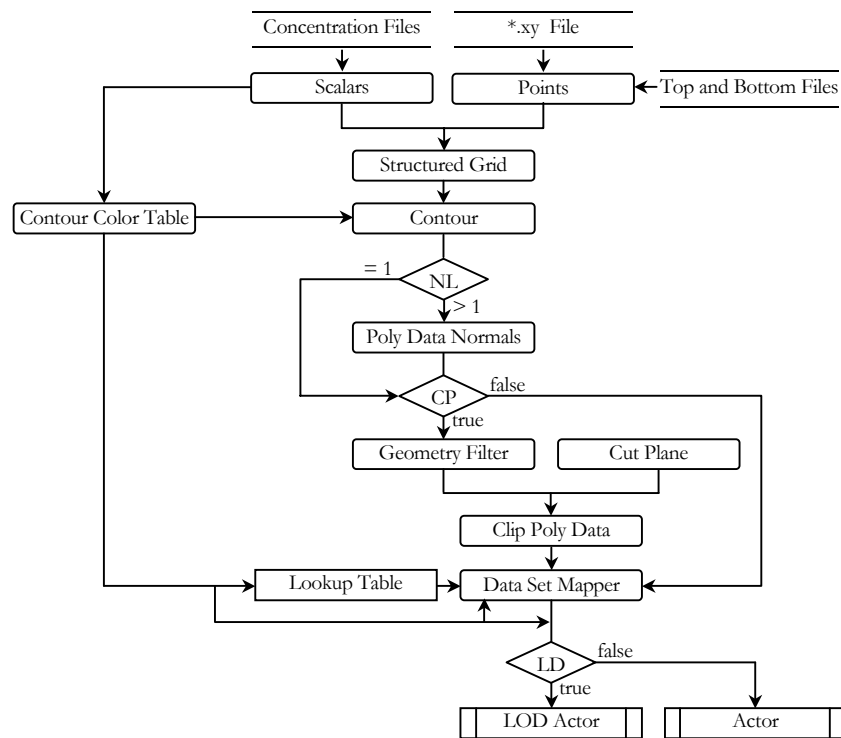


Figure 5.21: Isosurface visualization pipeline.

Dataset geometry is created from the information in the **\*.xy File** and **Top and Bottom Files** with *top* and *bot* file extensions, as discussed in Chapter 3 “*PMWIN File Output*”. Geometry is only created in the middle of each model cell in the **Points** process, as shown for an example of four model cells in Figure 5.22.

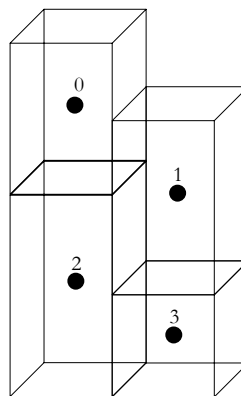


Figure 5.22: Geometry for Isosurface visualization object.

Listing 5.9 lists the pseudo code that creates the geometry for the **Structured Grid** dataset.

Listing 5.9: Pseudo code, creating geometry for **Structured Grid** dataset of the Isosurface visualization object.

```

001 Create middle row and column coordinates.
002 Create a point array.
003 For all the model layers.
004 {
005     Read the top elevation for all cells into the top elevation array.
006     Read the bottom elevation for all cells into the bottom elevation array.
007     For number of rows.
008         For number of columns.
009             Insert the point into the point array, using the middle row and
                column coordinates and the middle of the top and bottom
                elevation.
010 }

```

The scalar dataset attribute for the **Structured Grid** dataset are read from the **Concentration Files**. Geometry and the scalar dataset attribute are used as input into the **Structured Grid** dataset. Listing 5.10 lists the pseudo code that creates the scalar dataset attribute.

Listing 5.10: Pseudo code, creating the scalar dataset attribute for **Structured Grid** dataset of the Isosurface visualization object.

```

001 Create a scalar array.
002 For all the model layers.
003 {
004     Read the concentration for all cells into concentration array.
005     For number of rows.
006         For number of columns.
007             If it is an inactive concentration cell.
008                 Insert next scalar with a value of 0.
009             Else
010                 Insert next scalar using the cell concentration.
011 }

```

Dataset topology still does not exist, and is created using the **Contour** filter. Figure 5.23 shows an isosurface of a specific concentration for four active cells.

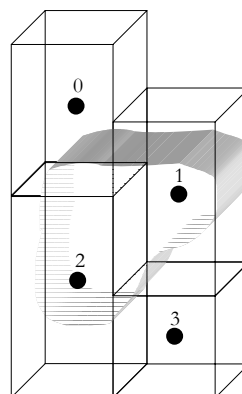


Figure 5.23: Topology for Isosurface visualization object.

If a groundwater model has more than one layer, the polygonal data normals are created using the **Polydata Normals** filter. However, if the groundwater model consists of only one layer the output of the **Contour** filter is polygonal lines, for which polygonal data normals cannot be created. As with all the previous visualization modules the data flow is the same from the **Poly Data Normals** filter onwards. See the discussion in the “*Geospatial Model*” section for an explanation on the other filters and Appendix B for the object structure. The only difference is the **Geometry Filter**, which is used to obtain the geometry from the **Contour** or **Poly Data Normals** filters, depending on the number of layers of the groundwater model. The reason for using the **Geometry Filter** is that a structured grid dataset has been used and not a polygonal dataset, as within the previous visualization modules. A structured grid dataset is used, because the **Contour** filter can only be used with a **Structured Grid** dataset. The **Clip Poly Data** filter, on the other hand, can only be used with polygonal data. This is why the **Geometry Filter** is used to convert the structured grid dataset to a polygonal dataset.

## 5.7 Hydraulic Components

The Hydraulic Components visualization object colors groundwater model cells which contain hydraulic component information. Figure 5.24 shows colored model cells for discharge and recharge wells.

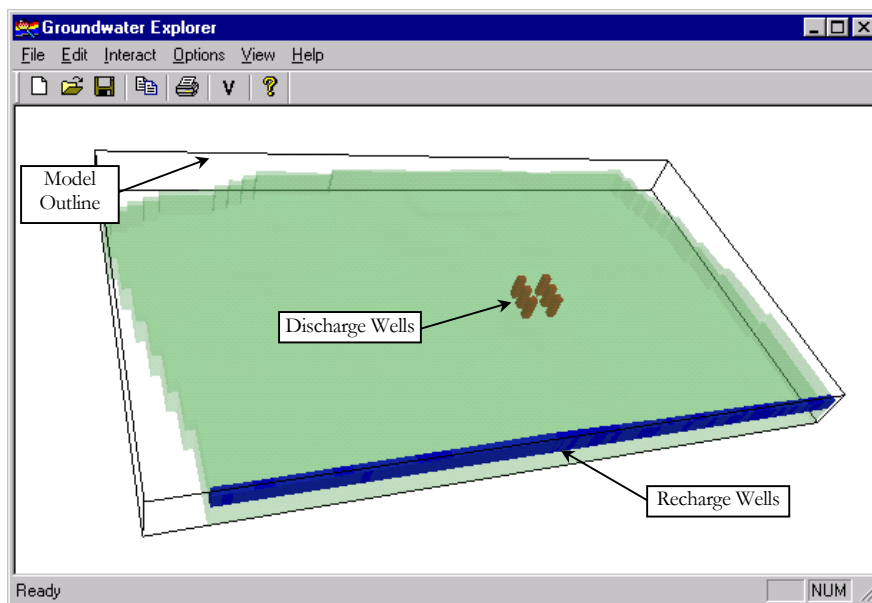


Figure 5.24: Hydraulic Components visualization object.

The visualization pipeline diagram as seen in Figure 5.25, summarizes the design of the Hydraulic Components program module. Hydraulic components are divided into time dependent and time independent components and are listed below:

Time dependent hydraulic components:

- Discharge Well
- Drain
- General Head Boundary
- Recharge Well
- River
- Time Variant Specified Concentration
- Time Variant Specified Head

Time independent hydraulic components:

- Fixed Concentration
- Fixed Heads
- Horizontal Flow Barriers
- Reservoir



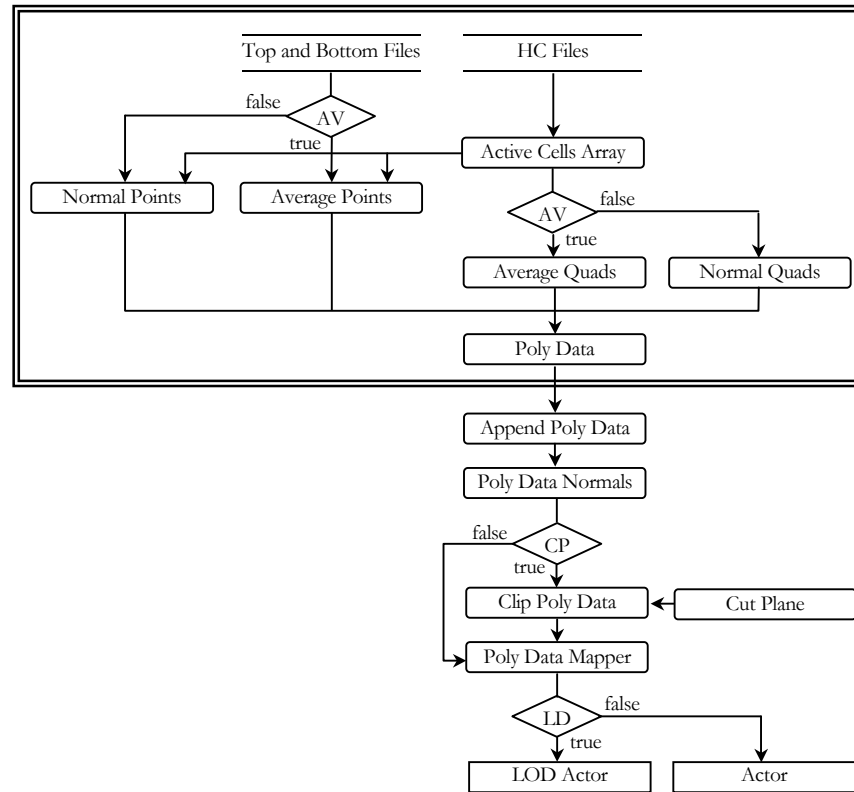


Figure 5.25: Hydraulic Components visualization pipeline.

The processes inside the double line border are repeated for each layer. Geometry, topology and a dataset are created for each layer. The geometry that is created is the same as that of the Geospatial Model, as discussed in the “*Geospatial Model*” section. Figure 5.9 is therefore also an example of geometry for the Hydraulic Components visualization object and Figure 5.11 is of topology. The pseudo code listings for normal and average, geometry and topology for the Geospatial Model visualization object, also apply for the Hydraulic Components visualization object. The only difference is that, for the Geospatial Model the user had the option of viewing the top, sides and bottom part of the Geospatial Model for each layer. With the Hydraulic Components module the user does not have the option, and the top, sides and bottom are always created and shown.

For the previous modules, one actor is used for each dataset, but in the Hydraulic Components visualization module, one actor represents all the layers. The **Append Poly Data** filter is therefore used to append the datasets of all the layers resulting into one dataset. See the “*Geospatial Model*” section for a discussion on the visualization pipeline filters used on data flow from the **Append Poly Data** filter onwards.

## 5.8 Parameter

The Parameter visualization object colors model cells that contain parameter information. Figure 5.26 shows the Effective Porosity parameter, where the two different values of effective porosity were assigned different colors.

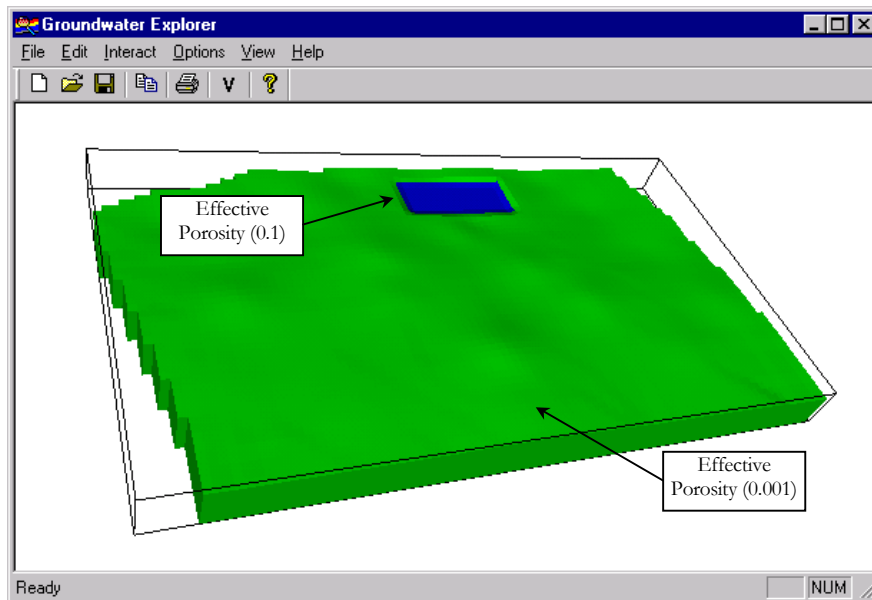


Figure 5.26: Parameter visualization object.

The following parameters can be visualized if model cells were assigned values:

- Horizontal Hydraulic Conductivity
- Vertical Hydraulic Conductivity
- Specific Storage
- Transmissivity
- Vertical Leakance
- Storage Coefficient
- Effective Porosity
- Specific Yield

The visualization pipeline diagram, as seen in Figure 5.27, summarizes the design of the Parameter program module.

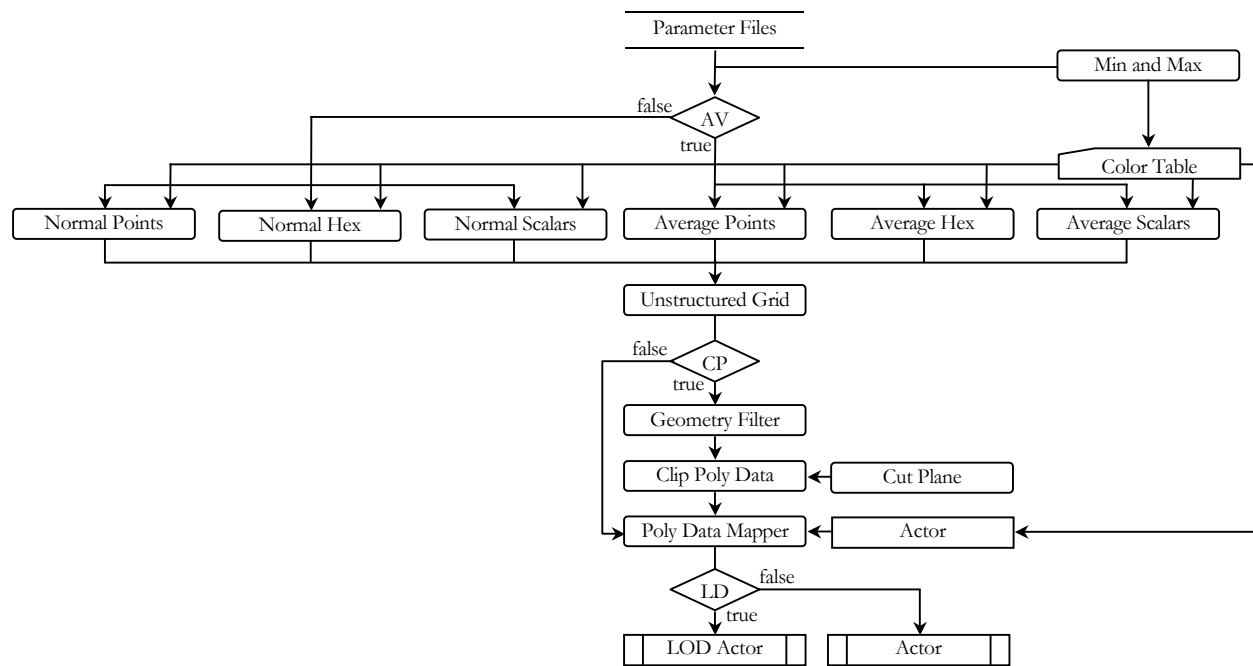


Figure 5.27: Parameter visualization pipeline.

The **Min and Max** process obtains the minimum and maximum parameter values from the information contained in the **Parameter Files**. The **Parameter Files** are output files of PMWIN, as discussed in Chapter 3, “*PMWIN File Output*”. The **Color Table** is then created using the minimum and maximum parameter values as the scalar value range. Geometry and topology to be created, depends on the **AV** (average) user option. The **Normal Points**, **Normal Hex** and **Normal Scalars** processes are used for visualization when average is not used. If the user wants to apply the average option with the visual effect, as seen in Figure 5.13, the **Average Points**, **Average Hex** and **Average Scalars** processes are used.

As with the Hydraulic Components module, the geometry that is created for the Parameter module is the same as that of the Geospatial Model, as discussed in the “*Geospatial Model*” section. Figure 5.9 is therefore also an example of geometry. Figure 5.11 is an example of topology for the Parameter visualization module. The pseudo code listings creating normal and average geometry for the Geospatial Model module also apply for the Parameter module.

Topology for the Parameter module is not created in the same way as topology for the Geospatial Model module. Parameter topology consists of hexahedrons and not quadrilaterals, as used with the Geospatial Model module. The reason is that color values are assigned to whole model cells for the Parameter module, and each cell can have a

different color. Figure 5.28 is an example of topology for the Parameter visualization object, and Listing 5.11 lists the pseudo code creating the topology and the scalar dataset attribute.

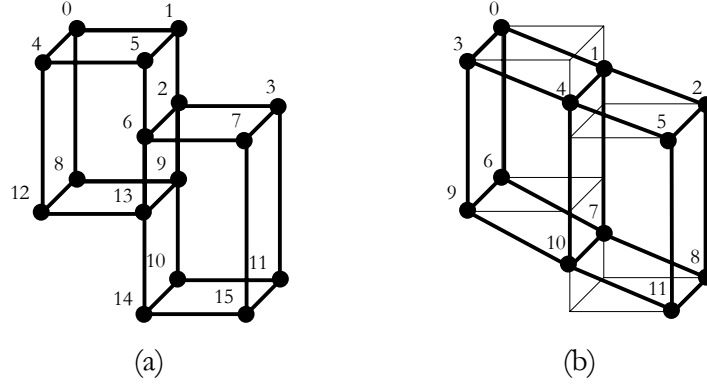


Figure 5.28: (a) Normal and (b) average topology for Parameter visualization object.

Listing 5.11: Pseudo code, creating normal topology and the normal scalar dataset attribute for **Unstructured Grid** dataset of the Parameter visualization object.

```

001 Create a cell array.
002 Create a scalars array.
003 For all the model layers.
004 {
005     Read parameter value for all cells into parameter array for specific layer.
006     For number of rows.
007         For number of columns.
008             If it is an active cell and parameter value is in Color Table
009             range.
010             {
011                 Create 8, point indexes for hexahedron into point array.
012                 Insert hexahedron into cell array, using the 8, point
013                 indexes.
014                 Insert parameter value into scalar array.
015             }
016         }
017     }
018 }

```

The code creating average topology is the same as for creating normal topology, with the only difference in the calculation of the point indexes.

The geometry and topology created are used as input into an **Unstructured Grid** dataset. The filters and other processes, onwards from the dataset, do the same job as discussed for the other visualization modules in the previous sections.

## 5.9 Summary

The visualization tool can add an arbitrary number of visualization objects, as discussed in the previous sections, to a visualization scene by making use of polymorphism. The visualization objects are as follows:

- Model Outline – The bounding box of a groundwater model.
- Geospatial Model – Topography of each model layer.
- Potentiometric Surface – Calculated hydraulic head values.
- XYZ Surface – Measured values.
- Isosurface of concentration values calculated with MT3DMS.
- Hydraulic Components, for example pumping wells.
- Model parameter values.



# Chapter 6

---

## Evaluation, Future Research and Conclusion

### 6.1 Introduction

In this chapter the visualization tool is evaluated to determine if the tool addresses the problems described in Chapter 1. It also evaluates how the tool addresses the issues discussed in the “*Hypothesis and Research*” section in Chapter 1, as well as the “*Advantages and Disadvantages*” section in Chapter 2. Naturally, after evaluation, ideas for future research on how to improve and expand the tool are pointed out. The chapter ends with a conclusion.

### 6.2 Evaluation

The key problems identified in the “*Problem Description*” section in Chapter 1, and examples of how the visualization tool is used to address these problems, are listed below:

- **Communicating and understanding conceptual models.** “*Conceptualize the Wetland Model*” in Chapter 4 of the “*User’s Guide*” in Appendix C, is an example of how the visualization tool can be used to help communicate and understand a conceptual model.
- **Verifying model input.** “*Step 3: Add Time Dependent Hydraulic Component*” in Chapter 4 of the “*User’s Guide*” in Appendix C, shows how to add discharge wells as a time dependent, hydraulic component to the visualization scene. From the visualization, it is easy to see if the pumping wells were assigned to the intended layers.
- **Explaining and communicating conclusions and recommendations.** Zhang used the visualization tool to explain and communicate the interface of fresh- and saltwater in his thesis, as seen in Figure 6.1.

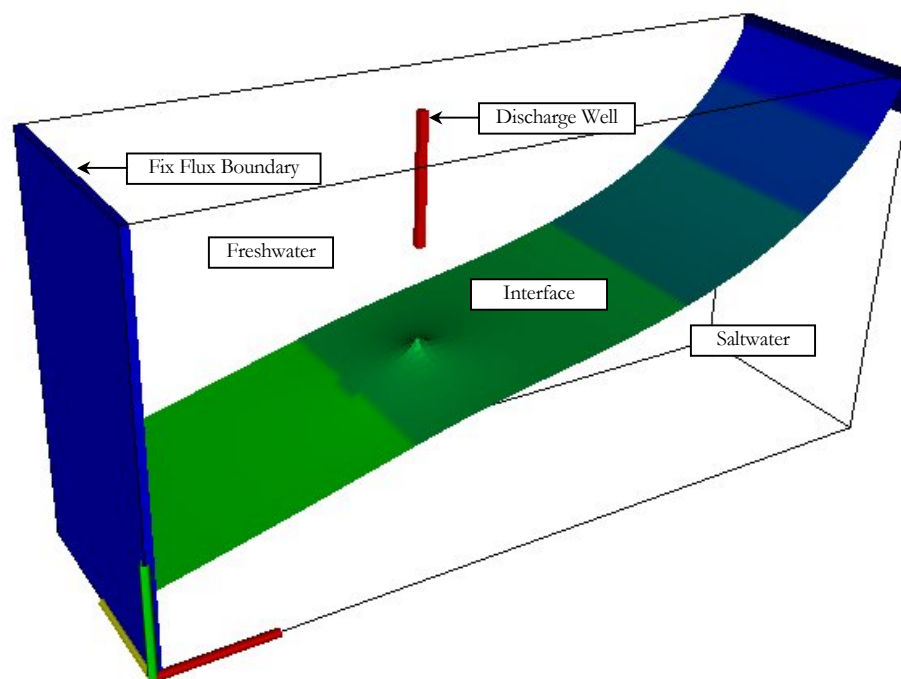


Figure 6.1: Fresh-, saltwater interface.

- Volumes of data.** The wetland model used for the tutorial in the “*User’s Guide*” is discretized into 50 rows, 90 columns and 2 layers, making the number of cells that contains data 9000 ( $50 \times 90 \times 2$ ). Imagine comparing 9000 values manually to one another for an understanding of the modeled result. “*Step 3: Add Time Dependent Hydraulic Component*” in Chapter 4 of the “*User’s Guide*” in Appendix C, shows how these 9000 cells of data are comprehended and summarized with the visualization tool.
- Motivating expenses.** A problem scenario introduced into the course for the wetland model, was pollution. The students had to model a solution to protect the wetland and well field. Cut-off walls are such a solution, and the visualization tool would have made its use easy to explain. The solution would be motivated to laymen, who are often the managers making budgetary decisions.

The issues in the “*Hypothesis and Research Questions*” section in Chapter 1, and the “*Advantages and Disadvantages*” section in Chapter 2, with the way they are addressed with the visualization tool, are listed as follows:

- Interactive visualization** in the form of *motion dynamics*, which enables the visualization objects to be rotated and zoomed, as well as *update dynamics*, where the visualization objects can be scaled, have been successfully implemented into the



visualization tool. Additional interaction in the form of clipping, averaging, decimation and smoothing, has been implemented. The user of the tool has the freedom for example to change the properties of the visualization objects to his own or meaningful liking.

- The human vision system is well adapted to take note of **changes** and therefore the visualization tool has been developed to be able to visualize time dependent data at different time intervals. Creating more than one visualization object at a different time interval enables the user to compare time related results, making it possible to note change. However, a nice addition to this static comparison would be animation. Animation for time-varying phenomena and animation properties, such as controlling the animation speed, portion of scene in a view, amount of detail shown and the geometric relationship of the objects in the scene to one another, would be a great improvement to the current state of the visualization tool.
- The visualization tool has been developed to run on a **PC**, and not on the traditional much more expensive workstations used for 3D computer graphics. This makes it inherently cheaper and more widely used. Despite the technological advances for visualization on a PC, interaction speed was still very much a design issue. LOD actors and decimation, for example have been implemented to keep interaction acceptable.
- Visualizing **abstract data** is a known advantage of visualization. The visualization tool allows the visualization of computational data, as for example fixed head cells, which do not really have physical meaning.
- Like the **money saving** advantage for designers who get visualizing preliminary results, models are sometimes used to investigate different scenarios to obtain the most economic solution. Visualization is an the amplification intelligence tool that will aid communication and understanding of these scenarios, resulting in choosing the best solution and therefore saving money.

### 6.3 Future Research

From the evaluation of the visualization tool, it was evident that animation would be an improvement. Other aesthetic improvements would be to use texture mapping with the Geospatial Model visualization object. Textures of geology will result in a more natural and realistic look. An even more important application of texture mapping for the Geospatial Model would be to map aerial photographs and other maps onto the top of the

first layer, bringing the advantages of Geographical Information Systems (GIS) into the visualization tool. Also for aesthetic improvement, a good idea is to use for example a cylinder shape to visualize discharge and recharge wells. At the moment, colored model cells that are cubes are used to visualize these wells. Why not take it even further? For presentation purposes it would be nice to be able to add trees and buildings.

At present the tool only uses the data from flow and transport models and does not calculate data itself. The model data are not visualized to its fullest potential, because with relative minimal calculation, other informative data can be generated. Flow vectors are some of the data which would be relatively easy to calculate. With excellent vector visualization paradigms in place, it would be a good addition to the visualization tool. Directed glyphs and streamlines are some of the paradigms that can be used to visualize the calculated vector data.

Other ideas are contours on cross-section planes, that cut through the data and isosurfaces for drawdown and hydraulic heads.

The ideas for future research are endless because of the evolving technology, software and ideas; making visualization an exciting challenge.

## **6.4 Conclusion**

The problems set out at the beginning of the study are well addressed by the visualization tool. With add-ons from future research, the visualization tool might become indispensable in the field of geohydrology.

# Appendix A

---

## Program Code

In this appendix, code as used in the visualization tool is listed for all the pseudo code listings in Chapter 5. The pseudo code is used as comments in the code and is in a **bold** typeface.

### A.1 Code creating, normal geometry for Poly Data dataset of the Geospatial Model visualization object.

```
void CGeometry::CreateNormalGeometry(int iLayer)
{
001  // Create a point array.
002  CreatePointArray();
003
004  // Set the size of the point array = 2 * 2 * number of rows * 2 * number of
// columns.
005  m_pcFloatPoints->SetNumberOfPoints(iN_CELL_POINTS
    * m_pcModelInfo->GetNumberOfRows() * m_pcModelInfo->GetNumberOfColumns());
006  CreateNormalGeometryForLayer(iLayer);
}
```

```
void CGeometry::CreateNormalGeometryForLayer(int iLayer,
int iLayerStartCellIndex /* = 0 */, int iLayerStartPointIndex /* = 0 */)
{
001  // Read the top elevation for all cells into a top elevation array.
002  float* pfTopArray = ReadTimeIndependentData("top", iLayer);
003  float fZTopValue = 0;
004
005  // Read the bottom elevation for all cells into a bottom elevation array.
006  float* pfBottomArray = ReadTimeIndependentData("bot", iLayer);
007  float fZBottomValue = 0;
008
009  // For number of rows.
010  for (int i = 0; i < m_pcModelInfo->GetNumberOfRows(); i++)
011      // For number of columns.
012      for (int j = 0; j < m_pcModelInfo->GetNumberOfColumns(); j++)
013      {
014          // If it is an active cell.
015          if (m_bActiveCellsArray[i * m_pcModelInfo->GetNumberOfColumns()
016              + j + iLayerStartCellIndex])
017          {
018              // Get top elevation of cell from top elevation array for
// current row and column.
019              fZTopValue = pfTopArray[i
    * m_pcModelInfo->GetNumberOfColumns() + j];
020
021              // Get bottom elevation of cell from bottom elevation
// array for current row and column.
022              fZBottomValue = pfBottomArray[i
    * m_pcModelInfo->GetNumberOfColumns() + j];
023          }
024          else

```

```

025      {
026          // Use minimum elevation of all bottom and top elevations
          // as top elevation.
027          fZTopValue = m_fMinimumValue;
028
029          // Use minimum elevation of all bottom and top elevations
          // as bottom elevation.
030          fZBottomValue = m_fMinimumValue;
031      }
032
033      // Insert 4 points into point array, using the 4 bottom point
      // indexes, row and column coordinates and bottom elevation.
034      InsertNormalPoints(i, j, fZTopValue, fZBottomValue,
          iLayerStartPointIndex);
035  }
036  delete [] pfTopArray;
037  delete [] pfBottomArray;
038  }

```

```

void CGeometry::InsertNormalPoints(int iRowIndex, int iColumnIndex, float fZTopValue,
float fZBottomValue, int iLayerStartPointIndex /* = 0 */)
{
001  // Create 2 column coordinates for current row and column.
002  float fXP02 = m_pcModelInfo->GetColumnCoordinate(iColumnIndex);
003  float fXP13 = m_pcModelInfo->GetColumnCoordinate(iColumnIndex + 1);
004
005  // Create 2 row coordinates for current row and column.
006  float fYP01 = m_pcModelInfo->GetRowCoordinate(iRowIndex);
007  float fYP23 = m_pcModelInfo->GetRowCoordinate(iRowIndex + 1);
008
009  // Calculate 4 top point indexes into point array for 4 top points of cell.
010  int iPTop[4];
011  CalculateTopIndexes(iPTop, GetP0NormalIndex(iRowIndex, iColumnIndex),
    P2NormalIndex(iRowIndex, iColumnIndex), iLayerStartPointIndex);
012
013  // Insert 4 points into the point array, using the 4 top point indexes, row
    // and column coordinates and top elevation.
014  m_pcFloatPoints->InsertPoint(iPTop[0], fXP02, fYP01, fZTopValue);
015  m_pcFloatPoints->InsertPoint(iPTop[1], fXP13, fYP01, fZTopValue);
016  m_pcFloatPoints->InsertPoint(iPTop[2], fXP02, fYP23, fZTopValue);
017  m_pcFloatPoints->InsertPoint(iPTop[3], fXP13, fYP23, fZTopValue);
018
019  // Calculate 4 bottom point indexes into the point array for 4 bottom
    // points of cell.
020  int iPBottom[4];
021  CalculateBottomIndexes(iPTop, iPBottom, GetNormalNumberOfLayerPoints());
022
023  // Insert 4 points into point array, using the 4 bottom point indexes,
    // row and column coordinates and bottom elevation.
024  m_pcFloatPoints->InsertPoint(iPBottom[0], fXP02, fYP01, fZBottomValue);
025  m_pcFloatPoints->InsertPoint(iPBottom[1], fXP13, fYP01, fZBottomValue);
026  m_pcFloatPoints->InsertPoint(iPBottom[2], fXP02, fYP23, fZBottomValue);
027  m_pcFloatPoints->InsertPoint(iPBottom[3], fXP13, fYP23, fZBottomValue);
028  }

```

```

void CIndexes::CalculateTopIndexes(int iTopPoint[4], int iP0, int iP2,
int iLayerStartPointIndex /* = 0 */)
{
001  iTopPoint[0] = iP0 + iLayerStartPointIndex;
002  iTopPoint[1] = iTopPoint[0] + 1;
003  iTopPoint[2] = iP2 + iLayerStartPointIndex;
004  iTopPoint[3] = iTopPoint[2] + 1;
005  }

```

```

void CIndexes::CalculateBottomIndexes(const int iPTop[4], int iPBottom[4],
int iNumberOfLayerPoints)
{
001  iPBottom[0] = iPTop[0] + iNumberOfLayerPoints;
002  iPBottom[1] = iPBottom[0] + 1;
003  iPBottom[2] = iPTop[2] + iNumberOfLayerPoints;
004  iPBottom[3] = iPBottom[2] + 1;
005  }

```

## A.2 Code creating, average geometry for Poly Data dataset of the Geospatial Model visualization object.

```

void CGeometry::CreateAverageGeometry(int iLayer)
{
001  // Create average points.
002  CreateAveragePoints(iLayer);
003
004  // Create a point array.
005  CreatePointArray();
006
007  // Set the size of the point array = 2 * (number of rows + 1)
    // (number of columns + 1).
008  m_pcFloatPoints->SetNumberOfPoints(2 * (m_pcModelInfo->GetNumberOfRows() + 1)
    * (m_pcModelInfo->GetNumberOfColumns() + 1));
009
010  // For indexes of point array.
011  for (int i = 0; i < 2 * (m_pcModelInfo->GetNumberOfRows() + 1)
    * (m_pcModelInfo->GetNumberOfColumns() + 1); i++)
012      // Calculate row and column indexes using the point array index.
013      // Insert a point into the point array where:
014      //     point.x = Get the column coordinate for column index.
015      //     point.y = Get the row coordinate for row index.
016      //     point.z = Get the average elevation from average point array.
017      m_pcFloatPoints->InsertPoint(
        i,
        m_pcModelInfo->GetColumnCoordinate(CalculateColumnIndex(i)),
        m_pcModelInfo->GetRowCoordinate(CalculateRowIndex(i)),
        m_pcAverageArray.GetAt(i)->GetAverage());
}

```

```

void CGeometry::CreateAveragePoints(int iLayer, int iLayerStartCellIndex /* = 0 */)
{
001  // Set the size of the average point array = 2 * (number of rows + 1)
    // (number of columns + 1).
002  m_pcAverageArray.SetSize(2 * (m_pcModelInfo->GetNumberOfRows() + 1)
    * (m_pcModelInfo->GetNumberOfColumns() + 1), 1);
003
004  // Initialize the average point array.
005  for (int i = 0; i < 2 * (m_pcModelInfo->GetNumberOfRows() + 1)
    * (m_pcModelInfo->GetNumberOfColumns() + 1); i++)
006  {
007      CAverage* pcAverage = new CAverage();
008      m_pcAverageArray[i] = pcAverage;
009  }
010
011  // Read the top elevation for all cells into a top elevation array.
012  float* pfTopArray = ReadTimeIndependentData("top", iLayer);
013
014  // Read the bottom elevation for all cells into a bottom elevation array.
015  float* pfBottomArray = ReadTimeIndependentData("bot", iLayer);
016
017  // For number of rows.
019  for (i = 0; i < m_pcModelInfo->GetNumberOfRows(); i++)
020      // For number of columns.
021      for (int j = 0; j < m_pcModelInfo->GetNumberOfColumns(); j++)
022          // If it is an active cell.
023          if (m_bActiveCellsArray[i * m_pcModelInfo->GetNumberOfColumns()
    + j + iLayerStartCellIndex])
024              InsertAveragePoints(i, j, pfTopArray, pfBottomArray);
025
026  // clean
027  delete [] pfTopArray;
028  delete [] pfBottomArray;
}

```

```

void CGeometry::InsertAveragePoints(int iRowIndex, int iColumnIndex,
float* pfTopArray, float* pfBottomArray)
{

```

```

001 // Calculate 4 top average point indexes into average point array for cell.
002 int iPTop[4];
003 CalculateTopIndexes(    iPTop,
                        GetP0AverageIndex(iRowIndex, iColumnIndex),
                        GetP2AverageIndex(iRowIndex, iColumnIndex));
004
005 // Add the top elevation to the average point array, using the 4 top
// average point indexes.
006 for (int iPointIndex = 0; iPointIndex < 4; iPointIndex++)
007     m_pcAverageArray[iPTop[iPointIndex]]->AddToSum(pfTopArray[iRowIndex
    * m_pcModelInfo->GetNumberOfColumns() + iColumnIndex]);
008
009 // Calculate 4 bottom average point indexes into average point array for cell.
010 int iPBottom[4];
011 CalculateBottomIndexes(iPTop, iPBottom, GetAverageNumberOfLayerPoints());
012
013 // Add the bottom elevation to the average point array, using the 4
// bottom average point indexes.
014 for (iPointIndex = 0; iPointIndex < 4; iPointIndex++)
015     m_pcAverageArray[iPBottom[iPointIndex]]->AddToSum(
    pfBottomArray[iRowIndex * m_pcModelInfo->GetNumberOfColumns()
    + iColumnIndex]);
}

```

### A.3 Code creating, normal top topology for Poly Data dataset of the Geospatial Model visualization object.

```

void CTopQuadTopology::CreateNormalTopology(vtkCellArray* pcCellArray, BOOL bAverage,
int iNumberOfNormalLayerPoints /* = 0 */)
{
001     // Horizontal top.
002     CreateHorizontalTopology(pcCellArray, bAverage, iNumberOfNormalLayerPoints);
003
004     // Right vertical.
005     CreateNormalRightVerticalTopology(pcCellArray, iNumberOfNormalLayerPoints);
006
007     // Front vertical.
008     CreateNormalFrontVerticalTopology(pcCellArray, iNumberOfNormalLayerPoints);
}

```

```

void CTopQuadTopology::CreateHorizontalTopology(vtkCellArray* pcCellArray,
BOOL bAverage, int iNumberOfNormalLayerPoints /* = 0 */)
{
001     // For number of rows.
002     for (int i = 0; i < m_pcModelInfo->GetNumberOfRows(); i++)
003         // For number of columns.
004         for (int j = 0; j < m_pcModelInfo->GetNumberOfColumns(); j++)
005             // If it is an active cell.
006             if (m_bActiveCellsArray[i * m_pcModelInfo->GetNumberOfColumns()
    + j])
007                 // Insert the next quadrilateral into the cell array, using
// the 4 horizontal, point array indexes.
008                 InsertNextCell(    pcCellArray,
                                GetHorizontalTopQuadIndexes(
                                bAverage, i, j,
                                iNumberOfNormalLayerPoints));
}

```

```

CQuadIndexes CIndexes::GetHorizontalTopQuadIndexes(BOOL bAverage, int iRowIndex,
int iColumnIndex, int iNumberOfNormalLayerPoints)
{
001     CQuadIndexes cQuadIndexes;
002
003     // Calculate 4 horizontal top or bottom, point array indexes for
// quadrilateral.
004     cQuadIndexes.iP[0] = GetP0Index(bAverage, iRowIndex, iColumnIndex)
    + iNumberOfNormalLayerPoints;
}

```

```

005     cQuadIndexes.iP[1] = cQuadIndexes.iP[0] + 1;
006     cQuadIndexes.iP[2] = GetP2Index(bAverage, iRowIndex, iColumnIndex)
      + iNumberOfNormalLayerPoints;
007     cQuadIndexes.iP[3] = cQuadIndexes.iP[2] + 1;
008     return cQuadIndexes;
}

```

```

void CTopQuadTopology::CreateNormalRightVerticalTopology(vtkCellArray* pcCellArray,
int iNumberOfNormalLayerPoints /* = 0 */)
{
001     // For number of rows.
002     for (int i = 0; i < m_pcModelInfo->GetNumberOfRows(); i++)
003         // For number of columns - 1.
004         for (int j = 0; j < m_pcModelInfo->GetNumberOfColumns() - 1; j++)
005             // If it is an active cell and cell to right is an active cell.
006             if (m_bActiveCellsArray[i * m_pcModelInfo->GetNumberOfColumns()
      + j]
      && m_bActiveCellsArray[i * m_pcModelInfo->GetNumberOfColumns()
      + j + 1])
007                 // Insert the next quadrilateral into the cell array, using
      // the 4 right vertical, point array indexes.
008                 InsertNextCell( pcCellArray,
      GetNormalRightVerticalQuadIndexes(
      i, j, iNumberOfNormalLayerPoints));
}

```

```

CQuadIndexes CIndexes::GetNormalRightVerticalQuadIndexes(int iRowIndex, int
iColumnIndex, int iNumberOfNormalLayerPoints)
{
001     CQuadIndexes cQuadIndexes;
002
003     // Calculate 4 right vertical, point array indexes for quadrilateral.
004     cQuadIndexes.iP[0] = GetP0NormalIndex(iRowIndex, iColumnIndex) + 1
      + iNumberOfNormalLayerPoints;
005     cQuadIndexes.iP[1] = cQuadIndexes.iP[0] + 1;
006     cQuadIndexes.iP[2] = GetP2NormalIndex(iRowIndex, iColumnIndex) + 1
      + iNumberOfNormalLayerPoints;
007     cQuadIndexes.iP[3] = cQuadIndexes.iP[2] + 1;
008     return cQuadIndexes;
}

```

```

void CTopQuadTopology::CreateNormalFrontVerticalTopology(vtkCellArray* pcCellArray,
int iNumberOfNormalLayerPoints /* = 0 */)
{
001     // For number of rows - 1.
002     for (int i = 0; i < m_pcModelInfo->GetNumberOfRows() - 1; i++)
003         // For number of columns.
004         for (int j = 0; j < m_pcModelInfo->GetNumberOfColumns(); j++)
005             // If it is an active cell and cell in front is an active cell.
006             if (m_bActiveCellsArray[i * m_pcModelInfo->GetNumberOfColumns()
      + j]
      && m_bActiveCellsArray[i * m_pcModelInfo->GetNumberOfColumns()
      + j + m_pcModelInfo->GetNumberOfColumns()])
007                 // Insert the next quadrilateral into the cell array, using
      // the 4 front vertical, point array indexes.
008                 InsertNextCell( pcCellArray,
      GetNormalFrontVerticalQuadIndexes(
      i, j, iNumberOfNormalLayerPoints));
}

```

```

CQuadIndexes CIndexes::GetNormalFrontVerticalQuadIndexes(int iRowIndex, int
iColumnIndex, int iNumberOfNormalLayerPoints)
{
001     CQuadIndexes cQuadIndexes;
002
003     // Calculate 4 front vertical, point array indexes for quadrilateral.
004     cQuadIndexes.iP[0] = GetP2NormalIndex(iRowIndex, iColumnIndex)
      + iNumberOfNormalLayerPoints;
005     cQuadIndexes.iP[1] = cQuadIndexes.iP[0] + 1;

```

```

006     cQuadIndexes.iP[2] = cQuadIndexes.iP[0] + iN_EDGE_POINTS
      * m_pcModelInfo->GetNumberOfColumns();
007     cQuadIndexes.iP[3] = cQuadIndexes.iP[1] + iN_EDGE_POINTS
      * m_pcModelInfo->GetNumberOfColumns();
008     return cQuadIndexes;
    }

```

## A.4 Code creating, average top topology for Poly Data dataset of the Geospatial Model visualization object.

```

void CTopQuadTopology::CreateHorizontalTopology(vtkCellArray* pcCellArray,
BOOL bAverage, int iNumberOfNormalLayerPoints /* = 0 */)
{
001     // For number of rows.
002     for (int i = 0; i < m_pcModelInfo->GetNumberOfRows(); i++)
003         // For number of columns.
004         for (int j = 0; j < m_pcModelInfo->GetNumberOfColumns(); j++)
005             // If it is an active cell.
006             if (m_bActiveCellsArray[i * m_pcModelInfo->GetNumberOfColumns()
+ j])
007                 // Insert the next quadrilateral into the cell array, using
008                 // the 4 horizontal, point array indexes.
                 InsertNextCell( pcCellArray,
                                GetHorizontalTopQuadIndexes(
                                    bAverage, i, j,
                                    iNumberOfNormalLayerPoints));
}

```

## A.5 Code creating, normal sides topology for Poly Data dataset of the Geospatial Model visualization object.

```

void CSidesQuadTopology::CreateSidesTopology(vtkCellArray* pcCellArray, BOOL bAverage)
{
001     // For number of rows.
002     for (int i = 0; i < m_pcModelInfo->GetNumberOfRows(); i++)
003         // For number of columns.
004         for (int j = 0; j < m_pcModelInfo->GetNumberOfColumns(); j++)
005             // If it is an active cell.
006             if (m_bActiveCellsArray[i * m_pcModelInfo->GetNumberOfColumns() + j])
007             {
008                 // left
009                 CreateLeftSidesTopology(pcCellArray, bAverage, i, j);
010
011                 // right
012                 CreateRightSidesTopology(pcCellArray, bAverage, i, j);
013
014                 // back
015                 CreateBackSidesTopology(pcCellArray, bAverage, i, j);
016
017                 // front
018                 CreateFrontSidesTopology(pcCellArray, bAverage, i, j);
019             }
}

```

```

void CSidesQuadTopology::CreateLeftSidesTopology(vtkCellArray* pcCellArray, BOOL
bAverage, int iRowIndex, int iColumnIndex)
{
001     // If it is the first column or if the cell to the left of cell is inactive.
002     if ((iColumnIndex == 0)
        || (!m_bActiveCellsArray[iRowIndex * m_pcModelInfo->GetNumberOfColumns()
+ iColumnIndex - 1]))

```



```

003      // Insert the next quadrilateral into the cell array, using the 4, point
      // array indexes.
004      InsertNextCell(pcCellArray, GetLeftSidesQuadIndexes(bAverage, iRowIndex,
          iColumnIndex));
    }

```

```

CQuadIndexes CIndexes::GetLeftSidesQuadIndexes(BOOL bAverage, int iRowIndex,
int iColumnIndex)
{
001     CQuadIndexes cQuadIndexes;
002
003     // Calculate 4, point array indexes for quadrilateral.
004     cQuadIndexes.iP[0] = GetP0Index(bAverage, iRowIndex, iColumnIndex);
005     cQuadIndexes.iP[1] = GetP2Index(bAverage, iRowIndex, iColumnIndex);
006     cQuadIndexes.iP[2] = cQuadIndexes.iP[0] + GetNumberOfLayerPoints(bAverage);
007     cQuadIndexes.iP[3] = cQuadIndexes.iP[1] + GetNumberOfLayerPoints(bAverage);
008     return cQuadIndexes;
}

```

```

void CSidesQuadTopology::CreateRightSidesTopology(vtkCellArray* pcCellArray, BOOL
bAverage, int iRowIndex, int iColumnIndex)
{
001     // If it is the last column or if the cell to the right of cell is inactive.
002     if ((iColumnIndex == m_pcModelInfo->GetNumberOfColumns() - 1)
        || (!m_bActiveCellsArray[iRowIndex * m_pcModelInfo->GetNumberOfColumns()
        + iColumnIndex + 1]))
003         // Insert the next quadrilateral into the cell array, using the 4, point
        // array indexes.
004         InsertNextCell(pcCellArray, GetRightSidesQuadIndexes(bAverage, iRowIndex,
            iColumnIndex));
}

```

```

CQuadIndexes CIndexes::GetRightSidesQuadIndexes(BOOL bAverage, int iRowIndex,
int iColumnIndex)
{
001     CQuadIndexes cQuadIndexes;
002
003     // Calculate 4, point array indexes for quadrilateral.
004     cQuadIndexes.iP[0] = GetP0Index(bAverage, iRowIndex, iColumnIndex) + 1;
005     cQuadIndexes.iP[1] = GetP2Index(bAverage, iRowIndex, iColumnIndex) + 1;
006     cQuadIndexes.iP[2] = cQuadIndexes.iP[0] + GetNumberOfLayerPoints(bAverage);
007     cQuadIndexes.iP[3] = cQuadIndexes.iP[1] + GetNumberOfLayerPoints(bAverage);
008     return cQuadIndexes;
}

```

```

void CSidesQuadTopology::CreateBackSidesTopology(vtkCellArray* pcCellArray, BOOL
bAverage, int iRowIndex, int iColumnIndex)
{
001     // If it is the first row or if the cell to the back of cell is inactive.
002     if ((iRowIndex == 0)
        || (!m_bActiveCellsArray[iRowIndex * m_pcModelInfo->GetNumberOfColumns()
        + iColumnIndex - m_pcModelInfo->GetNumberOfColumns()]))
003         // Insert the next quadrilateral into the cell array, using the 4, point
        // array indexes.
004         InsertNextCell(pcCellArray, GetBackSidesQuadIndexes(bAverage, iRowIndex,
            iColumnIndex));
}

```

```

CQuadIndexes CIndexes::GetBackSidesQuadIndexes(BOOL bAverage, int iRowIndex,
int iColumnIndex)
{
001     CQuadIndexes cQuadIndexes;
002
003     // Calculate 4, point array indexes for quadrilateral.
004     cQuadIndexes.iP[0] = GetP0Index(bAverage, iRowIndex, iColumnIndex);
005     cQuadIndexes.iP[1] = cQuadIndexes.iP[0] + 1;
006     cQuadIndexes.iP[2] = cQuadIndexes.iP[0] + GetNumberOfLayerPoints(bAverage);
007     cQuadIndexes.iP[3] = cQuadIndexes.iP[2] + 1;
008     return cQuadIndexes;
}

```

```
}

```

```
void CSidesQuadTopology::CreateFrontSidesTopology(vtkCellArray* pcCellArray, BOOL
bAverage, int iRowIndex, int iColumnIndex)
{
001  // If it is the last row or if the cell to front of the cell is inactive.
002  if ((iRowIndex == m_pcModelInfo->GetNumberOfRows() - 1)
003      || (!m_bActiveCellsArray[iRowIndex * m_pcModelInfo->GetNumberOfColumns()
+ iColumnIndex + m_pcModelInfo->GetNumberOfColumns()])))
004      // Insert the next quadrilateral into the cell array, using the 4, point
// array indexes.
005      InsertNextCell(pcCellArray, GetFrontSidesQuadIndexes(bAverage, iRowIndex,
iColumnIndex));
}

```

```
CQuadIndexes CIndexes::GetFrontSidesQuadIndexes(BOOL bAverage, int iRowIndex,
int iColumnIndex)
{
001  CQuadIndexes cQuadIndexes;
002
003  // Calculate 4, point array indexes for quadrilateral.
004  cQuadIndexes.iP[0] = GetP2Index(bAverage, iRowIndex, iColumnIndex);
005  cQuadIndexes.iP[1] = cQuadIndexes.iP[0] + 1;
006  cQuadIndexes.iP[2] = cQuadIndexes.iP[0] + GetNumberOfLayerPoints(bAverage);
007  cQuadIndexes.iP[3] = cQuadIndexes.iP[2] + 1;
008  return cQuadIndexes;
}

```

## A.6 Code creating, normal geometry and the normal scalar dataset attribute for Poly Data dataset of the Potentiometric Surface visualization object.

```
void CPotentiometricSurface::CreateNormalGeometryAndScalarDatasetAttribute()
{
001  // Create a point array.
002  CreatePointArray();
003
004  // Set the size of the point array = 4 * number of rows * number of columns.
005  m_pcFloatPoints->SetNumberOfPoints(4 * m_pcModel->GetNumberOfRows()
* m_pcModel->GetNumberOfColumns());
006
007  // Create a scalar array.
008  CreateScalarArray();
009
010  // Set the size of the scalar array = 4 * number of rows * number of columns.
011  m_pcScalars->SetNumberOfScalars(4 * m_pcModel->GetNumberOfRows()
* m_pcModel->GetNumberOfColumns());
012
013  // head
014  float fHead;
015
016  // For number of rows.
017  for (int i = 0; i < m_pcModel->GetNumberOfRows(); i++)
018      // For number of columns.
019      for (int j = 0; j < m_pcModel->GetNumberOfColumns(); j++)
020      {
021          // If it is an active cell and head value is in Color Table range.
022          if (m_bActiveCellsArray[i * m_pcModel->GetNumberOfColumns() + j]
&& m_pfHeads[i * m_pcModel->GetNumberOfColumns() + j]
>= m_pcColor->m_cColorTable.GetMinimumScalarValue()
&& m_pfHeads[i * m_pcModel->GetNumberOfColumns() + j]
<= m_pcColor->m_cColorTable.GetMaximumScalarValue())
023              // Get groundwater head of cell from groundwater head array for
// current row and column.
024              fHead = m_pfHeads[i * m_pcModel->GetNumberOfColumns() + j];

```

```

025         else
026             // Use minimum groundwater head of all groundwater heads as
             // groundwater head.
027             fHead = m_pcColor->m_cColorTable.GetMinimumScalarValue();
028
029             // Insert normal points and scalars
030             InsertNormalPointsAndScalars(i, j, fHead);
031     }
}

```

```

void CPotentiometricSurface::InsertNormalPointsAndScalars(int iRowIndex, int
iColumnIndex, float fHead)
{
001     // Create 2 column coordinates for current row and column.
002     float fXP02 = m_pcModel->GetColumnCoordinate(iColumnIndex);
003     float fXP13 = m_pcModel->GetColumnCoordinate(iColumnIndex + 1);
004
005     // Create 2 row coordinates for current row and column.
006     float fYP02 = m_pcModel->GetRowCoordinate(iRowIndex);
007     float fYP13 = m_pcModel->GetRowCoordinate(iRowIndex + 1);
008
009     // Calculate 4 point indexes into point array for 4 points of cell.
010     int iTopPoint[4];
011     CalculateTopIndexes(iTopPoint, GetP0NormalIndex(iRowIndex, iColumnIndex),
        GetP2NormalIndex(iRowIndex, iColumnIndex));
012
013     // Insert 4 points into point array, using the 4, point indexes, row and column
    // coordinates and head value.
014     m_pcFloatPoints->InsertPoint(iTopPoint[0], fXP02, fYP02, fHead);
015     m_pcFloatPoints->InsertPoint(iTopPoint[1], fXP13, fYP02, fHead);
016     m_pcFloatPoints->InsertPoint(iTopPoint[2], fXP02, fYP13, fHead);
017     m_pcFloatPoints->InsertPoint(iTopPoint[3], fXP13, fYP13, fHead);
018
019     // Insert 4 scalars into scalar array, using the 4, point indexes and head value.
020     for (int iIndex = 0; iIndex < 4; iIndex++)
021         m_pcScalars->SetScalar(iTopPoint[iIndex], fHead);
}

```

## A.7 Code creating, average geometry and the average scalar dataset attribute for Poly Data dataset of the Potentiometric Surface visualization object.

```

void CPotentiometricSurface::CreateAverageGeometryAndScalarDatasetAttribute()
{
001     // Create average points.
002     CreateAveragePoints();
003
004     // Create a point array.
005     CreatePointArray();
006
007     // Set the size of the point array = (number of rows + 1)
    // * (number of columns + 1).
008     m_pcFloatPoints->SetNumberOfPoints((m_pcModel->GetNumberOfRows() + 1)
        * (m_pcModel->GetNumberOfColumns() + 1));
009
010     // Create the scalar array.
011     CreateScalarArray();
012
013     // Set the size of the scalar array = (number of rows + 1)
    // * (number of columns + 1).
014     m_pcScalars->SetNumberOfScalars((m_pcModel->GetNumberOfRows() + 1)
        * (m_pcModel->GetNumberOfColumns() + 1));
015
016     // For indexes in point array.
017     for (int i = 0; i < (m_pcModel->GetNumberOfRows() + 1)
        * (m_pcModel->GetNumberOfColumns() + 1); i++)
018     {

```

```

019      // Calculate row and column indexes using the point array index.
020      // Insert a point into the point array where:
021      // point.x = Get column coordinate for column index.
022      // point.y = Get row coordinate for row index.
023      // point.z = Get average groundwater head from average point array.
024      m_pcFloatPoints->InsertPoint(
          i,
          m_pcModel->GetColumnCoordinate(CalculateColumnIndex(i)),
          m_pcModel->GetRowCoordinate(CalculateRowIndex(i)),
          m_pcAverageArray.GetAt(i)->GetAverage());
025
026      // Insert average groundwater head into the scalar array.
027      m_pcScalars->SetScalar(i, m_pcAverageArray.GetAt(i)->GetAverage());
028  }
}

```

```

void CPotentiometricSurface::CreateAveragePoints()
{
001  // Set the size of the average point array = (number of rows + 1)
    // * (number of columns + 1).
002  m_pcAverageArray.SetSize((m_pcModel->GetNumberOfRows() + 1)
    * (m_pcModel->GetNumberOfColumns() + 1), 1);
003
004  // Initialize the average point array.
005  for (int i = 0; i < (m_pcModel->GetNumberOfRows() + 1)
    * (m_pcModel->GetNumberOfColumns() + 1); i++)
006  {
007      CAverage* bAverage = new CAverage();
008      m_pcAverageArray[i] = bAverage;
009  }
010
011  // For number of rows.
012  for (i = 0; i < m_pcModel->GetNumberOfRows(); i++)
013      // For number of columns.
014      for (int j = 0; j < m_pcModel->GetNumberOfColumns(); j++)
015          // If it is an active cell and head value is in Color Table range.
016          if (m_bActiveCellsArray[i * m_pcModel->GetNumberOfColumns() + j]
    && m_pfHeads[i * m_pcModel->GetNumberOfColumns() + j]
    >= m_pcColor->m_cColorTable.GetMinimumScalarValue()
    && m_pfHeads[i * m_pcModel->GetNumberOfColumns() + j]
    <= m_pcColor->m_cColorTable.GetMaximumScalarValue())
017              InsertAveragePoints(i, j);
}

```

```

void CPotentiometricSurface::InsertAveragePoints(int iRowIndex, int iColumnIndex)
{
001  // Calculate 4 point indexes into average point array for cell.
002  int iTopPoint[4];
003  CalculateTopIndexes(iTopPoint, GetP0AverageIndex(iRowIndex, iColumnIndex),
004      GetP2AverageIndex(iRowIndex, iColumnIndex));
005
006  // Insert 4 points into average point array, using the point indexes and head.
007  for (int iPointIndex = 0; iPointIndex < 4; iPointIndex++)
008      m_pcAverageArray[iTopPoint[iPointIndex]]->AddToSum(m_pfHeads[iRowIndex
009      * m_pcModel->GetNumberOfColumns() + iColumnIndex]);
}

```

## A.8 Code creating, geometry and the scalar dataset attribute for Poly Data polygonal dataset of the XYZ Surface visualization object.

```

void CXYZSurfaceGeometryAndTopology::CreateGeometryAndScalarDatasetAttribute(
    CString sPathName, CColor* pcColor)
{
001  // Create a point array.

```

```

002  if (m_pcFloatPoints)
003      m_pcFloatPoints->Delete();
004  m_pcFloatPoints = vtkFloatPoints::New();
005
006  // Create a scalar array.
007  if (m_pcScalars)
008      m_pcScalars->Delete();
009  m_pcScalars = vtkScalars::New();
010
011  ifstream waterLevelFile(sPathName, ios::in);
012  float xyz[3];
013
014  // While not end of file.
015  while (!waterLevelFile.eof())
016  {
017      // Read x, y and z coordinate from file.
018      waterLevelFile >> xyz[0] >> xyz[1] >> xyz[2];
019
020      // If z coordinate is in Color Table range.
021      if (xyz[2] >= pcColor->m_cColorTable.GetMinimumScalarValue()
022          && xyz[2] <= pcColor->m_cColorTable.GetMaximumScalarValue())
023      {
024          // Insert x, y and z coordinate as a point into point array.
025          m_pcFloatPoints->InsertNextPoint(xyz);
026
027          // Insert z coordinate as a scalar into scalar array.
028          m_pcScalars->InsertNextScalar(xyz[2]);
029      }
030  }
031  waterLevelFile.close();
032  }

```

## A.9 Code creating, geometry for Structured Grid dataset of the Isosurface visualization object.

```

void CGeometry::CreateMiddleGeometry(int iNumberOfPoints /* = 1 */)
{
001  // Create a point array.
002  if (m_pcFloatPoints)
003      m_pcFloatPoints->Delete();
004  m_pcFloatPoints = vtkFloatPoints::New();
005
006  // For all the model layers.
007  for (int iLayer = 1; iLayer <= m_pcModelInfo->GetNumberOfLayers(); iLayer++)
008  {
009      // Read the top elevation for all cells into the top elevation array.
010      float* pfTopArray = ReadTimeIndependentData("top", iLayer);
011
012      // Read the bottom elevation for all cells into the bottom elevation array.
013      float* pfBottomArray = ReadTimeIndependentData("bot", iLayer);
014
015      // For number of rows.
016      for (int i = 0; i < m_pcModelInfo->GetNumberOfRows(); i++)
017      {
018          // For number of columns.
019          for (int j = 0; j < m_pcModelInfo->GetNumberOfColumns(); j++)
020          {
021              // Insert the point into the point array, using the middle row
022              // column coordinates and the middle of the top and bottom
023              // elevation.
024              float zIncrement = (pfTopArray[i
025                  * m_pcModelInfo->GetNumberOfColumns() + j] - pfBottomArray[i
026                  * m_pcModelInfo->GetNumberOfColumns() + j]) / (1
027                  + iNumberOfPoints);
028              for (int k = 1; k <= iNumberOfPoints; k++)
029                  m_pcFloatPoints->InsertNextPoint(
030                      m_pcModelInfo->GetMiddleColumnCoordinate(j),
031                      m_pcModelInfo->GetMiddleRowCoordinate(i),
032                      pfBottomArray[i
033                          * m_pcModelInfo->GetNumberOfColumns() + j]
034                          + k * zIncrement);
035          }
036      }
037  }
038  }

```

```

024     }
025     delete [] pfTopArray;
026     delete [] pfBottomArray;
027 }
}

```

## A.10 Code creating, the scalar dataset attribute for Structured Grid dataset of the Isosurface visualization object.

```

void CMT3DMS::CreateScalarDatasetAttribute()
{
001  // Create a scalar array.
002  if (m_pcScalars)
003      m_pcScalars->Delete();
004  m_pcScalars = vtkFloatScalars::New();
005
006  // BOOL to mark the first value that can be used for minimum and maximum
007  BOOL bFirst = TRUE;
008
009  // For all the model layers.
010  for (int layer = 1; layer <= m_pcModel->GetNumberOfLayers(); layer++)
011  {
012      // file name
013      char cName[180];
014      GetConcentrationFileName(cName);
015      float pfTime = m_dTotalElapsedTime;
016      long int liNX = m_pcModel->GetNumberOfColumns();
017      long int liNY = m_pcModel->GetNumberOfRows();
018      long int liNZ = m_pcModel->GetNumberOfLayers();
019      long int liLayer = layer;
020      long int liError = 0;
021      float* pfCon = new float[liNX * liNY];
022
023      // Read the concentration for all cells into concentration array.
024      READCON(cName, // file to read data from
              &pfTime, // output pfTime
              pfCon, // one dimensional array for concentration
              &liNX, // number of columns
              &liNY, // number of rows
              &liNZ, // number of layers
              &liLayer, // layer to read from
              &liError); // error
025
026      // get the minimum and maximum concentration and assign scalar values
027      // For number of rows.
028      for (int i = 0; i < m_pcModel->GetNumberOfRows(); i++)
029          // For number of columns.
030          for (int j = 0; j < m_pcModel->GetNumberOfColumns(); j++)
031          {
032              // If it is an inactive concentration cell.
033              if (m_pcModel->GetConcentrationInactive() != pfCon[i
                  * m_pcModel->GetNumberOfColumns() + j])
034              {
035                  if (bFirst)
036                  {
037                      bFirst = FALSE;
038                      m_fMinimumValue = pfCon[i
039                      * m_pcModel->GetNumberOfColumns() + j];
040                      m_fMaximumValue = pfCon[i
041                      * m_pcModel->GetNumberOfColumns() + j];
042                  }
043                  else
044                  {
045                      if (pfCon[i * m_pcModel->GetNumberOfColumns() + j]
046                      < m_fMinimumValue)
047                          m_fMinimumValue = pfCon[i
048                          * m_pcModel->GetNumberOfColumns() + j];
049                      if (pfCon[i * m_pcModel->GetNumberOfColumns() + j]
050                      > m_fMaximumValue)

```

```

047                                     m_fMaximumValue = pfCon[i
                                * m_pcModel->GetNumberOfColumns() + j];
048                                     }
049                             }
050                             for (int k = 1; k <= m_iNumberOfPoints; k++)
051                                 if (m_pcModel->GetConcentrationInactive() == pfCon[i
                                * m_pcModel->GetNumberOfColumns() + j])
052                                     // Insert next scalar with a value of 0.
053                                     m_pcScalars->InsertNextScalar(0);
054                             else
055                                 // Insert next scalar using the cell concentration.
056                                 m_pcScalars->InsertNextScalar(pfCon[i
                                * m_pcModel->GetNumberOfColumns() + j]);
057                             }
058                             delete [] pfCon;
059     }
}

```

## A.11 Code creating, normal topology and the normal scalar dataset attribute for Unstructured Grid dataset of the Parameter visualization object.

```

void CHexahedronTopology::CreateNormalOrAverageTopology(
vtkScalars* pcScalars,
CString sFolderAndFileName,
CColor* pcColor,
int (CIndexes::*pFuncP0Index)(int, int) const,
int (CIndexes::*pFuncP2Index)(int, int) const,
int (CIndexes::*pFuncLayerStartPointIndex)(int) const,
int iNumberOfLayerPoints)
{
001     // Create a cell array.
002     if (m_pcCellArray)
003         m_pcCellArray->Delete();
004     m_pcCellArray = vtkCellArray::New();
005
006     // For all the model layers.
007     for (int iLayer = 1; iLayer <= m_pcModelInfo->GetNumberOfLayers(); iLayer++)
008     {
009         if (m_bVisibleLayersArray[iLayer - 1])
010         {
011             float* pfParameterArray = new float[m_pcModelInfo->GetNumberOfRows()
                                * m_pcModelInfo->GetNumberOfColumns()];
012
013             // Read parameter value for all cells into parameter array
            // for specific layer.
            m_cReadFile.ReadTimeIndependentData(sFolderAndFileName,
                                                pfParameterArray,
                                                m_pcModelInfo->GetNumberOfRows(),
                                                m_pcModelInfo->GetNumberOfColumns(),
                                                iLayer);
014
015             // For number of rows.
016             for (int i = 0; i < m_pcModelInfo->GetNumberOfRows(); i++)
017                 // For number of columns.
018                 for (int j = 0; j < m_pcModelInfo->GetNumberOfColumns(); j++)
019                     // If it is an active cell and parameter value is in
                    // Color Table range.
020                     if (m_bActiveCellsArray[GetLayerStartCellIndex(iLayer) + i
                                * m_pcModelInfo->GetNumberOfColumns() + j]
                        && pfParameterArray[i * m_pcModelInfo->GetNumberOfColumns()
                                + j] >= pcColor->m_cColorTable.GetMinimumScalarValue()
                        && pfParameterArray[i * m_pcModelInfo->GetNumberOfColumns()
                                + j] <= pcColor->m_cColorTable.GetMaximumScalarValue())
021                     {
022                         // Calculate 4 top point indexes into point array
                        // for 4 top points of cell.
023                         int iPTop[4];

```

```
024         CalculateTopInd(iPTop, (this->pFuncP0Index)(i, j),  
025                             (this->pFuncP2Index)(i, j),  
026                             (this->pFuncLayerStartPointIndex)(iLayer));  
027  
028         // Calculate 4 bottom point indexes into the point  
029         // array for 4 bottom points of cell.  
030         int iPBottom[4];  
031         CalculateBottomIndexes(iPTop, iPBottom,  
032                                 iNumberOfLayerPoints);  
033  
034         // Insert hexahedron into cell array,  
035         // using the 8, point indexes.  
036         InsertHexahedronCell(iPTop, iPBottom);  
037  
038         // Insert parameter value into scalar array.  
039         pcScalars->InsertNextScalar(  
040             pfParameterArray[i  
041                 * m_pcModelInfo->GetNumberOfColumns() + j]);  
042     }  
043     delete [] pfParameterArray;  
044 }  
045 }
```

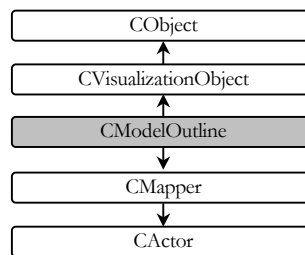


# Appendix B

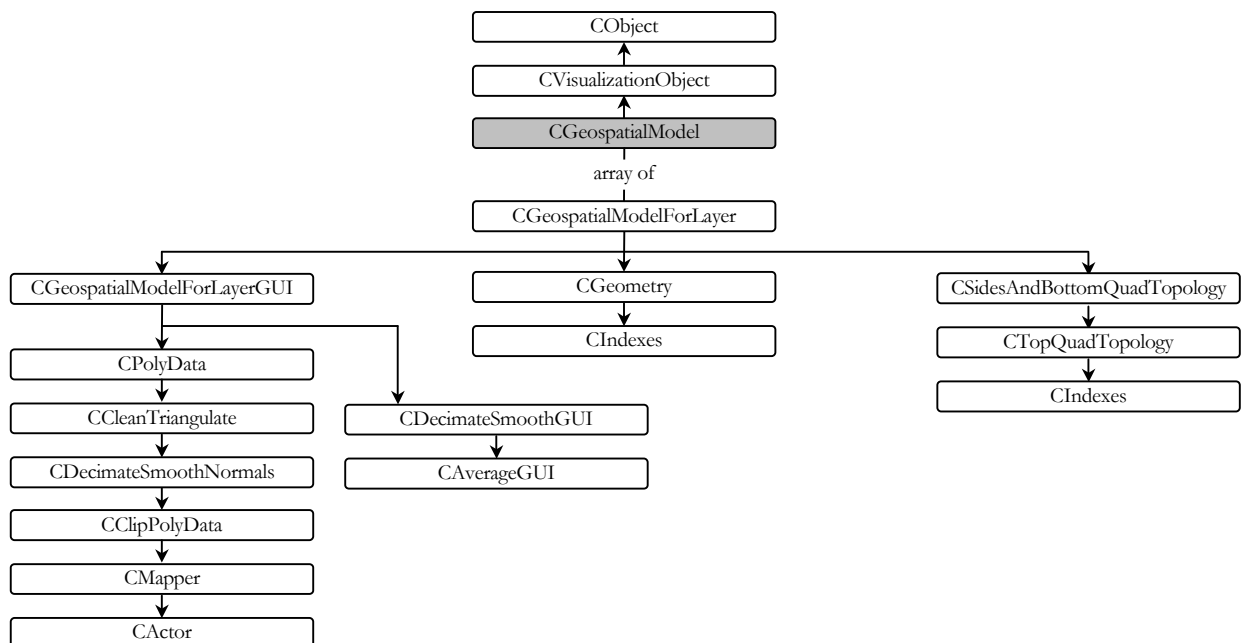
---

## Object Structure

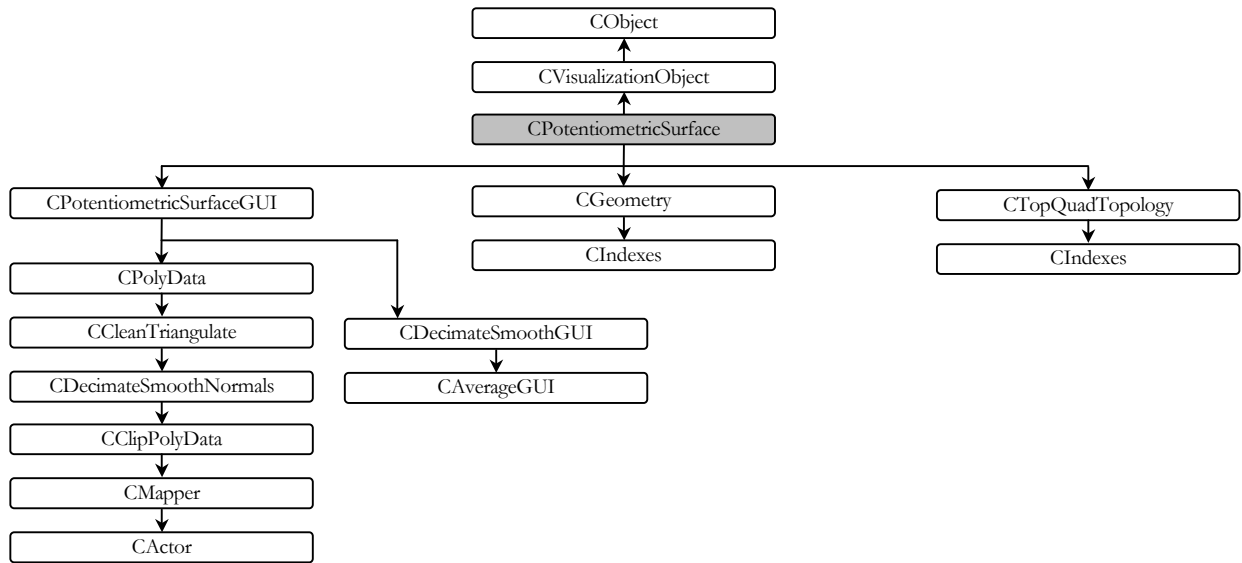
### B.1 Model Outline



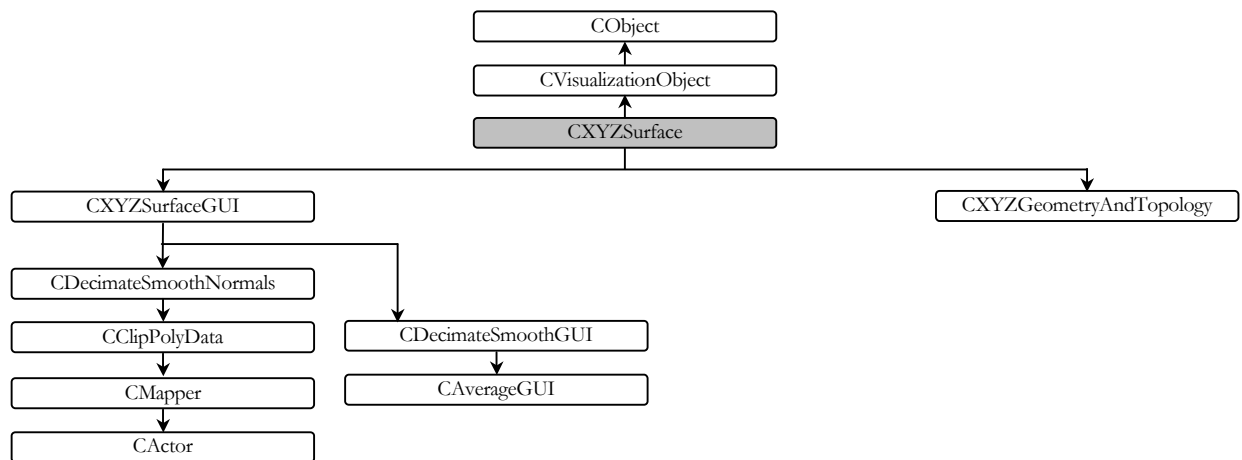
### B.2 Geospatial Model



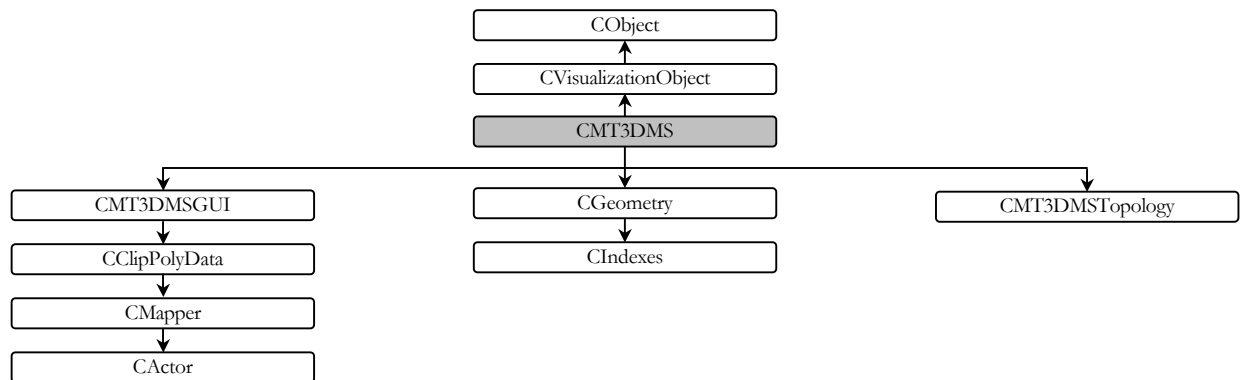
### B.3 Potentiometric Surface



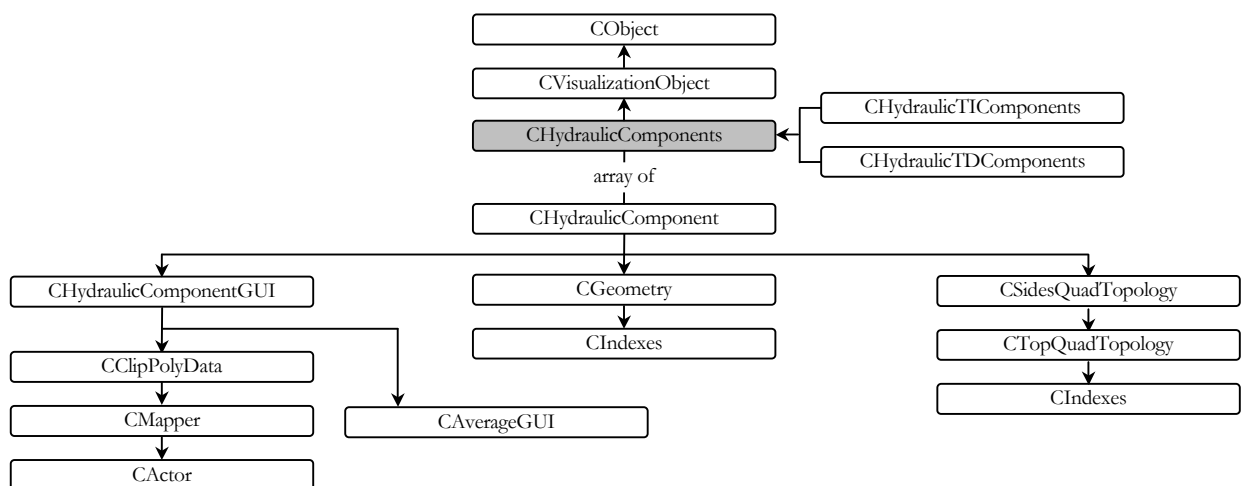
### B.4 XYZ Surface



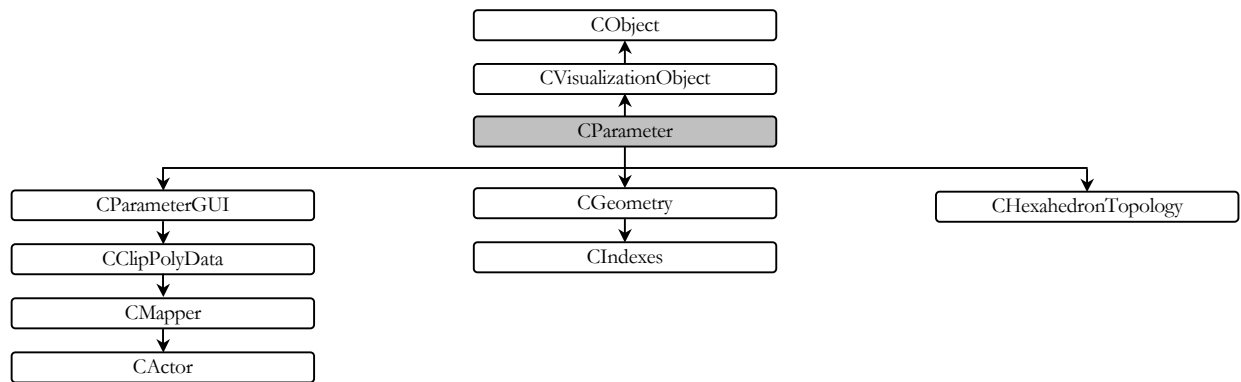
## B.5 Isosurface



## B.6 Hydraulic Components



## B.7 Parameter



# *Appendix C*

---

## **User's Guide**

<b>List of Figures.....</b>	<b>iii</b>
<b>Chapter 1 Introduction.....</b>	<b>1</b>
1.1    Groundwater Explorer .....	1
1.2    Documentation .....	1
1.3    System Requirements .....	2
1.4    Installation.....	3
1.5    Structure of the Following Chapters .....	3
<b>Chapter 2 Getting Started .....</b>	<b>5</b>
2.1    Menus .....	5
2.1.1    File .....	5
2.1.2    Edit.....	7
2.1.3    Interact .....	7
2.1.4    Options.....	8
2.1.5    View.....	11
2.1.6    Help.....	11
2.2    Toolbar .....	11
2.3    Status Bar .....	12
2.4    Control the Scene with the Mouse and Keyboard .....	12
<b>Chapter 3 Visualization Objects.....</b>	<b>13</b>
3.1    Introduction .....	13
3.2    Model Outline .....	13
3.2.1    General.....	14
3.3    Geospatial Model .....	15
3.3.1    General.....	15
3.3.2    Visible.....	18
3.3.3    Average .....	18
3.4    Potentiometric Surface .....	21
3.4.1    General.....	21
3.4.2    Average .....	22
3.4.3    Color .....	23
3.5    XYZ Surface .....	24
3.5.1    General.....	25
3.5.2    Decimate and Smooth.....	26
3.5.3    Color .....	27
3.6    Isosurface .....	27

3.6.1	General .....	28
3.6.2	Contour and Color .....	29
<b>3.7</b>	<b>Hydraulic Components .....</b>	<b>30</b>
3.7.1	General .....	31
3.7.2	Average .....	32
<b>3.8</b>	<b>Parameter .....</b>	<b>33</b>
3.8.1	General .....	34
3.8.2	Average .....	34
3.8.3	Color .....	35
<b>Chapter 4</b>	<b>Tutorial .....</b>	<b>37</b>
<b>4.1</b>	<b>Introduction .....</b>	<b>37</b>
<b>4.2</b>	<b>Conceptualize the Wetland Model.....</b>	<b>37</b>
4.2.1	Step 1: Start GE .....	38
4.2.2	Step 2: Open the Wetland Model.....	38
4.2.3	Step 3: Add Geospatial Model.....	39
4.2.4	Step 4: Scale.....	42
4.2.5	Step 5: Add Potentiometric Surface.....	43
<b>4.3</b>	<b>Contaminated Wetland.....</b>	<b>46</b>
4.3.1	Step 1: Start GE .....	47
4.3.2	Step 2: Open the Wetland Model.....	47
4.3.3	Step 3: Add Time Dependent Hydraulic Component .....	47
4.3.4	Step 4: Add Isosurface .....	49
4.3.5	Step 5: Scale.....	52
4.3.6	Step 6: Change Contours .....	52

# *List of Figures*

---

Figure 1.1:	Visualization objects and scene.....	2
Figure 2.1:	File menu.....	5
Figure 2.2:	Save As dialog box for saving the visualization scene as a Groundwater Explorer file. ....	6
Figure 2.3:	Save As dialog box for saving the visualization scene as a VRML file.....	6
Figure 2.4:	Edit menu.....	7
Figure 2.5:	Interact menu. ....	7
Figure 2.6:	Clip dialog box. ....	8
Figure 2.7:	Scale dialog box.....	8
Figure 2.8:	Options menu.....	9
Figure 2.9:	Axes dialog box.....	9
Figure 2.10:	Visualize dialog box.....	10
Figure 2.11:	Add button items.....	10
Figure 2.12:	View menu. ....	11
Figure 2.13:	Help menu.....	11
Figure 2.14:	Toolbar. ....	11
Figure 2.15:	Status bar. ....	12
Figure 3.1:	Minimum and maximum corners of Model Outline visualization object.....	14
Figure 3.2:	General page of the Model Outline dialog box. ....	14
Figure 3.3:	Geospatial Model visualization object. ....	15
Figure 3.4:	General tab of the Geospatial Model dialog box.....	16
Figure 3.5:	Color Spectrum dialog box. ....	16
Figure 3.6:	Color dialog box. ....	17
Figure 3.7:	Potentiometric Surface visualization object can be seen inside the Geospatial Model visualization object of which the layer opacity values were set to 0.2. ....	17
Figure 3.8:	Visible page of the Geospatial Model dialog box. ....	18
Figure 3.9:	(a) Top, (b) Sides and (c) Bottom of a Geospatial Model visualization object for a layer.....	18
Figure 3.10:	Average page of the Geospatial Model dialog box. ....	19
Figure 3.11:	(a) Layer 1 of Geospatial Model visualization object with Average checked. (b) Layer 1 of Geospatial Model visualization object with Average not checked. ....	19
Figure 3.12:	(a) No decimation. (b) Reduction = 0.3. (c) Reduction = 0.5. (d) Reduction = 0.7. ....	20
Figure 3.13:	(a) No smoothing. (b) Smoothing with Iterations = 10 and Relaxation = 0.2. ....	20
Figure 3.14:	Potentiometric Surface visualization object.....	21
Figure 3.15:	General page of the Potentiometric Surface dialog box. ....	22
Figure 3.16:	Average page of the Potentiometric Surface dialog box. ....	22
Figure 3.17:	Color page of the Potentiometric Surface dialog box. ....	23
Figure 3.18:	Color Level dialog box.....	23
Figure 3.19:	Color Spectrum dialog box.....	24
Figure 3.20:	Color dialog box. ....	24
Figure 3.21:	XYZ Surface visualization object. ....	25
Figure 3.22:	General page of the XYZ Surface dialog box. ....	25
Figure 3.23:	An example of an XYZ file edited in WordPad. ....	26
Figure 3.24:	Decimate and Smooth page of the XYZ Surface dialog box.....	26
Figure 3.25:	Color tab of the XYZ Surface dialog box.....	27

Figure 3.26:	Concentration Isosurface (MT3DMS) visualization object.....	28
Figure 3.27:	General page of the Concentration Isosurface (MT3DMS) dialog box.....	28
Figure 3.28:	Contour and Color page of the Concentration Isosurface (MT3DMS) dialog box.....	29
Figure 3.29:	Color Level dialog box.....	29
Figure 3.30:	Hydraulic Components visualization object.....	30
Figure 3.31:	General page of the Hydraulic Components (Time Independent) dialog box.....	31
Figure 3.32:	General page of the Hydraulic Components (Time Dependent) dialog box.....	32
Figure 3.33:	Average page of the Hydraulic Components dialog box.....	33
Figure 3.34:	Parameter visualization object.....	33
Figure 3.35:	General page of the Parameter dialog box.....	34
Figure 3.36:	Average page of the Parameter dialog box.....	35
Figure 3.37:	Color page of the Parameter dialog box.....	35
Figure 4.1:	Wetland model in PMPATH.....	38
Figure 4.2:	Open... item for opening an existing model.....	38
Figure 4.3:	Open dialog box.....	39
Figure 4.4:	Label Axes, Color Axes and Model Outline visualization object.....	39
Figure 4.5:	Visualize... item.....	40
Figure 4.6:	Visualize dialog box.....	40
Figure 4.7:	Add button items.....	40
Figure 4.8:	General page of the Geospatial Model dialog box.....	41
Figure 4.9:	Visualize dialog box with Geospatial Model added.....	41
Figure 4.10:	Label Axes, Color Axes, Model Outline and Geospatial Model visualization objects.....	42
Figure 4.11:	Scale... item.....	42
Figure 4.12:	Scale dialog box, Z = 20.....	43
Figure 4.13:	Scaled Geospatial Model visualization object.....	43
Figure 4.14:	Visualize... item.....	44
Figure 4.15:	Visualize dialog box.....	44
Figure 4.16:	Add button items.....	44
Figure 4.17:	General page of the Potentiometric Surface dialog box.....	45
Figure 4.18:	Visualize dialog box with Potentiometric Surface added.....	45
Figure 4.19:	Label Axes, Color Axes, Model Outline, Geospatial Model and Potentiometric Surface visualization objects.....	46
Figure 4.20:	Visualize... item.....	47
Figure 4.21:	Visualize dialog box.....	47
Figure 4.22:	Add button items.....	48
Figure 4.23:	General page of the Hydraulic Components (Time Dependent) dialog box.....	48
Figure 4.24:	Visualize dialog box with Hydraulic Components (Time Dependent) visualization object added.....	49
Figure 4.25:	Label Axes, Color Axes, Model Outline and Hydraulic Components (Time Dependent) visualization objects.....	49
Figure 4.26:	Visualize... item.....	50
Figure 4.27:	Visualize dialog box.....	50
Figure 4.28:	Add button items.....	50
Figure 4.29:	General tab of the Concentration Isosurface (MT3DMS) dialog box.....	51
Figure 4.30:	Visualize dialog box with Concentration Isosurface (MT3DMS) visualization object added.....	51
Figure 4.31:	Label Axes, Color Axes, Model Outline, Hydraulic Components and Concentration Isosurface (MT3DMS) visualization objects.....	52
Figure 4.32:	Visualize... item.....	52



---

Figure 4.33:	Visualize dialog box.....	53
Figure 4.34:	Concentration Isosurface (MT3DMS) dialog box. ....	53
Figure 4.35:	Contour and Color page.....	54
Figure 4.36:	Color Level dialog box.....	54
Figure 4.37:	Changed Color Level dialog box. ....	54
Figure 4.38:	Contour and Color page.....	55
Figure 4.39:	Color dialog box. ....	55
Figure 4.40:	Contour and Color page.....	56
Figure 4.41:	5 µg/l Isosurface.....	56



# *Chapter 1*

---

## **Introduction**

### **1.1 Groundwater Explorer**

Groundwater Explorer (GE) is a software tool for the visualization of data from groundwater flow and transport models. The present version of GE visualizes models created by the simulation system, PMWIN (Chiang and Kinzelbach, 2000). This guide explains the different components of GE and guides the user to easily use GE. At the end of the guide a step-by-step tutorial explains how to use GE for the Wetland model.

### **1.2 Documentation**

Formatted text in this guide is used as follows:

- A **Bold** typeface is used for menus, dialog box names and tab labels.
- An *Italic* typeface is used for folder paths and filenames.
- ..... (dotted lines) are used to show where pictures were cut.

Terms used in this guide are as follows:

- Visualization Object (Figure 1.1) – An object that represents an aspect of groundwater model data in a 3D space.
- Scene (Figure 1.1) – Collection of visualization objects, visible on the display. GE can add arbitrary numbers of visualization objects of the following types to a scene. Refer to Chapter 3 for details.
  - Model Outline – The bounding box of a groundwater model.
  - Geospatial Model – Topography of each model layer.
  - Potentiometric Surface – Calculated hydraulic head values.
  - XYZ Surface – Measured values.

- ❑ Isosurface of concentration values calculated with MT3DMS (Zheng, 1998), MT3D99 (Zheng and Wang, 1999), PHT3D (Prommer, 2000) and RT3D (Clement, 1997).
- ❑ Hydraulic Components, for example pumping wells.
- ❑ Model parameter values.

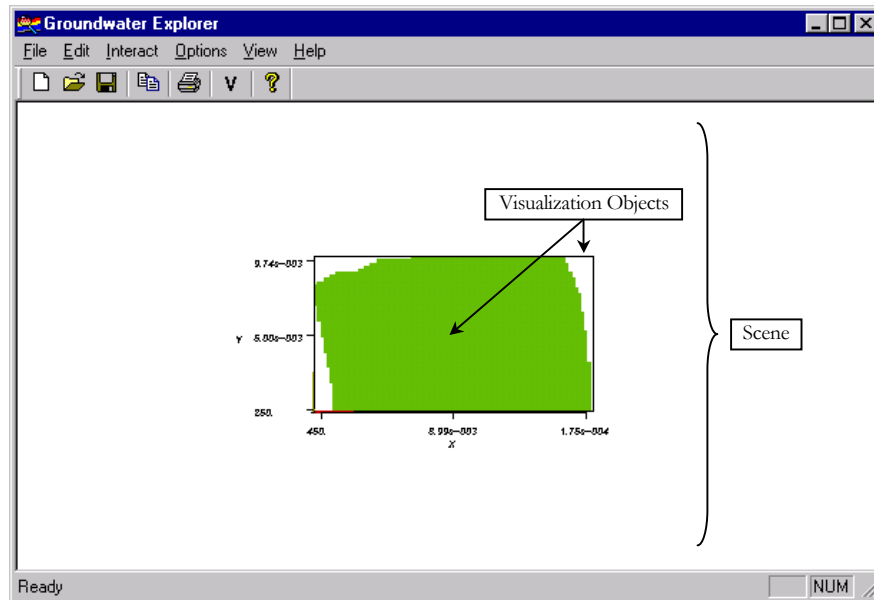


Figure 1.1: Visualization objects and scene.

### 1.3 System Requirements

GE runs on a personal computer (PC) with Windows 98 or Windows NT as the operating system. The recommended hardware is as follow:

- Pentium III Processor
- 128 MB RAM
- 4MB AGP Video Card
- SVGA Monitor
- Mouse

## **1.4      Installation**

Insert the GE CD into the CD-ROM drive. Use Windows Explorer to browse for the installation program *Setup.exe*, double click the file and follow the instructions to install GE.

## **1.5      Structure of the Following Chapters**

Chapter 2 helps you to get started. Chapter 3 explains the details of the visualization objects. It also explains how to create and change the visualization objects for certain visual effects. The last chapter is a step-by-step tutorial.



# Chapter 2

---

## Getting Started

### 2.1 Menus

GE's menus are described in the following sections. Most of the menus are gray and inactive and cannot be used until a scene or groundwater model is opened.

#### 2.1.1 File

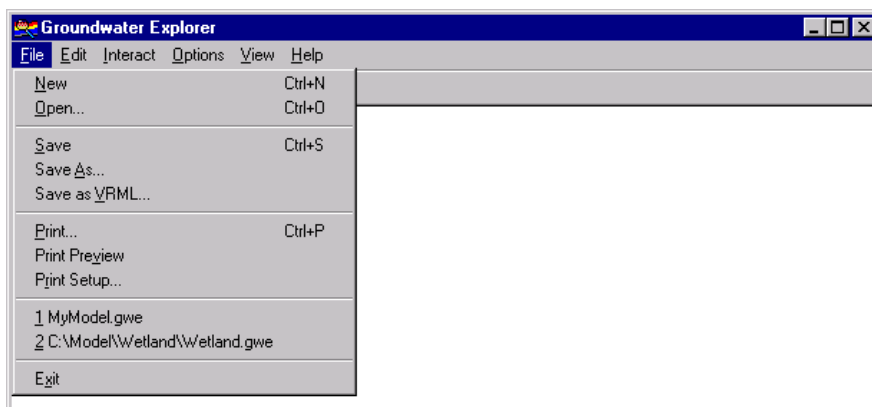


Figure 2.1: File menu.


The File menu (Figure 2.1) consists of the following items:

- **N**ew – Clean the visualization tool's scene for a new scene.
- **O**pen... – Opens an existing visualization scene or a groundwater model with an Open dialog box. You can open a saved scene (with a *gwe* file extension) or a PMWIN model (with a *pm5* file extension). GE searches for all available information (input data, calculated results etc.) within the specified file folder.
- **S**ave – Saves the scene using the current filename displayed on the title bar. If a filename is not specified, this item displays the **Save As** dialog box (see below).
- **S**ave **A**s... – Displays the **Save As** dialog box (Figure 2.2) for saving the scene.



Figure 2.2: **Save As** dialog box for saving the visualization scene as a Groundwater Explorer file.

To save a scene:

1. Specify the filename to be used in the **File name** box.
  2. Choose **Groundwater Explorer (\*.gwe)** as the **Save as type**.
  3. Click the  button. The *gwe* file extension is automatically added to the given filename. In Figure 2.3 for example, the current scene will be saved as *Wetland.gwe* in the wetland folder.
- **Save as VRML...** – Displays the **Save As** dialog box (Figure 2.3) for saving the visualization scene as a VRML file.

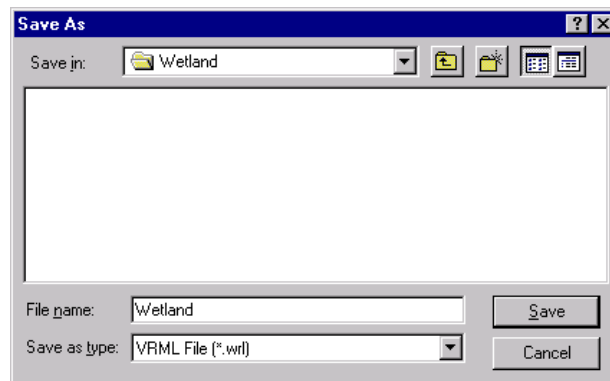



Figure 2.3: **Save As** dialog box for saving the visualization scene as a VRML file.

To save a scene:

1. Specify the filename to be used in the **File name** box.
2. Choose **VRML File (\*.vrl)** as the **Save as type**.
3. Click the  button. The *vrl* extension is automatically added to the given filename. In Figure 2.3 for example, the current scene will be saved as *Wetland.vrl* in the wetland folder.



VRML files can be viewed with Cosmo Player (<http://www.cai.com/cosmo/>) and it is a fast and easy way to put interactive visualization scenes on the World Wide Web.

- **P**rint... – Prints the scene.
- **P**rint **P**review – Displays a preview of how the scene is going to be printed on a page.
- **P**rint **S**etup... – Changes the print setup.
- **E**xit – Exits the program.

### 2.1.2 Edit

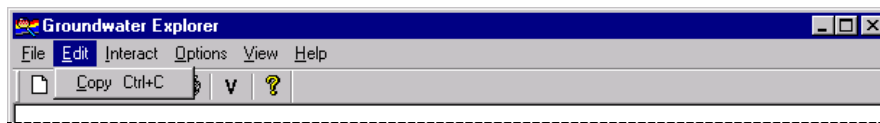


Figure 2.4: Edit menu.

Currently the Edit menu (Figure 2.4) contains only one item:

- **C**opy – Copies the current scene as an image to the clipboard. The image can be pasted into most word- or graphics-processing software environments by using Ctrl+V.

### 2.1.3 Interact

The Interact menu (Figure 2.5) groups items that have an influence on all visualization objects.

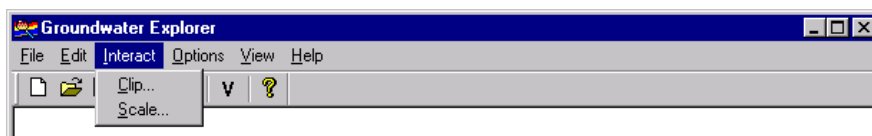


Figure 2.5: Interact menu.

The Interact menu consists of the following items:

- **C**lip... – Displays the **C**lip dialog box (Figure 2.6) for clipping visualization objects of the scene.

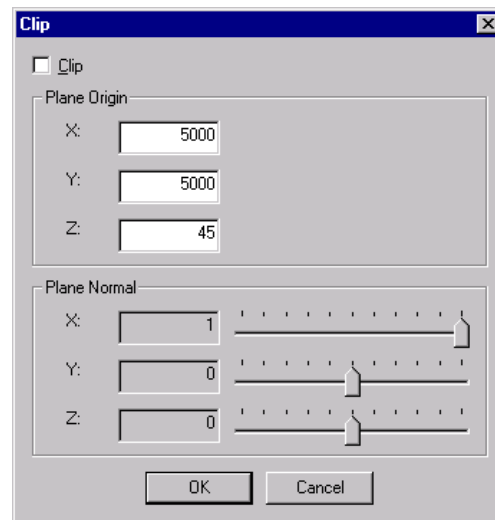


Figure 2.6: **Clip** dialog box.

The clipping is done when the **Clip** box is checked (☒) and **OK** is pressed. The **Plane Origin** is the origin of the plane used for clipping and the **Plane Normal** is a normal vector on the plane, which determines the direction of clipping. Valid values for **X**, **Y** and **Z** of the **Plane Normal** range from  $-1$  to  $1$ .

- **Scale...** – Displays the **Scale** dialog box (Figure 2.7).

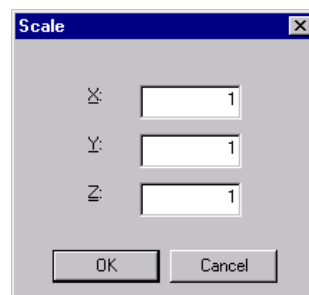


Figure 2.7: **Scale** dialog box.

Visualization objects can be scaled in the positive or negative axes' directions. Valid scaling values range from  $-100$  to  $100$  for all directions.

### 2.1.4 ***Options***

The **Options** menu (Figure 2.8) is used to make changes to the axes and visualization objects.

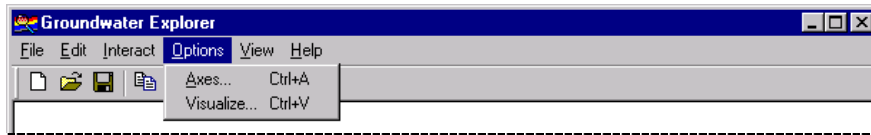


Figure 2.8: **Options** menu.

The **Options** menu consists of the following items.

- **Axes...** – Displays the **Axes** dialog box (Figure 2.9). The **Axes** dialog box is used to set the properties of the **Label Axes** and **Color Axes**. The **Label Axes** object displays axes with tick marks and corresponding coordinates. The displayed coordinate values are scaled by the factors set in the **Scale** dialog box (Figure 2.7). The **Color Axes** object displays axes, which are fixed at the bottom, left and lower corner of the model.

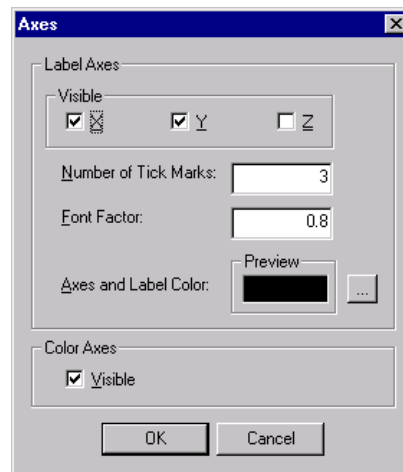


Figure 2.9: **Axes** dialog box.

Check (☒) the **Visible** check boxes to make the axes visible and clear (☐) them to make them invisible. In Figure 2.9, the **X** and **Y** axes of the **Label Axes** and the **Color Axes** are checked (☒) as visible and the **Z** axis of the **Label Axes** is invisible.

- **Visualize...** – Displays the **Visualize** dialog box (Figure 2.10), which is used to add or remove visualization objects, or to change properties of existing visualization objects.

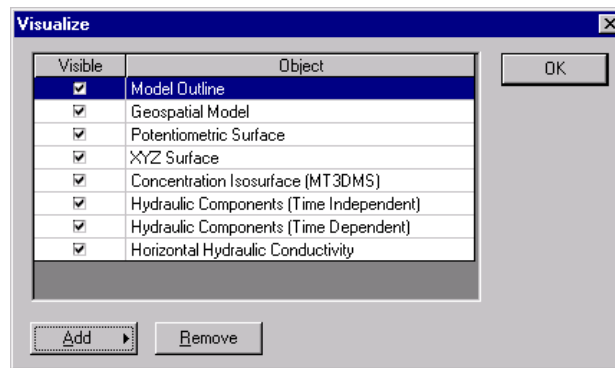
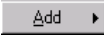





Figure 2.10: **Visualize** dialog box.

- To add a visualization object:
  1. Click the  button to display the visualization objects menu (Figure 2.11).
  2. Select an appropriate item from the visualization objects menu. A dialog box displaying a **General** page for the specific visualization object appears.
  3. Type in a description and specify other data for the visualization object (refer to Chapter 3 for details), then click .
- To remove a visualization object:
  1. Select a visualization object from the Visualize dialog box.
  2. Click the  button.
- To modify the properties of a visualization object:
  1. Double click a visualization object to open a dialog box containing its available properties.
  2. Modify the properties, then click . Refer to Chapter 3 for details.

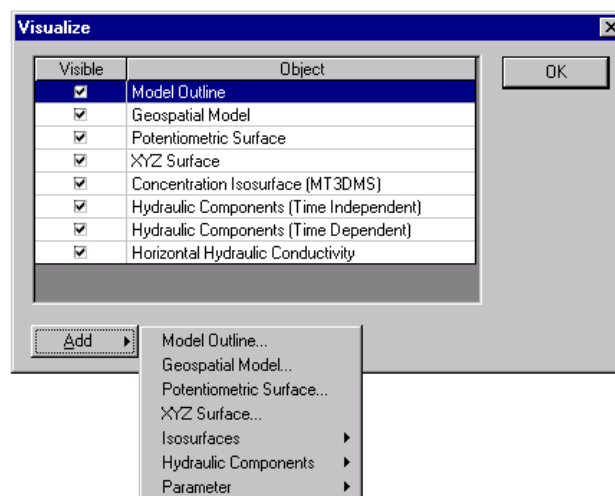


Figure 2.11: **Add** button items.

### 2.1.5 View

The **View** menu (Figure 2.12) groups items that switch the display of the **Toolbar** and **Status Bar** on or off.

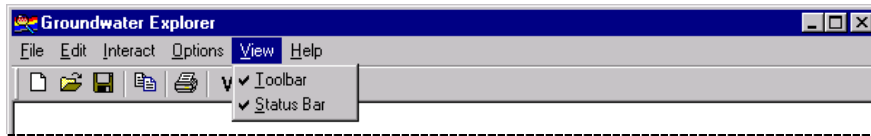


Figure 2.12: **View** menu.

The **View** menu consists of the following items:

- **Toolbar** – The toolbar is displayed if the item is checked.
- **Status Bar** – The status bar is displayed if the item is checked.

### 2.1.6 Help

The **Help** menu (Figure 2.13) item groups help related items.

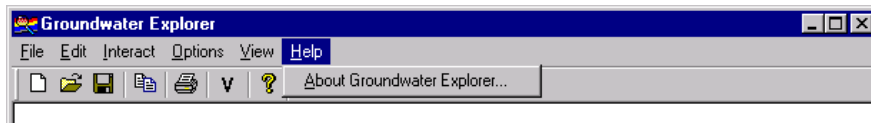


Figure 2.13: **Help** menu.

The **Help** menu consists of the following item:

- **About Groundwater Explorer...** – Displays GE's **A**bout dialog box.

## 2.2 **T**oolbar

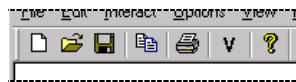









Figure 2.14: Toolbar.

The toolbar buttons are shortcuts to menus. The buttons and the corresponding menus are listed in Table 5.1.

Table 2.1: Toolbar buttons.

	<b>N</b> ew
	<b>O</b> pen...
	<b>S</b> ave
	<b>C</b> opy
	<b>P</b> rint...
	<b>V</b> isualize...
	<b>A</b> bout Groundwater Explorer...

## 2.3 Status Bar



Figure 2.15: Status bar.

The status bar displays processing and help information in the left corner. In Figure 2.15 the system is specified as being **Ready**. On the right side of the status bar, indicators show the status of the Caps Lock (**CAP**) and Num Lock (**NUM**) keys.

## 2.4 Control the Scene with the Mouse and Keyboard

Groundwater Explorer allows you to control the scene by using the mouse or keyboard in the following ways:

- Rotate the scene by holding down the left mouse button and moving the mouse.
- Zoom into the scene by holding down the right mouse button and moving the mouse pointer to the top half of the scene.
- Zoom out of the scene by holding down the right mouse button and moving the mouse pointer to the bottom half of the scene.
- The **w** key switches the display of all objects to the wireframe mode. The **s** key switches the display of all objects to the surface mode.

# *Chapter 3*

---

## **Visualization Objects**

### **3.1 Introduction**

The visualization objects that can be added to a scene in GE are: Model Outline, Geospatial Model, Potentiometric Surface, XYZ Surface, Isosurface, Hydraulic Components and Parameter.

The following sections describe the groundwater model data used for each visualization object, and show examples of the objects. The components of the dialog pages are also explained.

### **3.2 Model Outline**

The Model Outline visualization object is a wire frame bounding box of the model domain. The bounding box is defined by specifying the bounds of the model domain. Minimum and maximum x, y and z real-world, corner coordinates define the bounds. Figure 3.1 shows a Model Outline visualization object with its minimum and maximum corners for which real-world, bound coordinates are specified.

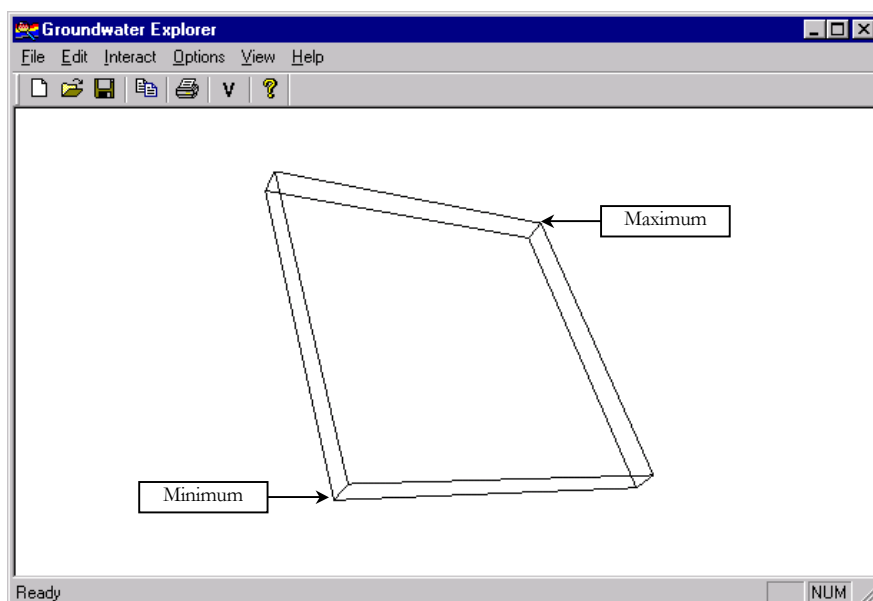


Figure 3.1: Minimum and maximum corners of Model Outline visualization object.

### 3.2.1 General

Figure 3.2 shows the **General** page of the **Model Outline** dialog box.

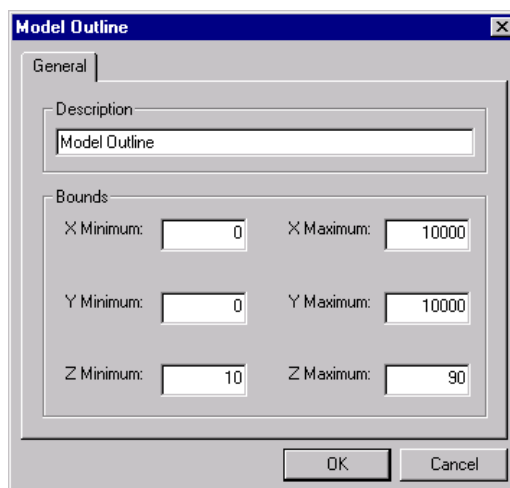


Figure 3.2: **General** page of the **Model Outline** dialog box.

**Description** is a text description by which the Model Outline visualization object is identified. **X Minimum**, **Y Minimum**, **Z Minimum**, **X Maximum**, **Y Maximum** and **Z Maximum** are the minimum and maximum x, y and z real-world, coordinate values of the minimum and maximum corners, which defines the bounding box.



### 3.3 Geospatial Model

The Geospatial Model visualization object is an outline polygonal mesh of the active model cells for each layer. Inactive groundwater model cells are not visualized. Different Geospatial Model visualization properties can be set for each layer. Figure 3.3 shows a Geospatial Model visualization object for a two-layer groundwater model. The outline polygonal mesh is built from polygonal quadrilaterals that are triangulated for decimation and smoothing.

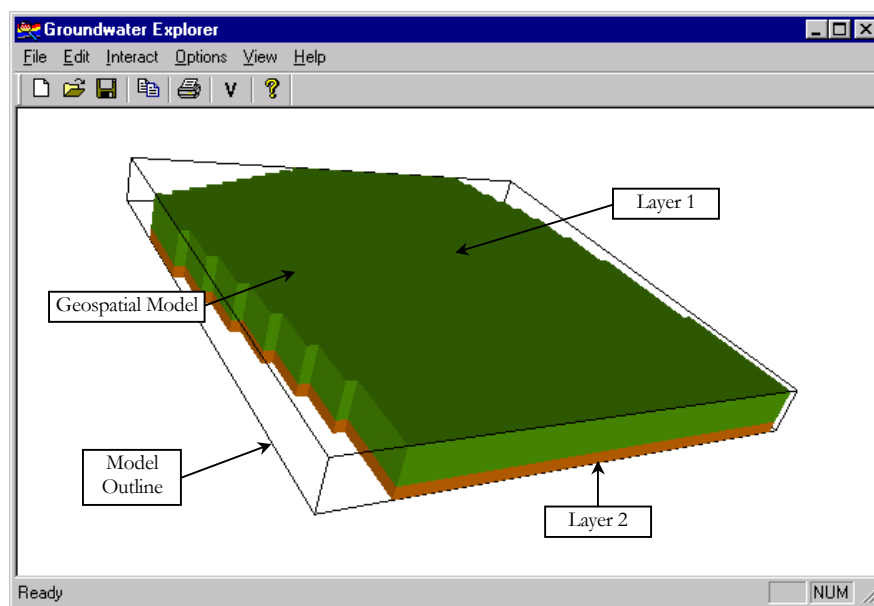


Figure 3.3: Geospatial Model visualization object.

#### 3.3.1 *General*

Figure 3.4 shows the **General** page of the **Geospatial Model** dialog box.

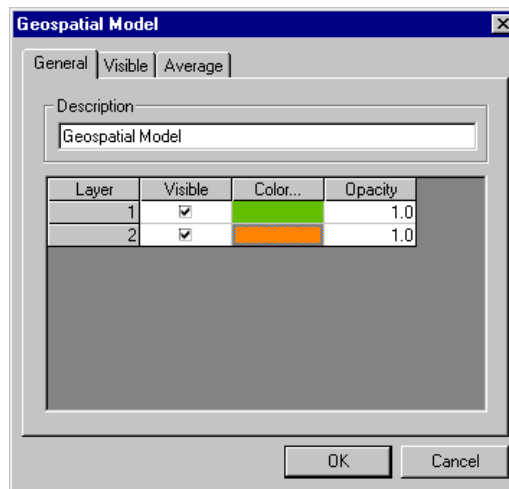


Figure 3.4: **General** tab of the **Geospatial Model** dialog box.

**Description** is a text description by which the Geospatial Model visualization object is identified. Check (☒) the **Visible** check boxes to make a layer's geospatial model visible, or clear (☐) them to make them invisible. For example in Figure 3.4 Layer 1 and Layer 2 are checked to be visible. Double click a cell of the **Color...** field to change the color of an individual layer or click the **Color...** header button to display the **Color Spectrum** dialog (Figure 3.5). On the Color Spectrum dialog, **Minimum Color** is the color used for the first layer, and **Maximum Color** is used for the last layer. Layers between the first and last layer are assigned color values using a linear scale.

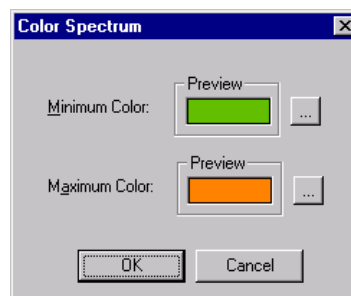



Figure 3.5: **Color Spectrum** dialog box.

Click the  button of the **Minimum Color** or **Maximum Color** to change their respective colors. The new color for **Minimum Color** or **Maximum Color** is selected from the **Color** dialog box (Figure 3.6).

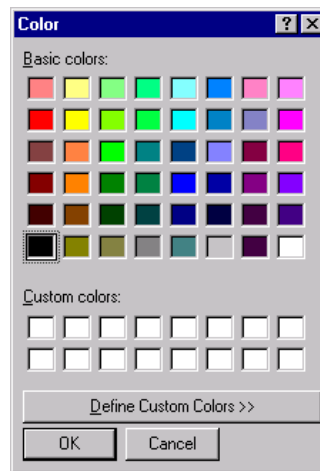


Figure 3.6: **Color** dialog box.

A color is selected by clicking on a color from the **Basic colors**, or a color is custom defined by clicking on the **Define Custom Colors**.

The **Opacity** fields in Figure 3.4 for Layer 1 and Layer 2 are set to 1. Opacity values range from 0 to 1. 0 sets a visualization object translucent and a value of 1 opaque. Figure 3.7 shows a Geospatial Model visualization object where the opacity values for both layers were set to 0.2. Setting the opacity values to 0.2 makes it possible to see the Potentiometric Surface inside the Geospatial Model.

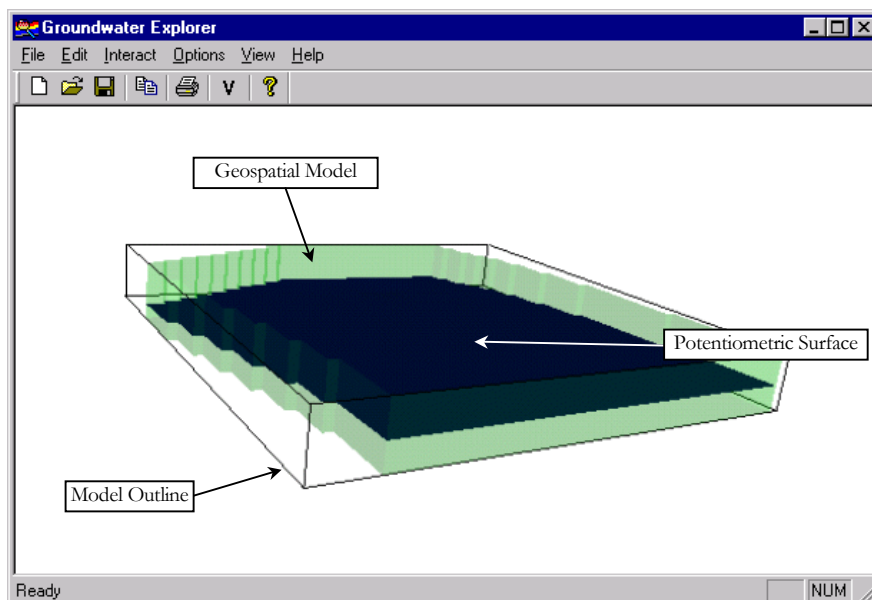


Figure 3.7: Potentiometric Surface visualization object can be seen inside the Geospatial Model visualization object of which the layer opacity values were set to 0.2.

### 3.3.2 Visible

Figure 3.8 shows the **Visible** page of the **Geospatial Model** dialog box.

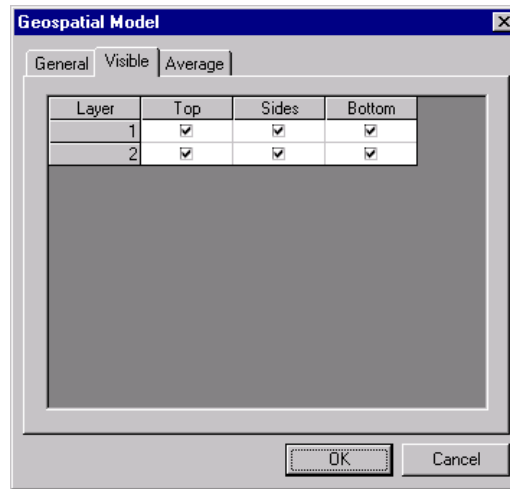


Figure 3.8: **Visible** page of the **Geospatial Model** dialog box.

Check (☒) or clear (☐) the check boxes in the **Top**, **Sides** and **Bottom** fields, to set the visibility for the top, sides and bottom of a Geospatial Model, for a specific layer. The **Top**, **Sides** and **Bottom** of a Geospatial Model for a layer are shown in Figure 3.9.

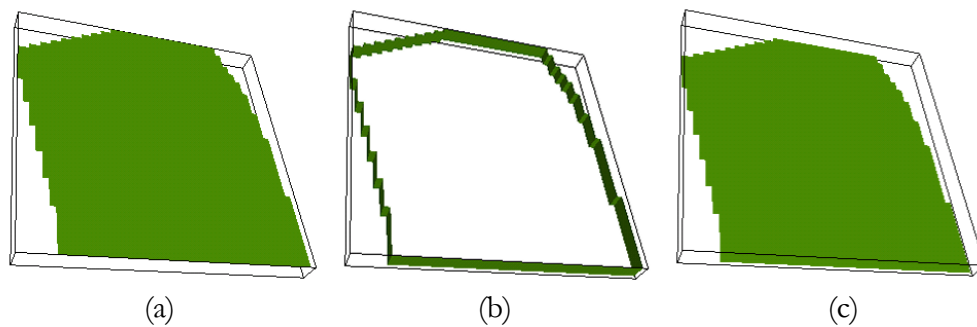


Figure 3.9: (a) **Top**, (b) **Sides** and (c) **Bottom** of a Geospatial Model visualization object for a layer.

### 3.3.3 Average

Figure 3.10 shows the **Average** page of the **Geospatial Model** dialog box.

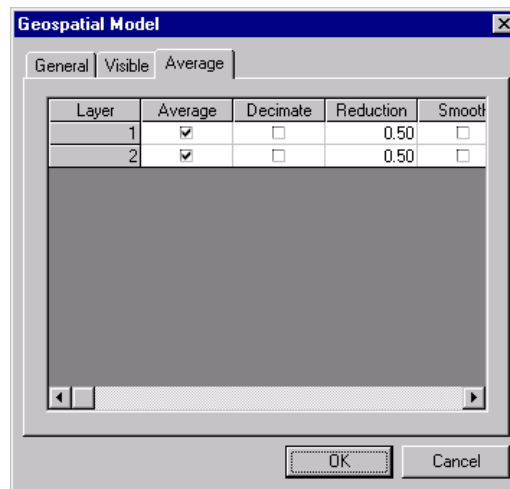


Figure 3.10: **Average** page of the **Geospatial Model** dialog box.

Check (☒) or clear (☐) the check boxes in the **Average** field to switch averaging on or off for a layer. Averaging has a visual smoothing effect (Figure 3.11).

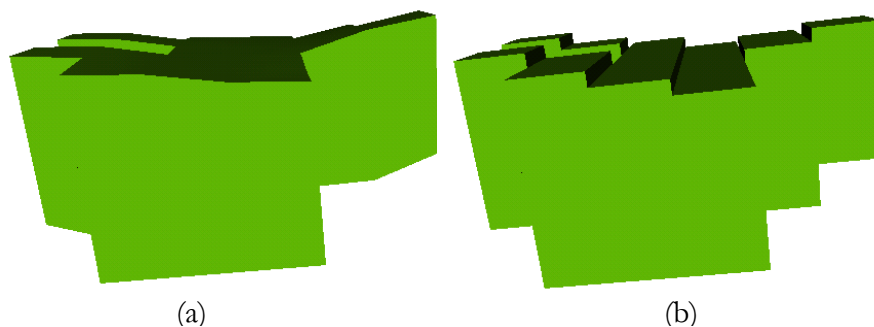


Figure 3.11: (a) Layer 1 of Geospatial Model visualization object with Average checked. (b) Layer 1 of Geospatial Model visualization object with Average not checked.

Check (☒) the **Decimate** check box for decimation to be done. Decimation results in fewer triangles used to create the polygonal mesh of the Geospatial Model, and has the advantage of faster rendering and interaction times.

Reduction specifies the desired reduction in the total number of triangles that make up the Geospatial Model, using decimation. The level of reduction may not be realized. Reduction values range from 0 and 1. Figure 3.12 shows the decimation results on the top part of a Geospatial Model visualization object. The model used to create the Geospatial Model is a one layer model with two columns and two rows. For (a) no decimation was done. For (b), (c) and (d) the target reduction was set to 0.3, 0.5 and 0.7, respectively.

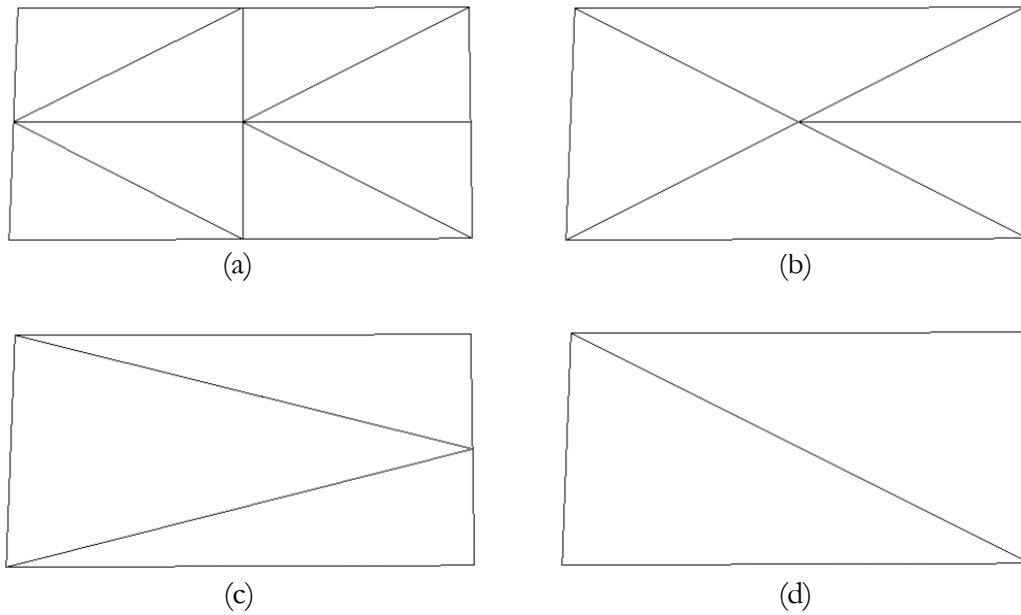


Figure 3.12: (a) No decimation. (b) Reduction = 0.3. (c) Reduction = 0.5. (d) Reduction = 0.7.

The fields associated with **Smooth**, are the number of **Iterations** and the **Relaxation** factor. Smoothing is used with **Average** and optionally with **Decimate**. Figure 3.13 (a) is the **Top** part of a Geospatial Model visualization object without smoothing. Figure 3.13 (b) shows the result of smoothing where **Iterations** was set to 10 and **Relaxation** to 0.2.

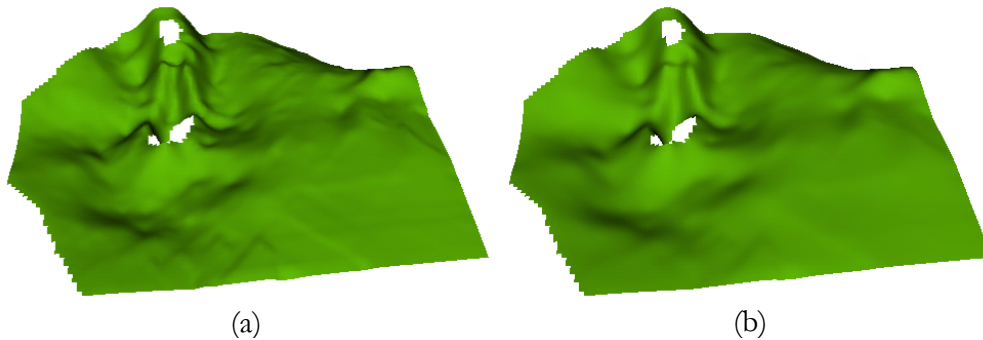


Figure 3.13: (a) No smoothing. (b) Smoothing with **Iterations** = 10 and **Relaxation** = 0.2.

**Iterations** sets the number of iterations for Laplacian smoothing, and **Relaxation** specifies the relaxation factor for Laplacian smoothing. As with all iterative methods, the stability of the process is sensitive to the **Relaxation** field. In general, small relaxation factors and large numbers of iterations are more stable than larger relaxation factors and a smaller number of iterations.

### 3.4 Potentiometric Surface

The Potentiometric Surface is a visualization object (Figure 3.14) which visualizes groundwater heads in the highest active cells for all layers, or the groundwater heads in active cells for a specified layer. The groundwater heads are also visualized for a specified stress period and time step.

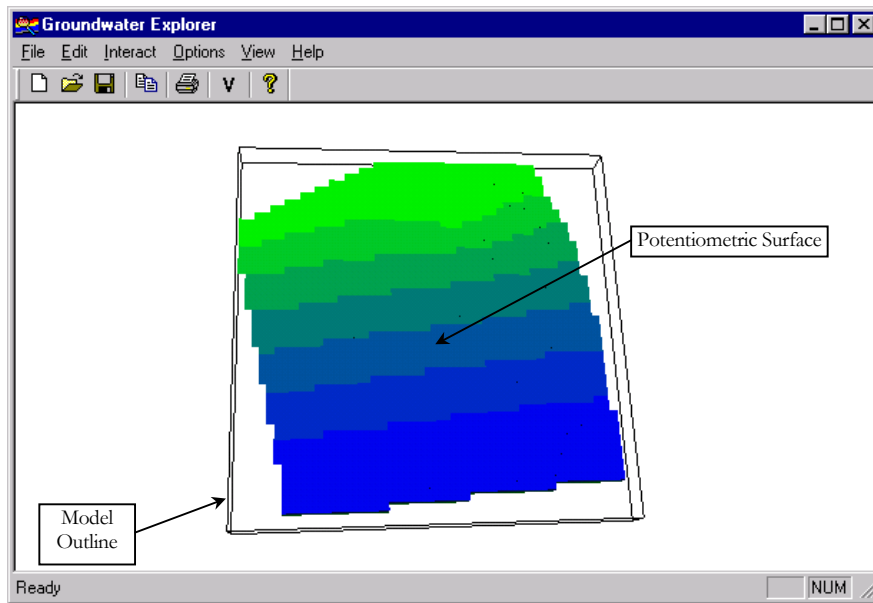


Figure 3.14: Potentiometric Surface visualization object.

#### 3.4.1 *General*

Figure 3.15 shows the **General** page of the Potentiometric Surface dialog box.

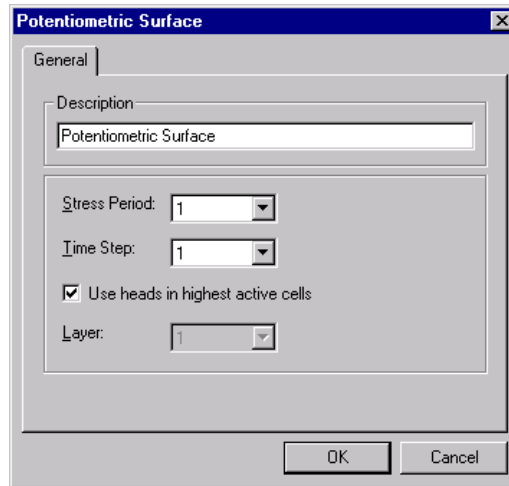


Figure 3.15: **General** page of the **Potentiometric Surface** dialog box.

**Description** is a text description by which the Potentiometric Surface visualization object is identified. **Stress Period** is a list of available stress periods from which one is selected. **Time Step** is the selected time step, from a list of available time steps for the selected **Stress Period**. If the groundwater heads in the highest active cells are to be visualized, check (☑) **Use heads in highest active cells**. If the heads for a specific layer are to be visualized, clear (☐) the **Use heads in highest active cells** and select the required layer from the **Layer** list.

### 3.4.2 Average

Figure 3.16 shows the **Average** page of the **Potentiometric Surface** dialog box.

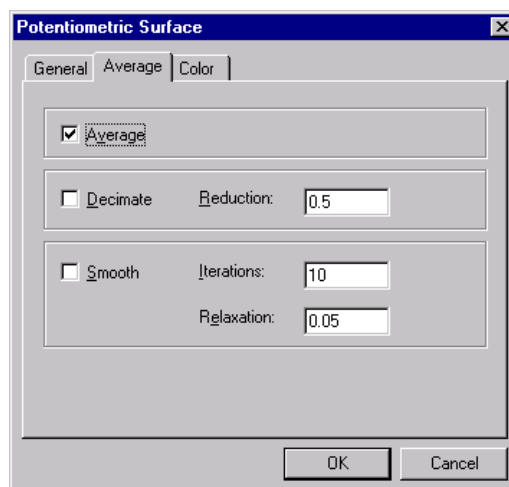


Figure 3.16: **Average** page of the **Potentiometric Surface** dialog box.



Section 3.3.3, “*Average*”, gives an explanation of the effect of the different components.

### 3.4.3 Color

Figure 3.17 shows the **Color** page of the Potentiometric Surface dialog box.

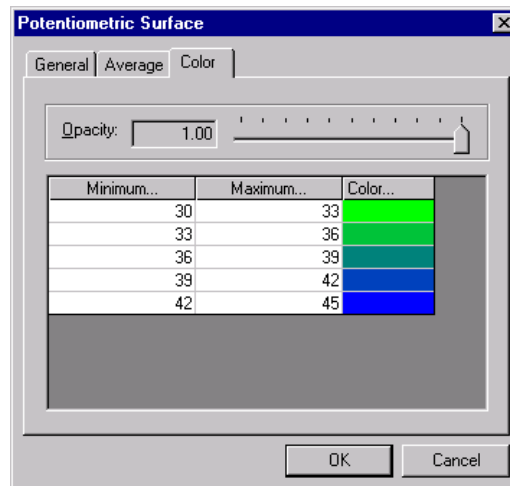


Figure 3.17: **Color** page of the **Potentiometric Surface** dialog box.

The **Opacity** value in Figure 3.17 for the Potentiometric Surface visualization object is set to 1. Opacity values range from 0 to 1. 0 sets a visualization object translucent and a value of 1, opaque.

Click the **Minimum...** or **Maximum...** header buttons to display the **Color Level** dialog box (Figure 3.18). The **Color Level** dialog box makes it possible to specify the minimum and maximum head values, and the number of intervals to associate a specific color with. In Figure 3.18 the **Minimum** head value that is to be displayed is 30 and the **Maximum** head value is 45, with the interval specified as 3. Specific colors will therefore be associated with each interval of 3 from 30 to 45. The result of using these values is shown in Figure 3.14.

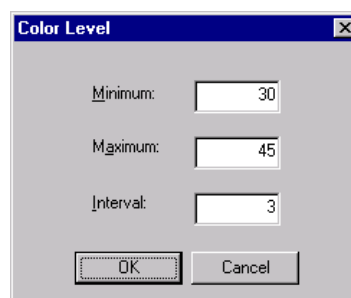


Figure 3.18: **Color Level** dialog box.

Double click a cell of the **Color...** field to change the color of an interval or click the **Color...** header button to display the **Color Spectrum** dialog (Figure 3.19). On the **Color Spectrum** dialog, **Minimum Color** is used for the first interval and **Maximum Color** is used for the last interval. Intervals between the first and last interval are assigned colors, using a linear scale.

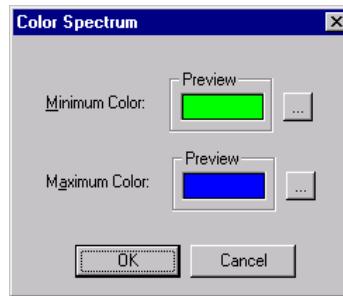



Figure 3.19: **Color Spectrum** dialog box.

Click the  button of the **Minimum Color** or **Maximum Color** to change their respective color. The new color is selected from the **Color** dialog box (Figure 3.20).

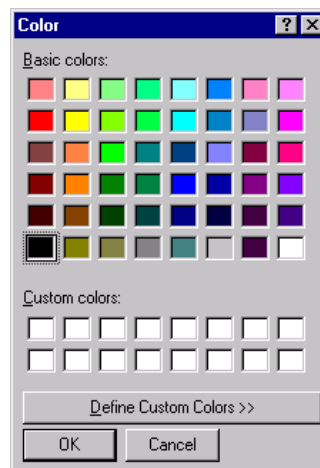


Figure 3.20: **Color** dialog box.

A color is selected by clicking on a color from the **Basic colors**, or a color is custom defined by clicking on the **Define Custom Colors**.

### 3.5 XYZ Surface

The XYZ Surface visualization object is created from x, y and z real-world coordinates (Figure 3.21).

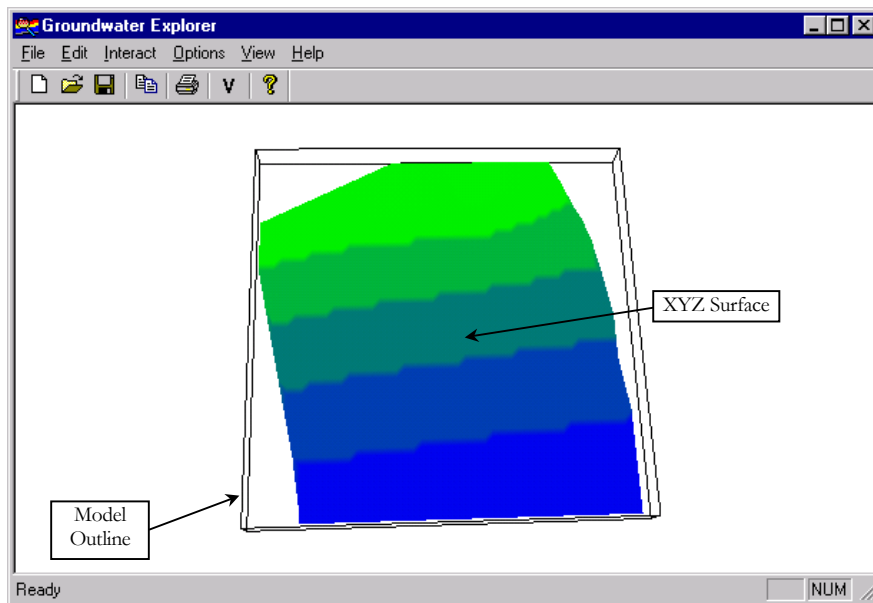


Figure 3.21: XYZ Surface visualization object.

### 3.5.1 General

Figure 3.22 shows the **General** page of the **XYZ Surface** dialog box.

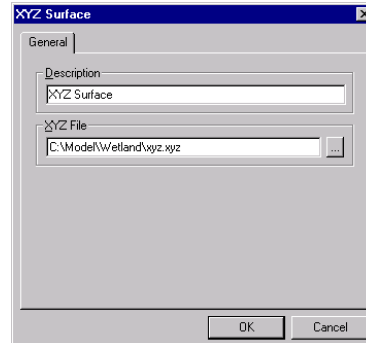



Figure 3.22: **General** page of the **XYZ Surface** dialog box.

**Description** is a text description by which the XYZ Surface visualization object is identified. The **XYZ File** is used to specify the XYZ folder and filename. The xyz file contains the x, y and z real-world coordinates from which a polygonal mesh is to be created. Type the folder and filename in the **XYZ File** text box or browse for it by clicking on the  button to display the **Open** dialog box, which is used to select the file that contains the x, y and z coordinates.

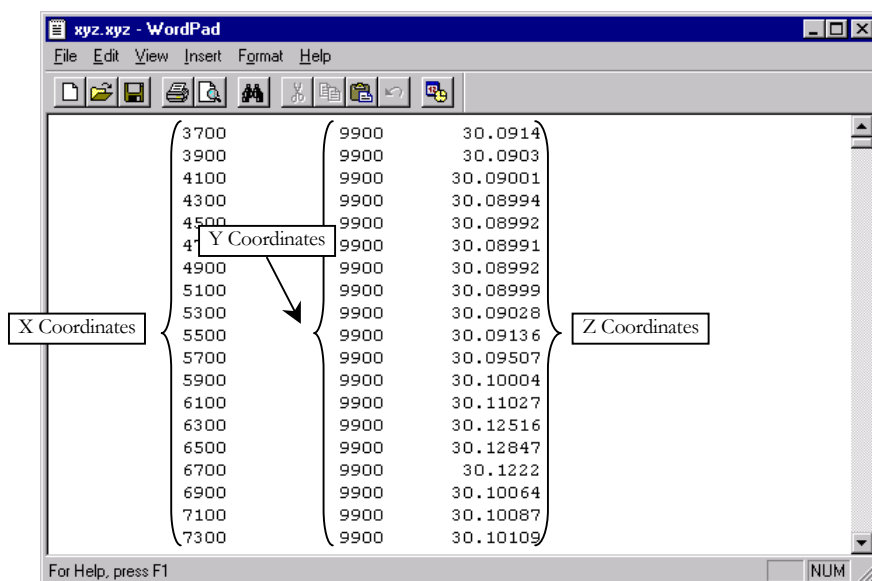


Figure 3.23: An example of an XYZ file edited in WordPad.

The x, y and z coordinates are inputted into a text file (Figure 3.23). The first column contains the x coordinates, the second the y coordinates and the third the z coordinates.

### 3.5.2 *Decimate and Smooth*

Figure 3.24 shows the **Decimate and Smooth** page of the **XYZ Surface** dialog box.

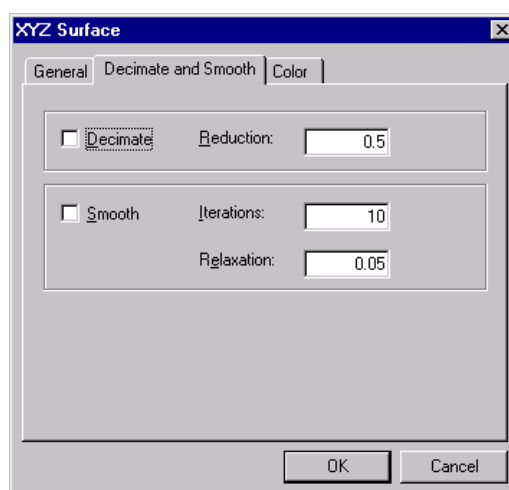


Figure 3.24: **Decimate and Smooth** page of the **XYZ Surface** dialog box.

Section 3.3.3, “*Average*”, gives an explanation on the effect of the **Decimate**, **Reduction**, **Smooth Iteration** and **Relaxation** components.

### 3.5.3 Color

Figure 3.25 shows the **Color** page of the **XYZ Surface** dialog box.

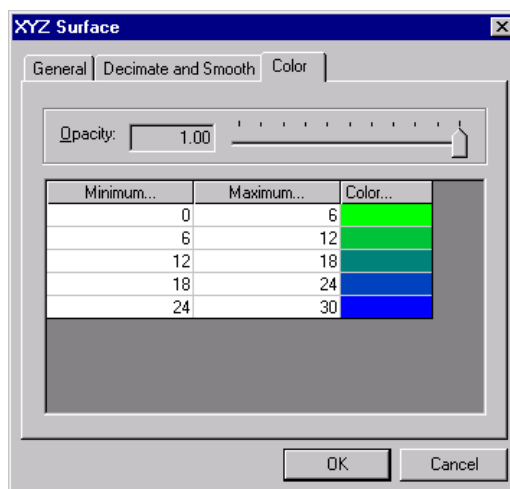


Figure 3.25: **Color** tab of the XYZ Surface dialog box.

Section 3.4.3, “*Color*”, gives an explanation on the effect of the **Opacity**, **Minimum...**, **Maximum...** and **Color...** fields.

## 3.6 Isosurface

Isosurfaces are created from model cell concentrations calculated by MT3DMS. Figure 3.26 shows two isosurfaces for two different concentrations. Isosurfaces can also be created for PHT3D and RT3D concentrations. The dialog components for these two models are exactly the same as for MT3DMS.

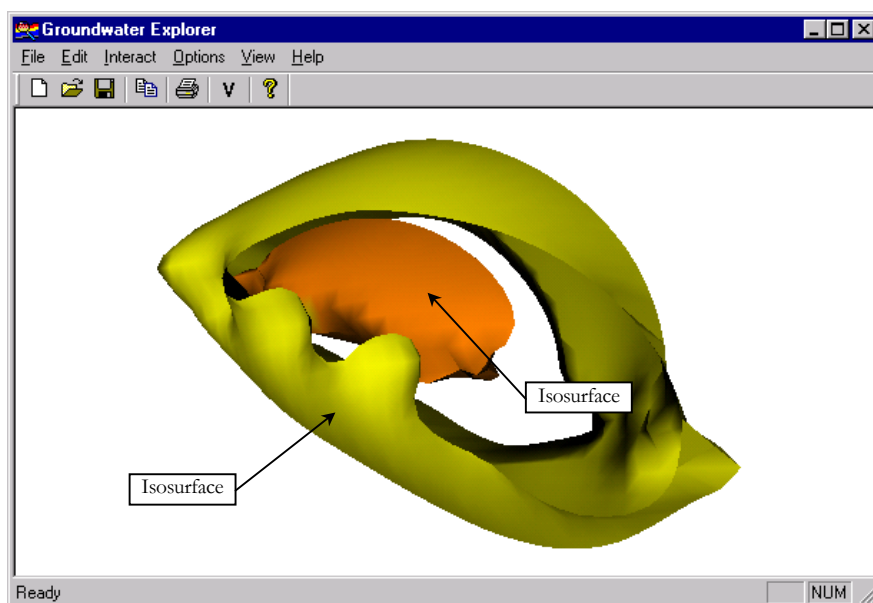


Figure 3.26: Concentration Isosurface (MT3DMS) visualization object.

### 3.6.1 General

Figure 3.27 shows the **General** page of the **Concentration Isosurface (MT3DMS)** dialog box.

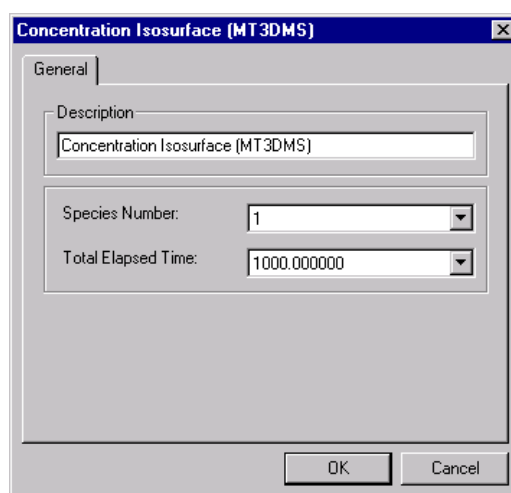


Figure 3.27: **General** page of the **Concentration Isosurface (MT3DMS)** dialog box.

**Description** is a text description by which the Concentration Isosurface (MT3DMS) visualization object is identified. **Species Number** is a list of available species from which

one is selected. **Total Elapsed Time** is the selected total elapsed time from a list of available total elapsed times for the selected species number.

### 3.6.2 *Contour and Color*

Figure 3.28 shows the **Contour and Color** page of the **Concentration Isosurface (MT3DMS)** dialog box.

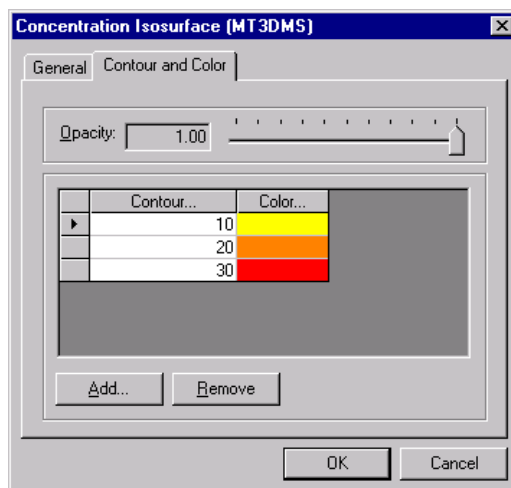


Figure 3.28: **Contour and Color** page of the **Concentration Isosurface (MT3DMS)** dialog box.

Isosurfaces are created for the specified contour values in the **Contour...** field and are colored with the specified colors in the **Color...** field. Double click a **Contour...** cell to change a contour value. The row mark with ► is removed when clicking on the **Remove** button. A row is added by clicking on the **Add...** button. Clicking on the **Contour...** header button displays the **Color Level** dialog box (Figure 3.29) which also changes the contour values. The **Color Level** dialog box is used to set the **Minimum** and **Maximum** contour values, and the interval for contour spacing.

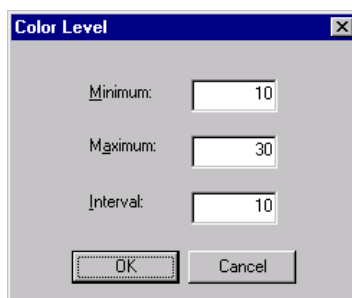


Figure 3.29: **Color Level** dialog box.

The isosurfaces color are changed in the same way as the colors for the Potentiometric Surface visualization object as explained in section 3.4.3, “Color”.

## 3.7 Hydraulic Components

The Hydraulic Components visualization object colors groundwater model cells which contain hydraulic component data. In Figure 3.30 the Hydraulic Components, visualization object consists of Discharge Wells and Recharge Wells.

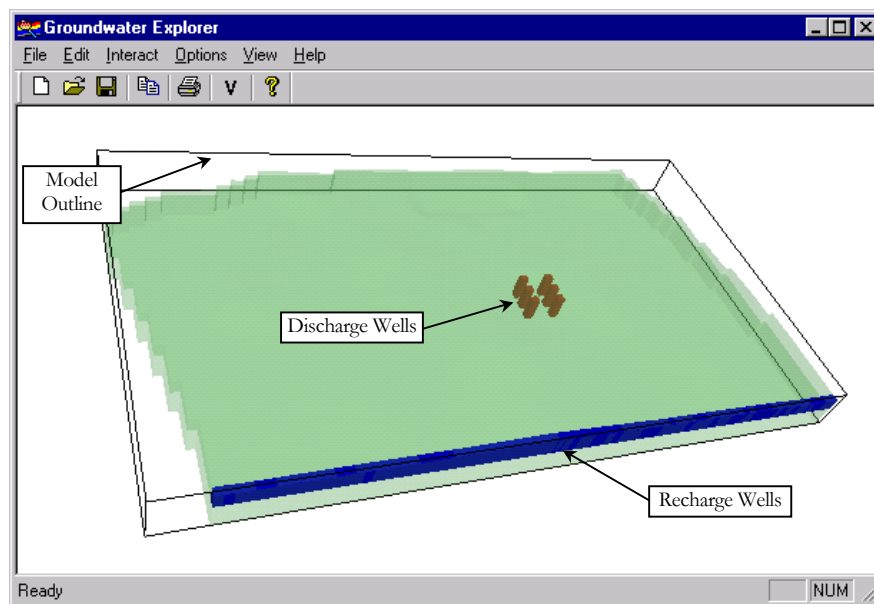


Figure 3.30: Hydraulic Components visualization object.

Hydraulic components are divided into time independent and time dependent components, and are listed below:

Time independent hydraulic components that can be visualized are:

- Fixed Concentration
- Fixed Heads
- Horizontal Flow Barriers
- Reservoir



Time dependent hydraulic components that can be visualized are:

- Discharge well
- Drain
- General Head Boundary
- Recharge well
- River
- Time Variant Specified Concentration
- Time Variant Specified Head

### 3.7.1 *General*

Figure 3.31 shows the **General** page of the **Hydraulic Components (Time Independent)** dialog box and Figure 3.32 shows the **General** page of the **Hydraulic Components (Time Dependent)** dialog box.

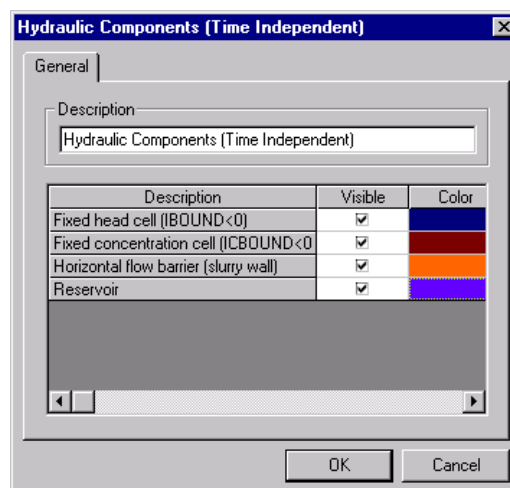


Figure 3.31: **General** page of the **Hydraulic Components (Time Independent)** dialog box.

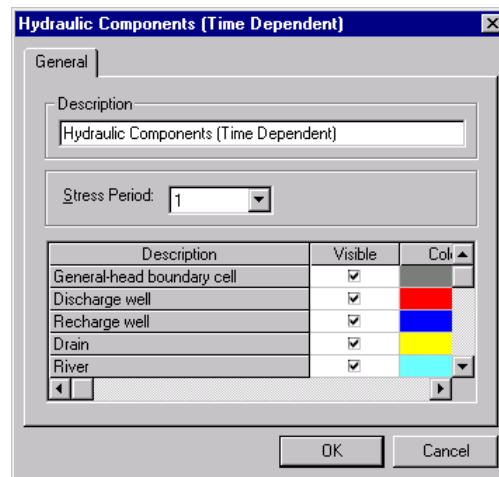


Figure 3.32: **General** page of the **Hydraulic Components (Time Dependent)** dialog box.

**Description** is a text description by which the Hydraulic Components visualization object is identified. Check (☒) the **Visible** check boxes to make a hydraulic component visible, or clear (☐) them to make them invisible. The **Color** field is used for specifying the color for the different hydraulic components. To change the color of a Hydraulic Component, double click the **Color** cell of the component where the color change is needed. Section 3.4.3, “*Color*”, explains how to use the **Color** dialog box.

The **General** page of the time dependent Hydraulic Components differs from the **General** page of time independent Hydraulics Components, because of the added **Stress Period** list. **Stress Period** is used to visualize the time dependent Hydraulic Components for a specific stress period, by selecting the desired stress period from the **Stress Period** list.

### 3.7.2 *Average*

Figure 3.33 shows the **Average** page of the **Hydraulic Components** dialog box.

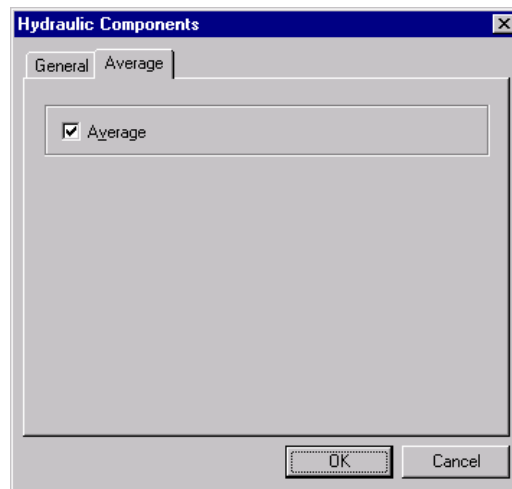


Figure 3.33: **Average** page of the **Hydraulic Components** dialog box.

Section 3.3.3, "*Average*", gives an explanation on the effect of averaging.

## 3.8 Parameter

The Parameter visualization object colors groundwater model cells that contain parameter data.

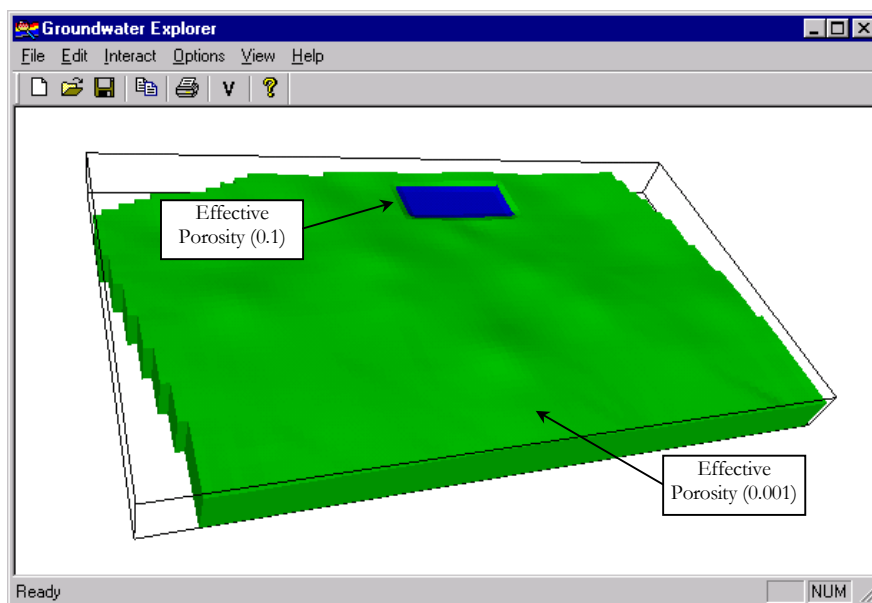


Figure 3.34: Parameter visualization object.

The following parameters can be visualized if groundwater model cells were assigned values:

- Horizontal Hydraulic Conductivity
- Vertical Hydraulic Conductivity
- Specific Storage
- Transmissivity
- Vertical Leakance
- Storage Coefficient
- Effective Porosity
- Specific Yield

### 3.8.1 *General*

Figure 3.35 shows the **General** page of the **Parameter** dialog box.

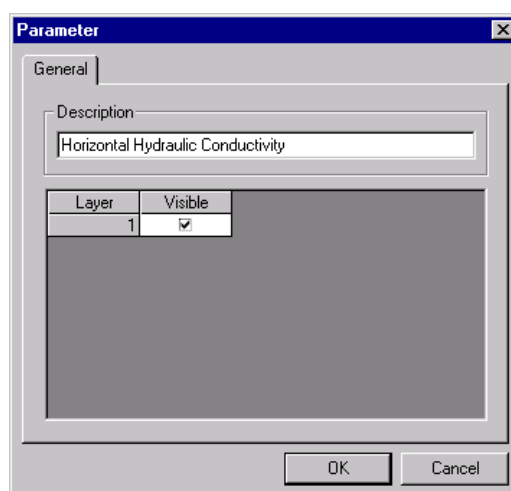


Figure 3.35: **General** page of the **Parameter** dialog box.

**Description** is a text description by which the Parameter visualization object is identified. Check (☒) the **Visible** check boxes to make a parameter visible for a layer, or clear (☐) them to make them invisible.

### 3.8.2 *Average*

Figure 3.36 shows the **Average** page of the **Parameter** dialog box.

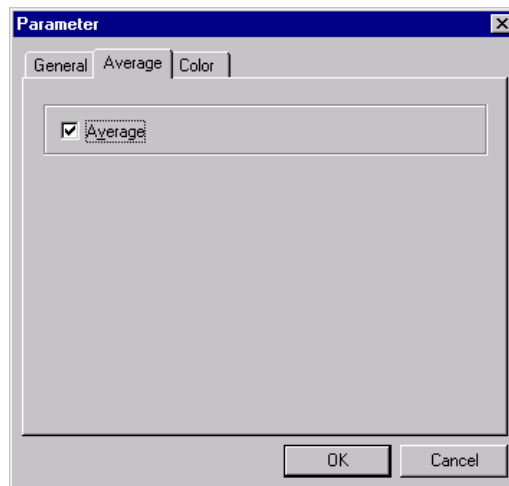


Figure 3.36: **Average** page of the **Parameter** dialog box.

Section 3.3.3, “*Average*”, gives an explanation on the effect of averaging.

### 3.8.3 *Color*

Figure 3.37 shows the **Color** page of the **Parameter** dialog box.

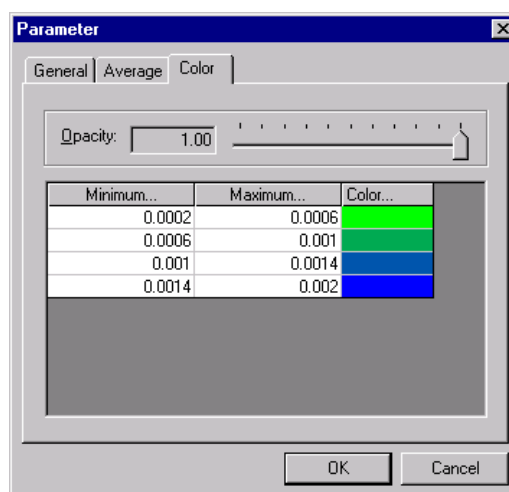


Figure 3.37: **Color** page of the **Parameter** dialog box.

Section 3.4.3, “*Color*”, gives an explanation on **Opacity**, **Minimum...**, **Maximum...** and **Color...**



# *Chapter 4*

---

## **Tutorial**

### **4.1 Introduction**

This chapter gives step-by-step instructions on how to visualize existing PMWIN models. The PMWIN models used for this visualization tutorial were course material for a short course titled “Applied Groundwater Modeling using PMWIN”, presented by Prof. Wen-Hsing Chiang at the University of the Free State, South Africa from the 27<sup>th</sup> to the 30<sup>th</sup> of August 1999.

The PMWIN models were created for a proposed well field near a wetland. The well field will supply drinking water and water for industrial use to a nearby community. The proposed pumping rate of the well field is 3600 m<sup>3</sup>/d, and is evenly distributed to six model cells (Figure 4.1).

### **4.2 Conceptualize the Wetland Model**

The first course task was to conceptualize the natural aquifer condition and to construct a flow model. The Wetland1 model resulted from this task. The following sections are instructions on how to use GE to help conceptualizing the natural condition of the wetland aquifer.

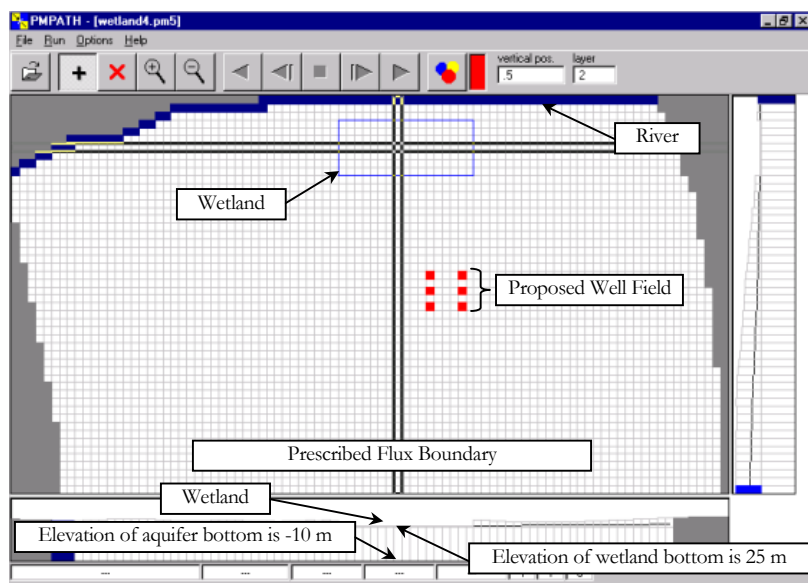


Figure 4.1: Wetland model in PMPATH.

#### 4.2.1 Step 1: Start GE

Start GE from the **Programs** menu list.

#### 4.2.2 Step 2: Open the Wetland Model

To start visualizing the PMWIN model, Wetland1 must be opened. From the **File** menu of GE, click the **Open...** item.

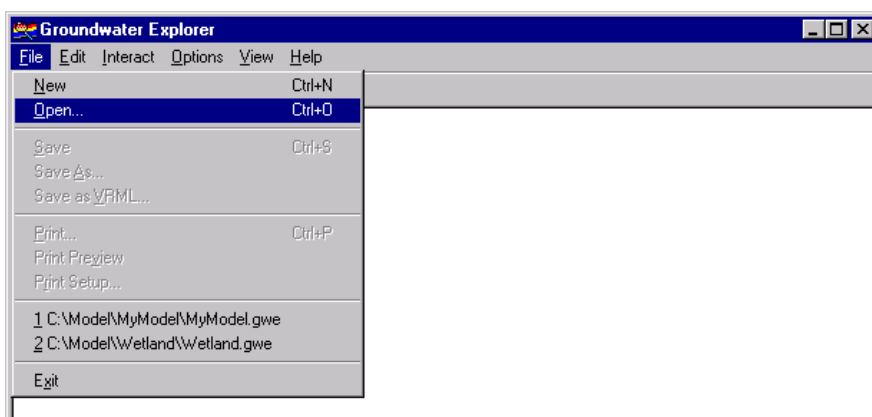


Figure 4.2: **Open...** item for opening an existing model.

After the **Open...** (Figure 4.2) item is clicked, an **Open** dialog box is displayed, which makes the selection of an existing PMWIN model possible. Browse for the



...\Models\Wetland1\ file folder on the accompanying CD and select the *wetland1.pm5* file (Figure 4.3).

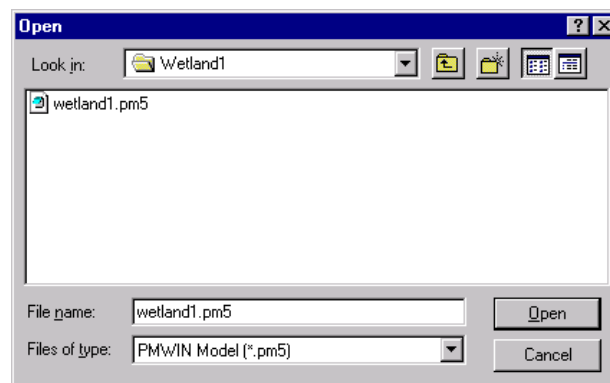
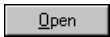


Figure 4.3: **Open** dialog box.

Click the  button to open the Wetland1 model for visualization. The result of opening the model is Label Axes, Color Axes and a Model Outline visualization object (Figure 4.4).

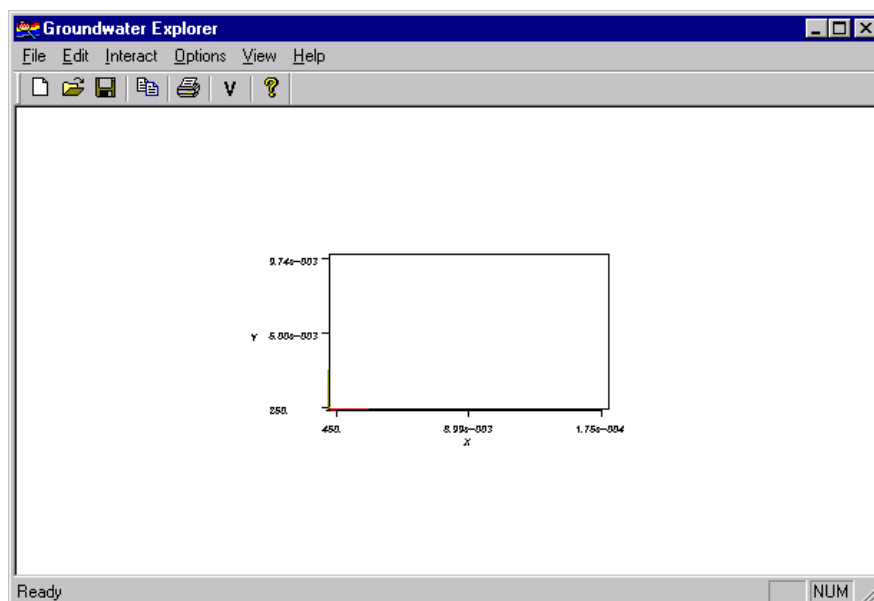


Figure 4.4: Label Axes, Color Axes and Model Outline visualization object.

### 4.2.3 Step 3: Add Geospatial Model

The Geospatial Model visualization object helps to conceptualize the Wetland model. Click **Visualize...** (Figure 4.5) to display the **Visualize** dialog box (Figure 4.6) from which the

Geospatial Model visualization object can be added. Note that GE has automatically added the Model Outline visualization object.

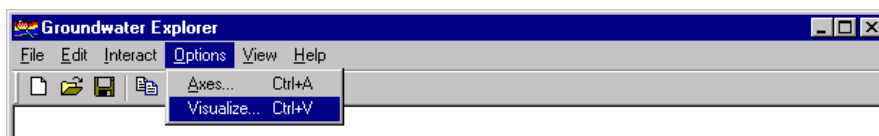


Figure 4.5: **Visualize...** item.

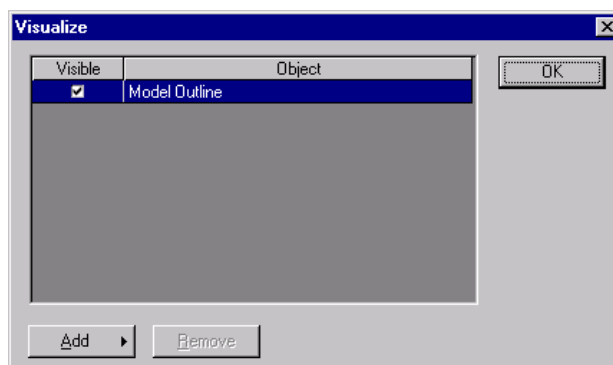



Figure 4.6: **Visualize** dialog box.

Click the  button to display the **Add** button items (Figure 4.7).

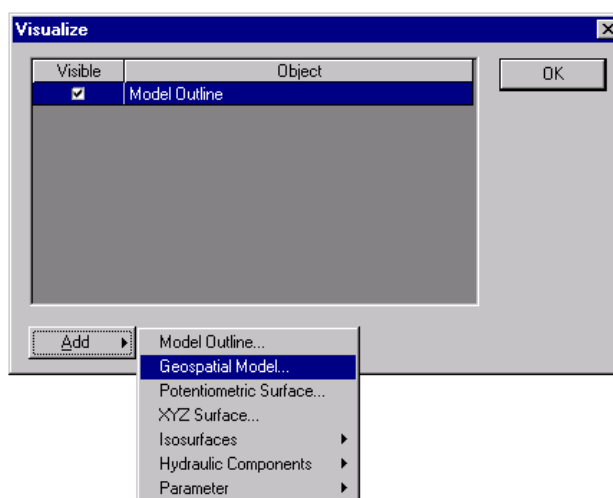


Figure 4.7: **Add** button items.

A new Geospatial Model visualization object is added by clicking on the **Geospatial Model...** item, which displays the **Geospatial Model** dialog box with the **General** page (Figure 4.8).

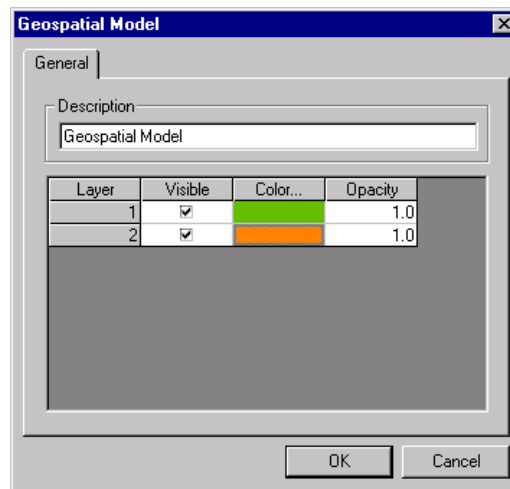



Figure 4.8: **General** page of the **Geospatial Model** dialog box.

Keep the default values for the text box and all the fields. Click the  button.

The Geospatial Model is added to the list of visualization objects in the **Visualize** dialog box (Figure 4.9).

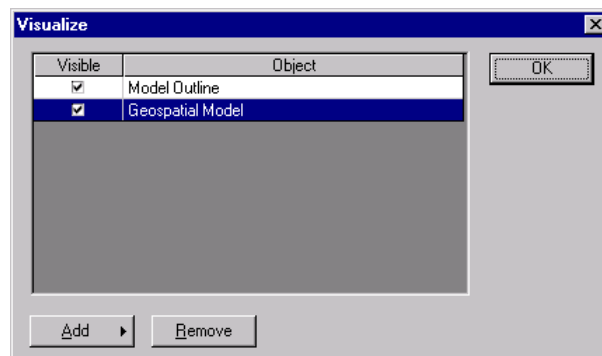
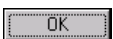


Figure 4.9: **Visualize** dialog box with Geospatial Model added.

Click the  button to see the result, as in Figure 4.10.

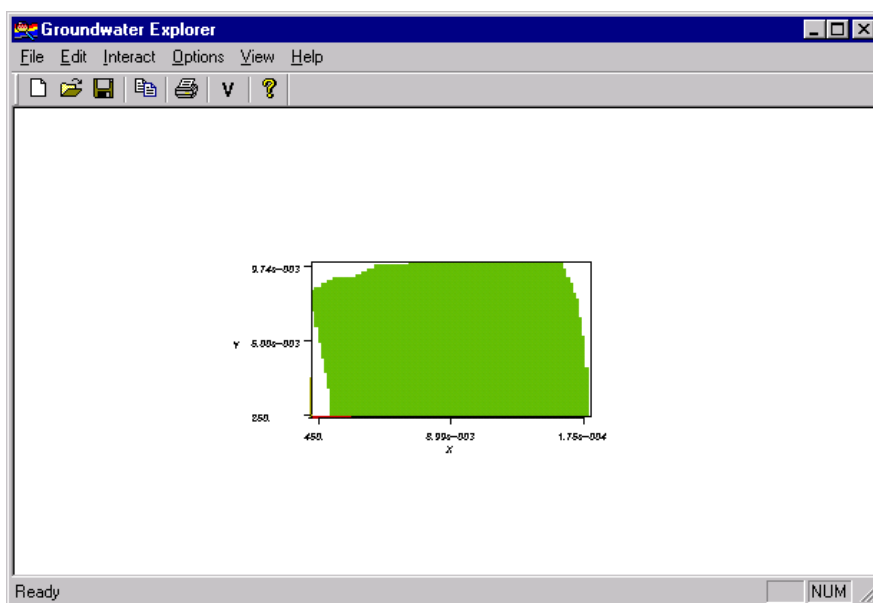


Figure 4.10: Label Axes, Color Axes, Model Outline and Geospatial Model visualization objects.

#### 4.2.4 Step 4: Scale

The visualization objects of a scene can be rotated and zoomed using the mouse. See section 2.4, “Control the Scene with the Mouse and Keyboard”, in Chapter 2, for an explanation on how to use the mouse to rotate and zoom.

The Geospatial Model is flat in the z coordinate direction, in comparison with the x and y coordinate directions. A good idea is to scale the z coordinate direction to make changes in elevation more obvious.

To scale all the objects in a visualization scene, click the **Scale...** item (Figure 4.11) to display the **Scale** dialog box (Figure 4.12).

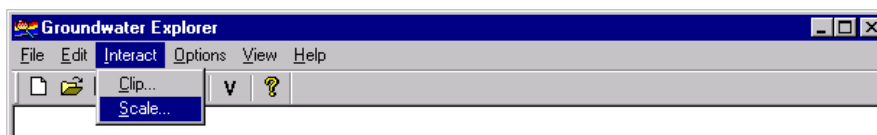


Figure 4.11: **Scale...** item.

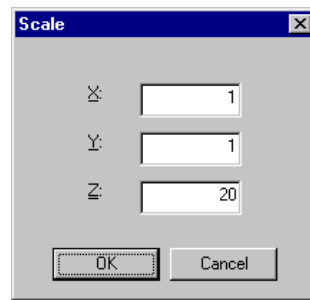


Figure 4.12: **Scale** dialog box, **Z** = 20.

Change the 1 in the **Z** text box to 20 and click the **OK** button.

Figure 4.13 shows the scaled, rotated and zoomed Geospatial Model. As a result of scaling, the wetland and two model layers are seen more clearly.

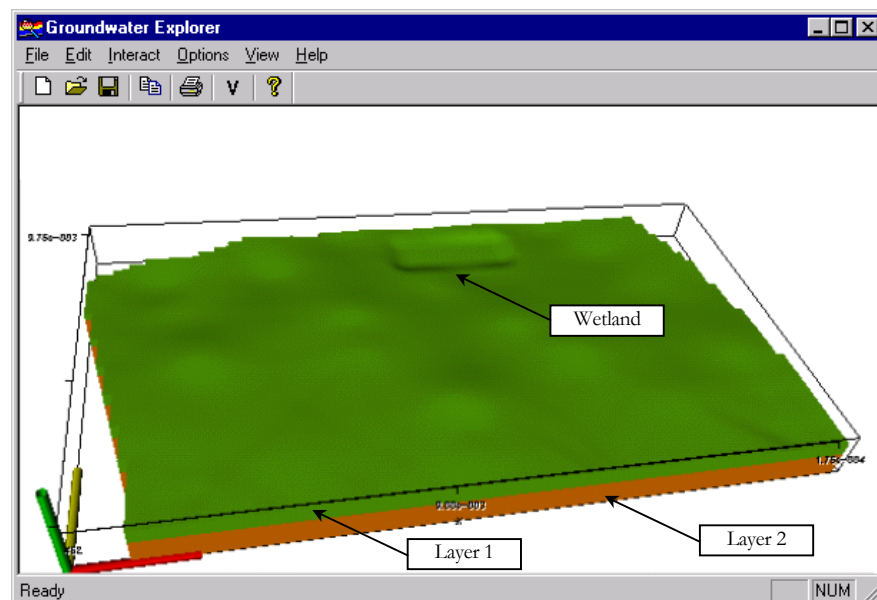


Figure 4.13: Scaled Geospatial Model visualization object.

#### 4.2.5 Step 5: Add Potentiometric Surface

The goal of the first Wetland model was also to create a flow model. GE can be used to visualize the groundwater heads which resulted from the flow model. The way to go about visualizing the groundwater heads is describe in the following sections.

Click **Visualize...** (Figure 4.14) to display the **Visualize** dialog box (Figure 4.15) from which the Potentiometric Surface visualization object can be added. The list of

visualization objects already contains the Model Outline and Geospatial Model to which the Potentiometric Surface visualization object is to be added.

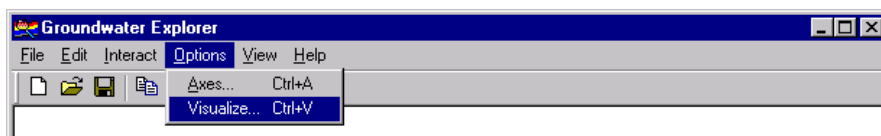


Figure 4.14: **Visualize...** item.

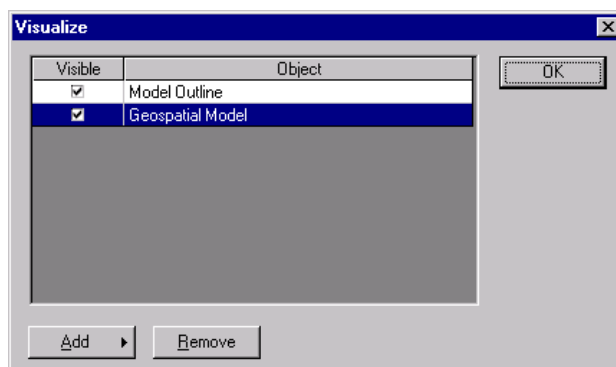



Figure 4.15: **Visualize** dialog box.

Click the  button to display the **Add** button items (Figure 4.16).

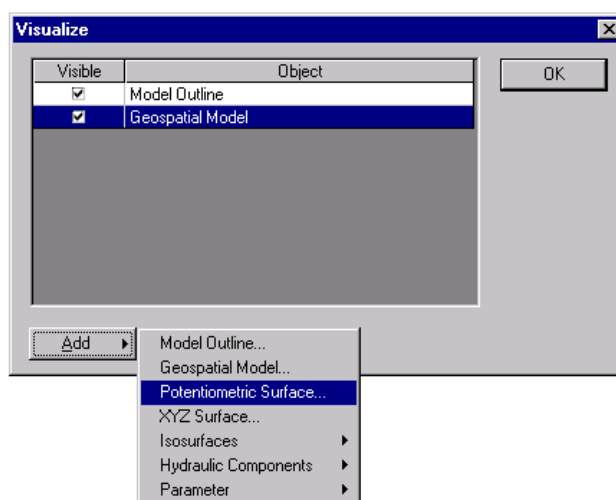


Figure 4.16: **Add** button items.

A new Potentiometric Surface visualization object is added by clicking on the **Potentiometric Surface...** item, which displays the **Potentiometric Surface** dialog box's **General** page (Figure 4.17).

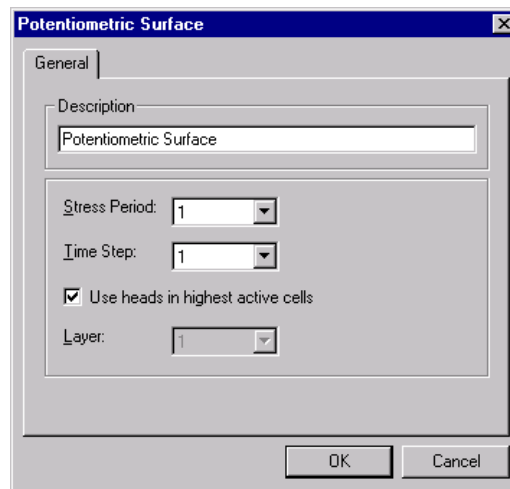



Figure 4.17: **General** page of the **Potentiometric Surface** dialog box.

Keep the defaults for all boxes and click the  button.

The Potentiometric Surface is added to the list of visualization objects in the **Visualize** dialog box (Figure 4.18).

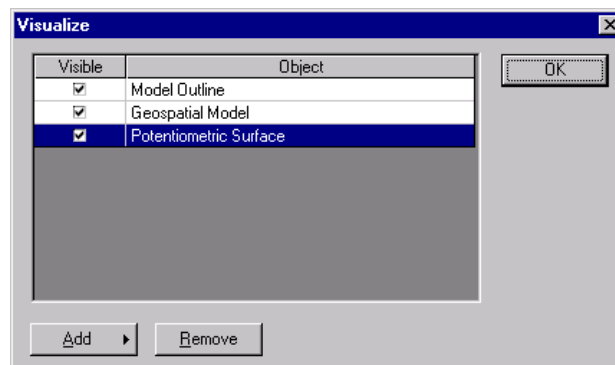
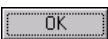


Figure 4.18: **Visualize** dialog box with Potentiometric Surface added.

Click the  button to see the result as in Figure 4.19.

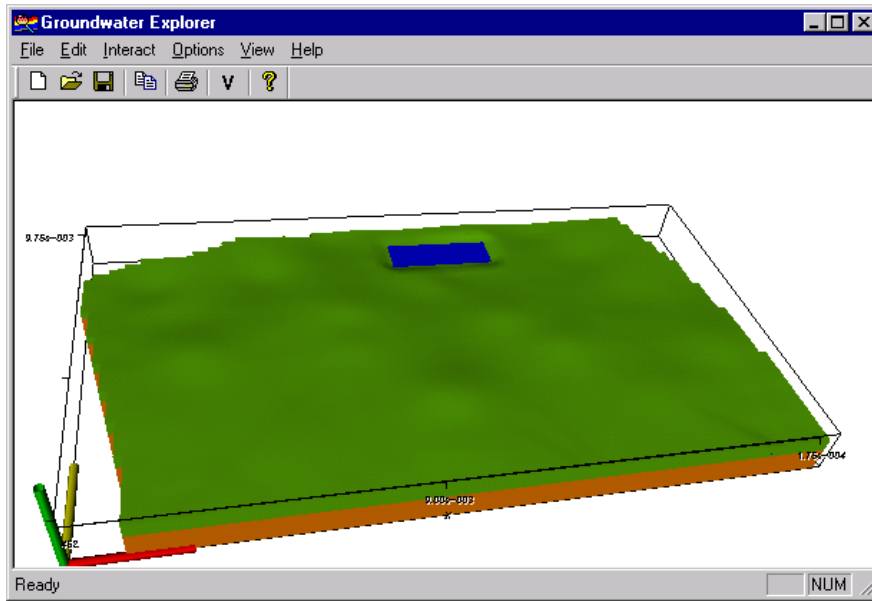


Figure 4.19: Label Axes, Color Axes, Model Outline, Geospatial Model and Potentiometric Surface visualization objects.

The nice thing about the Potentiometric Surface visualization object is that, as can be seen from Figure 4.19, the wetland is filled with water under natural conditions. Compare Figure 4.19 to Figure 4.13, where no Potentiometric Surface was added.

Another course task with the Wetland model was to see if the water level would drop lower than 25.8 m in the wetland if the recharge rate and inflow from the southern prescribed flux boundary were reduced to 50% of the long-term average values, and if a higher pumping rate of 6000 m<sup>3</sup>/d was required to cover an increased demand. The visualization of the Potentiometric Surface visualization objects in such a case shows the wetland dry.

### 4.3 Contaminated Wetland

Because of an accident, the aquifer upstream of the well field is highly contaminated with Benzene. The solute concentration at the PMWIN model cells [59, 39, 1], [59, 40, 1], [59, 39, 1] and [59, 39, 1] is (and remains) 10000 µg/l. According to the USEPA, the maximum contaminant level (MCL) of Benzene for drinking water is 5 µg/l. If the predicted concentration at the pumping well is higher than the MCL, something must be done to protect the wells.



### 4.3.1 Step 1: Start GE

Start GE from the **Programs** menu list.

### 4.3.2 Step 2: Open the Wetland Model

Follow the same steps as in section 4.2.2, “*Step 2: Open the Wetland Model*”, but open the model contained in the ...\\models\\wetland5\\ file folder on the accompanying CD, and select the *wetland5.pm5* file to open.

### 4.3.3 Step 3: Add Time Dependent Hydraulic Component

In order to know if any action is to be taken, we want to see if the maximum allowed concentration for Benzene reaches the well field. This tutorial step gives instructions on how to add the discharge wells as a visualization object to the scene.

Click **Visualize...** (Figure 4.20) to display the **Visualize** dialog box (Figure 4.21) from which the **Hydraulic Components, Time Dependent** visualization object can be added.

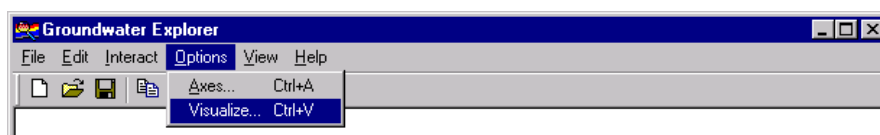


Figure 4.20: **Visualize...** item.

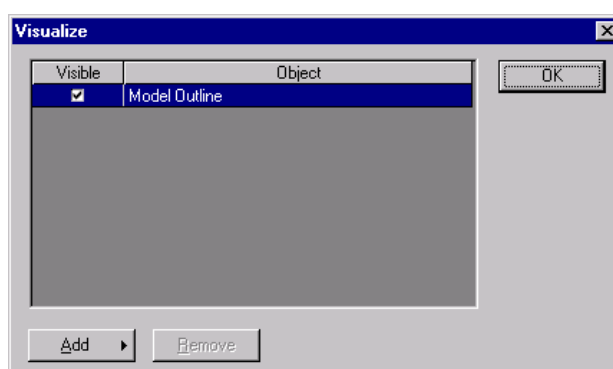



Figure 4.21: **Visualize** dialog box.

Click the  button to display the **Add** button items (Figure 4.22).

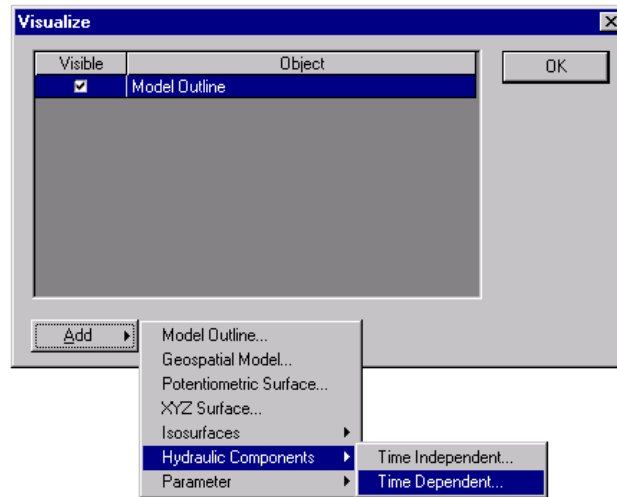


Figure 4.22: **Add** button items.

A new **Hydraulic Components, Time Dependent** visualization object is added by clicking on the **Time Dependent** item, which displays the **Hydraulic Component (Time Dependent)** dialog box with the **General** page (Figure 4.23). Keep the default values for all the boxes and fields and click the **OK** button.

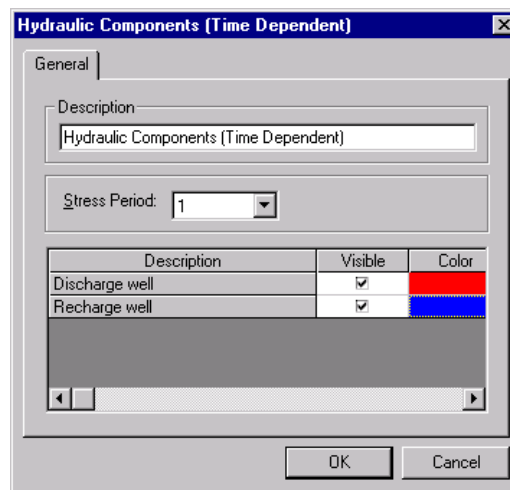


Figure 4.23: **General** page of the **Hydraulic Components (Time Dependent)** dialog box.

The Geospatial Model is added to the list of visualization objects in the **Visualize** dialog box (Figure 4.24).

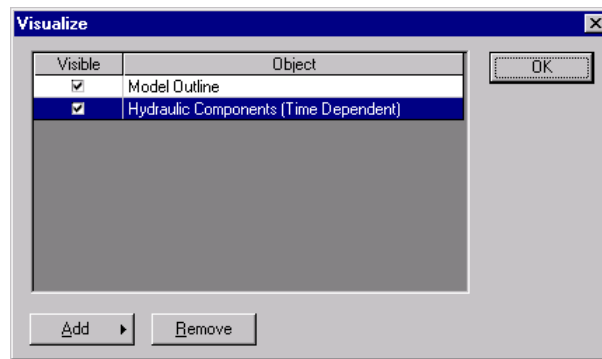



Figure 4.24: **Visualize** dialog box with **Hydraulic Components (Time Dependent)** visualization object added.

Click the  button to see the result as in Figure 4.25.

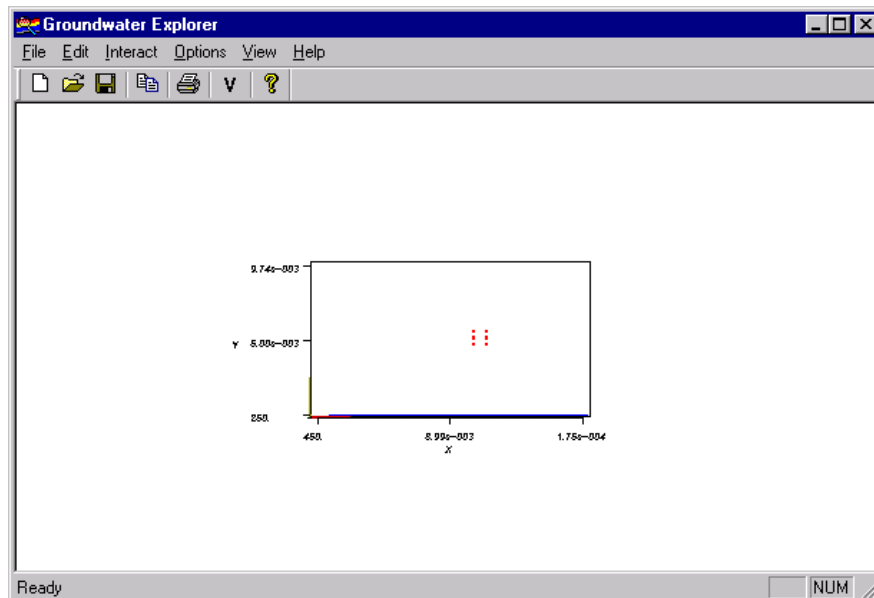
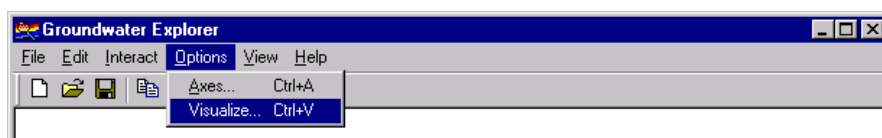
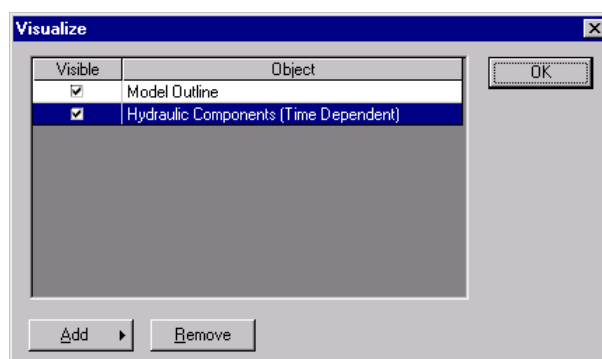
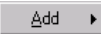


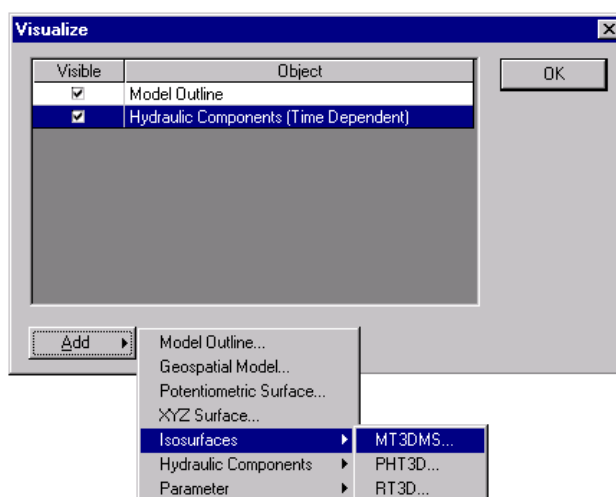
Figure 4.25: Label Axes, Color Axes, Model Outline and Hydraulic Components (Time Dependent) visualization objects.

#### 4.3.4 Step 4: Add Isosurface

Click **Visualize...** (Figure 4.26) to display the **Visualize** dialog box, from which the Isosurface visualization object can be added. The **Visualize** dialog box is shown in Figure 4.27.

Figure 4.26: **Visualize...** item.Figure 4.27: **Visualize** dialog box.

Click the  button to display the **Add** button items (Figure 4.28).

Figure 4.28: **Add** button items.

A new MT3DMS isosurface visualization object is added by clicking on the **MT3DMS...** item, which displays the **Concentration Isosurface (MT3DMS)** dialog box with the **General** page (Figure 4.29).

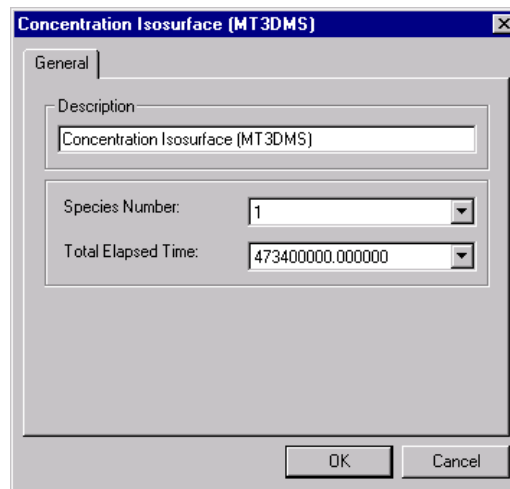
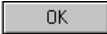


Figure 4.29: **General** tab of the **Concentration Isosurface (MT3DMS)** dialog box.

Keep the default values for all the boxes; accept select 473400000.000000 from the **Total Elapsed Time** dropdown list, and click the  button. It can be seen from the **Visualize** dialog box in Figure 4.30, that the Concentration Isosurface (MT3DMS) is added to the list of visualization objects.

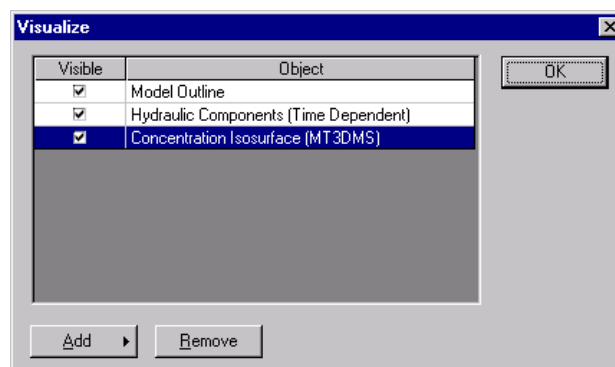



Figure 4.30: **Visualize** dialog box with **Concentration Isosurface (MT3DMS)** visualization object added.

Click the  button to see the result as in Figure 4.31.

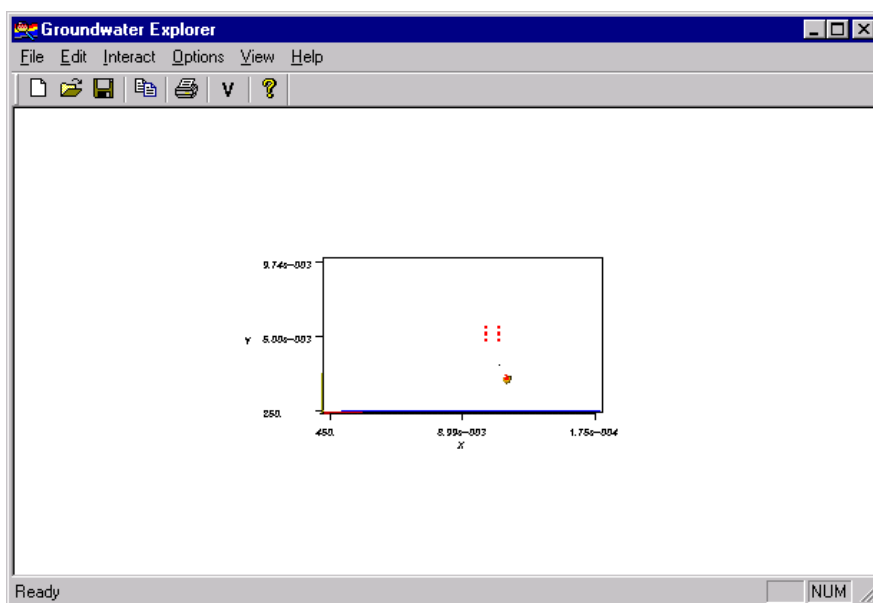


Figure 4.31: Label Axes, Color Axes, Model Outline, Hydraulic Components and Concentration Isosurface (MT3DMS) visualization objects.

#### 4.3.5 Step 5: Scale

Follow the same steps as in section 4.2.4, “Step 4: Scale”.

#### 4.3.6 Step 6: Change Contours

According to USEPA, the maximum contaminant level (MCL) for Benzene in drinking water is 5  $\mu\text{g/l}$ . In this tutorial step GE is used to see if such concentrations have reached the well field. Click **Visualize...** (Figure 4.32) to display the **Visualize** dialog box (Figure 4.32) from which the **Concentration Isosurface (MT3DMS)** visualization object can be changed.

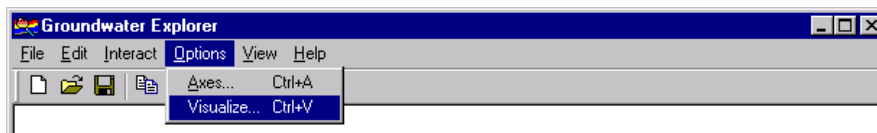


Figure 4.32: **Visualize...** item.

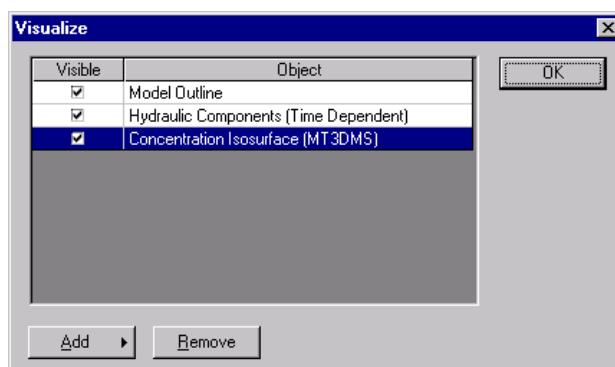


Figure 4.33: **Visualize** dialog box.

To make changes to an existing visualization object double click the description of the visualization object in the **Object** field. Double click **Concentration Isosurface (MT3DMS)** (Figure 4.33), which will display the **Concentration Isosurface (MT3DMS)** dialog box (Figure 4.34). Note that only the **Description** on the **General** page can be changed. If other isosurfaces are to be displayed for a different **Species Number** and **Total Elapsed Time**, add a new Isosurface visualization object from the **Visualize** dialog box with the **Add** button.

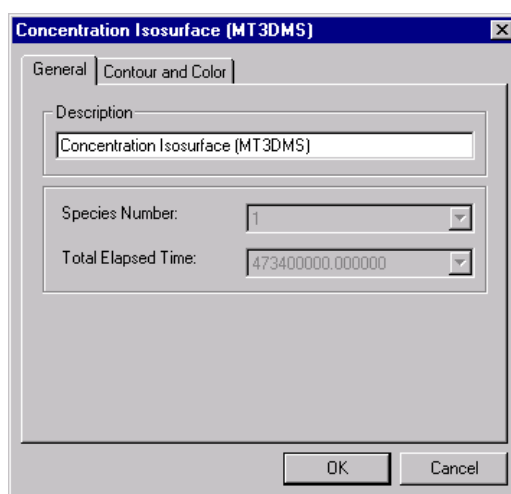


Figure 4.34: **Concentration Isosurface (MT3DMS)** dialog box.

The changes to the concentration isosurfaces must be made on the **Contour and Color** page. Click the **Contour and Color** tab to display the **Contour and Color** page (Figure 4.35).

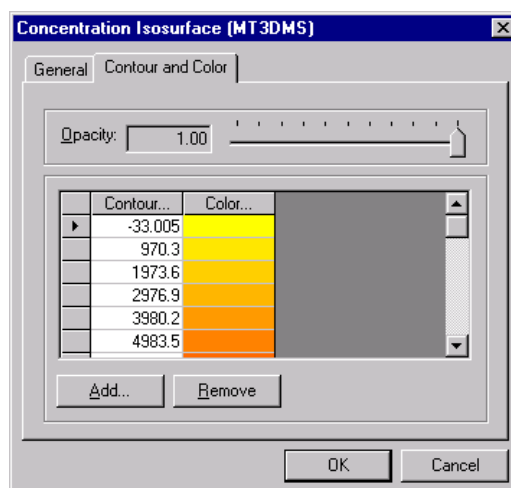


Figure 4.35: **Contour and Color** page.

Click the **Contour...** column header button, which will display the **Color Level** dialog box (Figure 4.36) used for specifying the minimum and maximum contour values and interval.

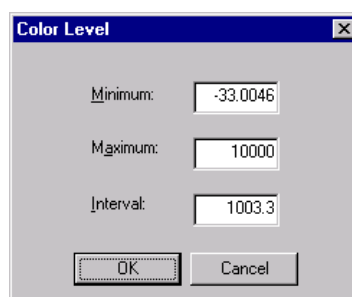


Figure 4.36: **Color Level** dialog box.

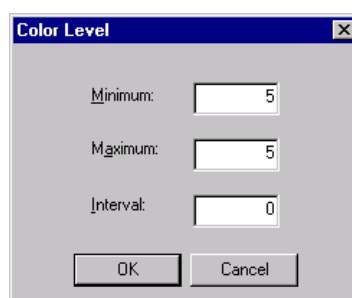


Figure 4.37: Changed **Color Level** dialog box.

Change the values of the **Minimum** and **Maximum** boxes to **5** and the value in the **Interval** box (Figure 4.37) to **0**. Click the **OK** button.

Because a Benzene concentration of 5  $\mu\text{g}/\text{l}$  is a health risk in drinking water, it is a good idea to change the color of the 5  $\mu\text{g}/\text{l}$  concentration isosurface to red.



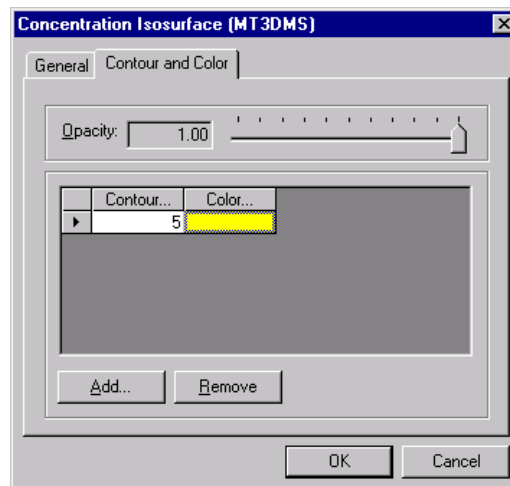


Figure 4.38: **Contour and Color** page.

To do that, double click the color cell in the **Color...** field, next to the **5**. This displays the **Color** dialog box (Figure 4.39).

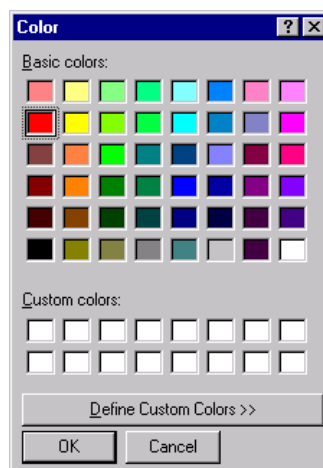


Figure 4.39: **Color** dialog box.

Click the red square (first column, second row) and then the **OK** button. Figure 4.40 shows the changed **Contour and Color** page.

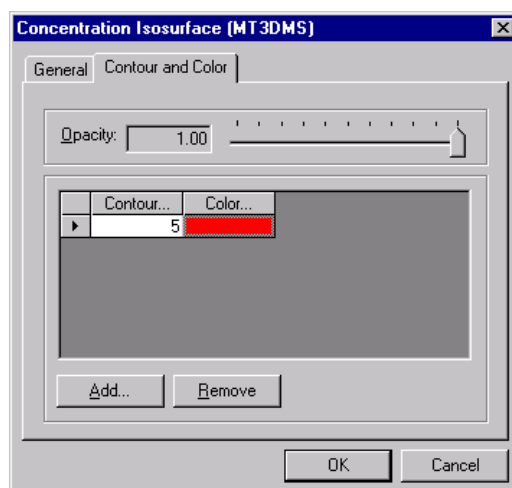


Figure 4.40: **Contour and Color** page.

The changes to Isosurface visualization object are done. Click the **OK** button. A rotated and zoomed result of the scene is shown in Figure 4.41. It is clear that a concentration of 5  $\mu\text{g/l}$  has reach the well field and therefore something needs to be done.

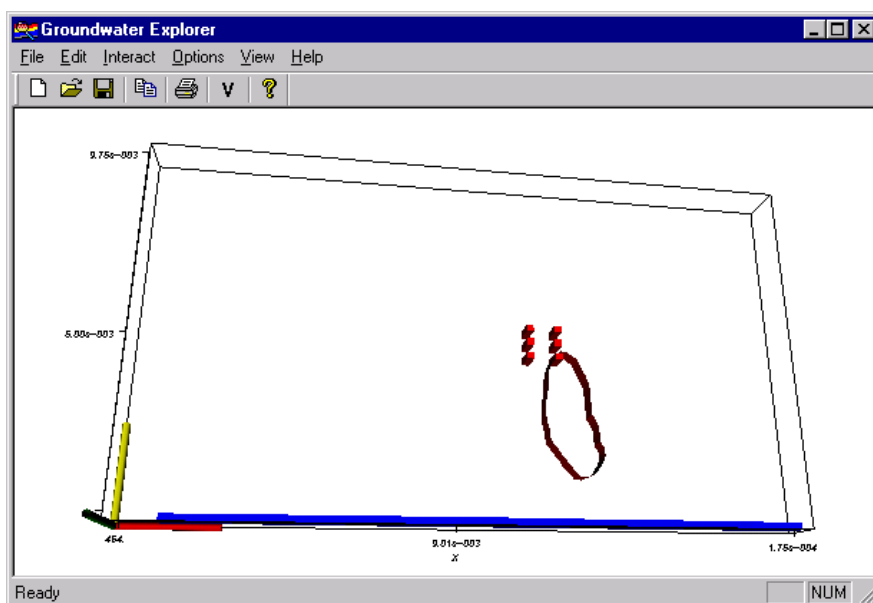


Figure 4.41: 5  $\mu\text{g/l}$  Isosurface.

# References

---

- Brooks, F.P. (1996). *The computer scientist as Toolsmith II*. Communications of the ACM, March 1996, Vol. 39, No. 3.
- Bajaj, C. (1999). *Data Visualization Techniques*. John Wiley and Sons Ltd.
- Botha, J.F., Verwey, J.P., Van der Voort, I., Vivier, J.J.P., Buys, J., Colliston, W.P. and Looock, J.C. (1998). *Karoo Aquifers. Their Geology, Geometry and Physical Properties*. WRC Report No TT 73/95. Water Research Commision, P.O. Box 824, Pretoria 0001.
- Bowie, J.E. (1995). *Data Visualization in Molecular Science. Tools for insight and Innovation*. Addison-Wesley Publishing Company, Inc.
- Chiang, W.H. and Kinzelbach, W. (2000). *3D Groundwater Flow and Transport Modeling with Processing MODFLOW*. Springer, Berlin, Heidelberg, New York.
- Clement, T.P. (1997). *RT3D: A Modular Computer Code for Simulating Reactive Multi-Species Transport in 3-Dimensional Groundwater Sytems*. The U.S. Department of Energy. Contract DE-AC06-76RLO 1830. Pacific Northwest National Laboratory Richland, Washington, 99352.
- Foley, J.D., Van Dam, A., Feiner, S.K. and Hughes, J.F. (1996). *Computer Graphics: Principles and Practice*. (2<sup>nd</sup> ed. in C). Addison-Wesley Publishing Company, Inc.
- Johnson, C., Parker, S.G., Hansen, C., Kindlmann, G.L. and Livnat, Y. (1999). *Interactive Simulation and Visualization*. Computer Graphics World. December Volume 32, Issue 12.
- Machover, C. (2000). *Computer Graphics and Visualization into the new Millennium*. A Keynote Speech at WSCG.
- Mahoney, D.P. (2000). *A new view on volumes: New technologies for rendering volume data could lure polygonal graphics diehards into the volumetric fold*. Computer Graphics World. July Volume 23, Issue 7.
- McDonald, M.G. and Harbaugh, A.W. (1998). *A Modular Three-Dimensional Finite-Difference Groundwater Flow Model*. U.S. Geological Survey Techniques of Water Resources Investigations.
- Moltenbrey, K. (1999). *Fields of dreams. VR provides its weight in (black) gold for locating hidden oil reserves*. Computer Graphics World. September Volume 22, Issue 9.
- Prommer, H. (2000, in preparation). *A coupled PHREEQC and MT3DMS for simulating multi-species reactive transport in groundwater*. Personal contact.

- Robertson, B. (2000). *Sea Change. ILM's effects crew plunged deep into state-of-the-art technology to create digital water for The Perfect Storm.* Computer Graphics World. July Volume 23, No. 7.
- Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991). *Object-oriented modeling and design.* Prentice Hall, Englewood Cliffs, New Jersey 07632.
- Stifter, J. (2000). *vtk Documentation 3.0.* Online vtk Documentation generated by Doxygen written by Dimitri van Heesch, © 1997-1999.
- Schroeder, W.J., Avila, L.S. and Hoffman, W. (2000). *Visualizing with VTK: A Tutorial.* IEEE Computer Graphics and Applications, September/October.
- Schroeder, W. J., Martin, K. M. and Lorensen, W. E. (1998a). *The Design and Implementation Of An Object-Oriented Toolkit For 3D Graphics and Visualization.* IEEE.
- Schroeder, W., Martin, K. and Lorensen, B. (1998b). *The Visualization Toolkit. An Object-Oriented Approach to 3D Graphics.* (2<sup>nd</sup> ed.). Prentice Hall PTR. Upper Saddle River, NJ.
- Schroeder, W.J. and Martin, K.M. (1998c). *User's Guide.* Kitware, Inc.
- Ware, C. (2000). *Information Visualization, Perception for Design.* Academic Press.
- Zhang, J. (2000). *Estimation of the Preliminary Groundwater Reserve using Numerical Models.* Submitted in fulfillment of the requirements for the degree of Master of Science in the Faculty of Natural Sciences, Institute for Groundwater Studies at the University of the Free State. April.
- Zheng, C. (1990). *MT3D. A Modular Three-Dimensional Transport Model.* S.S. Papadopoulos & Associates, Inc, Rockville, Maryland.
- Zheng, C. and Wang, P.P. (1998). *MT3DMS. A Modular Three-Dimensional Multispecies Transport Model.* Waterways Experiment Station, US Army Corps of Engineers, Vicksburg, Mississippi, 39187.
- Zheng, C. and Wang, P.P. (1999). *MT3DMS. A modular three-dimensional multispecies model for simulation of advection, dispersion and chemical reactions of contaminants in groundwater systems; Documentation and User's Guide.* Contract Report SERDP-99-1, U.S. Army Engineer Research and Development Center, Vicksburg, MS.

# ***Abstract***<sup>\*</sup>

---

Groundwater flow and transport models produce large amounts of data, which the human brain cannot possibly grasp. Taking advantage of the natural abilities of the human vision system, 3D visualization is often the tool of choice for understanding and communicating conceptual models, verifying model input, understanding model output, explaining and communicating conclusions and recommendations, and motivating expenses. A 3D visualization tool has therefore been developed for intelligence amplification of model data. The tool is based on a groundwater modeling system (Processing MODFLOW) and makes use of the results from existing groundwater flow (MODFLOW) and transport models (MT3DMS, PHT3D and RT3D). The Visualization Toolkit (vtk), a C++ class library for visualization was used to render 3D geohydrological objects. Realistic scenes of 3D geospatial models and 3D distributions of geohydrological properties, such as hydraulic conductivity, heads and solute concentrations, can be rendered.

The advantages of 3D visualization are evident by applying the visualization tool to case studies.

**Keywords:** visualization, computer graphics, groundwater flow models, groundwater transport models

<sup>\*</sup>Parts of this abstract were submitted to the International Association of Hydrogeologists (IAH) Congress, 26 November – 1 December 2000 in Cape Town, South Africa, but were not accepted. The co-author of these parts is Prof. W.H. Chiang.



# *Opsomming*

---

Grondwatervloei- en -transportmodelle produseer groot volumes data wat die menslike brein nie maklik kan verstaan nie. Dit is duidelik wanneer die natuurlike vermoë van menslike visie in ag geneem word, dat 3D-visualisering 'n goeie manier is om konseptuele modelle te verstaan en oor te dra, modelinvoer te ondersoek, modelafvoer te verstaan, gevolgtrekkings en voorstelle te verduidelik en oor te dra en om uitgawes te motiveer. 3D-visualiserings sagteware is dus ontwikkel vir intelligensie versterking van modeldata. Die sagteware gebruik as basis 'n grondwater-modelleringsstelsel (Processing MODFLOW) en maak ook gebruik van bestaande grondwatervloei (MODFLOW) en -transportmodelle (MT3DMS, PHT3D en RT3D). "The Visualization Toolkit" (vtk) wat uit C++ klasse bestaan is gebruik om 3D geohidrologiese objekte te vertoon. Realistiese tonele van georuimtelike modelle en die 3D verspreiding van geohidrologiese eienskappe soos hidroliesegeleiding, watervlakke en besoedelingskonsentrasies kan vertoon word.

Die voordele van 3D-visualisering is opmerklik uit die toepassings van die visualiserings sagteware op gevallestudies.