GWVIS: A TOOL FOR COMPARATIVE GROUND-WATER DATA VISUALIZATION

By

DANIEL BEST

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

MAY 2009

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of DANIEL BEST find it satisfactory and recommend that it be accepted.

Robert R. Lewis, Chair

MD. Akram Hossain

Wayne O. Cochran

ACKNOWLEDGEMENT

The following individuals deserve thanks many times over. RuthAnn, my wife, who watched our little girl while I was doing homework, patiently listened to “computer speak”, and helped make me into a better writer. Ember, my daughter, for providing a ray of happiness during even the hardest times. Dr. Lewis, for the countless hours discussing the project, and strengthening my understanding of the need to know what the data is doing and where it is at at all times. Dr. Hossain for being the patient domain expert whom I could continually “bring a rock” and receive valuable input. Finally, Battelle Memorial Institute for funding my way through school.

GWVIS: A TOOL FOR COMPARATIVE GROUND-WATER DATA VISUALIZATION

Abstract

by Daniel Best, M.S.
Washington State University
May 2009

Chair: Robert R. Lewis

The Ground-Water Visualization system (GWVis) presents ground-water data visually in order to educate the public on ground-water issues. It is also intended for presentations to government and other funding agencies. GWVis works with ground-water level elevation data collected or modeled over a given time span, together with a matching fixed underlying terrain.

Our tool is developed using the Python programming language in conjunction with associated extension packages and application program interfaces such as OpenGL™to improve performance and allow us fine control of attributes of the model such as lighting, material properties, transformations, and interpolation.

There are currently several systems available for visualizing ground-water data. One class, research oriented models, are overly complex and require training to interact with. Another class, static, presentation based representations (i.e., on paper) are not engaging, and have difficulty to representing multiple data dimensions.

GWVis bridges the gap between static and research based visualizations by providing an intuitive, interactive design that allows participants to view the model from different perspectives and to infer information about simulations. By incorporating scientific data in an environment that be easily understood, GWVis allows that

information to be presented to a large audience base.

Additionally, GWVis provides the capability to compare two datasets, visualizing key differences. Typically, one dataset is field information and the other is a simulation based on the original data. However, the comparison capability is also available for two simulation sets if desired. This comparison capability, coupled with the simplified layout of GWVis, provides a rich environment for presentations and core analytic capability that make it valuable to researchers modeling ground-water flow as well as to the public.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS iii

ABSTRACT v

LIST OF TABLES ix

LIST OF FIGURES x

CHAPTER

1. INTRODUCTION 1

1.1 Purpose 1

1.2 Scope of Work 2

2. BACKGROUND 3

2.1 Ground-Water 3

2.2 OpenGL 4

2.3 OpenDX 5

2.4 Python 7

2.5 NumPy 8

2.6 wxPython 9

3. RELATED WORK 10

3.1 Static, Presentation-Oriented (SPO) 10

3.2 Dynamic, Research-Oriented (DRO) 10

3.3 Spatially Distributed Data Interaction 13

4. DEVELOPMENT 15

4.1 Problem Formation 15

4.2 Analysis of the Problem 15

4.3 Method 17

4.4 Prototyping 17

4.5 Incorporating Related Work 18

4.6 Iterative Development 19

4.7 Prototype in OpenDX 20

4.7.1 Color 20

4.7.2 Downhill Vector 22

4.7.3 Interpolation 25

5. IMPLEMENTATION 27

5.1 Model View Controller 27

5.2 Interaction 28

5.2.1 Trackball 28

5.2.2 Azimuth and Altitude 29

5.2.3 Encoding Elevation with Histogram Equalization 31

5.3 Comparison 32

5.4 Text 35

5.5 Efficiency 35

5.5.1 Memory 36

5.5.2 Time 37

6. EVALUATION 40
6.1 Comparison 40
6.2 Presentation 41

7. CONCLUSIONS 43

8. FUTURE WORK 44

BIBLIOGRAPHY 46

APPENDIX

A. DERIVATON OF THE NORMAL TO A LEAST SQUARES FIT PLANE . . 49

LIST OF TABLES

Page

5.1 User Interaction 30

LIST OF FIGURES

Page

2.1 OpenDX data flow canvas for importing water data. 6

3.1 Changes in Ground-Water Levels for the Spokane Valley-Rathdrum Prairie Aquifer. 11

3.2 IGW Modeling Interface. 12

3.3 Google Earth. 14

4.1 Green Saturation Color Map. 21

4.2 Sine Wave Color Encoding. 22

4.3 Downhill Direction Diagram. 23

4.4 Direction of Downhill Vector. 25

5.1 GWVis visualization interface. 33

5.2 GWVis Rate Canvas. 34

5.3 GWVis Difference Canvas. 34

6.1 Addition Ease Comparison Concept from [9]. 41

Dedication

To RuthAnn and Ember – my two little stars that helped me through it all.

CHAPTER ONE

INTRODUCTION

Visualization of spatially-distributed data in general – and in particular ground-water – is a growing field of research. Prior to three dimensional graphical computer models, ground-water was most often represented as a two dimensional static image. Hydrology now has research tools that allow for analysis of complex problems in three dimensions [11]. These research models have been created for use by domain experts.

Recently, interaction techniques for complex models have been advancing in usability [4]. GWVis attempts to stand as an intermediary between the classic static presentation-oriented (SPO) two dimensional maps and the dynamic research-oriented, (DRO) three dimensional visualizations. The following sections describe the purpose and the scope of work to build a prototype, then an implementation of a solution to this problem. We then provide background on the current field of ground-water visualization, discuss the analysis of the problem, describe our prototype and solution to the problem, present our results, and finally discuss possible future work.

1.1 Purpose

This project provides a visual analytic tool that can be used for presentations and public dissemination. Visual analytic tools used to convey information, such as GWVis, are attractive because of the way human perception operates, allowing aspects of perceiving information to happen without conscious thought [3]. Spatially-distributed data visualization harnesses this natural human capability to present large datasets which would normally produce a cognitive overload.

Our aim is to produce a presentation-ready visual tool that allows comparisons of

ground-water flow simulations. To be presentation-ready the tool will be a simplified visualization that conveys ground-water concepts without the use of complex interaction requirements. Comparison is a key element of GWVis, one that sets our tool apart from other visual tools. The capability to compare two simulations is an advantage to ground-water researchers utilizing the MODFLOW system [13]. Instead of reading text files and comparing numeric values, we provide the capability to immediately present differences visually.

1.2 Scope of Work

The GWVis project consists of a visualization system that will continue development beyond the scope of this thesis (et seq.). This enables the project to continue incorporating changes that will provide valuable visual capabilities past the initial development.

GWVis audience is comprised of individuals who are interested in ground-water research. Interest can be founded in a desire to understand more about the environment: To know, for example, the result of adding more wells to an aquifer, or the need to perform further research when a model does not predict data accurately.

Data used for the tool is provided by the study conducted by Hsieh et al. in the form of MODFLOW data files [7]. Information given to the project consists of aquifer elevation data for head and underlying terrain, along with a multi-attribute dataset for the Spokane River that includes depth, width, and inflow.

The aquifer head elevation data consists of three layers of aquifer information in a grid of 256 by 172 floating point values. Grid coverage is 326 square miles of the Spokane Valley-Rathdrum Prairie aquifer. All three layers have information recorded for 181 months starting September 1990 and going until September 2005.

CHAPTER TWO

BACKGROUND

GWVis's method of development benefits from technologies that can be developed in a way that reduces the time between the identification of the need for a feature by the domain expert and the presentation of that feature's implementation. The ability to produce a result quickly reduces the iteration cycle time needed between discussing additions with the domain expert and gaining insight on their effectiveness.

In addition, technologies that allow for the creation of three dimensional visualizations are needed to produce the final result. In the following sections background information is provided about the domain GWVis focuses on and the technologies used to develop the prototype and implementation.

2.1 Ground-Water

GWVis is a subset of the GIS visualization domain, focusing on ground-water. The study of ground-water looks at water located underneath the Earth's surface. Water in all forms, not just flowing water, is part of this field including soil moisture, permafrost, and immobile water. Water flows through the soil at different rates due to soil composition and state [22]. When the water in the ground reaches a point of saturation, this is called the water table. The flow of water typically travels from the surface into the aquifer system. However, water can flow from the aquifer back to the surface. Two major categories of aquifers are considered here: confined and unconfined. Confined aquifers contain layers of soil that is impermeable to water such as clay or bed rock. Absence of clay or bed rock signifies the aquifer is unconfined.

Research topics for hydrologists include topics such as pollution of the water, energization of the aquifers, and the overuse of aquifer water. When an aquifer is overused it can have repercussions such as the inability of a ecosystem to sustain itself due to the lack of water. Or, where humans are concerned, it may cause preexisting wells to no longer be deep enough to reach the water table. If this happens, the well will need to be made deeper so that it can resume pumping water. Scenarios run by MODFLOW, providing GWVis with data, create what-if analysis of placing wells in various locations in the aquifer. By running an analysis and comparing it to actual retrieval data an analyst can quickly show negative or positive outcomes of well placement.

The aquifer used in the model is the Spokane Valley-Rathdrum Prairie (SVRP) aquifer. This aquifer supplies water to more than 500,000 residents in Spokane County, Washington, and Bonner and Kootenai Counties, Idaho [7]. The site underwent a study to develop a computer model to produce data that allows analysis of aquifer inflows and outflows, and simulate the effects of future changes in ground-water.

2.2 OpenGL

OpenGL is an application program interface (API) to graphics hardware that allows the programmer to produce interactive three dimensional applications[20]. OpenGL is the standard for three dimensional graphics. Many popular three dimensional visualization tools, such as OpenDX [23], use OpenGL as their foundation.

All shapes in OpenGL are constructed using geometric primitives (points, lines, and triangles) [20]. Visualization systems build the shapes the user sees out of typically thousands of these primitives.

The API is not tied to a specific windowing or operating system, thus providing cross-platform capability. However, it is output-only: The user must provide the input part of the

graphical user interface (GUI) by other means.

To use the API a developer binds “hooks” into the windowing system events that the program will utilize for drawing and interacting with the visualization. Common methods for creating a visualization include initializing, drawing, and redrawing. During initialization a developer will set attributes for the lighting environment, which also controls the color for the image, and establish if the model will be rendered using perspective or orthographic projection. During drawing the API is told how to draw the elements of the visualization. Object are drawn as points, lines, and triangles. This is slightly abstracted by allowing the use of squares, and series of shapes to help with speed. Redrawing of the visualization allows the reuse of data stored in vertex buffer objects (VBO) or display lists. By using stored instructions or data to redraw the image takes less time to render.

2.3 OpenDX

OpenDX started as a visualization tool developed by IBM known as the “Visualization Data Explorer”, “Data Explorer”, and “DX”[23]. DX was later released as open source software under the name OpenDX. The tool provides many powerful capabilities that allow a developer to rapidly produce two and three dimensional visualizations of information. Thompson et. al. describe the visualization process within OpenDX as:

1. Gather, collect, or create the data to be visualized
2. Investigate the best way to visualize the data
3. Prepare the data for the OpenDX data model and work through data import
4. Design the visual analysis and visual transformations required to achieve the vision
5. Determine and understand the output requirements

By following these steps, a scientist can quickly develop an application that provides valuable insight into their data. A key attribute of the tool is the ease which the process can be iteratively applied leading to a more refined solution. Helping to facilitate the visualization process, there are several tools at the hands of a developer. These tools include the data explorer, the Visual Program Editor (VPE), modules, and macros.

The OpenDX data explorer steps through the process of importing data. It requires in-depth understanding of the format the data is in, as well as its topology: a regular grid, a deformed regular grid, an irregular grid, or scattered. Also, one must know the dependencies of the data; whether the data is dependent on location or space. By knowing these items along with what the data represents a user of the system can create a `.general` file which can then later be used in the VPE to import the data to the tool.

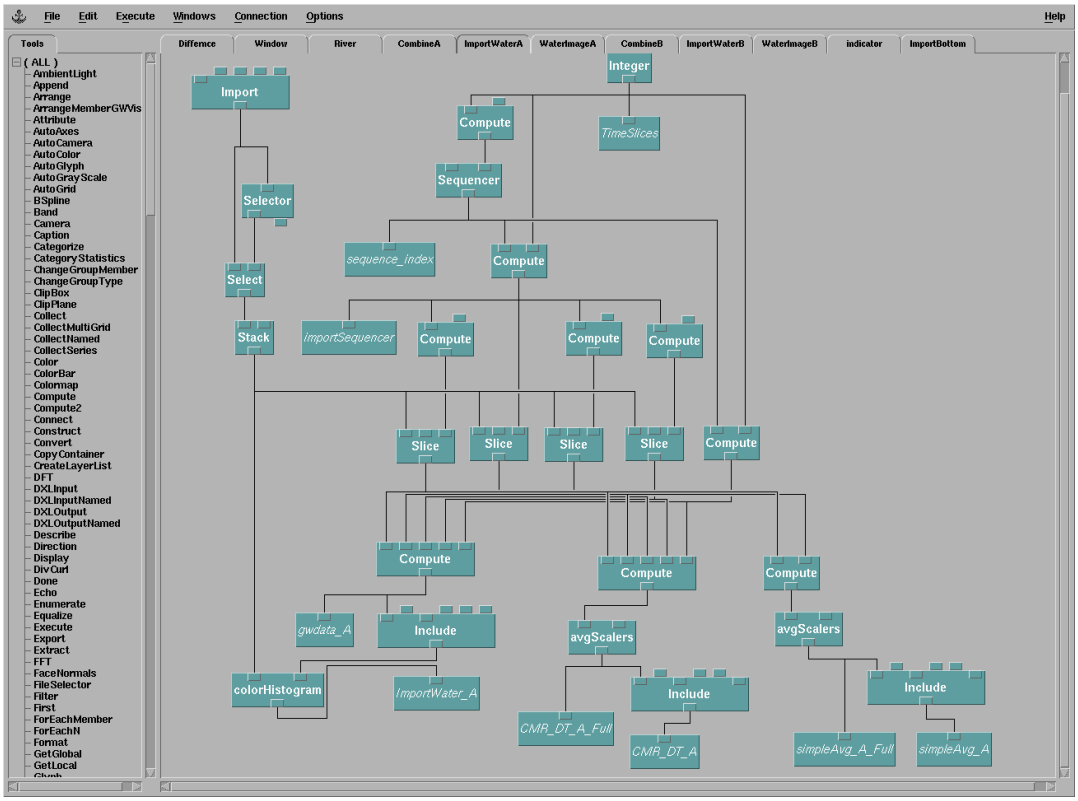

Figure 2.1: OpenDX data flow canvas for importing water data.

The “programming canvas”, like that seen in Figure 2.1, uses modules and macros to produce an image. The canvas is a visual programming language (VPL), that allows a developer to create an program using models instead of text. A module is an object that can be dragged onto the programming canvas using the mouse. There are many modules available to a developer that perform tasks from basic computation to complex creation of “rubber sheet” datasets. Each module has a number of inputs and outputs of varying data types. Inputs allow the module to perform its function, while the outputs are fed into another module or produce a final output such as a visualization or file. To connect the output of one module to the input of another module a developer clicks and drags an arrow representing the direction of data. Output is represented at the bottom of the module image and input is at the top of the image, allowing for hierarchical program views.

Importing the data can be accomplished in several ways such as importing a spreadsheet or using a `.general` file developed with the data explorer. Once imported, the data can be fed through modules that perform color mapping, calculations, “rubber sheet” creation, glyph creation, and other useful visual transformations. If a module is not provided by default, a developer has the choice of creating a macro or implementing their own in C. A macro is a collection of modules that are packaged together and have their own inputs and outputs, represented on the programming canvas using the same image as a standard module.

When an image is produced, interaction with the image is included by default. Users have the ability to zoom, pan, rotate, and animate the model if time series data is present.

2.4 Python

Python is a high level programming language that has attributes that make it attractive for use in visualization [10]. While easy to read, Python remains a compact language, which

results in less code doing more work as opposed to a C implementation. For GWVis, this represents the ability to adapt quickly and increase the tool's maintainability.

Being compact and powerful, coupled with features which are useful to both novice and experienced users, are what draw developers to the language [12].

Python is object oriented, therefore all the base objects provided by Python can be built upon and adapted for other purposes or improved. Objects are available to developers through Python's standard library which is mature enough to be sufficient for most programming tasks. Tasks that require extensions enjoy the diverse ways that the language allows for interaction with its code. If desired, modules for Python can be written in Python (like the standard library) or in another high level language such as C, C++, or Java. By being flexible in this manner, the language allows developers to find the best solution for complex problems; developing high performance modules in the best language possible that can then be utilized in the Python language.

Because of Python's popularity there are many modules and API's, like OpenGL, NumPy, and wxPython. The availability of these modules and API allow for the development of GWVis without the need to rewrite many popular capabilities.

2.5 NumPy

NumPy is a numerical extension to the Python programming language, and is a cornerstone of scientific computing in Python [10, 24]. A core component of NumPy is the `ndarray` object, which can contain a wide variety of scalar types with variable size and dimensionality. A given `ndarray` may only have a single scalar type. NumPy provides access to common scientific calculations such as vector, matrix, and scalar procedures over these arrays.

Also present in the code, is the ability to "slice" an array, and to select elements of an

array based on another array, or masking. [14, 15]. Slicing is a technique that allows a developer to effectively iterate through an array without using the computationally expensive (in Python) loop structure. NumPy allows developers to program numerically intensive programs while maintaining a reasonable level of time performance.

2.6 wxPython

wxPython is a GUI API built upon wxPython extensions [16]. The wxPython extensions are in turn built upon the original wxWidgets windowing framework API, written in C++ [21]. Because of this, wxPython retains the cross platform capabilities of the base API.

Like many GUI implementations, wxPython is an event-driven environment where user inputs such as button clicks and mouse drags or system inputs such as timers cause some action to be taken when that event is caught. wxPython calls user-specified "handler" functions to respond to the events in the program asynchronously.

To have a program use the wxPython library the application must have an instance of the `wx.App` class. This class initializes many of the underlying wxWidgets objects and provides functionality to start and stop the application. Since wxWidgets objects are being initialized when the `wx.App` class is created the class should be established in the initialization method for the program. In addition to the `wx.App`, a program must have at least one top level window object such as a frame, dialog, miniframe, or multiple document interface (MDI) frame objects.

CHAPTER THREE

RELATED WORK

There are many applications and models that cover aspects of viewing information associated with computational hydrology. Models range from hand-drawn two-dimensional models to complex interactive three-dimensional models. The following models exhibit attributes that relate [our] work.

3.1 Static, Presentation-Oriented (SPO)

Examples of two dimensional drawing can be seen in the USGS report by Hsieh et al.[7] In the study there are many different views of the same imagery of the Spokane Valley-Rathdrum Prairie Aquifer. Each image shows different information regarding the study. Areas of shallow bedrock, water purveyor service areas, areal distribution of water purveyor wells, and sewer hookup density. Figure 3.1 is one of these images. The drawing is showing the changes in ground-water levels for the aquifer over time.

Multiple dimensions of data are difficult to represent on the same image, hence there are 51 different figures in the report, each of which has some important role in understanding the information being presented. While the information on any particular figure is being presented in an understandable manner, it is difficult to see correlations of the various attributes all at once. A reader of the report must turn to one particular view and then back to another to make any connections between them.

3.2 Dynamic, Research-Oriented (DRO)

A second domain of study for ground-water visualization centers around research. Analysts research topics ranging from pollution plumes to available resources.

The performance of today's workstations makes feasible a new type of analytical tool.

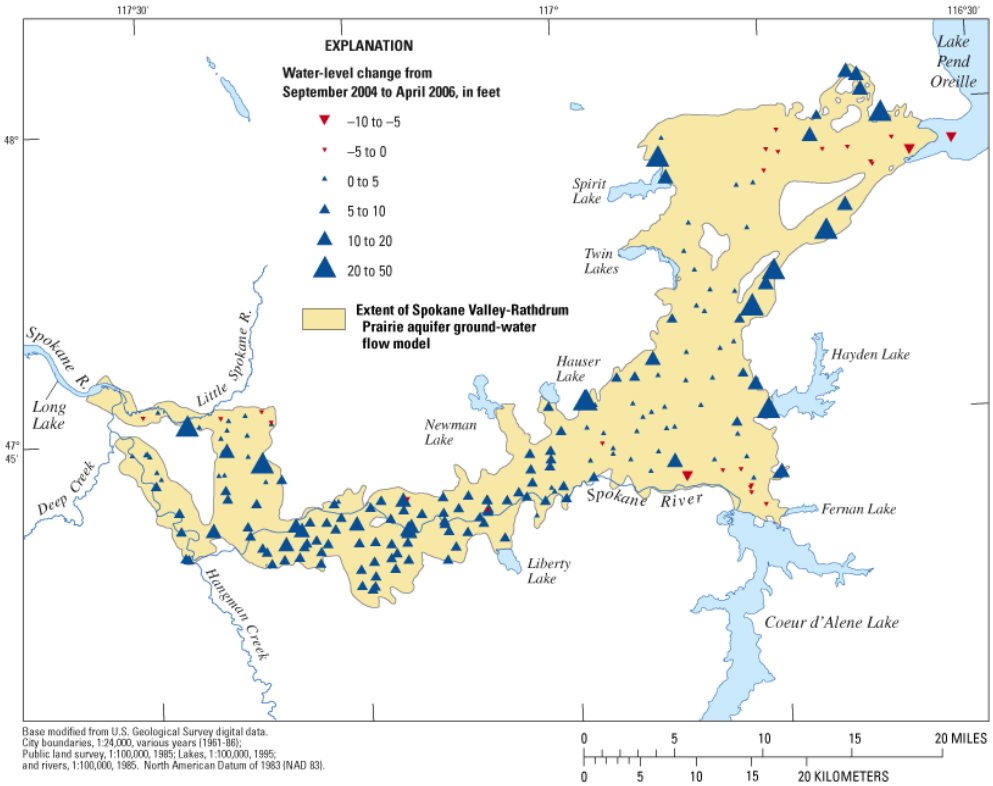

Figure 3.1: Changes in Ground-Water Levels for the Spokane Valley-Rathdrum Prairie Aquifer.

Research models that a user interacts with have been developed, such as the Interactive Ground-Water (IGW), Visual MODFLOW, Groundwater Vistas, and Aquaveo (formally GMS) [1, 11, 18, 25].

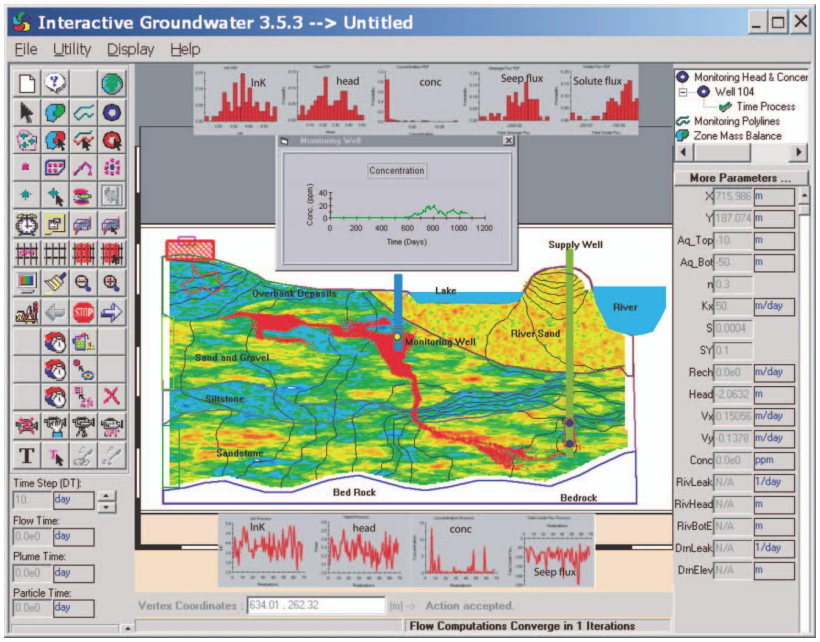

Figure 3.2: IGW Modeling Interface.

An example environment that can simulate a scenario established by a researcher is IGW. The model simulation can be stopped and adjusted at any time. It incorporates "level-of-detail" control: As a user zooms in and out of the model, the amount of information presented is increased or decreased accordingly to avoid clutter.

To allow for general research use, IGW has been developed with eight modeling, analysis, and visualization engines. The system is a powerful analytic tool allowing the researcher to perform stochastic analysis, modify properties, and develop hierarchically-nested sub-models, among other tasks. Figure 3.2 shows the IGW's GUI for changing model parameters. The model is a powerful analytic tool allowing a modeler to accomplish the following tasks:

- Modify the conceptual model such as boundaries, structures, properties and stresses
- Modify numerical representation for time step, grid spacing, layers, and solution methods
- Initiate particle tracking and/or transport modeling
- Develop hierarchically nested submodels
- Perform stochastic analysis or Monte Carlo simulation (MCS)
- Present model characteristics and results

During modeling, a researcher has the ability to interact with the data and steer the direction the model is heading. This may be to refine previous assumptions, or possibly to implement a new idea developed while running the simulation. IGW provides the following capabilities to steer the model:

- Step through inner iterations
- Step through outer iterations
- Step through the hierarchical modeling process
- Step through time increments
- Step through stochastic realizations

3.3 Spatially Distributed Data Interaction

Recent research in the field of spatially-distributed data visualization includes human factors. A major contributor to the development of this field is Google Earth. By default, Google Earth initially presents an image of the world. It is then left to the user to either

type in a location or to use the mouse inputs to navigate. Navigation within the model is accomplished by having the user click, grab, and move the model with the mouse, producing an easily learned movement.

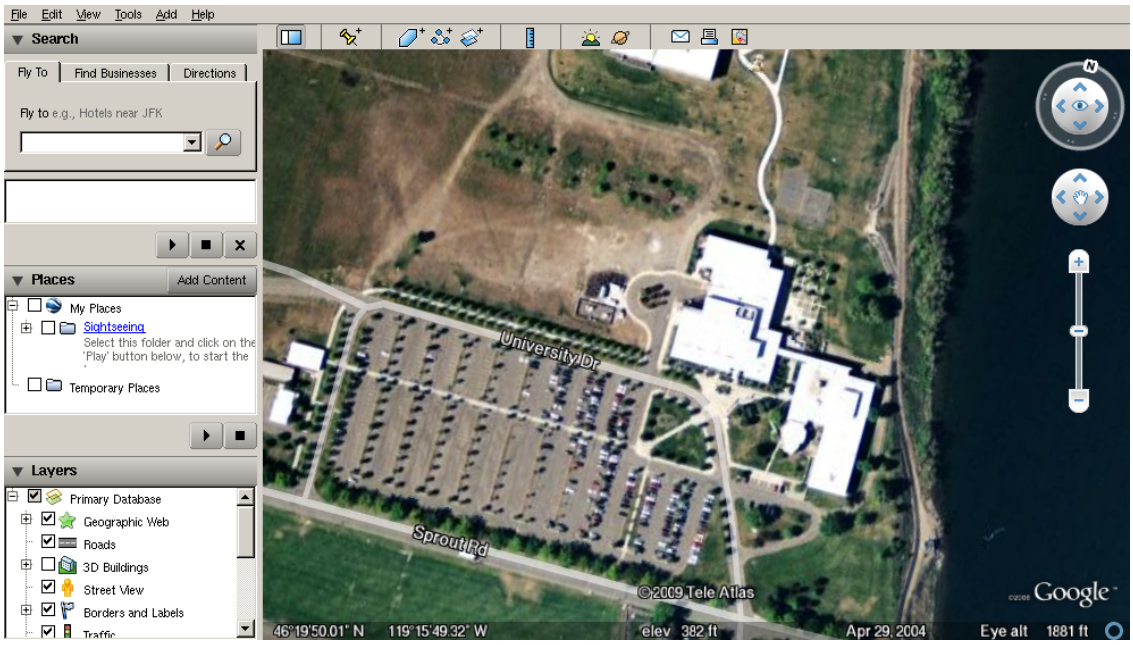

Figure 3.3: Google Earth.

Compieta et. al. look to eliminate the visual / cognitive overload possible with research models by providing two interfaces [4]. The first utilizes the Google Earth style interaction, while the other incorporates the analytic tools desired by a domain expert to data mine the information.

In that paper, the human aspect of GIS visualization is of special interest. The application looks to take advantage of the humans ability to perceive visual patterns and interpret them, and to provide a interaction means that is intuitive. System interaction is developed through the use of Google Earth technology (Figure 3.3) giving the system the ability to render the information with six degrees of freedom.

CHAPTER FOUR

DEVELOPMENT

4.1 Problem Formation

During standard course curriculum for Computer Science 538, Scientific Visualization, there is a requirement for all graduate students to complete a project using real world problems and their datasets. To accommodate this requirement the GWVis project was started. Initially the goal was focused on short term course work, however as the project progressed the model evolved into a comparative three dimensional model for ground-water data.

The initial work for Md. Akram Hossain is basic three dimensional rendering of a elevation field of data. This interaction with Dr. Hossain lead to further discussions about what the data was for and how it could possibly be used. Because he has in-depth knowledge of hydrology, and the aquifer data in particular, Dr. Hossain assisted the project as a ground-water domain expert. We determined that if the information could be displayed in an understandable and engaging manner, then the model could be used for presenting to the public and government and funding agencies.

GWVis is a product of those discussions. The goal was established to create a model for public dissemination and funding presentations. What would be shown and how were topics of discussion and research. To spur development we chose an iterative prototyping methodology to elicit feedback from the domain expert.

4.2 Analysis of the Problem

The goal is simple, show the ground-water in an engaging and intuitive manner that can be understood by non ground-water experts. To accomplish the task, the elements that make up the goal where broken out to their discrete tasks. While the possible solutions were

vague at first, the main issues with creating the model were apparent after discussions with the domain expert. These elements to implement are time, elevation change, positioning information, and how to interact with the model.

Time is not shown on the two dimensional visualizations using the image of an aquifer. Instead this concept is left to line graphs that plot a particular value over the given amount of time. When looking at how to help show the entire picture of the retrieval, we decided that showing the model in a sequence of months would be beneficial. By showing each consecutive month an observer can analyze how attributes change and determine what changed them.

The information GWVis uses to construct its model is almost solely aquifer head elevation. It is important to show the head elevation value at each point and how that point changes from month to month. Scenarios are simulated using various inputs which can alter the elevation of the aquifer. By showing a scenario alongside the actual data, a quick analysis can be accomplished by visually inspecting the models. Other cues to help indicate a rise and fall between the scenario and the gathered data were developed later, after determining visualizing the difference would be helpful. This is shown using a third model canvas.

While the viewers of the model will tend to be the public and funding agencies, they will likely have an understanding of the area being depicted by the model. It is important for GWVis to provide visual cues to establish the location of the model. Also important is where the buttons to control the model are located.

By proving a common media player interface, GWVis gives the user a sense of familiarity, easing adoption of the environment.

Finally, the display of information needs to be presented coherently. For example having the elevation information depicted as a rubber sheet and showing the rise in elevation

below the sheet is counterintuitive to the viewer. GWVis needs to show attributes of the data in a way the user does not second guess their meaning. There is a element of the graphics design process which the goal is to reveal and not obscure [3]. The interface must be presented in a way that does not over complicate the visualization.

Maintaining a user focus, GWVis also should provide a navigation control set that is intuitive to the users. Again, there are applications that already utilize interaction techniques that users are possibly familiar with. A major contributer to this type of development is Google, with their Google Earth project. When using Google Earth for the first time one can easily find the correct use of the mouse to move around the model. This type of easy interaction is key for GWVis if it is going to be used by individuals who may not have intimate knowledge of computers, or computer graphics. Using special key functions to move the model should be avoided if possible.

4.3 Method

GWVis is intended to develop a capability that can be used to display comparative visualization content which can be understood by the public. Our end goal was not strictly defined due to the uncertainty of visual techniques as applied to ground-water model comparisons. To accommodate this level of uncertainty, the project began with a fairly small requirement set, and then progressed after that point depending on results of development activities. GWVis used iterative design and implementation cycles coupled with prototyping techniques to speed implementation of changes.

4.4 Prototyping

The feasibility of a new function of the model was discussed to help eliminate tasks that could not be be implemented either in our time alloted, technology of choice, or at all.

This discussion would then lead to a form-study prototype, where elements of the solution would be drawn on a white board to develop the understanding of what was going to be accomplished.

Certain problems lent themselves to a physical prototype using stick and ball construction. This was especially the case during development of direction of downhill vectors and other geometric discussions.

Once the prototype was in a state where the basic concept was understood, the next step was to develop a small application that implemented the concept behind the form study. This visual prototype may undergo several iterations to improve upon the initial concept. If the prototype is deemed a valuable addition to the main model it is then added to the functional prototype. Finally, concepts added to the functional prototype are incorporated into the final GWVis application. GWVis allows for the customization of the code from the prototype which can overcome limitations.

4.5 Incorporating Related Work

Although two dimensional models have an engagement and information display limitation, there are attributes that GWVis borrow. Information that is important to gather from these types of models consists of data that will cement the viewer into the model space. To provide a solution that is usable, informative, and easy to comprehend, GWVis incorporates attributes from SPO, DRO, and HCI domains.

Attributes GWVis uses from the SPO domain are:

- visualization of change in aquifer elevation information,
- simple layout of information using data encoding, and
- geographical data that shows terrain, surface water, and aquifer location.

GWVis uses the following attributes from the DRO domain:

- incorporation of time and the ability to control the animation,
- ability to interact with the model during the animation,
- ability to pan and zoom into specific areas of interest, and
- displaying some analytic features.

Attributes included from HCI visualizations are:

- interaction mechanism,
- limiting cognitive overload, and
- usability-focused development.

4.6 Iterative Development

The first version of the model was developed toward a set of requirements developed during initial analysis of the problem. During development of this prototype, we met with the domain expert to ensure the information being represented was correct and appropriate for the task. Once a model was created that could be displayed and used for walk through purposes, the iterative development began. A standard iteration of a research task consisted of the following steps:

1. Discussion of the current model with the domain expert
2. Analysis of the model and discussion of the visualization with the domain expert
3. Development of a short list of possible solutions to incorporate the changes obtained in the previous steps

4. The list of possible solutions are analyzed for feasibility and how to accomplish the task

5. Solution is incorporated into the model

6. Process starts again

There have been many of these iterations throughout the development life cycle of GWVis. As such, the usability of the model has progressed with time.

4.7 Prototype in OpenDX

The prototype is developed using the OpenDX application. Several research paths were explored using the prototype to enable rapid turn around prior to implementation in the final model. Main topics included how to represent the elevation field information through the use of color, how to show the rise and fall of head elevation, and how to animate the model.

4.7.1 Color

We explored several different ideas for how to color the model. Throughout the different ways of encoding elevation with color, a saturation range is used. Varying the saturation while keeping the hue the same value for the data encoding clearly displays the value range, while representing the data as a set.

The first mechanism used was a simple color map as shown in Figure 4.1. In that model, the range in color is spread evenly from the minimum data point to the maximum data point. This was unsuccessful at showing the range of data however, since the majority of the elevation information resides in a small cluster of values resulting in the model being mostly a single color. Later revisions of the color map reduced the range in which

the saturation was altered, allowing more range to be shown on the image, however the relation of the elevation to the color lost much of its meaning.

Figure 4.1: Green Saturation Color Map.

Next, exploring how to show the amount of change in the elevation field, a sine wave was was applied to the color map. OpenDX was instructed to start at the minimum elevation and using specified parameters for the number of periods it would generate the sine wave color mapping as seen in Figure 4.2. A result of this mapping was that when the model was animated the dark blue bands appeared to travel in the direction of elevation change. If the elevation was being reduced on average in the model (such as during summer months), the bands would travel down hill. However, this also applies when the elevation is rising on average for the model, the bands appear to travel uphill. The direction of water flow is never uphill in the aquifer so the sine wave pattern was abandoned since it was easily misinterpreted to show water moving in the opposite direction that it actually was.

Finally, we implemented color histogram equalization. In this technique we establish a

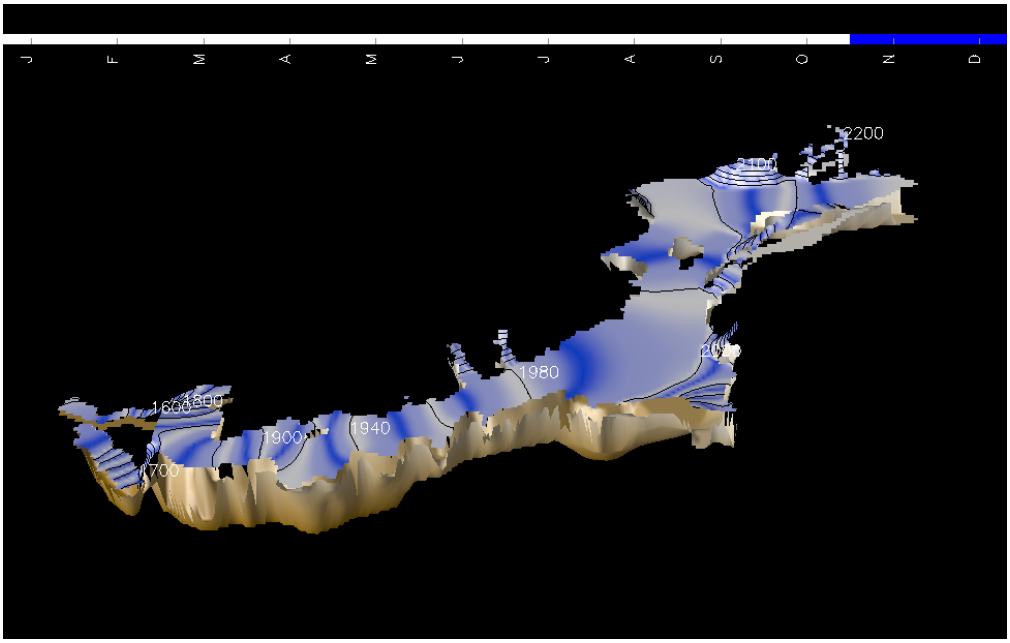

Figure 4.2: Sine Wave Color Encoding.

number of bins which will be filled with elevation information. The binning of the elevation data in this way allows for greater detail to be shown when there are a large number of elevation elements of similar value. Previously, a few elevation data points (along the edge of the aquifer) skewed the color map such that the majority of the model was the same color. Histogram equalization resolved this issue, and was implemented in the final GWVis model.

4.7.2 Downhill Vector

In the images found in the report by Hsieh et. al. [7] there are generalizations of the direction of flow for the ground-water. The prototype has access to data that could be used to generate actual direction patterns. Therefore, it was a research task to determine how to implement the direction of downhill vectors and find out whether or not the results produced would be feasible in the GWVis application. To find the downhill direction we used a least squares plane found for each point by looking at surrounding points.

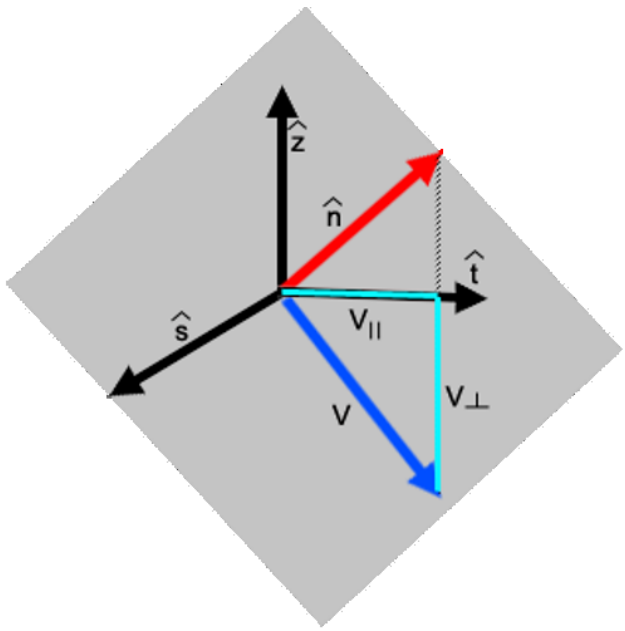

Figure 4.3: Downhill Direction Diagram.

Using the normal as found in Appendix A we find the vector corresponding to the downhill vector for the given point. We let $\vec{V}$ represent the shadow that the normal casts onto the plane. This is depicted in Figure 4.3. We decompose $\vec{V}$ into its components $V_{\|}$ and $V_{\perp}$:

$$\vec{V} = V_{\|}\hat{t} + V_{\perp}\hat{z} \tag{4.1}$$

where $\hat{z}$ is the vertical direction and $\hat{t}$ is a (unit) tangent vector lying somewhere in the xy plane. $\hat{z}$ is known, therefore we must find $\hat{t}$, $V_{\|}$, and $V_{\perp}$.

To find $\hat{t}$, we use an intermediate vector $\vec{s}$ that is perpendicular to both $\hat{n}$ and $\hat{z}$ and then make $\hat{t}$ perpendicular to $\hat{s}$ and therefore coplanar with $\hat{n}$ and $\hat{z}$:

$$\begin{aligned} \hat{z} \times \hat{n} &= \vec{s} \\ \vec{s} \times \hat{z} &= \vec{t} \end{aligned}$$

$$\hat{t} = \frac{\vec{t}}{|\vec{t}|}$$

We note that the length of the projection of both $\vec{V}$ and $\hat{n}$ onto $\hat{t}$ must be the same, $V_{\parallel}$, so

$$V_{\parallel} = \hat{t} \cdot \hat{n} \tag{4.2}$$

We solve for $V_{\perp}$ by noting that $\vec{V}$ is perpendicular to $\hat{n}$, so:

$$\vec{V} \cdot \hat{n} = 0 \tag{4.3}$$

So substituting Equations 4.1 and 4.2 into Equation 4.3 and solving for $V_{\perp}$, we get:

$$V_{\perp} = \frac{-(\hat{t} \cdot \hat{n})^2}{\hat{z} \cdot \hat{n}} \tag{4.4}$$

Equation 4.1 becomes:

$$\vec{V} = (\hat{t} \cdot \hat{n})\hat{t} - \left(\frac{(\hat{t} \cdot \hat{n})^2}{\hat{z} \cdot \hat{n}}\right)\hat{z}$$

When the plane is relatively flat, $\vec{V}$ is small, when the plane changes elevation rapidly $\vec{V}$ is large. Figure 4.4 removes the normal vectors and only shows $\vec{V}$.

As Figure 4.4 shows the direction vectors $\vec{V}$ are usually very small, with some large vectors along the edges of the aquifer. The direction vectors are small in the center of the aquifer because $\hat{n}$ closely matches $\hat{z}$, and the surface of the aquifer is nearly flat. When the model is animated the vectors change in size based on the change in head elevation. On a static image the vectors display useful information.

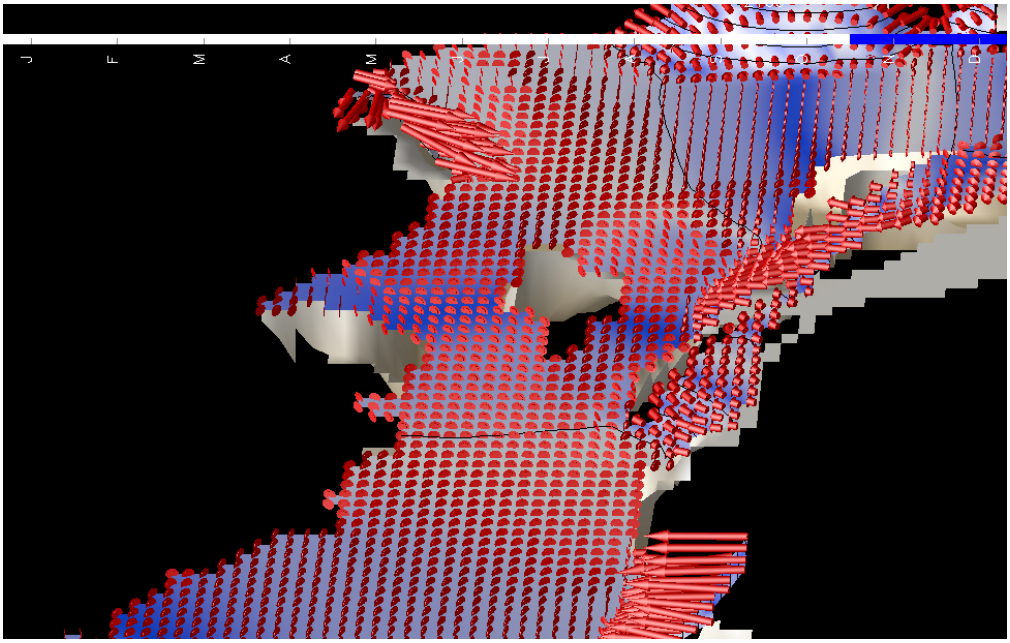

Figure 4.4: Direction of Downhill Vector.

The use of three dimensional arrow glyphs were used in the the prototype to visually confirm the shadow on the plane created by the normal was being calculated correctly, provide insight on what information could be shown, and produce an image to show the domain expert. When discussing the prototype of the downhill direction with the domain expert, it was determined that the focus should be changed to show the rate of change of a given cell. The visualization of direction was not adapted further based on that input.

4.7.3 Interpolation

Elevation information is retrieved in month intervals for the GWVis data set. When the model began animation sequences, we determined the images appeared erratic at the month boundaries. Data points had occurrences where one month the elevation was declining and the next month it was increasing, causing a dramatic shift in the column depicting change. A interpolation algorithm is a good solution to this type of problem. The first attempt at interpolation was a simple linear algorithm looking at one month to the next.

This function yielded adequate results however, we decided to pursue a smoother function; while wanting to remain fairly simple for performance and development considerations. Catmull-Rom was chosen as the favored interpolation technique. The Catmull-Rom spline has the advantage of only needing four data points to find the interpolated value. We used the following Hermite basis function polynomials in the spline calculation:

$$
\begin{aligned}
H_1(t) &= 2t^3 - 3t^2 + 1 \\
H_2(t) &= t^3 - 2t^2 + t \\
H_3(t) &= -2t^3 + 3t^2 \\
H_4(t) &= t^3 - t^2 \\
p(t) &= H_1(t)p_k + H_2(t)m_k + H_3(t)p_{k+1} + H_4(t)m_{k+1}
\end{aligned}
$$

having

$$m_k = \frac{1}{2}(p_{k+1} - p_{k-1})$$

The resulting animation smoothly transitioned from one month to the next.

CHAPTER FIVE

IMPLEMENTATION

The implementation of GWVis took lessons learned from a prototype and expanded upon them, with the goal of producing a usable system which [is] easily understood by the general public, and adaptable for future work. The program is written in Python using the wxPython, pyOpenGL, and NumPy packages heavily. GWVis consists of 14 different classes that encapsulate functionality, and uses over 2300 lines of Python code. The standard graphics API used for [rendering] three dimensional models is OpenGL; GWVis uses the Python extension of the library. The prototype, being written in OpenDX, abstracted the low level programming needed to render the model.

GWVis encountered new issues that had not been addressed previously. These issues were resolved using the same development mechanisms the prototype used, prototyping and iterative design. Focus areas for GWVis to ensure its goal are the basic interaction mechanics, data encoding of the model, the ability to compare scenarios, and efficiency.

5.1 Model View Controller

A standard architecture for client applications is the model view controller (MVC) pattern [5]. In this design, information regarding the data is housed within a model object. The visualization and graphical user interface (GUI) code is in the viewer object. Interaction between the model, events, and the viewer are managed in the controller object. An example interaction between the objects is when a user clicks on the canvas. The viewer will receive the click event and will raise the event so the controller can handle it. The controller will then call the viewer object, informing it that a change on the interface has just occurred, and which action to take. The viewer then changes the position of the camera,

possibly calling upon the model to retrieve the data to display again.

Advantages of this approach are maintainability and adaptability. Because interaction and display code is separated from the GUI event model, the library used to create the GUI could be changed at a later date with minimal code changes in either the viewer or model objects. Maintaining the code base is made easier by providing a distinct separation of duties between objects. If necessary, each object can be debugged separately from the rest of the model, allowing for isolation of an issue. GWVis is intended to live beyond the scope of the current work being done, therefore it is important for the model to be maintainable.

5.2 Interaction

Beyond the initial rendering of the model, interaction is the attribute that most affects how people perceive the utility of GWVis. Interaction with the model was continuously adapted during development such as being able to zoom in to a particular section, and how to move about the model. Two main techniques were used to allow movement around the model. The first mechanism was a trackball, where the model is simulated as being in a spinnable sphere. The second is more like Google Earth, where the camera moves along the model, and the model stays static.

5.2.1 Trackball

Trackball interaction for the project was developed using C program written by Ciemiewicz et al. in 1994 [8]. The trackball class establishes a rotation matrix created by the use of quaternions and the movement of the mouse [19]. Using the matrix, generated by mouse movements, as a transform on the model before rendering, produces the desired effect of altering the view.

To create the matrix the process is as follows:

1. A controller object is established for interaction with a particular canvas, setting the

current rotation matrix to the identity matrix.

2. The controller object is bound to mouse movement and click events.

3. While mouse movement is detected and the left mouse button is pressed, the controller gets the screen x and y coordinate and compares it to the last known position, calculating the change in x and the change in y.

4. Using the change in x and y, the trackball class computes the rotation matrix based on the model being inside a sphere and reports that matrix back to the controller.

5. The controller then instructs the viewer that it needs to redraw the model and gives it the new rotation matrix.

6. The model is rendered using the correct model information and applying the rotation matrix given by the controller.

Steps three through six are continuously applied until the user stops the mouse movement or releases the left mouse button. The effect of this process is the model moves around as the user moves the mouse, giving immediate feedback to the user. This allows them to make the determination if they wish to stop and view the current location.

During implementation of this method, the time efficiency issues of rendering the model became readily apparent. Drawing the model several times per second used up system memory and would result in halting of the program, or worse halting of the X server. Trackball was removed in favor of a simpler interaction, one that did not allow view such as the model rotated upside-down.

5.2.2 Azimuth and Altitude

After moving away from trackball interaction, the interactive attributes of Google Earth were evaluated for use in the model. Google Earth provides an interface similar to our

model. In addition, the core functionality of Google Earth interaction is a standard technique used by three dimensional models [6]. The next step was to implement that type of navigation into the model.

To provide a view that is more natural to the user, the model is developed using perspective instead of orthographic viewing. In addition, a camera class was developed that would report camera position and center direction, the latter parameterized by angles ϕ and θ. ϕ is the azimuth of the camera, and θ is its altitude. By altering ϕ the model allows for the user to look three hundred and sixty degrees. θ allows for the user to look up and down.

Camera direction is given an initial setting pointed down at the model. When the user interacts with the model to move along the terrain, the only attribute of the camera class that is specifically altered is the camera position. When the user interacts with the model to alter either ϕ or θ, then the angles are additive in the camera model, which produces the appropriate effect the next time the model is rendered. The camera will remain constant, yet where the camera is pointing will change. Finally, the ability to zoom into the model is supported by using the scroll wheel of the mouse. To change the zoom amount, the camera position is changed, moving it along the camera direction.

Table 5.1: User Interaction

Interaction	**Effect**
Click and drag left or right	Changes the azimuth of the viewer (pitch)
Click and drag up and down	Changes the altitude of the viewer (yaw)
Mouse scroll wheel forward or backwards	Moves the user forward or backward along the line of sight (dolly)
Control Button + Mouse scroll back and forth	Zooms the viewer in and out, changing the field-of-view

Interaction with the model using these methods allows a user to zoom,look left and

right, look up and down, and move through the model at a constant elevation. All canvases are synchronized so that the user is viewing the same section of the aquifer at the same time in all of them, providing a mechanism for quick comparison of the images being shown.

Animation control is constructed using typical media player input buttons (play, forward, backward, rewind, and pause). Playing the animation iterates through all data month-by-month so a user can see which areas of the aquifer are changing and which are remaining the same.

5.2.3 Encoding Elevation with Histogram Equalization

The ground-water elevation for the GWVis development data has a fairly small range: from 1400 to 2600 feet above sea level compared to 66.25 miles (349,800 feet) in horizontal extent. Most of the data values are clustered closely together. However, values change rapidly at the aquifer extremities. Because of the nonlinear data distribution, the color encoding of the model based on elevation data should not be a simple linear interpolation. To eliminate this issue, we incorporated color histogram equalization.

Histogram equalization is a standard and popular means to evenly distribute a nonlinear set of data points as seen in Bradski & Kaehler [2]. We apply this technique to evenly distribute the elevation information and encode elevation as the saturation value of blue. This technique produces a range of color from white to blue. This allows the user to infer which sections of the aquifer are higher or lower than others.

During equalization of the model, a discrete number of bins are created to place elevation data into. To create the bins, the valid data points for the aquifer are put into a one dimensional array and sorted. Next, the array is split into the number of bins desired. At each bin boundary, the elevation value was recorded. This produces the number of bins with an equal number of data points in each bin. Then, a lookup array is created using the max elevation subtracted from the min elevation array positions. During encoding, a

elevation value is used as an index for accessing the color value in the lookup array.

There are 181 months of data in the aquifer information. Each month has different minimum and maximum valid elevation values. To ensure that the entire visualization uses the same color for the same value throughout the animation, the lookup table is developed from the entire set of data. When analyzing the result of the equalization with only two bins, we expected to see half of the model one color and the other half the second color. In fact, this was not quite what was shown by the model. Instead, some months had more of the first color and less of the second, while other months had the opposite. To ensure the equalization produced correct result the number of data points in each bin for each month were summed. When reviewing the entire set of 181 months, the equalization resulted in accurate values.

5.3 Comparison

GWVis provides the ability to compare side-by-side scenario datasets during the animation of the model. SPO visualizations can provide this by having two images next to each other. The DRO visualizations do not provide a simplified visualization for this purpose. Showing the difference between two scenarios allows for basic analysis that would be beneficial to the intended audience. Layout for GWVis consists of two scenario "rate" canvases in the upper left and right quadrants and a "difference" canvas between the two scenarios taking up the lower half (lower left and lower right quadrants) of the window, as seen in Figure 5.1.

The Rate Canvases

In these canvases, GWVis uses color and position encoding to convey information about how the elevation for the visualization depicted on each canvas is rising or falling, as seen in Figure 5.2. The column is either green for increasing or red for decreasing elevation. An user can use this capability to determine which of the sets of data is changing the fastest

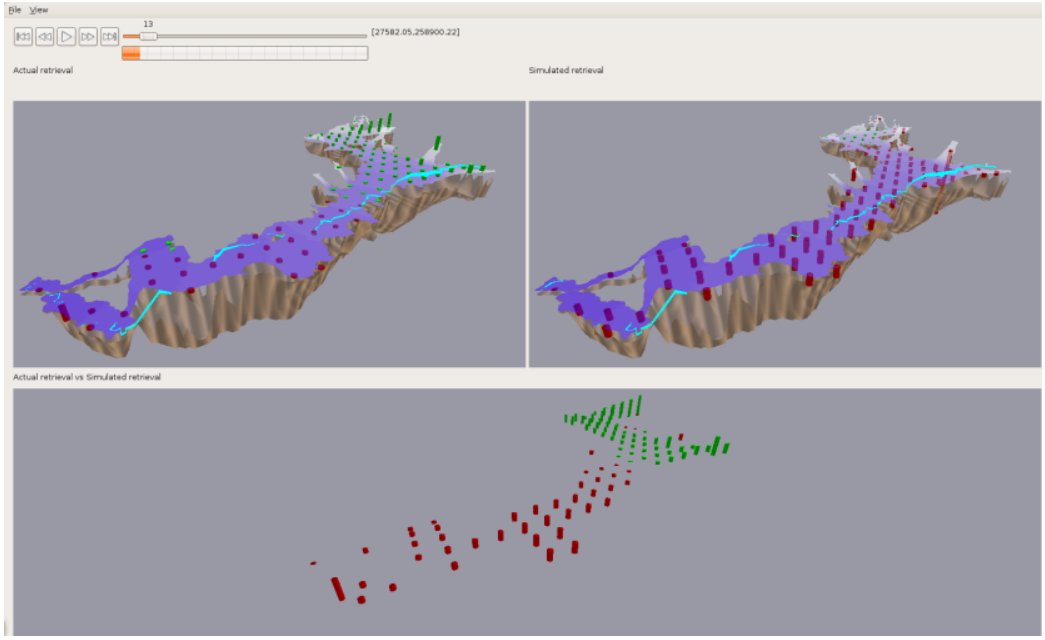

Figure 5.1: GWVis visualization interface.

where.

Because there are about 44,000 grid cells each month, a large number of columns could be shown. However, the domain expert advised that we simplify the visualization by reducing the number of columns depicted. To accomplish this, we take the mean of each 7 by 7 block of data points and draw a column with that value in the center of the block. Reducing the number of columns simplifies the visualization, reducing information overload.

The Difference Canvas

This canvas shows the difference between the two scenarios at the current time step as one image (Figure 5.3). The the right hand aquifer's elevations are subtracted from the left hand aquifer's. If, as we expect will be typical, the upper left quadrant is original flow data and the upper right quadrant is a simulation, this answers an important "what if" question for the user: "How would the changes being simulated affect the aquifer?"

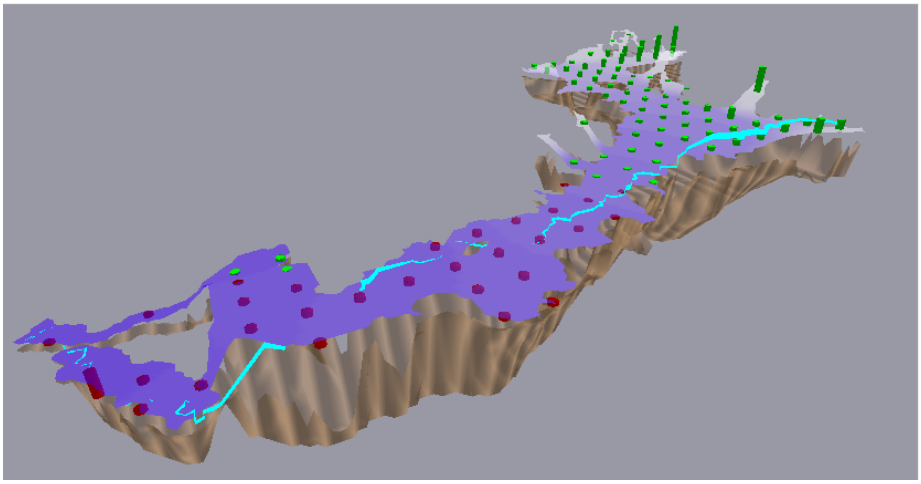

Figure 5.2: GWVis Rate Canvas.

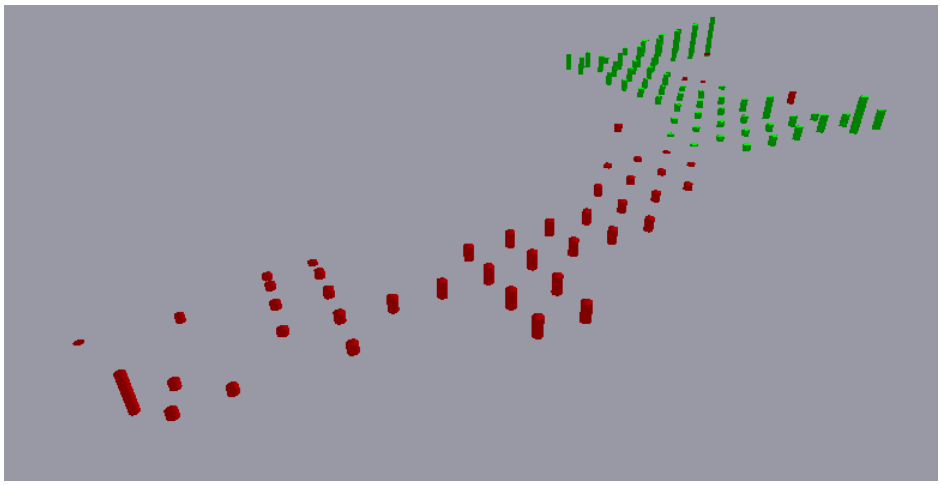

Figure 5.3: GWVis Difference Canvas.

5.4 Text

Currently minimal labeling has been added to the model. The intent is to keep the model simple yet convey the information needed. Information that has been added are labels describing the current scenarios and that the bottom panel is a comparison of the two top panels. In addition, the latitude and longitude of the current mouse position on the canvas is added to the interface.

To find the latitude and longitude, first we must find the world coordinates for where the mouse is pointing to the model. This is accomplished by using gluUnproject which creates a ray starting from the mouse position and using the current model and projection matrices to go the correct direction to intersect the model. First, the world coordinates are determined using the near and far clipping planes [23]. Next the line going between the two points is used to find the intersection of a plane representing where the model is located in three dimensional space. Once that position is found, the determination of the latitude and longitude is a transformation of feet information to coordinates. Let the equation of the plane be $ax + by + cz + d = 0$, p_0 be the world coordinates at the near clipping plane, and p_1 be the world coordinates at the far clipping plane. To find the three dimensional coordinates of the intersection with the model the following equations are used:

$$\begin{aligned} t &= \frac{-d - [a, b, c] \cdot p_0}{[a, b, c] \cdot (p_1 - p_0)} \\ r &= p_0 + (p_1 - p_0)t \end{aligned}$$

5.5 Efficiency

Python, being an interpreted language, is not as fast as some other choices could be. However, the benefits of using Python to develop the model (development community, available

libraries, and language familiarity) outweighed the possible negatives. However, there were issues that needed to be addressed in the implementation of the model that at first made it unusable. Since part of the project is to make the model usable to general public, attention was focused on eliminating time and memory overuse.

5.5.1 Memory

GWVis focused on both memory and time efficiencies. To reduce the memory footprint of the tool, several techniques were implemented, such as efficient memory allocation. To find where memory allocation could be reduced, we looked at where there were overlaps in necessary information. Improving memory allocation allows the model to be used on computer systems that do not have a lot of system memory such as an older laptop.

To find where objects could be minimized we looked at where there were overlaps in necessary information. Several items in the model class were used either once and remained in memory, or used multiple times but stored for each model. Solutions to remove unnecessary memory use were to:

1. OpenGL index arrays for the aquifer and the terrain below it are shared between models. This is allowed due to the data being collected in the same coordinates for both models. Using an index array enables us to store the value at a grid vertex once instead of multiple times for each geometric shape.

2. Coordinates for X and Y positions in three dimensions are stored only once in the main model class. The monthly classes then contain the water elevation providing the Z component of the coordinates. This removed 2 floating point numbers for both the aquifer and terrain data, for all 181 months.

3. Color encoding of the elevation data was changed to be computed only when a new month was being rendered. This was accomplished this due to the speed of NumPy

array slicing, generating the colors for the entire month in 0.03 seconds.

4. A large array containing 7.9 million elevation data points to generate the color lookup table was updated to be a function scoped variable so that the memory it occupied could be released. This reduced the footprint of the model once initialized by 30.4MB.

5. Where possible, data was placed into vertex buffer objects (VBO) and removed from the model [17]. When this is done the data is stored as efficiently as possible and the model does not need to worry about how it is done. The information is still available when rendering the model.

By paying close attention of where the model is using memory and how the data is allocated in that memory space we were able to reduce the resource requirement to approximately 120MB. The eighty-three percent reduction in memory costs for the model was sufficient to allow us to continue working on other areas of the model. It may be possible to reduce resource usage even farther. However, there is typically a trade off between time and memory efficiency.

5.5.2 Time

In all there are 8.1 million possible elevation values resulting in 93MB of position data points to load, process, and render. A model for loading that much information has time bottlenecks that need to be resolved. Being written in Python already brings inherent efficiency issues. However, it is important for the model to be fast during animation and interaction. Without an environment that is fast the model's usability suffers greatly. Our first iterations of GWVis took several minutes to load the initial model and over a second to redraw each canvas during animation and simple rotation. These issues were addressed by using vertex buffer object, display lists, index arrays, and the use of NumPy slicing.

Vertex buffer objects, like display lists, place the data associated with rendering an image on the graphics card. Unlike display lists the data is then available to change and retrieve when needed. Data after being pushed to the GPU can then be removed from the program. More importantly for time efficiency however is the fact the data is already on the graphics card and accessing that data is incredibly fast. Display lists are also used in GWVis to save processing time. Originally all 181 months were placed into individual display lists. This had the desired effect of decreasing the amount of time it took to render the image; however, it had the undesired effect of doubling all memory usage since the display lists make a copy of data used for the list. The copy is usually placed on the graphic card. However once that memory has been exhausted, system memory is used, causing the application to use more system memory than previously calculated. Now, each month is placed into a display list and remains there while the user is interacting with it. Once a new month is started the previous display list is cleared and reestablished with the new month's data.

Another task that slows down the processing of the model is if the 172 x 256 element array needs to be iterated by row and column for all 181 months. In some cases this was found to slow the processing down by as much as 6 seconds. When possible, the application uses NumPy array slicing and masking to find the appropriate data point to access or set it. Usually when using a mask in NumPy a copy of the array is made and you cannot assign a value using array[mask] = value. However, it is possible to set values in the array while still using a mask. An example that shows the power of array slicing and the ability to set a value is as follows:

```
for i,upper in enumerate(self.parent.bins):
    mask = (self.Water[...,0] >= lower) & (self.Water[...,0]<upper)
    colorArray[...,0:3][mask] = self.parent.blue[i]
```

```
    colorArray[...,3] = 0.5
    lower = upper
return colorArray
```

In the code snippet a mask is set using the bounding limits for the histogram equalization for a particular bin. Next the lookup table for values within the bound specified by the mask (where ever the colorArray returns true for the mask) is set to a particular color of blue. This is done for the correct number of bins. This method sets a boolean array for the entire water array for each bin and uses that boolean array to select values in the color array. It would seem this would be inefficient to call do this for all data points for all bins. However, it is remarkably fast iterating a particular month in only 0.03 seconds.

CHAPTER SIX

EVALUATION

The goal of GWVis is to produce a presentation-ready visual tool that allows comparisons of ground-water flow simulations. This is accomplished by incorporating elements of DRO and SPO visualization and bridging the two methods. Key attributes that set GWVis apart are the ability to compare simulations, and the style of interaction that lends itself to presentations.

6.1 Comparison

Of all the visualizations mentioned in Chapter 3, none provide the capability to show a comparison between models. Many have the capability to run multiple simulations in the same workspace. However, this mechanism does not yield the same functionality as GWVis provides.

It is easier to come to a conclusion to a problem when the correct information is present in an easy-to-see format [9]. As seen in Figure 6.1, it is easier to sum a list of numbers using ruled paper rather than randomly placed on a page.

The significance of this is that GWVis presents difference visually so the information is apparent. Other visual tools rely on the observer to compare one simulation to the next by looking at both windows. Each canvas in GWVis has the same transforms applied to it so that the views are synchronized. DRO views however, are not synchronized. Keeping the view the same enforces the ability to compare simulations together, viewing areas of interest and only having to navigate to that area in one canvas.

Evaluation of simulation differences provides functionaly for GWVis that is useful beyond presentations. An analyst can run a MODFLOW simulation and compare its results

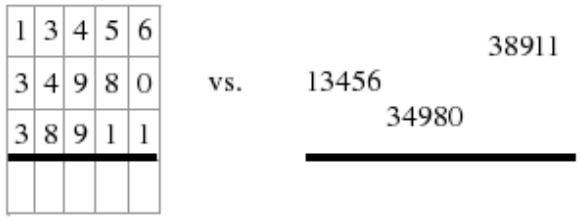

Figure 6.1: Addition Ease Comparison Concept from [9].

to field data, or to those of another simulation.

6.2 Presentation

GWVis is intended for use as a presentation tool in addition to basic analytic capabilities. The information shown in the visualization is kept minimal due to the main audience for the tool. When a visualization adds more information than is needed, or the information is hard to find and comprehend then the viewer will suffer from information anxiety [9]. This anxiety may lead to cognitive overload.

DRO models provide complex and powerful environments that allow analysis of flow simulations. Presenting a DRO model to the public or government and funding agencies will provide too much information for the purpose of telling a story about ground-water flow. There are buttons, labels, and capabilities that clutter the visualization. These are valid attributes of an analytic tool, however they can produce anxiety in individuals who are not trained in the environment.

SPO visualizations can be created in a way that minimizes information anxiety as GWVis does. However, the SPO models are unable to adapt their view to inquiries about specific areas of the aquifer. If the image was not created prior to the presentation, then the audience will be unable to view that requested information. GWVis can change its view

and focus in on areas of interest in an aquifer. The navigation is learnable so if desired the viewer can interact with the tool themselves.

CHAPTER SEVEN

CONCLUSIONS

In conclusion, GWVis has met the goals of the project. We visualize ground-water flow using a set of features that allow for analytic capability, while minimizing the effect of cognitive overload. Analytic capability is provided two ways. First, the rate of change canvases show a viewer how a simulation changes over time. The visualization include elements of ground-water, such as water elevation, river position, and underlying terrain, that are helpful in the decision making process. Rate of change allows a researcher to have an overall picture of what the aquifer is doing. Secondly, we provide a difference canvas which allows quick analysis of changes made in various simulations. Being able to compare two scenarios allows a researcher to analyze which changes to flow have affected a simulation in the ways they require for their study.

The visualization interface is kept simple, relying on user input to move around the model. Interacting with the model is developed in a way the users are able to learn quickly. Ease of interaction can be attributed to the use of standard mechanisms for input. While uncluttered with data, GWVis provides enough information to allow presentations and basic flow analysis.

We have adapted attributes of SPO and DRO visualizations and added a additional capabilities to produce a tool which allows comparison, is adaptable to MODFLOW scenarios, and is interactive.

CHAPTER EIGHT

FUTURE WORK

Not everything that can add benefit to the model could be added within the timeline of this project. Because GWVis is a research task of how to visualize ground-water in an intuitive way and provide basic analytic capability, new topics continue to be thought of that have yet to be explored. Some key areas that would be beneficial to the model include:

1. Close the gap between the bottom of the aquifer and the head. Currently when viewing the model, a user sees a layer of water elevation data. It would be more accurate to represent the area between the head of the aquifer and the bottom as volume of water.

2. Predefined fly-through routes. The capability of the model to move anywhere in the three dimensional space is already part of the project. It may be beneficial for display purposes to have a custom "flight path" file that would start the user in a location on the model and move without the need of user input.

3. Animation generation. By combining flight path animations, comparison animations, and text display, an overall animation could be created. This type of animation could be used for presentation, web site display, or as an additional media for information packages.

4. Addition of data dimensions. GWVis only has access to river, aquifer head, and aquifer bottom data. More data can be incorporated into the model such as the location of wells, active areas of the aquifer, and types of soil. It would benefit the model when done in a way that does not complicate the visualization further.

5. Integration with Google Earth. The aquifer information would be treated as a body of water and placed in the appropriate location. This would give the added benefit of utilizing functionality that is currently present in Google Earth such as satellite imagery.

BIBLIOGRAPHY

[1] Aquaveo. Aquaveo water modeling solutions. `http://www.aquaveo.com/gms`, 2008. Retrieved April 19, 2009.

[2] Gary Bradsky and Adrian Kaehler. *Learning OpenCV*. O'Reilly Media, Inc., Sebastopol, CA, USA, 2008.

[3] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman. *Readings in information visualization: using vision to think*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[4] P. Compieta, S. Di Martino, M. Bertolotto, F. Ferrucci, and T. Kechadi. Exploratory spatio-temporal data mining and visualization. *Journal of Visual Languages and Computing*, 18(3):p255 – 279, 20070601.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[6] Francis J. Hill. *Computer Graphics Using OpenGL*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[7] Paul A. Hsieh, Michael E. Barber, Bryce A. Contor, Akram Hossain Md., Gary S. Johnson, Joseph L. Jones, and Allan H. Wylie. Ground-water flow model for the spokane valley-rathdrum prairie aquifer, spokane county, washington, and bonner and kootenai counties, idaho. Scientific Investigations Report 2007-5004, U.S. Geological Survey, 2007.

[8] Mark J. Kilgard. *OpenGL programming for the X Window System*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1996.

[9] David Kirsh. A few thoughts on cognitive overload. *Intellectica*, 30, 2000.

[10] Hans Petter Langtangen. *Python Scripting for Computational Science*. Springer Verlag, Berlin, 2006.

[11] Shu-Guang Li and Qun Liu. A real-time, interactive steering environment for integrated ground water modeling. *Ground Water*, 44(5):p758 – 763.

[12] Mark Lutz. *Learning Python, 3rd Edition*. O'Reilly Media, Inc., October 2007.

[13] Michael G. McDonald and Arlen W. Harbaugh. The history of modflow. *Ground Water*, 41(2):p280 –.

[14] Travis E. Oliphant. Guide to numpy. `http://www.numpy.org`, December 2006.

[15] Travis E. Oliphant. Python for scientific computing. *Computing in Science and Engineering*, 9(3):10–20, 2007.

[16] Noel Rappin and Robin Dunn. *wxPython in Action*. Manning Publications Co., Greenwich, CT, USA, 2006.

[17] Dirk Reiners and Gerrit Voss. Climbing longs peak: The steep road to the future of opengl. *IEEE Computer Graphics & Applications*, 27(4):p84 – 89.

[18] Schlumberger. Visual modflow. `http://www.swstechnology.com/software_product.php?ID=29`. Retrieved April 19, 2009.

[19] Ken Shoemake. Animating rotation with quaternion curves. *SIGGRAPH Comput. Graph.*, 19(3):245–254, 1985.

[20] Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide*. Addison Wesley, fifth edition, July 2005.

[21] Julian Smart, Kevin Hock, and Stefan Csomor. *Cross-Platform GUI Programming with wxWidgets (Bruce Perens Open Source)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[22] Gary K. Speiran, Pixie A. Hamilton, and Michael D. Woodside. Natural processes of ground-water and surface-water interaction. Fact Sheet 178-97, U.S. Geological Survey, 1998.

[23] David Thompson, Jeff Braun, and Ray Ford. *OpenDX Paths to Visualization*. Visualization and Imagery Solutions, Inc., Missoula, MT, second edition, 2004.

[24] Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors. *Proceedings of the 7th Python in Science Conference*, Pasadena, CA USA, 2008.

[25] Groundwater Vistas. Groundwater vistas detailed description. `http://www.groundwater-vistas.com/gwv/detailed_description.php?products_id=43`, 2009. Retrieved April 19, 2009.

APPENDIX

APPENDIX ONE

DERIVATON OF THE NORMAL TO A LEAST SQUARES FIT PLANE

To find the normal first a plane is fit to the data, based on the equation of the plane the gradient is found. The normal follows from the gradient. Given that $z(x_i, y_i) = ax + by + c$ the equation of the plane can be found by finding the least squares:

$$\frac{\partial}{\partial \zeta} \sum (z_i - z(x_i, y_i))^2 = 0 \tag{A.1}$$

where

$$\zeta$$

can be a, b, or c

Equation A.1 can be simplified by moving the total derivative inside of the summation and removing a trivial factor of Z:

$$\sum (z_i - z(x_i, y_i)) \frac{\partial}{\partial \zeta} (z(x_i, y_i)) = 0$$

From $z(x_i, y_i) = ax + by + c$ we derive:

$$\begin{aligned} \frac{\partial z}{\partial a} &= x_i \\ \frac{\partial z}{\partial b} &= y_i \\ \frac{\partial z}{\partial c} &= 1 \end{aligned}$$

Substituting into the summation we get

$$\begin{aligned} \sum (z_i - z(x_i, y_i)) x_i &= 0 \\ \sum (z_i - z(x_i, y_i)) y_i &= 0 \\ \sum (z_i - z(x_i, y_i)) &= 0 \end{aligned}$$

Expanding out:

$$\begin{aligned} \sum (z_i x_i - a x_i^2 - b y_i x_i - c x_i) &= 0 \\ \sum (z_i y_i - a x_i y_i - b y_i^2 - c y_i) &= 0 \\ \sum (z_i - a x_i - b y_i - c) &= 0 \end{aligned}$$

Finally, the above 3 equations can be placed into matrix notation, moving the first terms of each equation over to the other side and multiplying by -1 to simplify. The following equation in the form of Ax = B is found. Solving for the values of a, b, and c the gradient f can be found.

$$\begin{bmatrix} x_i^2 & x_i y_i & x_i \\ x_i y_i & y_i^2 & y_i \\ x_i & y_i & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \sum x_i z_i \\ \sum y_i z_i \\ \sum z_i \end{bmatrix}$$

Using the gradient we can next find the normal to the calculated plane:

$$\widehat{n} = \frac{1}{\sqrt{a^2 + b^2 + c^2}} \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$