The Holy Grail of Adam's Ale

Locating Aquifers through Geostatistic Modeling

Team #65

April 7, 2010

New Mexico Supercomputing Challenge Final Report

Los Alamos High School

Team Members:

Gabriel Montoya
Rachel Robey
Orli Shlachter
Orion Staples

Teacher Sponsor:

Lee Goodwin

Mentors:

Robert Robey
Thomas Robey

Contents

List of Figures

Executive Summary

This project sought to develop a windows-based application to perform geostatistics, with a focus on its application to finding aquifers and other groundwater sources. Geostatistics is a branch of applied statistics used to calculate plausible values to fill the gaps in fragmented data sets. It depends on the idea of spatial correlation - that values located proximately are more likely to be similar. The application in hydro-geology was chosen because of New Mexico's dependency on groundwater; a case study was set up and real world data acquired from local boreholes.

This project encompassed an impressive feat of coding, incorporating C++, wxWidgets, OpenGL, and OpenCL. C++ is used for the computational and to interface with the user-interface done with wxWidgets. Visualization was done using OpenGL, and the beginning stages of optimization utilized OpenCL to run on the graphics processor.

While this project is not the first computer programming to be accomplished in the field of geostatistics, this project is a foundation for future studies. In the course of the year, several original algorithms were developed as well as integration of established methods, resulting in a working application that can reliably perform geostatistics of small problems. The case study on a real problem was partially successful and provided invaluable insights into future work.

1 Introduction

1.1 Problem Statement

A lack of complete data is a common problem faced across many fields of study. The solution is to estimate these unknowns, but making an accurate approximation becomes much more complex than simple means. The practice of this approximation is more difficult than the simplicity of the theory behind it. Geostatistics is a branch of statistics that can be used to make reliable predictions. It is based on the theory that data proximately located is more likely to be related. If the data is related then the unknowns can be approximated because the distance between data points would tell one how similar they should be.

Geostatistics is an effective method of filling in these "gaps" to logically create plausible data for the unknown points. This can be important especially in computer modeling where data is needed for every point, geostatistics can simplify the process.

The application of geostatistics to the discovery of aquifers was chosen because of New Mexico's shortage of water. Water is a very valuable commodity in the drier and more polluted regions of the world and the easier discovery of more water sources would help many people. To find new groundwater sources in such large expanses of land by testing every mile or every half mile would be difficult and inefficient. Using geostatistics to fill in gaps in the landscape will allow geologists and hydrologists to take far fewer samples and come up with more correct results. This will lessen the time and expense of finding groundwater sources, benefiting both economic and hydrological issues.

1.2 Objective

The purpose of this project was to write a windows-based program to perform geostatistics. The program was designed to approximate unknown values and show the detail of the terrain values in both color and height. There are many possible applications for geostatistics, and thus the usefulness of the program. The focus is on using a geostatistical model to find aquifers without taking inordinately large numbers of samples for a given area. To accomplish this goal a profusion of code had to be written for the many different facets of the program. The initial code to estimate the data using geostatistics was written in C++, the user interface was generated in wxFormbuilder (creating wxWidgets C++ code), and the graphics were rendered with OpenGL. The team wished to create a working program utilizing all three different programming languages that would accurately predict unknown values for a data set. These unknowns pertain to aquifer data so as to find more groundwater sources and alleviate problems in New Mexico and the rest of the world.

The program should reduce the time and money spent on geological surveying by a sizable margin and can be changed minimally to be applied to other problems.

1.3 Backround

1.3.1 Geostatistics

Geostatistics is a branch of statistics used to predict unknown values at specific locations, using the concept of spatially correlated data. That is, two values physically near each other are more similar

than two values farther apart. For example, in soil composition, samples taken closer together are more likely to be made up of similar minerals.

Geostatistics, originating in mining for the discovery of precious stones and metals, was first recognized as a reputable field theory in the 1960s in the French work "Theory of Regionalized Variables" which paved the way for inspirational work in the new discipline. Many changes were made to the math used in geostatistics and eventually it became applicable to many different fields besides mining. Now such employments as picture reconstruction and epidemiology are utilizing geostatistics[2].

To understand how geostatistics works one must understand the theories at the heart of the process. The Theory of Regionalized Variables states that it is possible to make a model of the spatial structure from known data and then use those known values to estimate the unknown ones[6]. The unknowns can be estimated because of the theory that data is spatially correlated. These are the underlying precepts behind geostatistics; the theories that make all others possible.

These postulates are used to determine the value of a given property in specific materials. This is done by applying the Theory of Regionalized Variables. There are two parts to regionalized variables:

- a *random* aspect, the unpredictable variation from point to point
- a *structured* aspect, the prevalent regional trend

The random aspect is the deviation from the normal that will throw off an approximation whereas the structured aspect is the normal trend which allows for the use of geostatistics in the estimations of unknown values.

Like most fields, there is some specialized language used in geostatistics. These terms will be defined as they appear, but are also described in the glossary (Appendix B on page 37)

1.3.2 Aquifers

The chosen application was locating aquifers. Backround information was needed to discover what characteristics to search for as an indication of an aquifer.

An aquifer is an underground layer of water-bearing permeable rock (Figure 1), which can be tapped by a well. The above diagram shows how the location of the water table is relative to the surface and to the surrounding geological points. An aquifer is a valuable commodity as a water source because, by definition, it readily transmits water to wells and springs. This means that it will be not be stagnant and undrinkable. Also because of the location of aquifers underground, the water cannot evaporate before its use. Unfortunately, aquifers are difficult to locate and can be contaminated. Aquifers are more likely to be closer to the surface because of the porous and permeable rocks there. Porosity refers to a rock's ability to retain water, while permeability is the capability of a porous rock to permit the flow of fluids through it[12]. The permeability and porosity generally decrease for larger distances from the surface since the cracks and fissures in a become diminished and close up as a result of the pressure of the rock overhead. However, this is not always the case and usable aquifers have been found in all surface depths[8].

More valuable results can be garnered from porosity and permeability because they are more reliable variables to use in a geostatistics model. This is because porosity and permeability more extensively affect the rock's ability to be a potable resource[8].

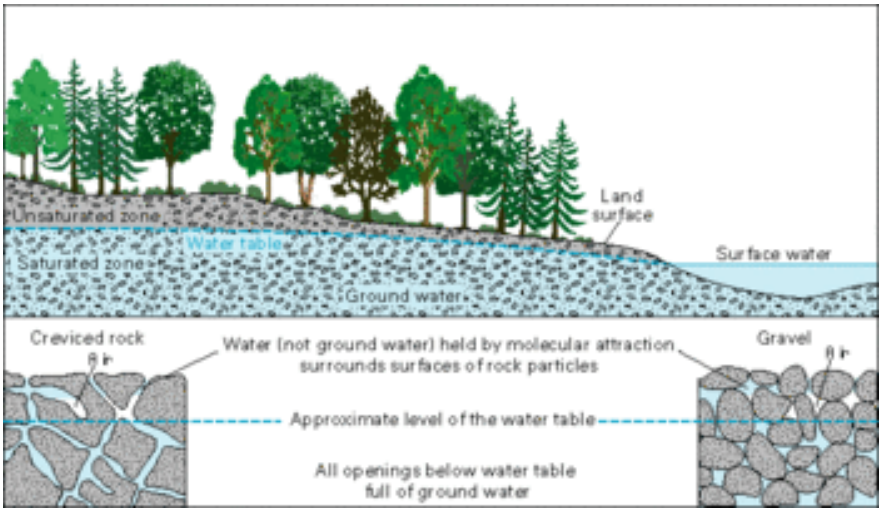

Figure 1: Diagram of an aquifer[8].

An aquifer can become contaminated as a result of human interference. Every time the water in an aquifer is used in a well, the level of the water table goes down and the water will be replaced with precipitation, known as recharge. The area in which an aquifer can benefit from the recharge is called a recharge zone. The larger an aquifer's recharge zone, the more wells can be drilled from it and the more often people can pump water for them. However, the larger an aquifer's recharge zone is the more opportunity for the aquifer to be contaminated. If an aquifer is contaminated it cannot be used thus creating a water shortage problem for its patrons.

Humanity's interference with the water table can also result in an aquifer's water pressure decreasing at an alarming rate[8]. If one were to pump too much water out of a well without allowing it its recharge period then surrounding wells could also go dry. This creates a problem for rural communities that depend on groundwater, making aquifers very sought-after resources[5]. Aquifers are a solution to the dire problem of acquiring water in the drier, more rural regions of the world. The case study of this project seeks to make them available by predicting their locations.

2 Mathematical Models

Geostatistics can be outlined with two main goals: to identify the spatial properties of the variable and to estimate gaps in incomplete data from the surrounding samples. These purposes are related, as characteristics of the spatial structure can be used to estimate unknowns. This is done by 1) constructing a semi-variogram, and 2) interpolating through the use of either inverse distance weighting or kriging.

2.1 Semi-variogram

The idea of spatial correlation discussed in the introduction to geostatistics is fairly intuitive. It makes sense that a value close to the unknown will be more similar to it than a value farther away. The semi-variogram is a way to quantify the variance in the values over space. It is fundamental to the idea of spatial correlation, and a crucial part of geostatistics.

The semi-variogram is unique for each material. Looking at many pairs of data at points about the same distance apart can provide an expected difference in value for a given distance. Described mathematically it is[2]:

$$\gamma * (h) = \frac{\sum [y(x) - y(x+h)]^2}{2n}$$

where $\gamma(h)$ is the semi-variogram[1] as a function of distance h between the data points,
$y(x)$ and $y(x+h)$ are the values at locations x, and x plus the distance h,
and n is the number of pairs of samples with distance h separating them.

These points are plotted with distance on the horizontal axis and semi-variogram on the vertical axis.

There are several specific terms associated with the semi-variogram. The distance at which the graph plateaus is called the range of influence, or simply the range. Any points farther apart than the range are completely uncorrelated, and thus are not helpful in accurately interpolating a value. The semi-variogram at that point is referred to as the sill. In experimental semi-variograms, it is possible that there will be a discontinuity at the origin called a nugget. In theory, this should not happen because the value at a point is equal to its own value; however, measurement errors and a random influence between the points can cause a nugget.

One of the things a semi-variogram can reveal about the data it represents is its isotropy. The material can be isotropic, meaning the spatial correlation is equal independent of the direction. If direction is an influence, it is anisotropic. Wood is a great example of this. There is a greater range so the relation extends much farther along the grain than against it.

There are several types of mathematical models which can be matched to the semi-variogram obtained from the data. A simple semi-variogram can be represented by a single type, but they can be combined for more complexity. Three of the most commonly used mathematical models are[2]:

Spherical

$$\gamma(h) = \begin{cases} C\left(\frac{3}{2}\frac{h}{a} - \frac{1}{2}\frac{h^3}{a^3}\right) & h \le a \\ C & h > a \end{cases}$$

Exponential

$$\gamma(h) = C\left[1 - \exp\left(-\frac{h}{a}\right)\right]$$

Gaussian

$$\gamma(h) = C\left[1 - \exp\left(-\frac{h^2}{a^2}\right)\right]$$

2.2 Interpolation

There are several different approaches to interpolating. Generally, this is done with the general equation for a weighted average shown below:

[1] The distinction between the *variogram*, $2\gamma(h)$, as opposed to the semi-variogram is important, and not always clear if semi-variograms are inattentively called just variograms.

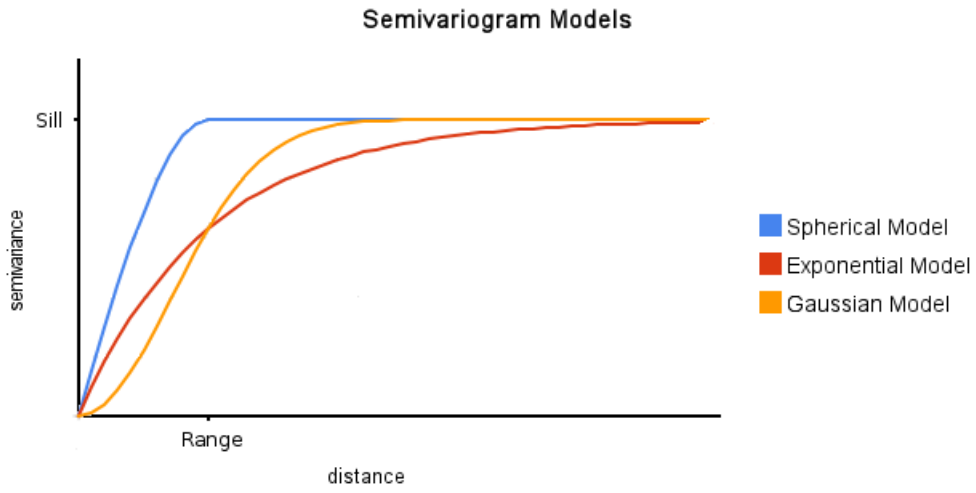

Figure 2: Example of the three common types of mathematical models for the semi-variogram with the same range/sill.

$$F(x,y) = \sum_{i=1}^{n} w_i f_i$$

which basically says that the value of the unknown point is a summation of the values of the points it is being interpolated from times its weight. The differences occur in determining the weights.

2.2.1 Inverse Distance Weighting

One of the simplest methods is Inverse Distance Weighting (IDW). The weights are purely dependent on the distance. The inverse of the distance for each of the points within range is found. If a point is 4 units from the unknown, it would be $\frac{1}{4}$. Then they must be summed and scaled to one. So the weight of each point is:

$$w_i = \frac{d_i^{-1}}{\sum_{j=1}^{n} d_j^{-1}}$$

where d is the distance for each point i of n points. The distances can be scaled to provide larger weights in some directions than others as determined in the semi-variogram this is discussed in more detail in Section 3.3. This method works fairly well for its simple approach and is a good comparison method to the kriging.

2.2.2 Kriging

Kriging is an interpolation method unique to geostatistics. It works by finding the "best" estimate. An explanation of how this is done will be given in terms of an unknown value T at a point A, as adapted from *Practical Geostatistics*[2].

If the value at the closest point is used as an estimation of T, it will incur an estimation error ε which is a measure of the difference between T and the estimated value $T*$:

$$\varepsilon = \mid T - T* \mid$$

Assuming there is no local trend, as the number of estimations increases towards infinity, the average error will approach zero. So, theoretically:

$$\overline{\varepsilon} = 0$$

The reliability of an estimator is rated by the spread of the errors. A 'good' estimator has errors consistently close to zero. If they range more widely, than the estimator is unreliable. The spread can be measured by the standard deviation of the estimation error - the standard error.

Consider the definition of standard deviation σ, the square root of the variance.

$\sigma^2 = \dfrac{\sum (X - \mu)^2}{N}$, the average of the squares of the difference from the mean. In case of the variance of errors, it follows that:

$=$ average of $(\varepsilon - \overline{\varepsilon})^2$

$=$ average of ε^2, since $\overline{\varepsilon} = 0$

$=$ average of $(T - T*)^2$

It is impossible to calculate these values directly since the actual value is unknown. A closer look at the definition provides a solution. The value of the point closest to point A is used for the estimator, and should vary from the actual value dependent on the distance from A. This expected difference is described by the variogram exactly, the average of the squared differences. Thus, the mathematical semi-variogram chosen to represent the spatial structure can be used to estimate the difference (multiplying by two yielding the variogram). So, finally, the variance of the errors is:

$$\sigma_\varepsilon^2 = 2\gamma(h)$$

As the estimate grows more complex with the addition of other points, the variance of errors is given as a weighted average of the variogram of each of them.

$$\sigma_\varepsilon^2 = 2\sum_{i=1}^{n} w_i \bar{\gamma}\,(P_i, A)$$

$\bar{\gamma}\,(P_i, A)$ is the average semi-variogram, as defined by the mathematical curve, between interpolation point P and the point being estimated, A. Kriging is unique in that it directly seeks to fid the 'best' estimate - that having the smallest estimation variance. The only values free to be altered are the weights of the weighted average, so the estimation variance is being minimized with respect to the weights. A minimum can be found by setting the differential equal to zero (i.e. the slope is zero as in Figure 3):

$$\frac{\partial \sigma_\varepsilon^2}{\partial w_i} = 0 \quad i = 1, 2, 3, 4...n$$

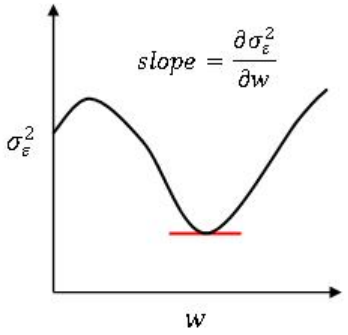

Figure 3: The differential of the variance of error with respect to the weights. Kriging seeks to minimize this variance to find the 'best' estimator.

While this will provide weights for the desired minimum, the sum of the weights must also be explicitly set to one so the related estimator will be a whole.

$$\sum w_i = 1$$

This added constraint consequently over-determines the system of equations - n weights, or variables, and $n+1$ equations. A Lagrangian Multiplier is introduced to balance this out. Rather than simply the estimation variance to be minimized, it is the term:

$$\sigma_\varepsilon^2 - \lambda\left(\sum w_i - 1\right)$$

When the sum of the weights is equal to one, $\sum w_i - 1 = 0$, thus nullifying the λ and taking the smallest variance as defined. The expanded system equations for three points is defined as:

$$w_1\overline{\gamma}(P_1,P_1) + w_2\overline{\gamma}(P_1,P_2) + w_3\overline{\gamma}(P_1,P_3) + \lambda = \overline{\gamma}(P_1,A)$$

$$w_1\overline{\gamma}(P_2,P_1) + w_2\overline{\gamma}(P_2,P_2) + w_3\overline{\gamma}(P_2,P_3) + \lambda = \overline{\gamma}(P_2,A)$$

$$w_1\overline{\gamma}(P_3,P_1) + w_2\overline{\gamma}(P_3,P_2) + w_3\overline{\gamma}(P_3,P_3) + \lambda = \overline{\gamma}(P_3,A)$$

$$w_1 + w_2 + w_3 = 1$$

On the left side of the equation are the weights times the variance between each point with each other point, and on the left is the variance between said point and the unknown. The system of equations follows the same pattern for any number of points used to interpolate. Solving these equations is a computational problem addressed in Section 3.2.

2.3 Sampling from Gaussian Distribution

If the goal of the interpolation is not to be accurate but to generate a sample set of data to be used, than it may be desired to address the random aspect of a regionalized variable. To give the impression of a degree of random variance in each point, a Gaussian Distribution is randomly sampled.

The interpolation gives the mean of the distribution, or the most likely value, but the value is free to vary from this. The magnitude of this variance is based on the variance of the unknown to the closest point as determined from the semi-variogram. This is more of a stylistic choice, and results in unknowns interpolating from farther afield points having a larger random element. This sampling is set up as the mean (μ) - or most likely value - as being the

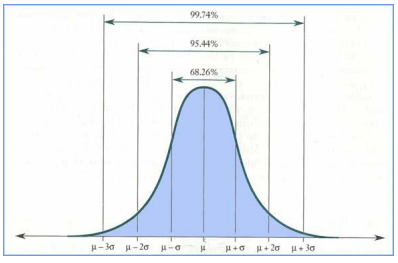

Figure 4: 'Normal' Gaussian Distribution. The mean is the interpolated value.

interpolated estimator. The standard deviation
(σ) is the square root of the variance found from the closest point. Values closer to the mean are more likely to be sample because of the bell shape of the curve, and the smaller the variance the smaller the range of possible values.

3 Computational Model

Since our program is windows based, the computational method is much more segmented. Each part is driven by user generated events (e.g., clicking buttons, typing).

After the data is input, there is a vector array of the grid points whose values are unknown. The user can interact with a plot to create a semi-variogram from the known data.

3.1 Semi-variogram

The program must calculate both experimental and mathematical semi-variograms. The experimental semi-variogram is somewhat simplified because the data is already broken up into equally spaced points. It did not take long to develop the basic method that was used. For convenience, the equation will be repeated here.

$$\gamma(h) = \frac{\sum [y(x) - y(x+h)]^2}{2n}$$

Part of the process is actually simplified because the data is already divided into equally spaced points. Each row or column begins at the first element and "jumps" over h points. If the values at both points are known, the semi-variogram is calculated.

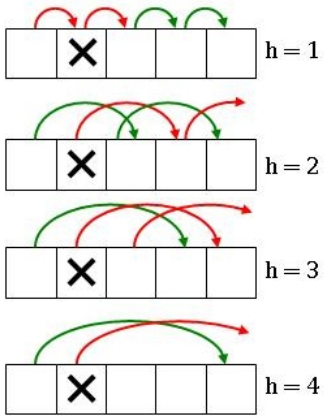

Figure 5: Computationally finding experimental semi-variogram.

This process is continued by incrementing up the row by one until there are no longer enough elements to skip h. Then h is increased.

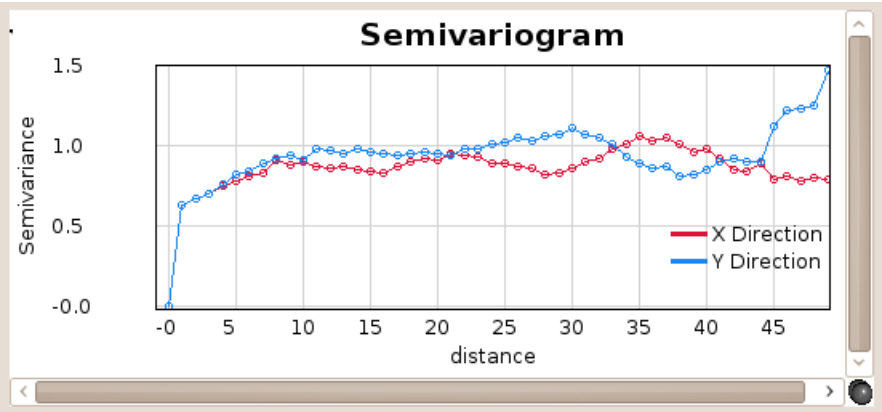

Figure 6: Screen capture of the plot section of the window showing an optimal experimental semi-variogram, created from sample data.

3.2 Solution of Kriging Equations

The kriging equations involved a more complex solution than IDW, using matrices to solve for all of the unknowns in the system. The added constraint on the sums makes the system of equations overdetermined, meaning there are more equations than variables. QR decompositions are a common solution for least squares problems with over-determined systems of equations[3].

QR factorization of the coefficient array was used to solve for the weights, and was computed using Givens rotations. QR factorization of a square matrix $A \in \mathbb{R}^{n\times n}$[2] is given by $A = QR$, where Q is orthogonal and R is upper triangular. The definitions of these special types of matrices are as follows:

A square matrix $Q \in \mathbb{R}^{n\times n}$ is orthogonal if $Q^TQ = QQ^T = I_n$, meaning its inverse is also its transpose[3]. An upper triangular matrix, also called right triangular, is also square ($n \times n$), with all the entries below the main diagonal zero:

$$U = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ 0 & 0 & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & u_{n-1,n} \\ 0 & 0 & 0 & 0 & u_{n,n} \end{bmatrix}$$

Givens rotations can be used for selectively zero elements, and calculate the decomposition of a matrix into its Q and R factors. Multiplication by a rotation matrix[4] performs a rotation in Euclidean space. To visualize a matrix geometrically, consider each column to be a set of coordinates to define the location of a point. Multiple points create a set of columns, a matrix, with each row the coordinates in the same dimension. So the matrix $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}^T$ performs a counterclockwise rotation of an angle θ about the origin of an x-y-plane - or alternatively viewed as the rotation of the coordinate system axes in the opposite direction. A rotation can be performed on a larger scale by expanding the previous rotation matrix to:

$$G(i,j,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cos\theta & \cdots & \sin\theta & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -\sin\theta & \cdots & \cos\theta & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

which is an identity matrix with the following substitutions: $g_{ii} = \cos\theta$, $g_{ij} = \sin\theta$, $g_{ji} = -\sin\theta$, $g_{jj} = \cos\theta$. The only rows affected are i and j, the others will remain the same, and thus may be

[2] This denotes the vector space of all real n-by-n matrices, essentially saying any matrix with the given dimensions.
[3] The transpose of a matrix is denoted with a superscript T.
[4] It should be noted that rotation matrices are orthogonal and have a determinant of one.

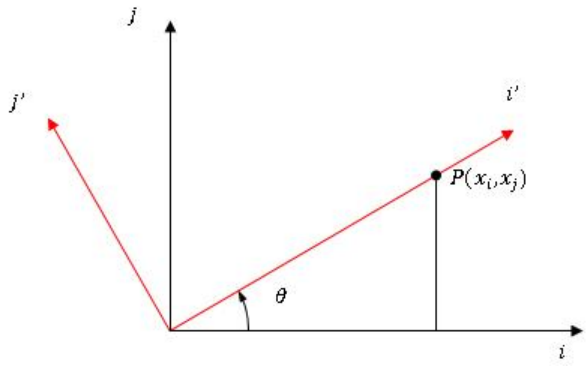

Figure 7: Illustration of concept behind Givens Rotation. The i axis is rotated so as to make point P lie upon it, zeroing its j' coordinate.

ignored.

A Givens rotation sets the angle to rotate the axis so the selected point lies on it, zeroing out the value of the other coordinate. If $x \in \mathbb{R}^n$, and $y = G(i,j,\theta)^T x$, then, by matrix multiplication:

$$y_k = \begin{cases} x_i \cos\theta - x_j \sin\theta & k = i \\ x_i \sin\theta + x_j \cos\theta & k = j \\ x_k & k \neq i, j \end{cases}$$

y_j can be forced to be zero when it lies on the perpendicular axis. Figure 7 illustrates the right triangle created by the previous axis and the new one that should run through point $P(x_i, x_j)$. This creates the desired angle of rotation. By directly using the definitions of the trigonometric functions used in the rotation, calculation of θ can be bypassed entirely. The Pythagorean Theorem ($c^2 = a^2 + b^2$)gives the length of the hypotenuse, and the sides of the triangle are known from the coordinates.

$$\cos\theta = \frac{adj}{hyp} = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}$$
$$\sin\theta = \frac{opp}{hyp} = \frac{-x_j}{\sqrt{x_i^2 + x_j^2}}$$

Substitute these definitions into the expression for y_j:

$$x_i \sin\theta + x_j \cos\theta$$
$$\frac{-x_j}{\sqrt{x_i^2 + x_j^2}} x_i + \frac{x_i}{\sqrt{x_i^2 + x_j^2}} x_j$$
$$\frac{-x_j x_i + x_i x_j}{\sqrt{x_i^2 + x_j^2}} = 0$$

And they result in zero because the first term of the numerator has a negative, making them additive inverses. This only happens in the specific case that was intentionally set up.

These Givens rotations can be used to find the QR factorization used to solve a matrix equation $Ax = b$. Each element of the coefficient matrix A where i is greater than j would be zeroed, creating an upper triangular matrix.

From the definitions of sine and cosine and the Pythagorean Theorem, they can be found as:
$\cos\theta = \frac{j}{h} = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}$ and $\sin\theta = \frac{i}{h} = \frac{-x_k}{\sqrt{x_i^2 + x_j^2}}$

Now the matrix equation has been put into the form $Ux = b$, it can be solved using back substitution. For a 2-by-2 example:

$$\begin{bmatrix} u_{1,1} & u_{1,2} \\ 0 & u_{2,2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

The bottom row has only one unknown, since the other variable is zeroed, and can be solved directly. Once that value is known, it can be substituted up into the next row so there is again only one unknown which can then be found. Starting at the bottom row, the unknown x's can be solved for sequentially:

$u_{2,2}x_2b_2$ is solved algebraically to be $x_2 = b_2/u_{2,2}$

$u_{1,1}x_1 + u_{1,2}x_2 = b_1$is $x_1 = (b_1 - u_{1,2}x_2)/u_{1,1}$

This back substitution can be represented by[3] :

$$x_i = \frac{b - \sum_{j=i+1}^{n} u_{i,j}x_j}{u_{i,i}}$$

This process provides the values of the weights in the weighted average. The corresponding code for a coefficient matrix A, right hand vector B, andmatrix size n, as based off of the psuedo code in *Matrix Computations[3]* is given in Listing 1. The solutoin is stored in vector x.

Listing 1: Matrix Solver using Givens Rotations

```
void GeoData::QRGivens(double **A, double *B,int n, double *x) {
    double c, s, tau, tau2;
    int i, j, k;
    for(j = 0; j < n; j++) {
        for(i = n-1; i > j; i--) {  //loop to zero all elements in lower triangle
            /* Determine variables for rotation */
            if(A[i][j] == 0.0) {    //already 0 - don't change
                c = 1.0;
                s = 0.0;
            }
            else if(abs(A[i][j]) > abs(A[j][j])) {
                tau = -A[j][j]/A[i][j];
                s = 1.0/sqrt(1.0+SQ(tau));
                c = s*tau;
            }
            else {
                tau = -A[i][j]/A[j][j];
                c = 1.0/sqrt(1.0+SQ(tau));
                s = c*tau;
            }
```

```
            for(k = 0; k < n; k++) {     //perform rotation on elements in two
                affected rows
                tau = A[j][k];
                tau2 = A[i][k];
                A[j][k] = c*tau - s*tau2;
                A[i][k] = s*tau + c*tau2;
            }
            /* Same rotation done on right hand solution vector */
            tau = B[j];
            tau2 = B[i];
            B[j] = c*tau - s*tau2;
            B[i] = s*tau + c*tau2;
        }
    }
    /* Back substitution stores solution in x */
    for(j = n-1; j>=0; j--) {
        x[j] = B[j];
        for(i = n-1; i > j; i--) {
            x[j] -= x[i]*A[j][i];
        }
        x[j] /= A[j][j];
    }
}
```

3.3 Anisotropy

Anisotropy is the property of being directionally dependent (as opposed to being isotropic). In geostatistics, this means having different spatial correlation in different directions. Wood is a good example of this characteristic. It is evident, even to the human eye, that it has a higher degree of correlation along the grain than against it. This might not be so apparent in the various qualities of different substances. Therefore, anisotropy must be identified using the semi-variogram. The plots of the experimental semi-variograms for the two directions will have different ranges of influence.

The solution is to make the ranges appear to be the same. This means adjusting the measurements so that one 'unit' in the semi-variogram may be 5m horizontally and 25m vertically.

The calculations rely on a distance method to calculate how far one point is from another. This is simple since they use a coordinate system. The distance formula is:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

If the formula is modified, it can account for the scaling of measurement. If s is the scale factor, in the form of a decimal percent (i.e. to have half the range, s would be 0.5), then the new formula can be expressed as:

$$d = \sqrt{[s_x (x_2 - x_1)]^2 + [s_y (y_2 - y_1)]^2}$$

The magnitude of the distance in each direction is scaled before the rest of the distance formula is performed. This scaling effect has also been applied to relative distances in the two directions. If it is one foot from one cell to the next horizontally, but two feet vertically, than the ratio is included

in the scale to make it a default of isotropic - the point above the unknown really is twice as far away as the one to the right of it.

3.4 Sampling from Gaussian Distribution

The Gaussian Distribution was fairly simple to set up with the use of a method from *Numerical Recipes in C*[9]. Once the interpolated value is set, the nearest point is found by searching through the list of points in range and the variance computed for the distance. The random sample returned from the method can be applied to the specific case.

```
double a, b;
a = estimate.getValue();
b = GetVariogram(estimate, GetClosestPoint(estimate));
return a + gaussRandom()*b;
```

The method gaussRandom() returns a sample from a Gaussian Distribution with a zero mean and unit variance. Multiplication by b stretches the curve horizontally and adding a shifts it horizontally.

Since this sampling was not important to the focus of this project, it has not been tied completely into the user interface: there is not an option in the window for it. The code needed to compute it is completed, but the method to apply it to each of the unknowns by choice of a user was not updated with the other code changes.

3.5 Multiple Points and Runs

Thus far, the discussion has been limited to the interpolation of a single point. To find all of the unknowns, they are ordered randomly to be calculated. This sequence is important because the points are interdependent - once an estimated value is found it is used in the interpolation of others.

Because of the random element of order, the answers will vary between runs, making it necessary to run multiple times. This repetition and integration of results, in the form of a mean, has been automated.

An algorithm had to be developed to randomly iterate through an array of the unknown points. Each element is a structure which contains the index of the unknown and a sum of the answers - which can then be divided by the number of runs to find the mean.

This is all done in a single array. A random index between zero and the maximum size of the array is chosen. Once the interpolation has taken place and added to the sum, the element is switched with the last element. The next random index is chosen, but this time excluding the final element which has already been found - that is between zero and one less

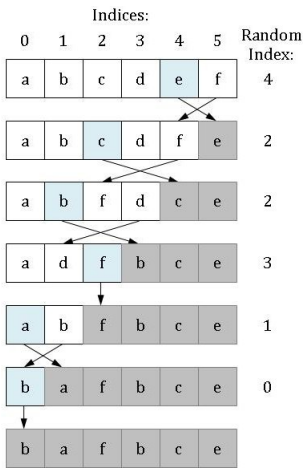

Figure 8: Algorithm developed to randomly iterate through unknowns. Elements are swapped to the back part of the array as they are randomly selected from the front part.

than the array size. This next element is switched
with the second to last element, creating a segment at the end of the array of the elements that have already been used. The next random element is searched for between zero and the maximum minus the number of completed elements.

This algorithm works fine until the problem of failures to calculate the points arises. If there are no points within the range of the current unknown, it has nothing to interpolate from. It must be skipped until later when more points have been calculated.

3.6 Optimization

Large scale problems are the norm in practical computing, calling for faster calculations to curb the lengthy run-times. Due to the sectionalized nature of the code, these beginning stages of optimization have been done within the separate methods. Later work may attempt to make this more streamlined in order to further improve the speed.

3.6.1 Variation on Random Iteration Algorithm

Though it is not necessarily characteristic of large problems, sparse data can become computationally expensive in the random iteration through the unknown points. In the original method outlined on Section 3.5, if a point fails - that is there are no points in range from which to interpolate - the unknown is left where it is in the array and the count of finished elements does not increase. This works when there is ample data and these types of points are rare. Another point is picked and eventually it will be successfully calculated as the points around it are found.

In a set of sparse data, however, there are so many points with nothing in range that the random index could continually hit these points and no progress would be made. An alternative algorithm was created, which, while it does not ensure equal chances in determining the random path, prevents already failed elements from being selected again before more points are filled in. This prevents repetitive sampling which may waste computation time.

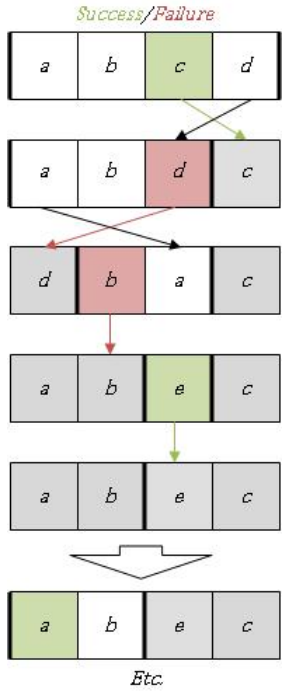

Figure 9: Variation on the random iteration algorithm, grouping 'failed' elements in the front to be retried after all the others have been iterated over.

This was accomplished by setting off elements at the beginning of the array, similar to the one on the end. If the calculation of the unknown at a random index is successful - that is there were points in range to interpolate from - it is swapped into the back section of the array. If it fails,

it is swapped into the front. Random indices are always selected from between the two boundaries of the separated sections by finding a random number for the range and offsetting it from the front:

index = rand() % (size−i−front) + front;

Once the boundaries converge, leaving no unknowns in the middle which have not been iterated over, the front boundary is reset to the beginning of the array. This puts all the previously failed elements in the middle to be retried. This continues until all the elements have been successfully completed and moved to the back - or if the boundaries converge with no new successfully completed unknowns, indicating the elements that are left are impossible to calculate with the current parameters.

The variation on the algorithm is implemented in the following fashion:

Listing 2: General Use Implementation of Variation of Random Iteration Algorithm

```
success = 0;
front = 0;
size = array.size();

for(i = 0; i < size; i++) { //iterator to determine when all elements are completed
    if(front == size - i) { //check if the boundaries have converged
        if(success < 1) //if there have been no successes here, the rest are impossible
            send error
        front = 0   //reset boundary and success count
        success = 0
    }
    index = rand() % (size-i-front) + front //in C++, find random index between
        boundaries
    if(calculation returns successful) {
        swap array[index] with array[size-i-1]
        success++
    }
    else {  //if the calculation failed
        swap array[index] with array[front]
        i-- //compensate for automatic increment of i, it wasn't successful
        front++ //move up boundary of front section
    }
}
```

Since there are potential consequences in the order of the random path, this alternative is only used with the selection of a sparse data option in the application window.

3.6.2 Transferring Matrix Solver to the GPU

The Givens Rotation QR decomposition was parallelized using OpenCL to send it to the Graphic Processor Unit (GPU). Large problems continued to use inordinate amounts of time without reaching completion. A smaller sized test problem was run to identify which methods were most computationally expensive. A significant 55.56% of the total time was spent in the QRGivens method (the solver for the kriging equations), which was also much larger than the next largest at 7.41%. This made the matrix solver a clear target for optimization.

The speed-up gained by operating on a GPU is mainly due to to the parallel computation - the same process is done on multiple data elements simultaneously[7]. The speed of calclulations on the GPU make loading data the primary concern. The coefficient and right hand vector of the matrix equation are passed in globally. For every rotation, only two rows are affected, as discussed in the description of Givens Rotations (Section 3.2) . Thus the rows can be loaded into local memory by

pairs, providing faster access.

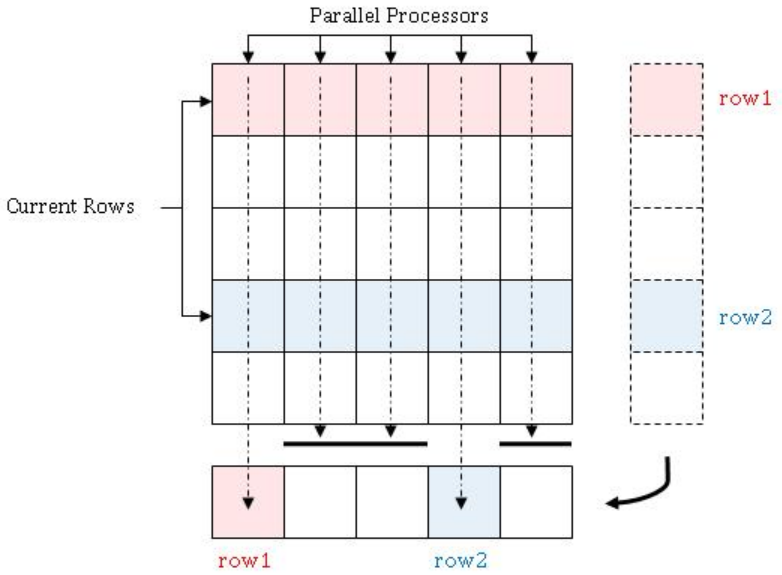

Figure 10: Break down of Givens Rotation to be done in parellel on the GPU where rows 1 and 2 are those affected by the current rotation.

Parallelization can only be used where the values are independent of eachother. Once the coefficients of the rotation are found, they can be applied to each of the elements individually - that is in parallel. This break down is shown in Figure 10. The current kernel implementation is shown in Listing 3.

Listing 3: OpenCL Givens Rotation Kernel

```
#define A(j,i) Coeff[j*npadded+i].s0
#define SQ(a) ((a)*(a))
__kernel void QRGivensGPU_kern(
    const int npadded,
    const int n,
    __global float2* BX,
    __global float2* Coeff,
    __local float* row1,
    __local float* row2,
    __local float* B)
{
    int giX = get_global_id(0);
    int tiX = get_local_id(0);
    int ngX = get_global_size(0);
    int ntX = get_local_size(0);
    float c, s, tau, tau2;
    int i, j, k;
    if(giX < n) {   //Only for processors that were not added for padding
        B[giX] = BX[giX].s0;
        for(j = 0; j < n; j++) {
            row1[giX] = A(j,giX);   //Each processor sets value in row1 from
                corresponding A
            for(i = n-1; i > j; i--) {
```

```
                row2[giX] = A(i,giX);
                barrier(CLK_GLOBAL_MEM_FENCE);  //Force wait until all processors
                    are done
                /* Set rotation values */
                if(row2[j] == 0.0f) {
                    c = 1.0f;
                    s = 0.0f;
                }
                else if(fabs(row2[j]) > fabs(row1[j])) {
                    tau = -row1[j]/row2[j];
                    s = 1.0f/sqrt(1.0f+SQ(tau));
                    c = s*tau;
                }
                else {
                    tau = -row2[j]/row1[j];
                    c = 1.0f/sqrt(1.0f+SQ(tau));
                    s = c*tau;
                }
                /* Perform rotation on each element - parallel */
                tau = row1[giX];
                tau2 = row2[giX];
                row1[giX] = c*tau - s*tau2;
                row2[giX] = s*tau + c*tau2;
                /* Perform rotation on appropriate elements of solution vector */
                if (giX == i || giX == j) {
                    tau = B[j];
                    tau2 = B[i];
                }
                barrier(CLK_GLOBAL_MEM_FENCE);
                if (giX==j) B[j] = c*tau - s*tau2;
                if (giX==i) B[i] = s*tau + c*tau2;

                A(i,giX) = row2[giX];   //Put updated values to the coefficent array
            }
            A(j,giX) = row1[giX];
        }
        /* Back substitute, putting answer into second vector component of BX */
        barrier(CLK_GLOBAL_MEM_FENCE);
        for(j = n-1; j>=0; j--) {
            for(i = n-1; i > j; i--) {
                B[j] -= B[i]*A(j,i);
            }
            B[j] /= A(j,j);
        }
        BX[giX].s1 = B[giX];
    }
}
```

4 Code

4.1 Overview and Structure

This is a large project and, especially with the GUI, there is a significant amount of code with a complex structure. There are several key parts that will be discussed in more detail in the next sections. This project was written primarily in C++, but also incorperated several different packages: wxWidgets (and its add-ons), OpenGL, and OpenCL.

The wxWidgets code which creates the windowing is generated using wxFormBuilder, but it is not used directly. In fact, the programmer should not hand edit it at all. Instead, a child class is created, inheriting the frame design and objects within in. It is in this class that the methods called on each event are implemented. The role of each class will be clarified with a description of their place in the structure and their methods.

AdamsAleAppGui is the file generated by wxFormBuilder, and actually contains several classes for the frame and each of the dialogs. Each one creates the window with the layout as designed in wxFormBuilder, but nothing is functional.

AdamsAleApp is the "main" class in the application. It calls the constructor of the frame displays it, and sets up the continuous rendering. Though these are its only tasks, they are important because external code (from the windowing) is needed to initialize the application.

AdamsAleAppFrame is the class inherited from the frame produced by the generated code. While the parent class has the layout, it is this class's job to fill in designated spaces such as the GLCanvasPane. It makes the original design functional by animating the controls, that is, defining what should be done for different user inputs. The frame has access to all of the objects within it, so it can retrieve data from inputs, display values, and call the methods of more complex objects (GLCanvasPane, PlotCtrlPane). The "command events" generated by clicking buttons, selecting menu items, etc., are directed to here. In wxFormBuilder, corresponding methods for each event can be set. These methods are defined here, generally calling on more specific methods in other classes. Otherwise, it calls dialogs and has all the file I/O.

GeoData was developed later to hold all of the variables and methods on the computational side of the application. It is purely computational, with no references to any of the user interface. When a new model is created or a file opened, an instance of the class is created and values passed in. The frame can then call any of the methods: from the semi-variogram to interpolating points.

GLCanvasPane is the pane in the window reserved for the visualization. An instance of this class is created as part of the constructor of the frame. The pointer to the data class is passed into this class after the unknowns have been interpolated so it has access to the values of each point.

PlotCtrlPane comes from the wxWidgetsAddition wxPlotCtrl with additional methods for its specific use in plotting semi-variograms.

The sizes of these classes may be estimated by the number of lines in each of their files[5]:

File	Line Count
AdamsAleApp	93
AdamsAleAppFrame	432
AdamsAleAppGui	763
DataPoint	65
GeoData	460
GLCanvasPane	301
PlotCtrlPane	184
Total	**2311**

There is an immense amount of code, and it cannot be recorded here in its entirety; major sections are included in pertinent sections. It is all online in the repository used during its development and can be viewed at: `http://code.google.com/p/adams-ale/source/browse/#svn/trunk/AdamsAle`.

4.2 WxWidget Windowing

The windowing and user interaction were a significant part of the program. Since there are so many different options in geostatistics, this format allows a user to choose the method that best fits the specific problem. The majority of the wxWidget C++ code was generated with wxFormBuilder, saving the time needed to write out the simple code by hand. The rest of the code has been integrated into this main application window as functions.

Figure 11 is a screen capture of the workspace in which the user interface is created. On the left is the hierarchy in which the elements of the page can be easily arrangedwithout disturbing the rest of the window. In the middle is a preview of the window the programmer is creating. On the far right is the "properties and events" window. The "properties" tab allows the programmer to set the size, labels, and properties of the object that was just created, while the "events" tab lets the programmer set the methods for the object. For example, the programmer can write "OnMouseClick" enable that object to be used when it is clicked on by a mouse by adding matching code later on. This object can now be used because it has a function. WxWidgets is used for creating the windowing in which a the program can operate. This tool allows the programmer to create a window with relative ease. It creates most of the 'cosmetic' code, while it only requires the programmer to write the code to animate the controls. This eliminates a lot of lines a programmer must write by hand. wxWidgets is also very useful for programmers using more than one platform as it is virtually the same on a PC and Mac - though this has not been attempted with this project . Tabbed windows allow more information to be seen on the same page. Using wxWidgets along with wxFormBuilder was a good choice because of the relative ease. because of the relative ease of using it. In wxFormBuilder, the programmer starts with a frame, which is a basic window. Then a sizer is added. Sizers organize the window and lay out the graphs and similar objects. After a sizer there are virtually endless possibilities from which the programmer can choose: toolbars, graphs, data tables, and graphics panes to be filled by other programmers.

[5]Line counts are from a single point in time and will vary a little as changes are made. Header files are included in the count.

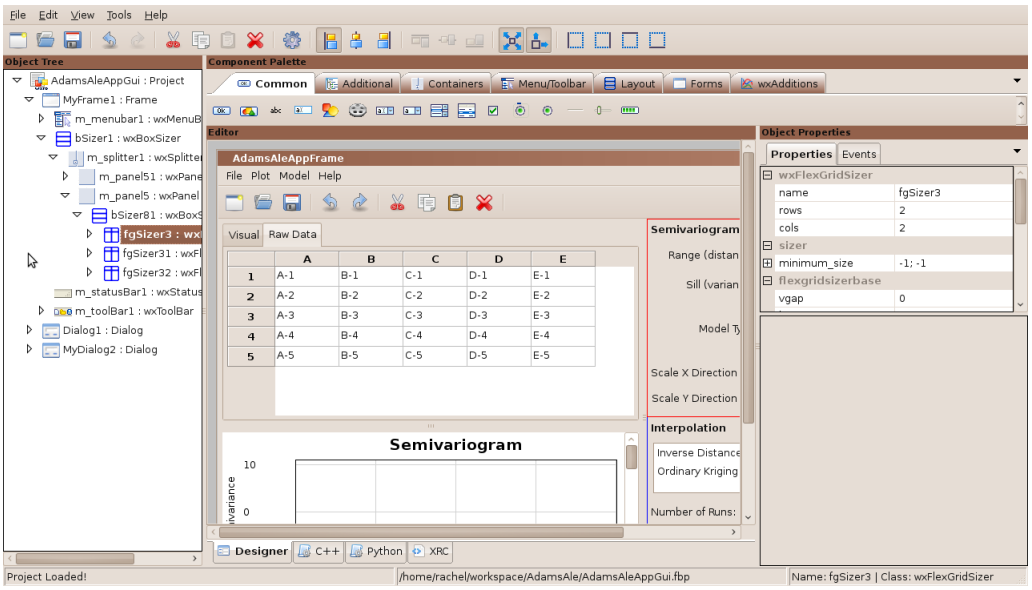

Figure 11: wxFormBuilder workspace

There are four different types of sizers: box, staticboxes, grids, and flexgrids. BoxSizers are the most common and ideal choice for a basic window. They space out the objects that are added to it equally, placing them either vertically or horizontally. A box sizer was used for the left hand side of the window. Static boxsizers are the basic variations. GridSizers are important because they are contain graphs and charts. The bottom and top parts of the window have gridsizers. All of the rows and collumns must be the same size. FlexGridSizer sizers are grid sizers with the exception that the programmer can manipulate the dimensions. In this application a FlexGridSizer is used to display the information on the right hand side of the window. Another important aspect utilized was sashed windows. Sashed windows allow the window to be stretched making it larger or smaller. For example, if there are seperate pieces of information such as a graph and dataset, the sash enables the user to enlarge one of the the two to see more of the other. wxFormBuilder alows the programmer to create buttons and windows with ease. The program writes all of the 'cosmetic' code while the programmer creates the little working code.

The user interface was designed to be very intuitive. The process starts with importing data from a specified borehole. This data is placed in a chart at the top of the screen. It is placed with two columns on opposite sides of the chart. This data could be porosity, amount of radiation, or resistivity, depending on the data from the borehole. Once the data has been imported into the chart, the user chooses the method of interpolation. After the method is chosen, the semi-variogram is plotted on the PlotCtrlPane graph below the chart. There is a key on the right side of the graph. In the right vertical portion of the window are many options for formatting the graph. The user can select from three different mathematical models: gaussian, spherical, and exponential. Once the user chooses his model the program will generate the plausible points. The user can also choose the range and sill as well as the scale in the x and y direction. He can also choose the number of times the program will run. There is a tab for a label named "visual." In that window a graph created in OpenGL will appear. The graphic will correspond with different numbers. For example, red would represent a higher number while blue would represent a lower number. A small user guide has been

compiled to show this (Appendix C).

4.3 Computational

The overall structure of the code was dominated by the windowing, so the numerical code was worked in as methods. Rather than mixing it in with the GUI, all of these methods were collected into their own class. An instance of this data class is created upon the opening of a new project, opening file, or data import. Accessors are primarily used in setting the initial values from user input or file I/O, but are then mainly internal because they are only used in computations which are inside the data class. Computations, such as interpolation, are instigated by a call of the method in the frame class, and then the values can be accessed for display.

This approach of isolating the computational code is much cleaner and makes changes in the user interface easier since only method calls must be moved rather than blocks of code. Storing the data in a data class also allows it to be passed to other parts of the frame. The graphic pane needs the data to display, and C++ inclusion/dependency makes it difficult to get the values if they are an intergal part of the frame.

Details on the implementation on key methods in the computational section of the code is detailed in Section 3.

4.4 OpenGL Graphics

The data set has been represented visually using OpenGL, an interface to graphics hardware. OpenGL is complex and powerful, and is used for rendering interactive color images of three dimensional objects. There were two different types of views created: a height map and a three-dimensional terrain. Due to the inexperience of the team members with OpenGL, the programming guide and tutorials were heavily relied upon[4, 10]. OpenGL creates smooth, aesthetically pleasing images by automatically blending the programmed colors in the window. It also makes the sizing of the screen simpler. The distance between a coordinate point on the graph and the origin will stay in proportion during a change in size of the overall image. This allows the screen to be shrunken, grown, or put into full screen while keeping the picture the same, which is especially important in an application setting. There are many other options set in OpenGL: lighting, surface materials, fog, movement, etc. which are beyond what this project requires.

In order to access the data to be visualized, the pointer to the data class which stores that information is passed to the GLCanvasPane after the interpolation is completed. This prevents anything from being rendered until after the interpolation is finished, or if everything is filled, the "Go" button is pressed. The dimensions of the grid in number of cells and maximum and minimum values are also needed for rendering.

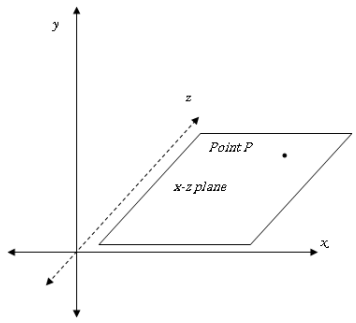

Figure 12: The x-z plane for the 2D grid. The y coordinates are set based upon the value at each point.

A terrain is a three dimensional surface with varying heights. It is created by having a two-dimensional mesh in the x-z plane and storing

heights (y coordinates) for each point. Specific to this project, this means taking the two dimensional slice that is being modeled, and using the values in each cell to set the height. A method is defined to return a height for a given set of two-dimensional coordinates:

```
float GetHeight(int x, int z) { return ((data->GetValue(x,z)-min)/(max - min) - 0.5f
    ); }
```

This takes the value relative to the minimum - which is the lowest point - and finds where it lies in the range. This returns a decimal, so subtracting 0.5 centers the object vertically. These heights are put out in the format of decimals so the programmer must convert them into values.

Besides setting the height, the color of the vertices are also set dependent on the value of each point. This was more complicated to do. First a color array is set up, ranging from blue to red. A red-green-blue (RGB) cube illustrates how the color changes from blue to green to red as different components are added and subtracted. The color array has structures for elements to store the red, green, and blue components of the colors. By setting the length of the color array proportional to the range of values, the index of a color can be calculated for a given value.

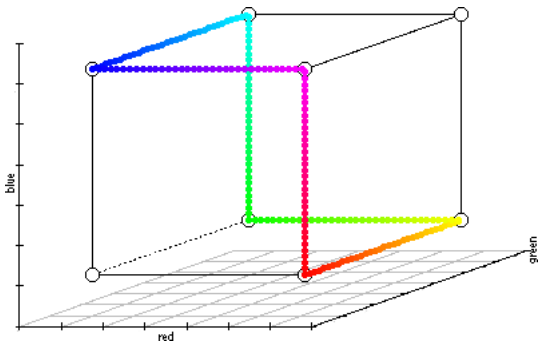

Figure 13: An RGB Cube - a graph of the colors with the red, green, and blue components on each axis.

$$\frac{index}{\#colors} = \frac{value}{max - min}$$

$$index = \frac{(value)(\#colors)}{max - min}$$

A method similar to the one for height is defined to return that index:

```
int GetColorIndex(int x, int z)
    { return ((data->GetValue(x,z)-min)*NCOLORS/(max-min)); }
```

The next step is to draw the actual surface. This is done with a triangle strip. A triangle strip takes given vertices and draws triangles for consecutive sets such as{1,2,3},{2,3,4},{3,4,5}. This is fast because there are fewer three-dimensional vertices that have to be sent to the graphics card. The method from the terrain tutorial was used for looping through the grid and setting the correct vertices. The routine used to render the terrain is as follows, with a sample result in Figure 14.

```
int z, x, index;
int sizing = 2.25;
float scale = 2.0f / MAX(width - 1, length - 1);
glScalef(scale, scale, scale);
glTranslatef(-(float)(width-1)/2, -0.5f, -(float)(length-1)/2);

for(z = 0; z < length - 1; z++) {
    glBegin(GL_TRIANGLE_STRIP);
```

```
    for(x = 0; x < width; x++) {
        index = GetColorIndex(x, z);
        glColor3f(Rainbow[index].Red, Rainbow[index].Green, Rainbow[index].Blue);
        glVertex3f(x, sizing*GetHeight(x, z), z);
        index = GetColorIndex(x, z+1);
        glColor3f(Rainbow[index].Red, Rainbow[index].Green, Rainbow[index].Blue);
        glVertex3f(x, sizing*GetHeight(x, z+1), z+1);
    }
    glEnd();
}
```

The other type of image was taken from the idea of a heightmap. A heightmap is an image used to store three-dimensional data. It is essentially the two-dimensional plane with a range of grayscale to represent height. It is easier to see the range of data in this heightmap, so it has been included as a viewing option. The x-y plane is used in this case, and only the colors are set, not depth. This allowed the estimations of the unknown geological values to be shown in a window for easier intelligence. The values of the makeup of the land were given specific colors and then graphed in a window to create a two-dimensional model of the landscape.

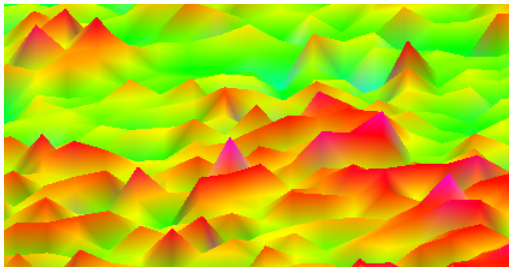

Figure 14: A cropped screen capture of a 3D terrain generated for sample data.

These graphics can be interpreted as red spots being the higher numbers with the other values colored accordingly. The preferable values and colors for finding an aquifer will differ between the graphs of different variables. In a graph of the porosity of the bore hole a higher number would be preferable to a lower because water will be retained in the rock more if the porosity is higher. However in a graph of gamma radiation a lower number would be better. This explains the lack of a key in the user interface window: making sense of the values is really up to the user.

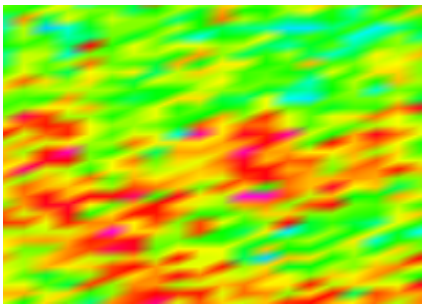

Figure 15: An example of the heightmap rendered for the same sample data as the terrain.

4.5 wxPlotCtrl Graphing

Even though there are several plotting packages available to use with wxWidgets, wxPlotCtrl is the one supported by wxFormBuilder. It is an interactive xy plot with options such as zooming, selection of points, and data processing. Many of these functions were hard to access and use owing to the lack of documentation. Despite this difficulty, an operational graph was created for the semi-variogram.

PlotCtrlPane is an object within the application frame. Once the semi-variogram is calculated,

the data is passed down to be plotted. The x and y directions are separate and distinguished by color in a key. The experimental semi-variogram is then interactively matched by the user. There are several options on the left sidebar. The data sets can be scaled to account for anisotropy by entering a percentage. This calls the function in the PlotCtrl which both performs the scale and saves the inverse. The next percentage that is entered will first use the inverse to revert back to the original size before applying the new scale. The sill/range of the semi-variogram is set by double clicking on the graph, which is interpreted by the library as coordinates. These are recorded in variables and used to calculate the mathematical models which are then added to the plot. These curves are created by setting the points every whole number, which can cause some misrepresentations for small horizontal sections. The sill and range can also be set originally from the sidebar. After they are set – either in the sidebar or with the mouse – the current values are displayed in the sidebar, updated in the idle loop, which prevents them from being set there again. Other methods of updating were attempted, but interclass communication and propagation of events has not been successful so far.

The wxPlotCtrl library also has some automatic functions that were useful. A click and drag of the mouse will zoom in on the selected section of the plot and can scroll along the axes. The title and labels can be edited, though they will always be reset.

5 Results

5.1 Case Study

This case study returns to the original application to aquifers: using data from boreholes to determine if water-bearing rock might be located in the ground between the holes. It is an ideal choice in some ways, since geostatistics was used in hydro-geology early on in its development. In most cases, it would be prudent to verify spatial dependence before interpolating, but for the limited scope of this project it has been assumed. This assumption is supported by the traditionalism of the field.

Regrettably, the two boreholes left to be used after one had to be dismissed were the farthest apart and proved too challenging for the present version of the program. In spite of these problems, progress has been made in completing the study and it has provided invaluble insights into what future work is required.

5.1.1 Data

Data from boreholes in the Los Alamos area were generously provided for use in this project (Section A.3). There are many different types of information collected from the boreholes; this project deals with depth, porosity, water flow, and radiation emission. While porosity or a different variable can provide valuable information about the locations of aquifers, it really is the *combination* of favorable qualities that will indicate a possible aquifer. This is because the perfect geological site for an aquifer is determined by many different variables acting together for the ideal surroundings. This

There was data for three boreholes in the Mortandad Canyon area (R-1, R-7, and R-33 located in Figure 17), but unfortunately, as it was prepared to be imported, it was realized that one did

not overlap with the other two depth wise and could not be used at this point. Borehole R-1 was eliminated and the other two were imported to test the program.

The same section of depth was observed by taking the elevations and depths for the boreholes. Even though the holes are not at the same elevation, it was possible to find an absolute measurement above sea level by decreasing the gound elevation by the depth of the hole at each point. This lined up sets of data, making it possible to take a consistant cross-section (Figure 16).

Five different sets of data were provided so that it would be relatively easy to write an import function to bring the sets into the program. The first set of measurements is standard Gamma Rays in API units. Gamma rays are used to find water and the different fluctuations in the rock patterns. The meters measure incoming gamma rays from radioactive elements in the surrounding rock. For example, if potassium or uranium are in a rock the water is most likely not there because they have a large nuclear signature. Since these elements are highly radioactive they will create quite a large signature and water will not be there because of lack of porosity. Even if there were porosity, it would be filled with the radioactive elements. The next type of data received was Deep-Reading Resistivity in ohm-meters. This basically reads the conductivity and resistivity of the rock at interval depths. The conductivity and resistivity of the different depths can be used to determine whether or not the rock holds any potential for water. The third set is a total porosity set which measures the total amount of pore space in the probed aquifer. This is a less effective way to find data because of its using all of the pore space no matter how small. More meaningful is the fourth data set, effective porosity. Effective porosity allows the probe to only find certain sized pores that allow effective flow of ground water measured by nuclear magnetic resonance (NMR). Last is the Logarithmic mean of T2. The Logarithmic mean is the average pore size of the probed area and can be used to tell how useful the aquifer is - the pores must be open enough to be interconnencted.

An import method was then written to bring the data into the application and was then used as a sample set to test the program with real world data.

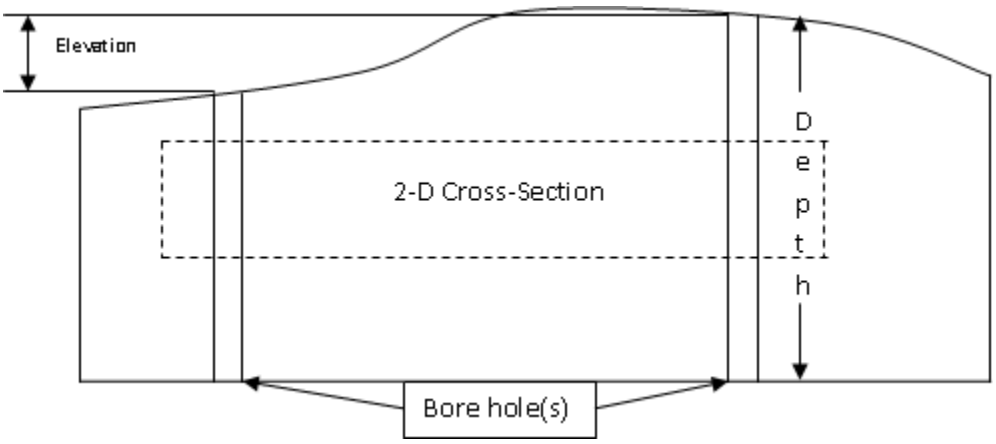

Figure 16: Two dimensional cross-section of boreholes with different elevations.

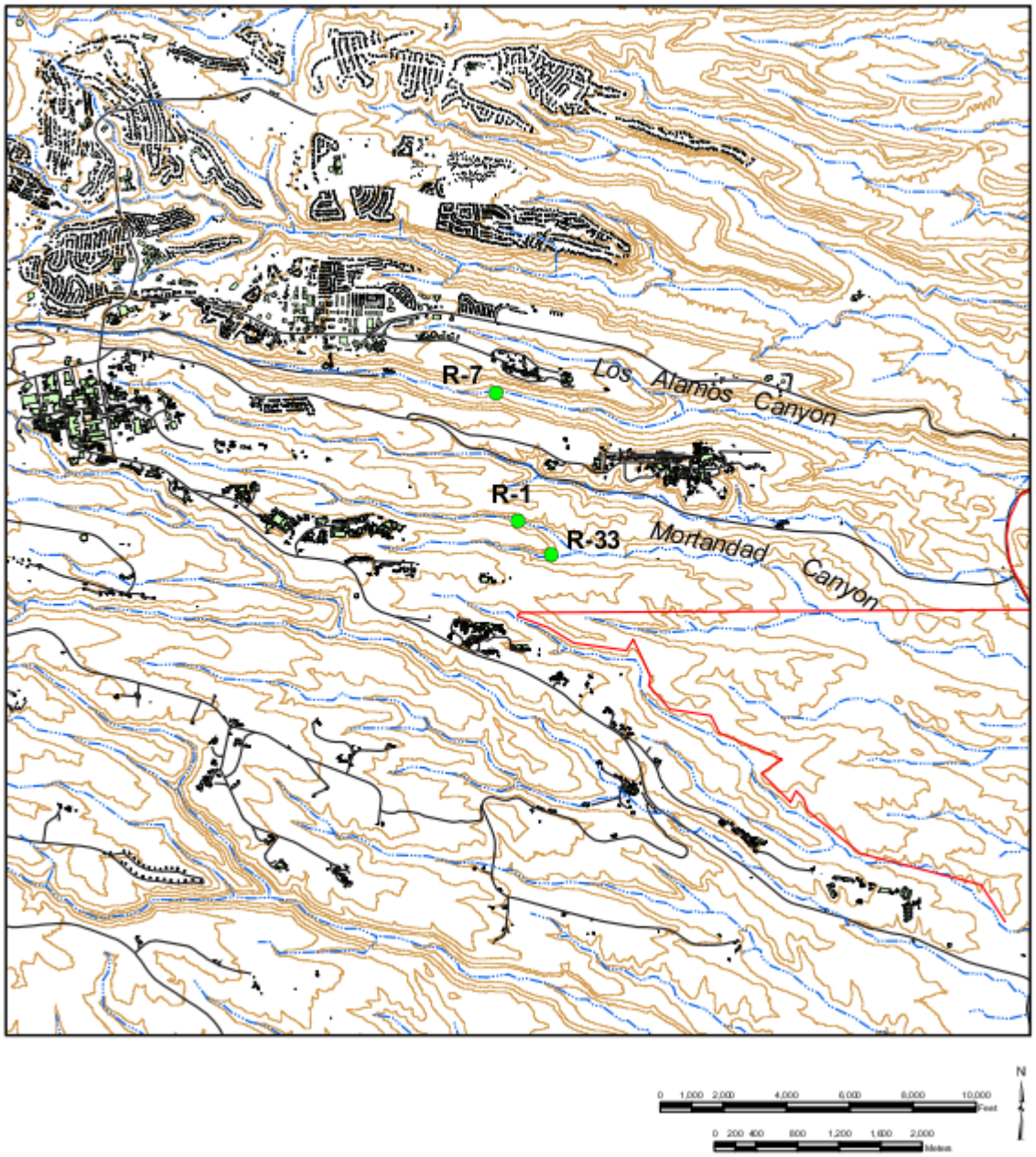

Figure 17: Location of the boreholes from the case study.

5.1.2 Semi-variogram

The semi-variogram for the various types of data was successfully plotted. The x direction has only one point because there is only the single distance from one borehole to the other. The y direction produced interesting plots which are inluded here in Figure 18. Note the large values of variance that probably arise when the data covers multiple rock structures. The large differences between them would increase the average variance.

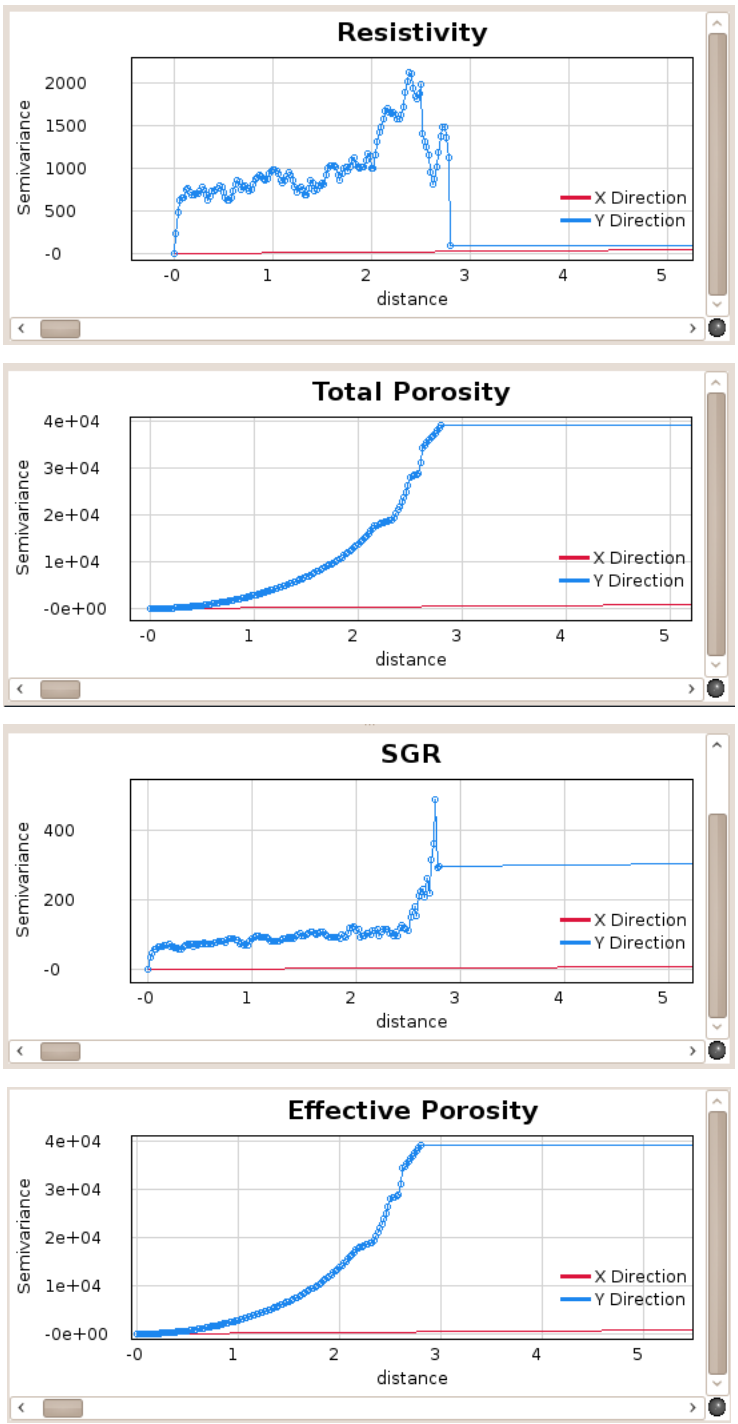

Figure 18: Semi-variogram of each of the data sets for the two boreholes. These are focused on the y direction as there is only one known point in the x direction.

5.1.3 Mixed Success of Interpolated Fields

There were mixed results between inverse distance weighting and kriging. The IDW was able to complete the interpolation, while kriging returned completely unreasonable numbers. These runs used the anisotropy and sparse data options and only run for each since the problem is so large.

The height maps produced for each data set using IDW are shown in Figure 19.

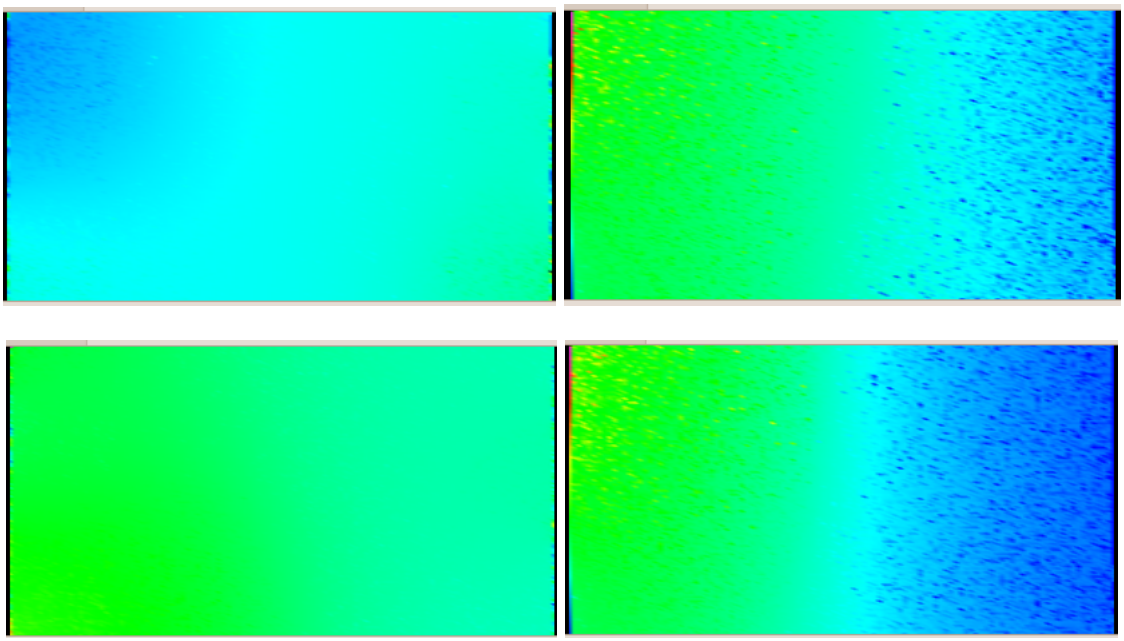

Figure 19: Height maps produced for inverse distance weighting interpolation between boreholes. From top to bottom they are: resistivity, total porosity, SGR, and effective porosity.

Since these results were only obtained at the end of the project, no attempt at numerical analysis has been made.

Attempts to run the borehole problems using kriging continued to be fruitless. Even though it was finally capable of finishing, the numbers were ridiculously large. In interpolation, results should fall within the range of the original data, further invalidating those values. It was theorized that the large variances used in matching the semi-variogram, coupled with the scale of the problem, simply overloaded the methods and caused it to return nonsense. Ultimately, a small section of the boreholes was to use instead, and itwas successfully completed when an much smaller variance was used (Figure 20).

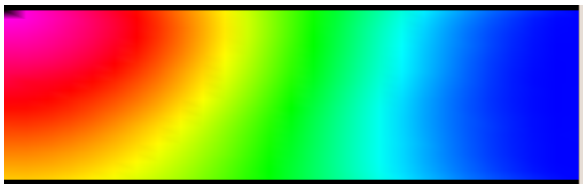

Figure 20: Visual results of small section of borehole interpolated with kriging.

6 Conclusions

There was a lot accomplished in the course of this project, but geostatistics is a complex field, so only the fundamental levels were covered in the available time. The science that was managed to be incorporated into the project is potentially useful in the a search for aquifers, as shown by the preliminary results, and was a significant learning experience for all involved.

The results of the case study demonstrated that there are still several adjustments to be made

for the program to be a viable tool in large scale problems. For smaller problems, it works well. This includes several options set by the user for desired effects. The user interface still has some pitfalls if it is not used the way it was designed, but turned out impressively.

Intersting characteristics of the two interpolation methods became apparant through the test runs. Kriging provides a smoother interpolation in contrast to the 'bull's eye' effect of inverse distance weighting. There are advantages of IDW which were demonstrated in the case study. Kriging is much more sensitive to the high variances and anisotropy, which likely caused the faulty results when it was used for the borehole data. IDW can complete this problem, returning results for similar situations that are not possible with kriging.

More than the cursory attention given to the results from the case study would be required to determine exactly how accuratly the program predicted possible aquifers. Unfortunately, these were not completed until the very end of the project, so this analysis was not possible. In and of itself, simply getting plausible results out of the program is an achievement.

6.1 Current Status

Despite hard work and good ideas there were many problems in the making of a working program. There was a memory leak that made the system crash if overly extensive data sets were attempted to be run. The memory leak was fixed, but large data sets still do not work, returning very large, faulty answers. Smaller data sets do run through the program and yield successful results. The case study involving the borehole data was imported into the program and helped to determine whether the process worked. To hasten the slow computation of the computer program an optimization of tasks was attempted. It was partially successful and made the program somewhat more efficient.

The graphics presented many difficulties and much time was spent in fixing them. Getting them to interface with the C++ was problematic and did not work for a long time. Making the graphic map three dimensional was a trial. Initially the graphics were being written in OpenGL as rows of colored boxes that would be colored by their specified values, however that could not be contrived to work correctly and was discarded in favor of triangle stripping. This worked much better and the graphics finally interfaced satisfyingly with the other code.

Fortunately, almost all of the problems encountered in the progress of "Adam's Ale" were dealt with and eliminated to create a program that does what it was designed for. Hopefully, it will be able to solve real world problems.

7 Teamwork

When a team has a small number of members, the confines of the duties are less well defined, as more work has to be done by fewer people. Team 65 had four members of whom each had a very specifically designated task. This system was designed for efficiency and effectiveness and was implemented with success. As well as giving each job to the most suitable team member, Team 65 attempted to involve each of their colleagues in their own allocated areas so as to provide instruction for everyone. The team members not already versed in programming learned the basics of OpenGL, wxFormbuilder, C and C++. Because of the program's application in geostatistics pertaining to aquifers, everyone mastered the information on aquifers needed to make this project a success.

One of the many ways Team 65 improved on communication was through the utilization of Google Code, Google Documents, and Google Calendar. Google Code granted the members of the team working on programming a repository for their specialized parts of the whole. The repository made the merging of the parts a quick and easy process. Google Documents was a repository for the members of the team working on the various reports so that the many different parts of the writing could be done more easily and the other teammates could immediately have the most current versions of the reports. The individuals of Team 65 are not especially noted for their organizational skills, however Google Calendar rectifies this matter. It could be said that having their schedule of events available for regular perusal impressed upon them the responsibilities of remembering a meeting. Not only would the other teammates be disappointed in the individual if he missed but would exclude him from the amusing antics Team 65 would routinely engage in while working diligently.

Overall, Team 65 functioned under very superior working conditions for the entirety of their project and were extremely pleased in the end result of all their hard work.

8 Recommendations

This project created a solid basis for future studies in the extensive and growing field of geostatistics. The discontinuity of success from small to large problems should be addressed, perhaps in the form of dividing the problem up into the separate structures. Additions of safe-guards to catch errors that may occur for incorrect user operations would prevent unexplained crashes. The user should be supplied with helpful messages to guide them to fix the problem in the input.

Efficiency in a computer program is always to be desired and optimization is the key to a faster running program and should definitely be undertaken to better the program. The operation on the GPU is still visibly slower than the CPU and requires more work to get any speed-up. This should be done by parallelizing more sections of the method and balancing the gain of the parallel processing with the cost of loading the data.

The case study involved in the program introduced the issue of creating an accurate semi-variogram as only one value could be calculated in the x direction - at the distance between the boreholes. It seems that this is still a dilemma of the field and that the graphing of semi-variograms from sparse data has yet to be performed in the field of geostatistics. If this project managed to achieve this milestone then many more could follow.

These improvements could take this project to a more complex and hopefully more useful level of practice.

A References

A.1 Bibliography

References

[1] Gaussian function. *Wikipedia*, 2009.

[2] Isobel Clark. *Practical Geostatistics*. Elsevier Applied Science, 1979.

[3] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, Maryland, 3rd edition, 1996.

[4] Bill Jacobs. Opengl video tutorial - terrain. 2008.

[5] Shari Kelley and Peggy Johnson. Frequently asked questions about water. *The New Mexico Bureau of Geology and Mineral Resources*, 2009.

[6] G. Matheron. The theory of regionalised variables and its applications. Technical report, 1971.

[7] NVIDIA. *OpenCL Programming Guide for the CUDA Architecture*, 2.3 edition, Mar. 2009.

[8] Howard Perlman. Water science for schools: Aquifers. *United States Geological Survey*, 2009.

[9] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, New York, 1988.

[10] Dave Shreiner. *OpenGL Programming Guide*. Addison-Wesley Professional, Upper Saddle River, New Jersey, 2009.

[11] 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.

[12] Kimberly J. Swanson. Aquifer characteristics. *Water Encyclopedia: Science and Issues*, 2009.

A.2 Software/Tools

Several programming tools and other applications were used in the development of this project:

wxWidgets/wxFormBuilder
Eclipse
Totalview
Doxygen
Google Code
Microsoft Office and OpenOffice
LyX

A.3 Acknowledgments

We would like to extend our deepest thanks to the people who volunteered so much of their time to help us with this project:

Robert Robey, for his general help with everything;

Thomas Robey, for his mentorship in the geostatistics and mathematics;

David Broxton, Danny Katzman, and Ned Clayton of Schlumberger, Inc for providing us with data, definitions, and meeting with us to work through it;

Mary Green, for her advice in geology and hydrology;

Larry Cox and Jorge Crichigno and for their advice and positive comments at the interim presentation.

B Glossary

Anisotropy property of directional dependence; in geostatistics this means that the data has different spatial correlation in the x and y directions. For example, wood will be more related along the grain than against it.

Geostatistics a branch of applied statistics that uses the interdependence of spatially correlated data to interpolate unknown values.

Isotropy property of consistency for all directions, in geostatistics the spatial correlation is independent of direction.

Kriging a common method of interpolation in geostatistics that uses a mathematical model of the semi-variogram.

Nugget (of semi-variogram) magnitude of discontinuity at the origin, usually a result of measuring/sampling errors.

Range (of semi-variogram) distance at which the semi-variogram plateaus and range at which points are correlated to some degree.

Semi-Variogram the formula and graph of the variation of data over distance, quantifying spatial correlation.

Sill (of semi-variogram) variance value for distances beyond the range or the value of the plateau

Spatial Correlation the idea of data being related as a function of its location

C User Guide

This program was created to be very user friendly. The window is a very simple design in which there are three large sections. The first section is a tabbed window in which the user can switch between the visual and the raw data. Below that is the semi-variogram. That is the area in which the data is graphed. On the right side is a vertical window in which the user can change the appearance of the semi-variogram. For example, the user can choose the range, sill, a model, the method of interpolation and the size at which the X and Y values are scaled.

-To start the process, either go to File->Open... or select File->New. Then select the set of data that is to be implanted or created.

-It will open a window that has the uploaded data; select the set of data.

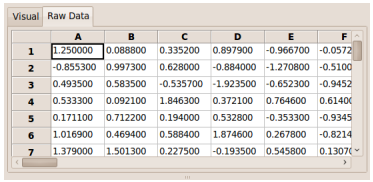

Figure 21: Data inserted

-Once that data set has been opened the chart at top of the window that reads “Raw Data” will fill with the selected set of data.

-Plot->Semivariogram, the graph will appear in the bottom section.

-Double click on the knee of the graph. Three more lines will appear, they are the three different mathematical models.

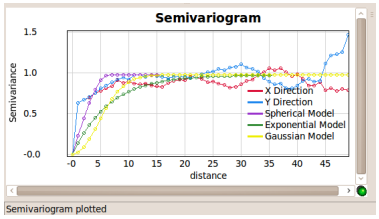

Figure 22: Semivariogram plotted, mathematical models chosen

-Select one of the models at the right in the “Model Type” scroll box. The model which has been selected will now be the only one on the graph.

-Choose your method of interpolation, the scale for each X and Y direction, and the number of runs.

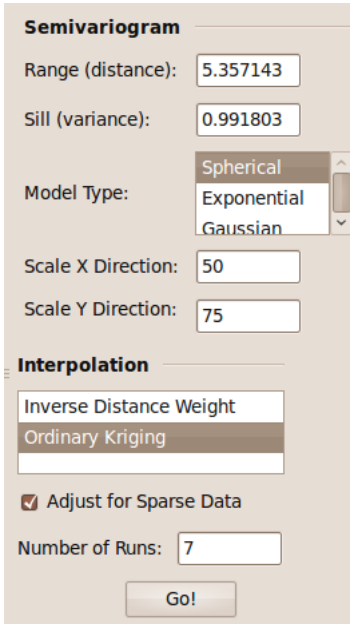

Figure 23: Model type, range and sill, scales, interpolation method, number of runs selected

-Hit “Go!”.

-There are many paths the user can take from this point.

-It is possible to view the data in which the semivariogram has “come up with”.

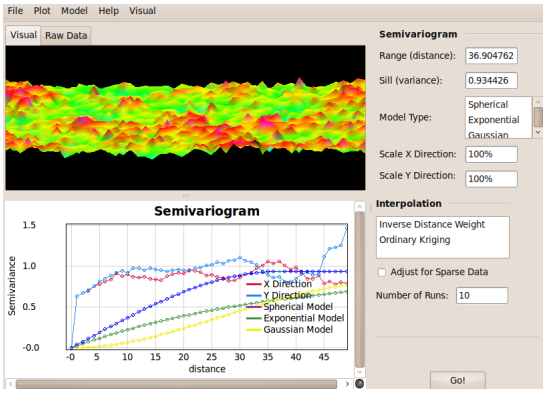

Figure 24: Final screen with a terrain map

-Also, the user can view two different versions of the visual with the "Visual" drop down menu. Choose between "Height Map" and "Terrain".

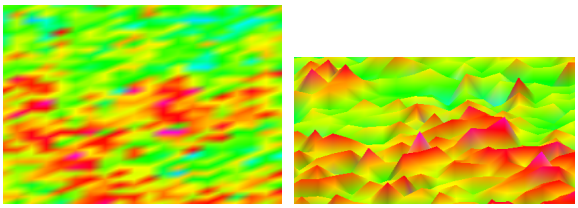

Figure 25: Left->Height Map, Right->Terrain