12-2008

Simulation Modeling of Karst Aquifer Conduit Evolution and Relations to Climate

John D. Broome
Western Kentucky University, john.broome@wku.edu

SIMULATION MODELING OF KARST AQUIFER CONDUIT EVOLUTION AND RELATIONS TO CLIMATE

A Thesis
Presented to
The Faculty of the Department of Geography and Geology
Western Kentucky University
Bowling Green, Kentucky

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science

By
John Broome

December, 2008

SIMULATION MODELING OF KARST AQUIFER CONDUIT EVOLUTION AND RELATIONS TO CLIMATE

Date Recommended October 1st, 2008 ______

Dr. Chris Groves ________________________
Director of Thesis

Dr. Stuart Foster ________________________

Dr. Stephen Kenworthy ___________________

Kevin Cary ____________________________

____

Dean, Graduate Studies and Research Date

ACKNOWLEDGEMENTS

At the completion of this thesis, I have a debt of gratitude to many parties. I would like to begin by thanking Almighty God for giving me both the opportunity and ambition to reach this personal milestone. I am also appreciative of the love and encouragement of my parents, David and Monica. Their faithful support of my decisions and goals has been a tremendous source of strength to me throughout my life.

Without the guidance of my thesis advisor, Dr. Chris Groves, I would not have had the stimulus for this research, nor the advantage of his wealth of geological expertise and experience. It is he who encouraged me to proceed during times when the purpose of our efforts seemed elusive. I would like to thank Dr. Groves for his friendship, and for the pleasure of many enlightening conversations about the intriguing world of karst and the essence of scientific research. Additionally, the original field data, which was an essential component of this project, came from the research of Dr. Groves and Joe Meiman (Mammoth Cave National Park).

In a similar vein, I would like to state my indebtedness to the other capable and professional members of my thesis committee (Dr. Stuart Foster, Dr. Stephen Kenworthy, and Kevin Cary). Their guidance and input was extremely valuable and appreciated greatly.

The logistical headaches of being a full-time employee, husband and father, a part-time grad student, and seemingly a full time commuter were partially mitigated by the help of my supervisors at the Planning Department in Nashville, and by Wendy DeCroix and Pat Kambesis at WKU. The financial burden was lessened by some funding received from the Hoffman Environmental Research Institute.

Finally, I would like to thank my wife, Stella, and my children, Patricia, Joseph, and Isaac, who make all my struggles worthwhile. During the pursuit of my academic goals, they have had to contend with my short temper, and the frustration of sharing my time and energy with my research. I sincerely thank them for their personal sacrifices that have made this thesis possible. It is to Stella, Patricia, Joseph, and Isaac that I dedicate this thesis.

FOREWORD

This note concerns the format of this Master's Thesis. Through time, MS level research projects conducted under the auspices of the Hoffman Environmental Research Institute in the Department of Geography and Geology have been more regularly published in professional journals, as is appropriate for high-quality environmental research. Mr. Broome's research described herein makes new contributions to the understanding of karst landscape evolution using a simulation modeling approach, and is indeed at such a level. With this in mind, and recognizing that publication of research results in peer-reviewed journals is the most appropriate method to disseminate research results, we are simultaneously evolving to a thesis format that is closer to that which is submitted for publication review, and we anticipate that this manuscript will be submitted for publication to the peer-reviewer journal *Geomorphology*. This thesis makes a stride in this evolution, at my suggestion and approval as the advisor of this research.

Chris Groves, PhD
Thesis Research Advisor

TABLE OF CONTENTS

LIST OF FIGURES

SIMULATION MODELING OF KARST AQUIFER CONDUIT EVOLUTION

Name: John Broome Date: December, 2008 Pages: 92

Directed by: Chris Groves, Stuart Foster, Kevin Cary, and Stephen Kenworthy

Department of Geography and Geology Western Kentucky University

ABSTRACT

Karst regions of the world that receive relatively similar amounts of precipitation display a wide variety of landscapes. It has been suggested (Groves and Meiman, 2005) that climates exhibiting larger discrete storm events have more dissolving power and consequently higher rates of conduit growth than climates with more uniform precipitation distributions. To study this concept, a computer program "Cave Growth" was developed that modeled the growth of a cross-section of a cave passage under dynamic flow and chemical conditions. A series of 46 simulation datasets were created to represent different climatic conditions. These simulations had the same total annual discharge, but demonstrated a range of flow distributions quantified by use of a gamma distribution index, along with two special theoretical cases.

After simulating a year of conduit growth for each of the various flow distributions in a series of model runs, and repeating these sets of simulations for three different passage cross-section geometries, it was evident that the annual temporal distribution of flow did indeed impact the amount of cave growth. However, an increase in the "storminess" of the climate did not simply equate to more dissolution and thus conduit growth. Rather, the quantity and duration of surface contact between water and the conduit walls combined with dissolution rates to affect the total growth. The amount of wetted perimeter (contact between fluid and passage floor/walls) generated by specific

flow levels depended upon the shape of the passage. Flow conditions that filled the conduit to capacity were shown to be very effective at growing the cave. Above this level, the dissolving power of additional water was essentially wasted. This investigation suggests that the maximum amount of passage growth occurs under flow conditions that result in the most wetted perimeter for the longest period of time at the highest dissolution rate.

INTRODUCTION

One of the quintessential tasks assumed by the discipline of Geoscience has been that of explaining the regional variation in the natural phenomena that are encountered across the globe. An example of this effort is the research that has been conducted in order to understand the unique scenery and subterranean features that characterize karst environments. Indeed, geologists and physical geographers have studied the processes and underlying conditions affecting the evolution of karst aquifers and their associated surface landscapes in great depth. This research has included the examination of many aspects of the relationships between the growth of the subsurface drainage networks that define these aquifers and the availability of groundwater. It has been shown that the total amount of groundwater that flows through a karst system each year has a direct impact on the rate of overall karst landscape denudation that occurs (White, 1988; Smith and Atkinson, 1976; Kiefer, 1990; Groves and Meiman, 2005). However, the influence of the annual temporal distribution of that flow, closely related to precipitation input rates, has been considered less carefully.

The rate of discharge in a system throughout the year is largely a reflection of the distribution of precipitation occurring at the surface. Precipitation quickly becomes recharge to the karst aquifer as well developed karst flow systems are characterized by low resistance and very rapid response to storm recharge events (White, 1988; Palmer, 1991; Ford and Williams, 2007). Based on analysis of one year of high resolution flow and chemical data from Logsdon River in Kentucky's Mammoth Cave System, Groves and Meiman (2005) suggested that flow distributions (and by inference, climates) exhibiting larger discrete storm events, that is, with less uniform rainfall distributions, have more dissolving power and consequently higher rates of conduit growth than climates with more uniform annual flow distributions. If it is generally true that the temporal distribution of discharge can affect the rate of cave passage expansion in an aquifer, this knowledge may help geoscientists to better understand the variety of landscapes found throughout the world's karst regions.

Figure 1a (left). High-relief karst towers developed in Paleozoic limestones of the Guangxi Autonomous Region, China.
Figure 1b (right). Low relief sinkhole plain developed on Paleozoic limestones of Kentucky.
Photos by Chris Groves.

For example, a region such as the Guangxi Autonomous Region in Southern China, that annually recieves about the same amount of precipitation as Southern Kentucky and is in some ways geologically similar, displays a quite different karst landscape. Its impressive karst towers (Figure 1a) and huge underground river passages (Yuan, 1988, 1991), in some places with widths in excess of 100m, contrast sharply with Southern Kentucky's gently rolling sinkhole plains (Palmer, 1981, White *et al*., 1970) (Figure 1b). While the factors that contribute to the evolution of these disparate karst landscapes are complex and multifaceted, and potentially involve differences in tectonic settings, an interesting distinction is that the annual precipitation in this region of China is concentrated into a short summer monsoon season (Ding, 1994), whereas Southern Kentucky experiences a comparatively uniform distribution of annual precipitation. Different recharge and discharge patterns in these landscape/aquifer systems may be an influential factor in the evolution of these two karst environments.

Any processes that influence an increase in the capacity of the underground drainage network to accept drainage consequently affect the speed at which the connected surface geography undergoes a transition from a fluvial to a karst landscape. As regions with soluble bedrock evolve, their surface topography shifts from landscapes defined by fluvial processes to those shaped primarily by subterranean drainage. Once most or all of a region's drainage has been diverted underground, non-fluvial landscape forming processes subsequently dominate surface forms. As more water is diverted underground through ever-enlarging conduits, surface streams become intermittent, and swallets eventually form. This can result in intermittent streams or dry valleys at the surface. Other karst features including sinkholes and fractures further disrupt surface drainage patterns and redefine a region's landscape. Ultimately, the rate of landscape denudation occurring in a karst region is shaped by factors that influence the growth of caves and conduits.

This research strives to explore the relations between one of these factors, the annual variability of discharge flowing through a karst aquifer, and evolution of the primary conduits carrying that water. To accomplish this task, a computer program "Cave Growth" was developed and simulation discharge datasets representing a series of varying climatic conditions, quantified by varying the temporal distribution of a fixed total annual quantity of water draining through the system (considered to be equal to precipitation minus evapotranspiration), were created and processed. Through this application, the growth of a specific cross-section of a karst conduit was analyzed under varying flow distributions in an attempt to gain insight into the relationship between these flow/precipitation distributions and conduit growth, and thus karst landscape evolution.

METHODOLOGY

Computer simulations and environmental models provide investigators with a means of analyzing conditions and processes that would otherwise be too large or small, too complex, or too time consuming to study in the real world (McCuen, 2002). The influence of climate on conduit growth within a karst aquifer is a good example of a subject that is difficult to study in the field. Underground sites are difficult to access and precise measurement of factors affecting conduit growth requires the undisturbed use of remotely implemented equipment over an extensive period of time. Thus, the purpose of the “Cave Growth” application developed for this research was to provide a method of exploring scenarios and conditions that affect this phenomenon that could be used to compliment real field studies.

While benefits such as those described are significant, there are caveats to simulating the real world environment that must be recognized when using them to make interpretations and draw conclusions. The most obvious problem with predictive modeling is that the simulation cannot hope to fully account for the complex conditions that exist in the real world. Additionally, the structure and design of the program can itself have impacts on the results that it generates. For example, the choice of grid cell size in a Geographic Information Systems (GIS) process that models surface drainage can affect the delineation of a drainage basin (Usery *et al*., 2004). Therefore, this discussion of the methodology utilized in this research attempts to clarify both the assumptions that were made and the logic of the programming code behind the Cave Growth application. The code itself is written in Microsoft's object oriented and event driven Visual Basic 6.0 programming language, and is listed in Appendix I.

Cave Growth was developed for this research project as an effort to create a computer model that could simulate the growth of cave conduits under dynamic flow and chemical conditions and also serve as a means of visually analyzing that growth. In an effort to isolate and study certain manageable facets of this highly complex phenomenon, the program represents conduit growth as the expansion in area of a two-dimensional cross-section of a cave passage that contains a flowing stream of potentially varying discharge. The rate and shape of that expansion are determined by evaluating the geometry of the cross-section in conjunction with the dissolution rate and the portion of the conduit that is underwater during each of a given series of time steps. The total change in the cross-sectional area of the cave over the course of a simulation run is considered a measure of growth associated with a particular combination of input parameters and simulation duration. By holding those parameters constant, while processing a series of simulation datasets demonstrating a broad spectrum of flow conditions, the Cave Growth application was utilized to examine the influence of flow distributions, and thus this aspect of climate on the rate of conduit growth.

The program's graphical user interface consists of a display grid (Figure 2-A), a toolbar (Figure 2-B), a message panel (Figure 2-C), and a configuration panel (Figure 2-D). These features allow the user to configure a simulation run and also to examine the

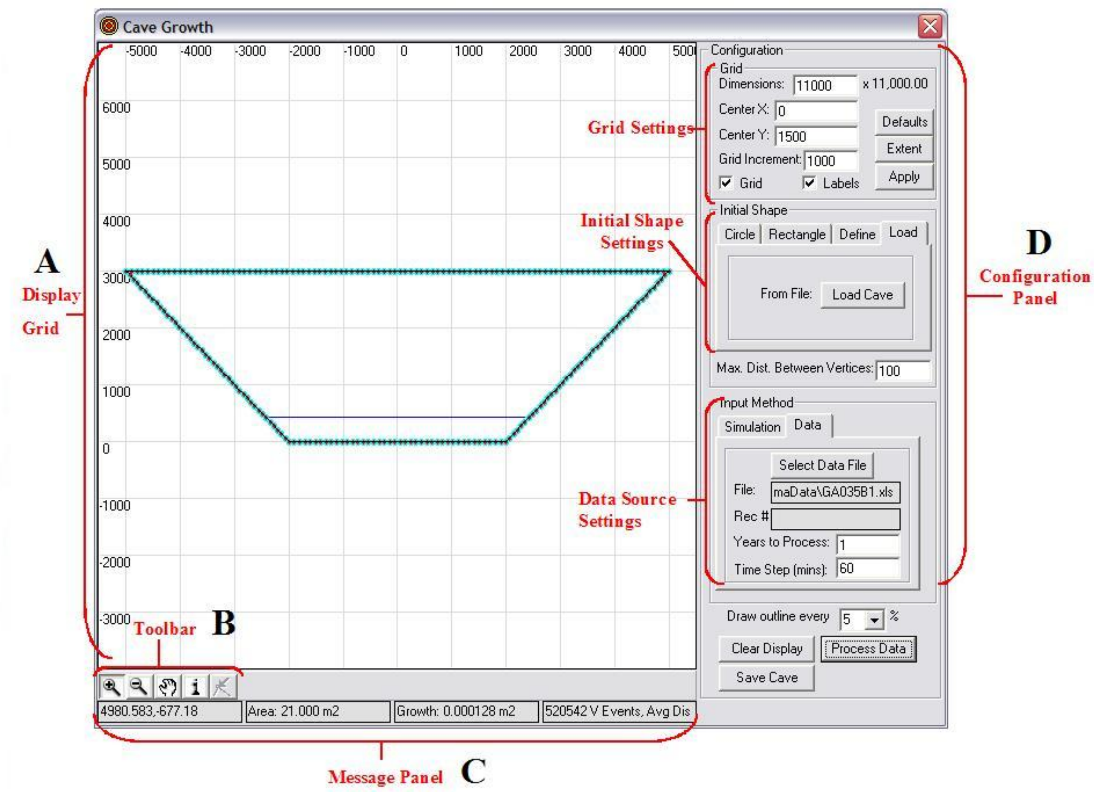


Figure 2. Graphical user interface (GUI) for the Cave Growth application.

results of that run. The display grid dominates the screen and is used to view the conduit cross-section and the accompanying reference grid. The tools that accompany the grid provide the user with simple navigational functionality allowing them to zoom in and out and pan around the viewing area and also provides them with the ability to identify vertices, which are specific points that together when connected define the passage cross-section. The change in a passage cross-section depends on the movement of the individual vertices. The message panel reports the location of the mouse cursor relative to the grid, the current cross-sectional area of the cave passage, the amount of growth recorded during a simulation, and the average dissolution rate and number of “vertex events” (described in more detail below) for the run. Among other things, the configuration panel of the application allows the user to set up the display grid, establish the geometric parameters, select a data source, specify processing time, and begin a simulation.

The initial shape of the conduit cross-section is one of the geometric parameters that must be established for a simulation run. Cave Growth has been designed to work with basic symmetrical profiles. In the configuration panel one can choose from a circular, rectangular, or user-defined initial shape. Once the geometry has been defined it can be stored and reloaded for efficiency across multiple runs. For this experiment, a $21m^2$ trapezoid representing the Logsdon River study site on which much of this research is based, a $21m^2$ circle, and a $21m^2$ (2m x 10.5m) rectangle were defined and reloaded for each run. These initial passage geometries are shown below in Figure 3. The shape chosen influences the amount of wetted perimeter that exists under given flow conditions and therefore affects the evolution of the passage and ultimately the amount of growth that occurs in a simulation run.

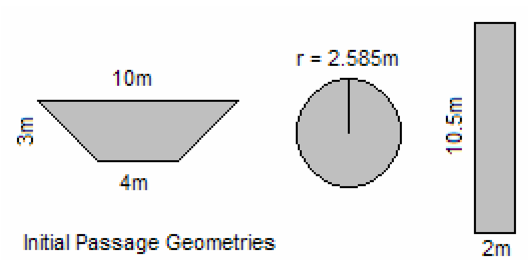


Figure 3. Passage geometries used in simulation runs

Although the cross-section is presented in the display grid as a polygon, within the application code, the cave walls are actually represented as a dynamic array of vertices. These vertices are individually evaluated and moved in accordance with shifting environmental conditions to reflect the retreat of the passage wall resulting from limestone dissolution. For every time step, each vertex is assessed to determine if it is underwater based on the current flow conditions and passage geometry. Only those vertices that are determined to be underwater are subject to dissolution in this model. The proximity of the vertices to one another is a configuration setting that was developed to allow the researcher to control the "geometric granularity" of the study (how many vertices are used to represent the passage wall). This setting establishes a maximum distance allowed between vertices, which is maintained after each time step by the insertion of a new vertex between any adjacent vertices that have strayed beyond the ascribed tolerance. A finer grained analysis can obviously be achieved by requiring vertices to be closer, but this has the cost of slower computational speed as more cave vertex objects have to be processed by the program. In the simulation runs for this experiment the maximum distance between vertices was set to 100 mm.

In the Cave Growth application, the continuous process of dissolution is represented as a series of discrete events, the temporal frequency of which can be varied, again at the cost of additional computer processing time for finer temporal resolution. The user is provided the means to determine this temporal resolution of the simulation run. While the program was designed to be able to utilize data containing varied time intervals, in the simulations reported here it processes those data in equal time steps, the length of which are defined by the user. This allows the user to examine the cumulative effect of multiple small-scale events that could be lost in a series of larger generalized events. For instance, the impact of shorter storm surges that bring the cave roof into contact with water would be averaged out with large processing steps. To evaluate the data in equal time intervals through the program code, dissolution rates and discharge information are multiplied by their corresponding duration and accumulated until the user-defined increment is met at which point average values for that increment can be determined. The datasets created to consider the question posed by this research were generated with uniform time intervals to make this task simpler. The time step value was set to one hour and the "years to process" was set to one year for the comparative simulation runs in this research.

As described, continuous variables such as the surface of the cave and the passage of time are simplified into discrete objects in this program. Furthermore, in order to model the complex subject of conduit growth in a karst aquifer, Cave Growth's processing structure is based on certain assumptions outlined below.

1) Zero permeability exists outside the conduit (all water flows through the conduit).
2) All conduit growth is by limestone dissolution, and abrasion by through-flowing sediments is negligible.
3) Limestone dissolution rates (in this work expressed as mm/yr of wall retreat) are assumed to be a function of bulk water chemistry, quantified by the Plummer *et al.* (1978) rate equation, and independent of fluid velocity at the conduit wall. Plummer *et al.* (1978) assumed reaction-limited dissolution kinetics by conducting stirred tank experiments, increasing velocities until dissolution rates became independent of stir velocity.
4) Limestone composition in these simulations is assumed to be homogenous.
5) All conduit walls are bare rock and no impacts of sediment barriers are considered in the current runs, though the program could be readily modified to explore this in future investigations.

The program affords two processing modes: a simulation mode in which static conditions (water flow rates and chemistry) are set and processed for a given number of events; and a data mode in which dynamic environmental conditions are read from an Excel spreadsheet. Regardless of the mode, for each time step, the water level that corresponds to a given cross-sectional area of water must be established in order to identify which vertices are underwater and therefore subject to dissolution.

If the wetted area from the spreadsheet (flow cross-section) is greater than that of the cave cross-section, all vertices are automatically classified as being underwater. Otherwise a vertex is picked and grouped with its equivalent from the opposite wall (one is created if necessary) and with all other vertices that are below this pair. The area of the polygon formed by these vertices is compared to that of the flow cross-section. If the polygon is larger than the flow cross-section, the water level is dropped to the next lower vertex and the process is repeated incrementally until the area formed by the group of vertices matches or is less than the flow cross-section area. The same approach is used in reverse if the wetted area is initially larger than the vertices being examined. Through this iterative process, the vertices that are underwater based on a specific flow and passage geometry can be deduced even though the precise water level has not been determined. Those vertices with y elevations at or beneath the vertex pair marking the approximate water level are flagged as being underwater and are passed to a function that moves them based on the dissolution rate and their position relative to their neighbors.

Cave Growth assumes that the solutional retreat of any vertex on the passage wall is perpendicular to the line segment formed between its immediate neighbors. To determine the direction in which to move a vertex (blue arrow in Figure 4), the slope of the line connecting its neighbors (red line in Figure 4) is calculated. The slope is entered into one of four equations, depending upon which quadrant the angle of the connecting line falls within (Figure 5). The angle returned is always perpendicular to the connecting line and external to the conduit polygon. In Figure 4, the angle of the red line joining vertices 25 and 27 is between 270° and 360°, thus, according to the chart in Figure 5, its slope is entered into the Arctan function, multiplied by 180/pi (to convert radians to degrees), multiplied by negative one, and finally added to 180° to arrive at the correct angle in which vertex 26 will travel (blue arrow).

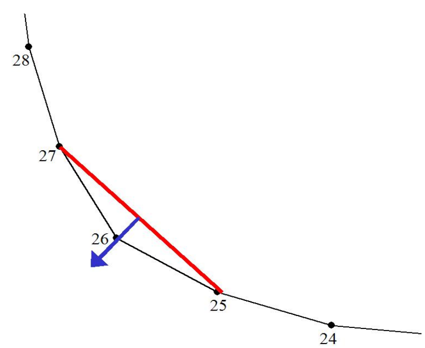


Figure 4. Direction of vertex movement in the Cave Growth Application

The x coordinate value of the new position of the vertex is ascertained by multiplying the dissolution distance (mm) with the sine of the bearing and adding that value to the existing x. Similarly, a vertex's new y position is determined by multiplying the dissolution distance with the bearing's cosine and adding it to the existing y. The effect of this process is visible in Figure 6, in which the display grid is focused on a section of a cave that has experienced an exaggerated amount of growth to illustrate this point.

In this manner, each underwater vertex is moved perpendicular to the existing

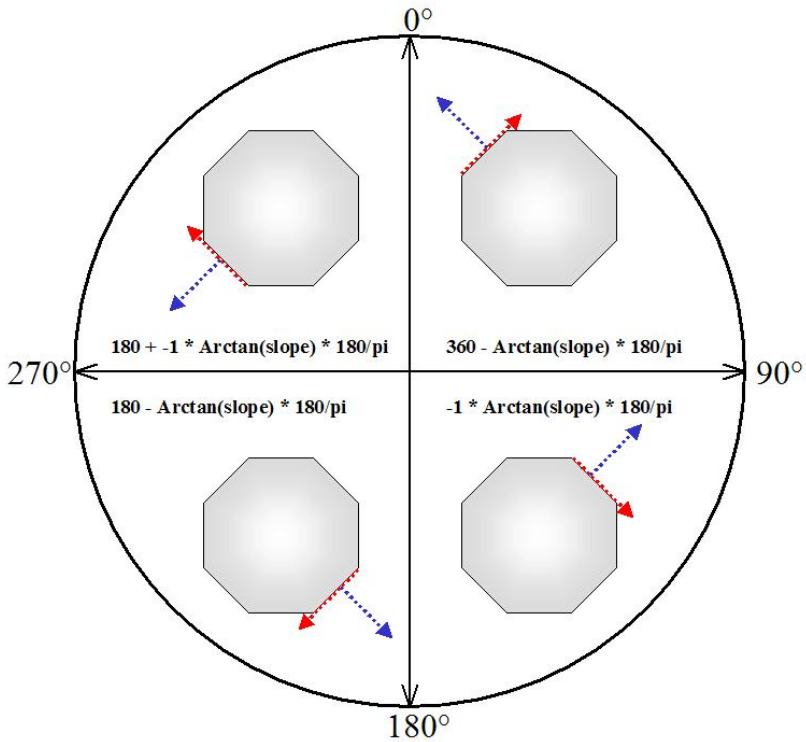


Figure 5. Formulas used in Cave Growth to deduce the angle of vertex growth (blue arrows) from the angle connecting a vertex's neighbors (red arrows).

cave wall at that location for each time step. The movement of an individual vertex is termed a "Vertex Event" in the Cave Growth application and the number of these events that occur over a simulation run is tracked and recorded in the message panel. Accumulating the distances traveled in each vertex event during a simulation run and dividing this value by the total number of events yields an average dissolution rate for the events that is also displayed in the message panel.

In addition to the application structure, this analysis was dependent upon the construction of a set of simulation datasets with realistic properties. Relationships that existed within a field generated dataset (Groves and Meiman, 2005) were used to develop a series of simulation datasets demonstrating a range of flow distributions. In turn, these flow distributions, based on a discharge-stage relationship specific to that field dataset, are presumed to result from a range of precipitation distributions varying from completely uniform to one in which almost all flow for the year came as a result of a few stormy hours. Manipulating climatic conditions in this manner, while holding total annual flow constant, allowed their impact on the amount of cave growth to be measured.

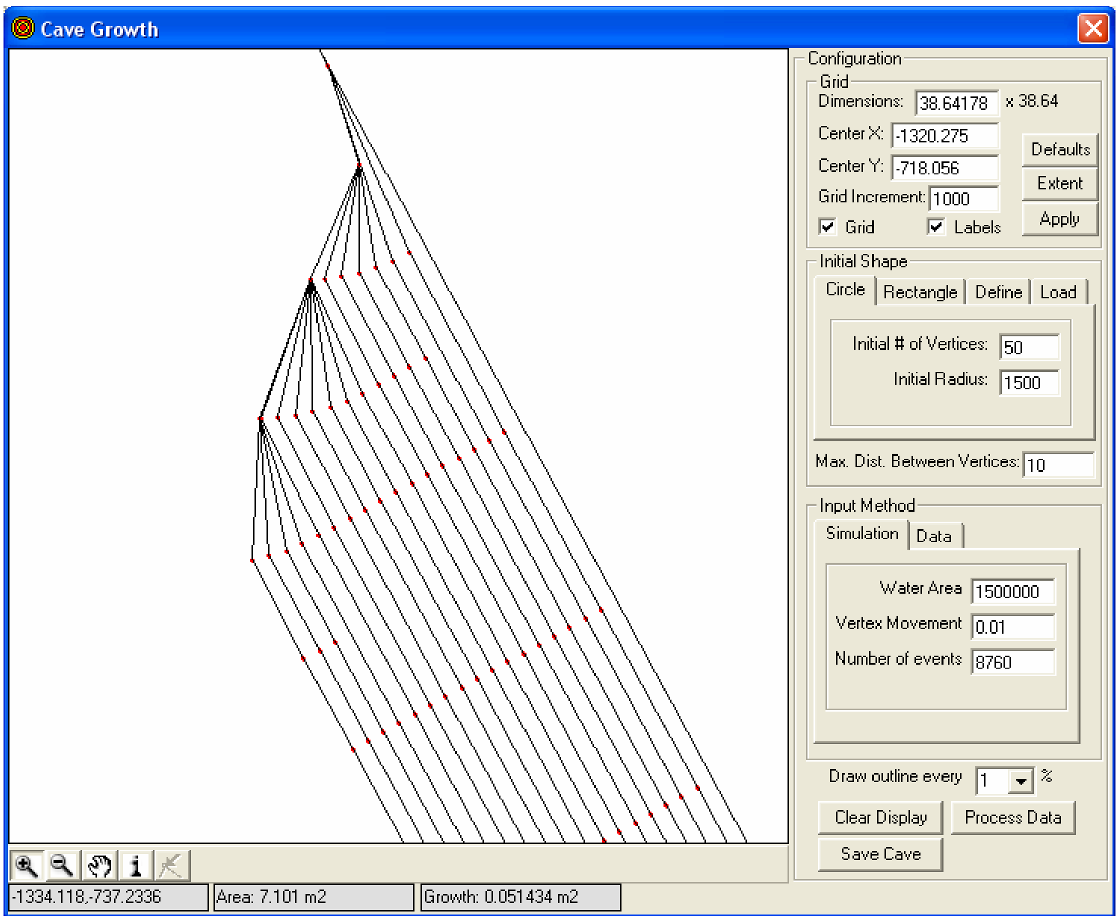


Figure 6. Insertion and movement of vertices by Cave Growth application

To generate the field data used as the basis for this investigation, Groves and Meiman (2000, 2001, 2005) conducted high-resolution hydrochemical monitoring at a study site in Logsdon River, a major underground stream within the Mammoth Cave System that drains the 25 km^2 Cave City groundwater basin. At this site (shown in Figure 7), there are two 145 m deep observation wells, one for collecting water samples

Figure 7. Observation wells at the Logsdon River study site, Mammoth Cave, KY
Photo: Chris Groves

from the surface with a pump while the other contains electronic probes that collect data on stage, velocity, temperature, and specific conductance. A Campbell CR10 multi-channel data logger queried the four probes every two minutes and recorded changes in their values. Over one year, between May 5th 1995 and May 4th 1996, 21,473 observations were made in this manner. Throughout the year and under a variety of flow conditions pH, calcium, and bicarbonate were physically measured from water samples taken at the site. Regression analysis determined a linear relationship between these three factors and specific conductance. As specific conductance was measured every two minutes, these relationships combined with temperature values allowed a dissolution rate to be calculated for each time step using the rate law of Plummer *et al.* (1978), where

$$Rate = k_1[H^+] + k_2[H_2CO_3*] + k_3[H_2O] - k_4[Ca_2^+][HCO_3^-] \quad (1)$$

with *Rate* expressed in mass of mineral lost per time per surface area of fluid/mineral contact, and where the *k*'s are temperature dependent kinetic rate constants (Plummer *et al.*, 1978). For input into the Cave Growth program used in the current research, dissolution rates were expressed as the rate of conduit wall retreat (mm/yr) following the example of Palmer (1991), assuming a constant calcite density of 2.7 g/cm^3.

A spreadsheet was derived from this detailed field data that contained columns for time (Julian days), flow cross-section area (m^2), a rate of passage wall retreat (mm/yr), and mean velocity (ft/s). The flow cross-section area had been determined by comparing the water level recorded by the stage probe with a detailed chart mapping out the actual cross-section of the Logsdon River conduit at the study site. From these data, discharge values were determined for each observation. Subsequently, the discharge units were converted into liters per second and multiplied by the interval in seconds that the record represented to arrive at a measurement of flow in liters for each record. Using this approach, it was determined that an estimated 12,255,649,896 liters of water had passed through the conduit over the course of a year. It was decided that for the "virtual" datasets created for this project the total annual flow would be held constant at this realistic level for each scenario. By controlling total flow, but varying its temporal distribution throughout the year in each dataset, it was possible to isolate the impact of this variable.

Each simulation dataset generated for this research contained a year's worth of data broken into 8,760 discrete records representing equal time increments of one hour. The challenge was to apportion the total annual flow across these time steps in patterns that satisfactorily represented different flow distributions. Furthermore, a timestamp in Julian years, a dissolution rate, and a wetted cross-sectional area had to be calculated for each record in these datasets.

As it has been shown to model precipitation distribution and other climatic variables effectively (Mooley, 1973; Ison *et al*., 1971; Thom, 1958), the "Gamma Distribution" was selected as the method for allocating the hourly flow values. The gamma distribution is a non-symmetric, continuous probability distribution with one-parameter (shape), two-parameter (shape and scale), and three-parameter (shape, scale, and location) versions (Aksoy, 2000). In this research, the statistical analysis software "S-Plus" was used to generate random flow values. A simple S-Plus script (Appendix II) was written based on that application's "rgamma" function. This function follows the two-parameter form of the gamma distribution and requires as inputs an alpha (shape) value, a beta (scale) value, and a total number of values to generate (8,760 in this case). By holding beta constant with a value of one and increasing the alpha parameter exponentially, it was possible to generate a series of flow distributions ranging from almost uniform to extremely concentrated. The script then scaled these random values to sum up to the required total flow. Holding beta at one and not including a location value effectively reduced the gamma probability distribution function to its simpler one-parameter form, also known as the standard gamma distribution (Nastos and Zerefos, 2007) that can be expressed as:

$$f(x) = \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}; x \geq 0 \qquad (2)$$

where α is the shape parameter and $\Gamma(\alpha)$ is the gamma function defined by the following integral:

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-x} dx; \alpha > 0 \tag{3}$$

In this manner, the shape could be isolated and manipulated for each simulation dataset.

To further tie the simulations to the Logsdon River data, distributions were limited to those producing hourly discharge rates that did not exceed 30,000 l/s, a ceiling derived from the approximated highest hour of flow found in the measured field data. Flow values were sorted in ascending order to facilitate the visual comparison of the associated discharge distributions. A selection of these distributions is displayed in figures 8a-8d demonstrating a broad range of flow patterns. Due to the extremely small magnitude of the conduit growth occurring over one year, it was assumed for this project that the influence of the order of these discrete dissolving events was negligible.

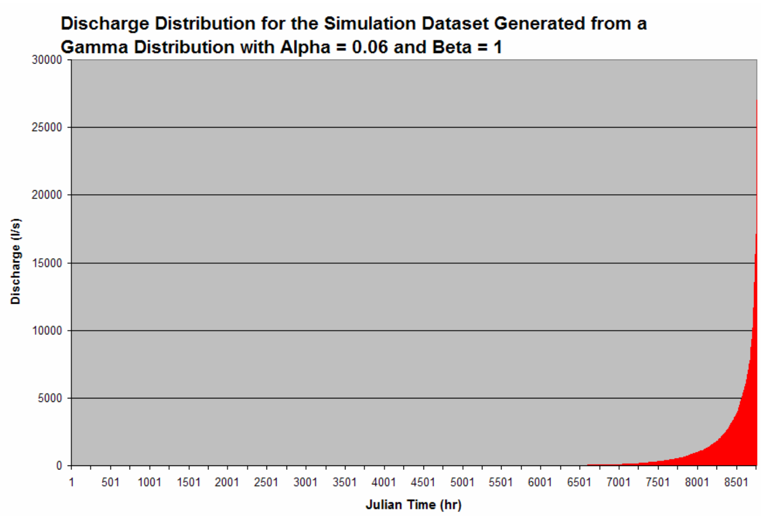


Figure 8a

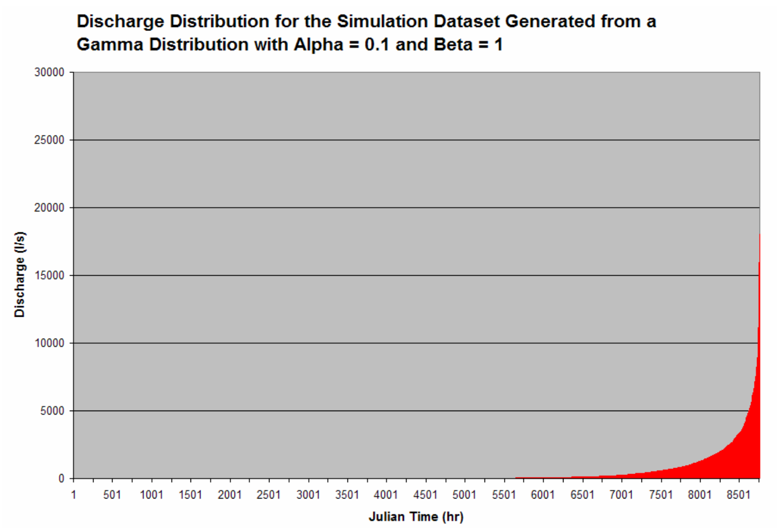


Figure 8b

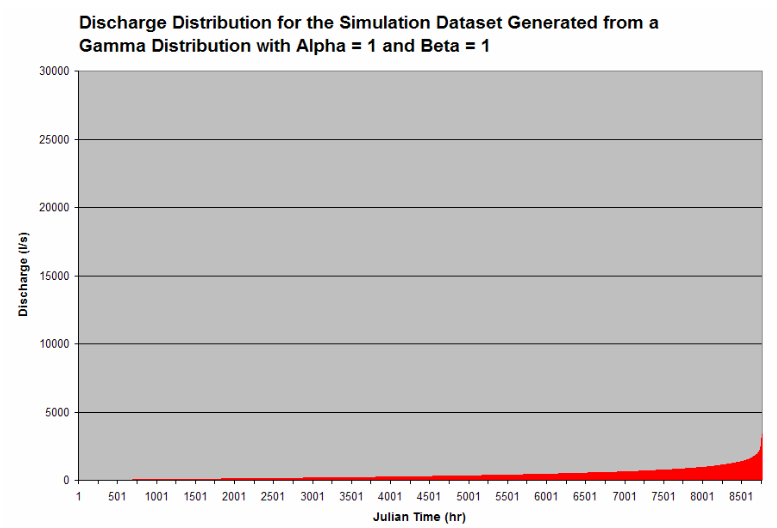


Figure 8c

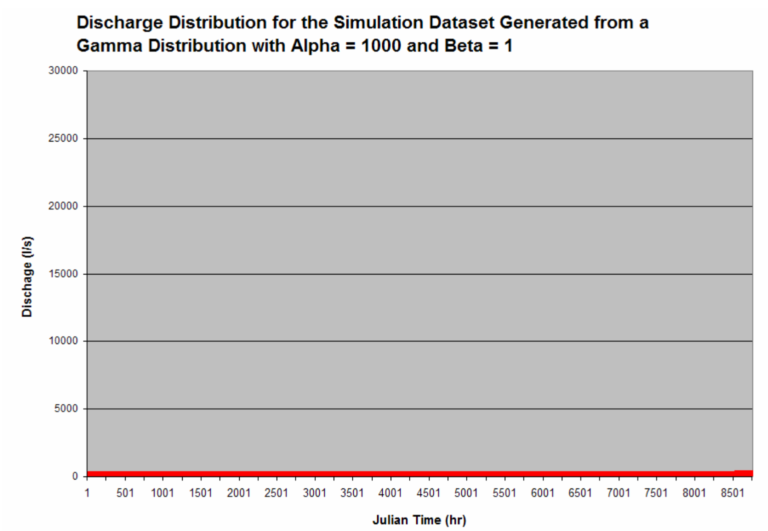


Figure 8d

A Visual Basic program (Appendix III) was written to generate Excel spreadsheets from these flow distributions and populate the three fields required by the Cave Growth application (Julian time, dissolution rate, and wetted cross-sectional area) using relationships found in the field data. First, discharge was obtained by dividing hourly flow values by 3,600 to produce a rate in liters per second. Regression analysis conducted on the Logsdon River data using the "Exponential rise to maximum" curve fitting method in the SigmaPlot software package produced the following formula that related "y" the dissolution rate (mm/yr) to "x" discharge (l/s).

$$y = 0.3412(1 - e^{-1.168*10^{-4} x}) \quad (5)$$

A scatter plot demonstrating this relationship in the original field data is shown in Figure 9. The r^2 value of 0.65 confirms a relationship between the dissolution rates and discharge that makes physical sense as higher discharges *tend* to produce waters more undersaturated with respect to calcite. There is noise in the relationship however,

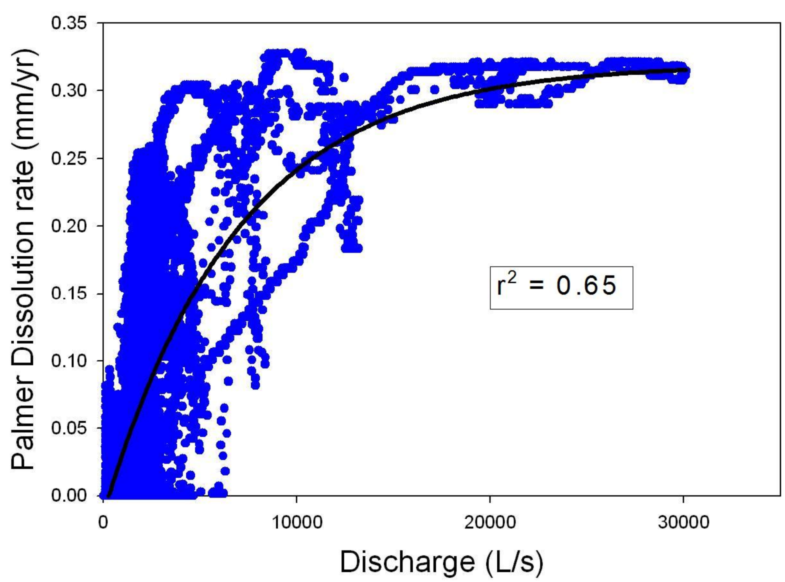


Figure 9. Dissolution rate versus discharge in the Logsdon River field dataset

primarily because water chemistry, on which dissolution rates are based, is not a unique function of discharge.

Further analysis of the field data established a two-part relationship between wetted cross-sectional area and the discharge rate. This relationship, developed from the inverted trapezoid passage at Logsdon River was applied to each of the three passage geometries in the simulation sets considered in this research. It was determined that discharge rates above 3,500 l/s would completely fill any of these 21 m^2 conduit geometries with water. Therefore, the program generating the virtual datasets simply ascribed a constant wetted cross-sectional area representing a full conduit to records with discharge rates at or above this limit. Below that threshold, the cross-sectional area could be described as a function of discharge using the formula below, which again was generated using SigmaPlot's regression analysis tools. This formula defined a relationship with an r^2 value of 0.781.

$$y = 9.284 * 10^{-3} x + 1.867 \tag{6}$$

Based on a wide spectrum of alpha inputs to the gamma function, forty-four datasets were created in this manner exhibiting a broad range of flow distributions. Two special simulation sets were also created to represent unique theoretical circumstances. The first, named "Even" in this discussion, contained a completely uniform distribution of flow representing a climate in which precipitation and the resultant discharge through the karst aquifer were non-varying throughout the year, with a total equal to the annual Logsdon River flow discussed above. The second dataset, referred to here as "Pipefull", was one in which full-passage conditions (Figure 10b) were met, but not exceeded, for the greatest length of time possible given the total amount of flow. The term “pipefull” in this research refers to flow conditions that fill the conduit exactly. This idea borrows from the conceptually similar “bankfull” stage in surface stream modeling in which a stream is full to the brim of its banks without actually overflowing them (Figure 10a). In fluvial geomorphology, this stage has been described as the discharge level “at which channel maintenance is most effective, that is, the discharge at which moving sediment, forming or removing bars, forming or changing bends and meanders, and generally doing work that results in the average morphologic characteristics of channels" (Dunne and Leopold, 1978, p608-609). While these surface processes are mechanical, the “pipefull” dataset was included in this study to consider if this “exactly full” discharge level had an influence on the effectiveness of work conducted in karst conduits by chemical dissolution.

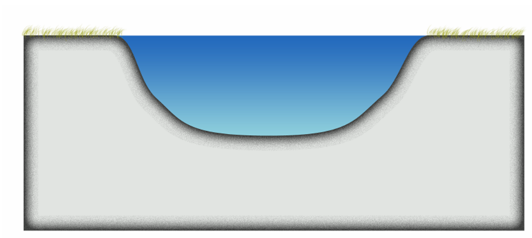


Figure 10a Bankfull stage in a surface stream.

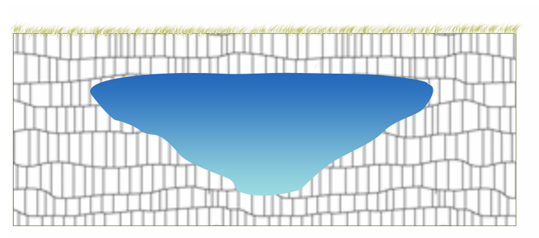


Figure 10b Pipefull stage in a karst conduit

To create the “Pipefull” distribution, the amount of flow required to fill the 21m^2 inverted-trapezoid for one hour was determined. The total annual flow was then divided by this value to calculate the number of records that would be given this “pipefull” hourly flow value of 7,515,000 liters. After the annual flow was allocated in this manner, the remaining flow was assigned to one record and all other time steps did not receive any flow.

Following the creation of the datasets, the simulation runs were configured and executed within the Cave Growth application. Three separate runs were generated from each of the 46 simulation datasets; one for each initial passage shape examined (a $21m^2$ trapezoid, circle, and rectangle.) In each run reported here, the configuration options were held constant (Figure 11). The maximum distance between vertices was set to 100 mm, and the time step was set to 60 minutes. At the completion of each run, the name of the data file, the initial geometry, and the measure of flow distribution (alpha input into the gamma function used to generate that dataset) were recorded in a spreadsheet. Corresponding values for total 2D passage growth, the average dissolution rate, and the number of "vertex events" were also recorded.

Conduit Geometry	Vertex Spacing	Time Step	Data
$21m^2$ Trapezoid	100mm	60 minutes	46 Simulation datasets: Pipefull, Even, and 44 Γ distributions
$21m^2$ Circle	100mm	60 minutes	46 Simulation datasets: Pipefull, Even, and 44 Γ distributions
$21m^2$ Rectangle	100mm	60 minutes	46 Simulation datasets: Pipefull, Even, and 44 Γ distributions

Figure 11. Configuration options used in the simulation runs

RESULTS

The results presented in this section are an effort to display the output from the 138 simulation runs (three runs for each of the 46 datasets) in a manner that addresses the question of whether the "storminess" of a climate's precipitation distribution has an effect on the evolution of karst aquifers. To accomplish this task, it is necessary to compare the amount of cave growth that occurred during each simulation run with a measure of climatic variability for that run. Climatic variation in this analysis is represented by the annual temporal distribution of flow within the aquifer and can be viewed in the form of discharge distribution graphs such as those in Figures 8a-8d. It is assumed, for the sake of this study, that the distribution of flow is directly related to the patterns of recharge producing precipitation occurring at the surface. It is understood that factors such as the delayed release of precipitation held in snow accumulation can confound this assumption in the real world.

The variability of discharge distributions within the simulation runs discussed here is related to the alpha (shape) parameter input into the gamma distribution formula from which the underlying datasets were derived (beta being held constant). Datasets with lower alpha input values reflect climates in which the distribution of precipitation is increasingly more discrete storm-driven and thus less uniform. Accordingly, the alpha input values were recorded for each dataset and used as a measure of climatic variability in the following graphs. As the pipefull and even datasets were not created from the gamma formula, faux alpha values of 0.225 and 20,000 respectively were generated for them so that they could be displayed on the graphs. A relationship between the alpha values and the standard deviation of the flow distributions in the gamma datasets provided a method to derive these coarse values.

The following graph (Figure 12) shows the two dimensional growth of the cave passage as a function of flow variability, or how storm-driven the climate is. The x-axis uses a logarithmic scale to display the input alpha values in a meaningful manner. Runs based on the three different initial passage geometries are identified by different symbols.

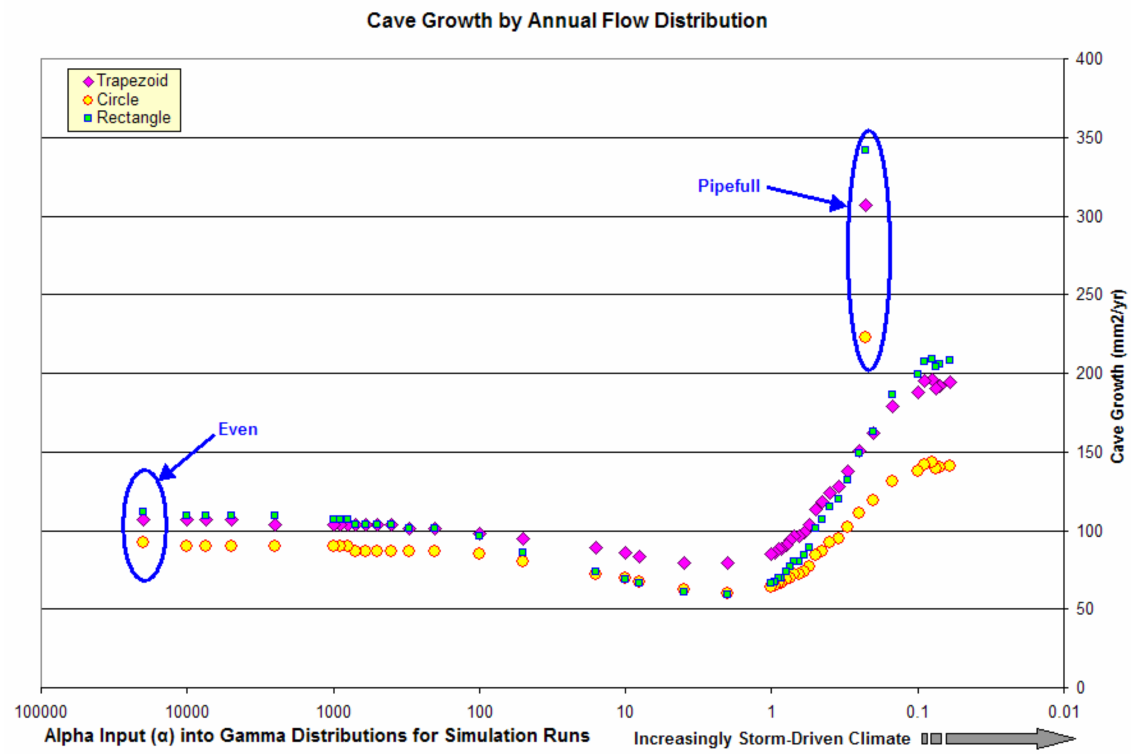


Figure 12. Cave growth vs. annual flow distribution.

Another quantity that was recorded was the number of "vertex events" that transpired during each simulation run. A "vertex event", in this research, is each occurrence of a vertex being moved as part of the Cave Growth application's modeling of passage wall retreat. For each time step in the simulation run, every vertex that is in contact with dissolving water, and is consequently moved, is counted and added to the total number of "vertex events". This measure can be considered an expression of the amount of wetted perimeter in the conduit multiplied by time over the course of a simulation run. On the graph below (Figure 13), the number of vertex events for each simulation run are plotted against climatic variability for the three different geometries considered.

An average dissolution rate, measured in mm/year was also recorded for every

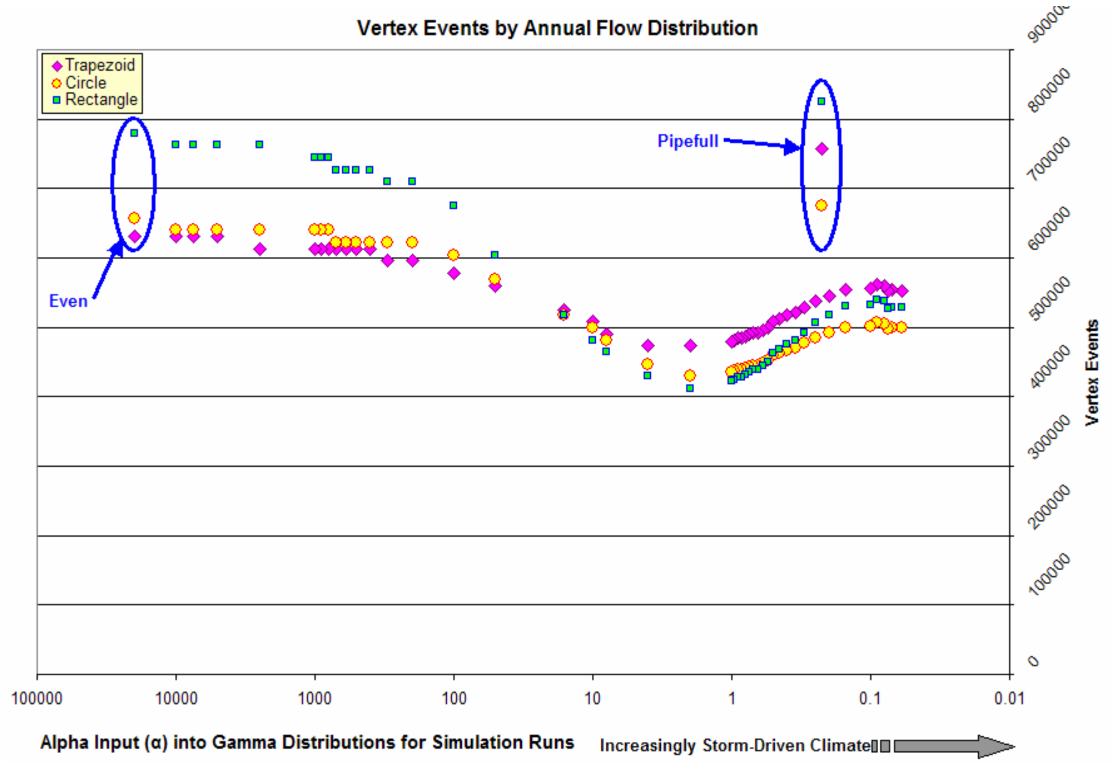


Figure 13. Vertex events vs. annual flow distribution.

simulation run processed by the Cave Growth application. For every vertex that the program moved during a simulation run, the dissolution rate associated with that particular vertex event was accumulated. At the end of the run, this total was divided by the number of vertex events to generate the average dissolution rate. Once calculated, the average dissolution rate for each run was plotted against the non-uniformity of a climate's precipitation distribution, in the same manner described for generating the previous graph. The resulting graph of the relationship between average dissolution rate and climate is displayed below in Figure 14.

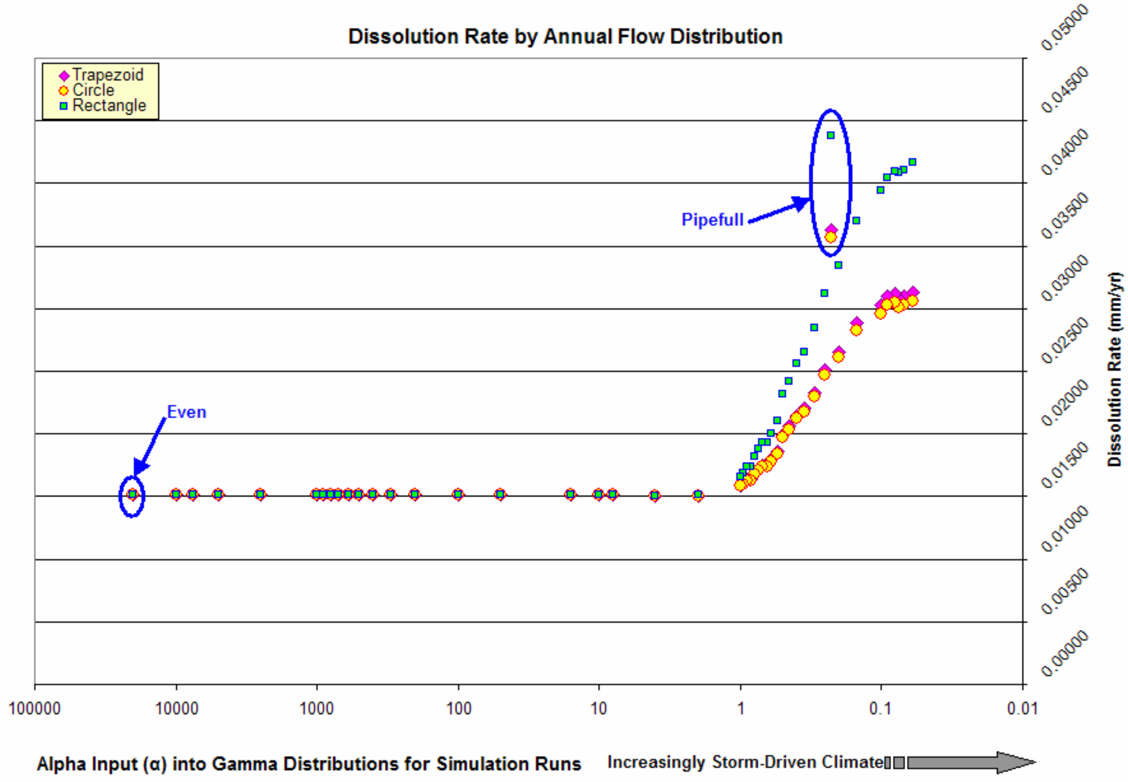


Figure 14. Dissolution rate vs. annual flow variation.

DISCUSSION

Before discussing the specific details of this research, it is worthwhile reiterating that a computer model, such as that presented here, allows researchers to isolate and study certain aspects of a system. It is understood that this model does not consider all the factors that affect the amount of cave growth that takes place within a karst system in the real world. For instance, in a real karst conduit the bedrock is not uniform and fractures in the limestone are exploited by water in ways that are not currently considered by the Cave Growth application. Similarly, conduits may contain a sediment layer across their floor that can act as a barrier to dissolution. If required, the application could be modified to consider these and other aspects of the physical world.

Another acknowledgment is that using annual discharge distributions to differentiate climates assumes a direct relationship between recharge producing precipitation on the surface and discharge rates within the aquifer. While evapotranspiration, snow accumulation, overland flow, and other factors can distort this relationship over different time scales, this approach remains a useful tool by which to represent climate. However, if discharge data were collected from field studies across a range of climates and normalized for total flow a more comprehensive analysis could be conducted.

A relationship between the amount of two dimensional cave passage growth and the "storminess" of the climate is clearly discernable from the graph of Cave Growth by Annual Flow Distribution (Figure 12). Indeed, a similar pattern is visible for all three sets of runs that are each based on different original passage geometries. From a starting point at the "even" simulation run, the observed amount of cave growth steadily declines as the climate represented by the simulation runs becomes progressively more storm-driven. This continues until an inflection point is reached, after which, observed cave growth increases as precipitation is concentrated into fewer, more intense, intervals until an apparent peak is reached. Beyond this, growth levels off and begins to decline slightly. For all three sets of runs, the passage growth generated by the pipefull dataset exceeds that of the even and gamma distribution runs by more than one third. It does not align with the other sets of results because its flow values do not follow the same type of distribution pattern. To graph the variability of flow distributions that are unrelated to the gamma formula, a different measure could be developed.

Based on the evidence visible in this graph, the relationship between climate and conduit growth in karst aquifers cannot be explained as one in which either 1) conduit growth is independent of rainfall and groundwater flow distributions, or in which 2) stormier climates simply equate to more conduit growth occurring. To better comprehend the inflections of the data exhibited on the graph described above, it is necessary to consider factors that are influenced by climate that impact the dissolution of the walls of a karst conduit. In the virtual environment discussed here, it should be possible to derive an explanation of the observed pattern from a closer examination of the limited number of variables that are considered by the Cave Growth application. Therefore, the average amount of wetted perimeter and the average dissolution rate (mm/yr) observed in each simulation run were studied in greater detail.

To consider the amount of passage wall exposed to the effects of dissolution, under different climatic conditions, the total number of vertex events for each run was graphed against the non-uniformity of flow distributions in Figure 13. It is apparent from this graph that climates in this study with more uniform annual precipitation distributions actually generated the most vertex events, except when compared to the results for the pipefull scenario. For each of the three sets of simulation runs, the average amount of wetted perimeter measured begins to decline as flow distributions becomes more concentrated. After a turning point is reached, the number of vertex events generated in a run increase as the simulated climates and associated patterns of flow become more storm-driven. Finally, a peak is reached and is then followed by a gradual decline.

To attempt to understand the nature of this relationship, it is useful to focus on the influence of the passage geometry on the amount of wetted perimeter that exists under given flow conditions. For instance, a very small amount of water flowing through a conduit with a wide level floor can contact a comparatively large amount of rock surface. As flow increases, the wetted perimeter will increase in relation to the geometry of the cave. If the passage widens as the water's depth increases, there will be a diminishing rate of return in terms of vertex events compared to any increase in flow as more of the underground river surface is exposed to the air rather than being in contact with the cave walls. Conversely, if the passage narrows as the depth increases, each additional unit of flow will generate progressively more wetted perimeter. After pipefull conditions are met, and the entire passage is in contact with the river, any increase in flow will just be converted to greater velocity in this model, as there is nowhere for the extra water to go. The wetted perimeter generating capacity of this additional water in the system is essentially "wasted" for discharges above that level.

Logically, more vertex events will be generated by a simulation run with flow conditions that maximize the proportion of wetted perimeter to the amount of flow cross-section area, for the greatest length of time. This situation is essentially a description of the pipefull simulation dataset explained earlier in this research. Depending upon the geometric details of their passages, changes in flow conditions across the study datasets induce different responses from the three simulation sets. However, once flow conditions are such that the entire passage is full, the excess water flowing during those stormy hours is wasted equally for each scenario.

The initial decline in the number of vertex events as the climates become more storm-driven is caused by more time in the simulation runs being spent in conditions that are geometrically inefficient at generating vertex events. Interestingly, the point on the graph where this decline ends, and the number of vertex events begin to increase with the non-uniformity of flow conditions, closely coincides with the first occurrence of the pipefull stage being met within the simulation datasets. The gamma dataset with an input alpha value of two is the first with time steps that contain enough flow to completely fill the conduit with water. The impact of the vertices along the cave roof being brought into contact with water varies according to the shape of the passage. Due to its wide level roof, the trapezoidal passage is affected most by this event. Consequently, the number of vertex events generated by the trapezoidal runs shifts from being the least to the most productive of the three simulation sets after the appearance of the pipefull events.

Those runs based on a rectangular passage generated significantly greater numbers of vertex events under the more stable of the simulated climates compared to the corresponding trapezoidal and circular runs. For example, 762,120 vertex events were observed for the simulation run based on the rectangular passage and the gamma dataset created with an alpha input of 1,000 and a beta input of one. The same dataset produced only 639,480 vertex events when the circular geometry was processed and just 630,720 when the initial geometry was the trapezoid (a reduction of more than 17%). The shape of the tall narrow rectangular passage meant that it did not lose much of its vertex event generating ability at these steady flow rates that filled only about a quarter of the passage. Comparatively, the circular and trapezoidal passage geometries resulted in much wider exposed water surfaces at these flow levels and, as a result, generated less wetted perimeter per unit of flow.

For all three sets of runs, as the number of time steps meeting or surpassing pipefull conditions rises with progressively stormier climates, the amount of vertex events also increases. It is important to remember that because all the simulations carry the same amount of total flow, higher flow conditions in these datasets are counteracted by more time also being spent under lower flow conditions. It is the interaction between the full spectrum of flow conditions in a simulation run and a specific passage geometry that culminate in the total number of vertex event produced. As the precipitation distribution within a dataset becomes more extreme, all three sets of runs eventually reach a second inflection point after which the average amount of wetted perimeter begins to level off. This is interpreted to mean that, after this point, any gains from meeting pipefull conditions are more than offset by wastage of vertex-event productivity during extreme storm events as discharge continues to increase above the exactly pipefull stage.

The average dissolution rate for the vertex events that took place in a simulation run is the other variable that warrants further exploration in this effort to understand the relationship between cave growth and climate in this study. Accordingly, this rate, measured in mm/yr was graphed against the non-uniformity of flow distributions in Figure 14. The relationship between the variables on this graph begins with an almost imperceptible decline in the average dissolution rate for all three simulation sets as the flow distributions contain more concentrated discharge events until a low point is reached. Subsequently, average dissolution rates increase sharply in response to higher dissolution rates associated with augmented flow conditions combined with greater numbers of vertex events occurring during those high flow events. Eventually, this increase levels off and a gradual decline ensues which is interpreted to be a result of the wasting of vertex events at higher flow levels described above. This would help explain why the pipefull scenario, which fills the passage to capacity with the least wastage of water, generated noticeably higher average dissolution rates for the vertex events than those produced by any of the gamma runs.

It is apparent from this discussion that the total number of vertex events and the average dissolution rates generated by a simulation run are closely connected. The combination of these factors helps explain the relationship between cave growth and the intensity of the annual flow distribution exhibited in the graph in Figure 12. At first, despite high rates of contact between water and the cave walls, the dissolution rates are comparatively low resulting in low levels of annual cave growth. Cave growth declines further as flow distributions tend toward those that are less efficient at generating wetted perimeter and still exhibit low average dissolution rates. The point of inflection on the graph where cave growth begins to rise with progressively more storm driven climates is that at which both average dissolution rates and the number of vertex events begin to increase. Annual cave growth eventually tapers off and begins to decline because the amount of wetted perimeter again starts to decrease as flow becomes too concentrated and pipefull stage is surpassed, wasting more of the dissolving power of the annual flow budget.

The different ranges of growth exhibited by the three simulation sets appear to be related to differences in the perimeters of their passage geometries. While all three conduit cross-sections in this investigation encompass the same $21m^2$ area, each has a different perimeter and ultimately a different degree of exposure to the dissolving effects of water when filled to capacity. The rectangular conduit has the largest perimeter at 25m and it also has the largest maximum cave growth ($342mm^2/yr$). In comparison, the trapezoidal cave has a smaller perimeter of roughly 22.5m and lower maximum growth values. The circular passage generated the least amount of wetted perimeter under pipefull conditions having a circumference of only about 16.25m. Indeed, the simulation runs for the circular passage generated both the smallest range of growth values and the lowest maximum cave growth of only 223mm^2/yr, more than a third less than the maximum for the rectangular simulation runs.

After examining the graphs, it is clear that the amount of cave growth occurring in a passage is affected by the intensity of precipitation events throughout a year. It is the interrelationship between the annual temporal distribution of discharge, the geometry of the passage, and the dissolution rate that influences the amount of dissolution that takes place. Climates that generate the most wetted perimeter per unit of flow at higher average dissolution rates appear to be those that conduct the greatest amount of "geomorphic work" in terms of conduit growth. In the case of a specific cross-section of a karst conduit, discharge distributions filling that passage to the brim for extended periods prove to be most effective at dissolving its walls.

It is important to realize that the flow conditions required to fill a particular section of an individual conduit with water does not represent pipefull conditions for the entire, three-dimensional, underground river basin. However, it could be conjectured that the same principals would apply to an entire system. Those climates with precipitation distributions that generate the highest combination of wetted perimeter and dissolution rates throughout the entire system would be most efficient at developing the drainage network.

The amount of flow required to meet these "network-full" conditions would depend on the volume and geometry of connected void space within the vadose zone of the aquifer. Logically, a younger system with a less developed drainage network would be less able to accommodate the amount of flow generated by storms. Therefore, much of the dissolving power of this water would be lost to overland flow or simply higher velocities of water passing through the cave passages. In contrast, based on the observations made in this analysis, it may be inferred that cave growth in mature systems with well-developed passages would respond more quickly to a precipitation distribution concentrated into storm events large enough to fill the passages to capacity.

CONCLUSION

Clearly, in an aquifer characterized by its soluble rock, the total amount of flow passing through the system affects the amount of dissolution that takes place. Therefore, climates that experience more precipitation and generate more groundwater flow will tend to give rise to more rapid development of underground drainage networks. Less obvious is the idea suggested by this research, that the temporal distribution of that flow, combined with the geometric configuration of the conduit network, can also have an important effect on the amount of geomorphic work (passage expansion) conducted in these subterranean drainage basins. Specifically, more dissolution occurs under precipitation conditions that result in more wetted perimeter for longer periods of time at higher dissolution rates. The results of this research suggest that climates exhibiting moderate storm events are more "efficient" at growing karst aquifer conduits than those in which precipitation and consequently flow are more evenly distributed.

Ultimately, understanding the impact of the temporal distribution of flow on the growth of karst conduits provides geoscientists with another tool to help interpret the rate of development of underground drainage networks, surface karst landscapes, and indeed overall landscape denudation. At a more specific level, knowing the frequency and "dissolving efficiency" of different flow levels as they relate to an individual cave may help explain its particular growth rate. Further refinement of the Cave Growth program could provide a more accurate understanding of these factors by including additional variables that affect the development of caves. Much would also be gained by ground truthing the findings presented here through analysis of field data on flow distributions and conduit growth collected from karst regions around the globe.

Figure 15. Very large river cave passage in Da Long Dong (Big Dragon Cave) in western Hunan Province, southwest China. In some places the width of this passage exceeds 100 meters. Photo: Kevin Downey

REFERENCE LIST

Aksoy H., 2000, Use of Gamma Distribution in Hydrological Analysis. *Turkish Journal of Engineering and Environmental Sciences* 24: 419-428.

Ding Y., 1993. *Monsoons Over China*. Kluwer Academic Publishers, Netherlands

Dunne T, Leopold LB. 1978. *Water in Environmental Planning*. W.H. Freeman Co.: San Francisco

Ford DC, Williams PW. 2007. *Karst Hydrogeology and Geomorphology*. Wiley: New York

Groves C, Meiman J. 2000. Regional atmospheric carbon sink within the south central Kentucky karst. In *Proceedings of the 8th Mammoth Cave Science Conference*. National Park Service: Mammoth Cave, KY; 131-141.

Groves C, Meiman J. 2001. Inorganic carbon flux and aquifer evolution in the south central Kentucky karst. *U.S. Geological Survey Water-Resources Investigations Report 01-4011*. 99-105.

Groves C, Meiman J. 2005. Weathering, geomorphic work, and karst landscape evolution in the Cave City groundwater basin, Mammoth Cave, Kentucky. *Geomorphology* 67: 115-126.

Ison NT, Feyerherm AM, Bark LD. 1971. Wet Period Precipitation and the Gamma Distribution. *Journal of Applied Meteorology* 10: 658-665.

Kiefer RL. 1990. Carbon Dioxide Concentrations and Climate in a Humid Subtropical Environment. *Professional Geographer* 42: 182-194.

Mooley DA. 1973. Gamma Distribution Probability Model for Asian Summer Monsoon Monthly Rainfall. *Monthly Weather Review* 101: 160–176.

McCuen R. 2002. *Modeling Hydrologic Change: Statistical Methods*. Lewis Publishers: Boca Raton

Nastos PT, Zerefos CS. 2007. On Extreme Daily Precipitation Totals at Athens, Greece, *Advances in Geoscience* 10: 59-66.

Palmer AN. 1981, *A Geological Guide to Mammoth Cave National Park*. Zephyrus Press: Teaneck, NJ

Palmer AN. 1991. The origin and morphology of limestone caves. *The Geological Society of America Bulletin* 103: 1-21.

Plummer LN, Wigley TML, Parkhurst DL. 1978. The kinetics of calcite dissolution in CO_2-water systems at 5 to 60^{o}C and 0.0 to 1.0 atm CO_2. *American Journal of Science* 278: 179-216.

Smith DI, Atkinson TC. 1976. Processes, landforms, and climate in limestone regions. In *Geomorphology and Climate*, Derbyshire E (ed.). Wiley: Chichester, UK; 367-409.

Thom, HCS. 1958. A Note on the Gamma Distribution. *Monthly Weather Review* 86: 117–122

Usery EL, Finn MP, Scheidt DJ, Ruhl S, Beard T, Bearden M. 2004. Geospatial data resampling and resolution effects on watershed modeling: A case study using the agricultural non-point source pollution model. *Journal of Geographical Systems* 6: 289-306.

White WB. 1988. *Geomorphology and Hydrology of Karst Terrains*. Oxford University Press: New York

White WB, Watson RA, Pohl ER, Brucker R. 1970. The Central Kentucky Karst. *Geographical Review* 60: 88-115.

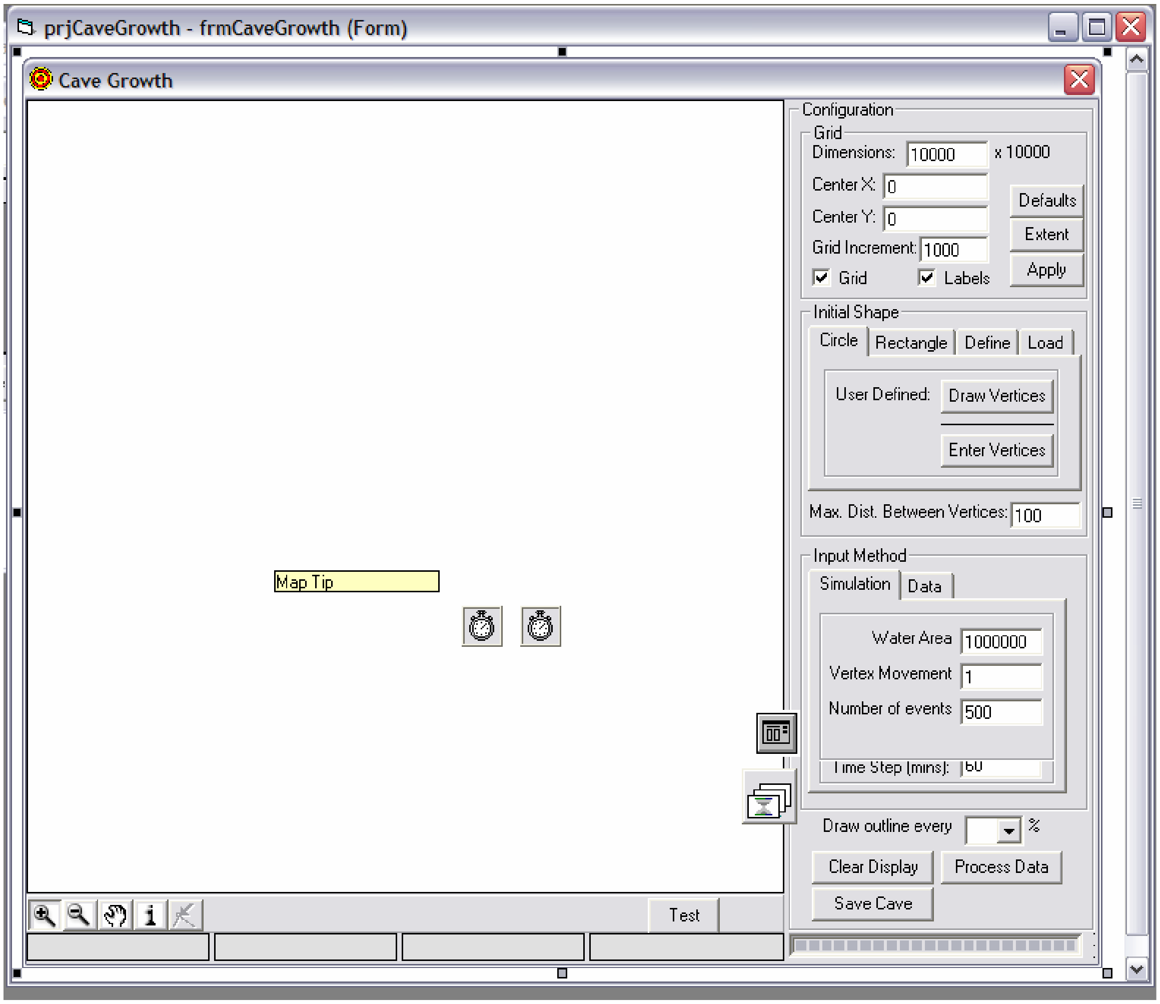
Yuan D. 1988. Environmental and engineering problems of karst geology in China. *Environmental Geology and Water Sciences*. 12: 79-87.

Yuan D. 1991. *Karst of China*. Geological Publishing House: Beijing

APPENDIX I

Cave Growth Application Code

VB6 Project: prjCaveGrowth
Form: FrmCaveGrowth




```
Option Explicit
Private WaterArea As Double
Private WaterLevel As Long
Private Pi As Double
Private TErosionDist As Double
Private CenterX As Double
Private CenterY As Double
Private Radius As Double
Private CurX As Single
Private CurY As Single
Private TimerX As Single
Private TimerY As Single
Private DataFile As String
Private NumRecs As Long
Private oConn As ADODB.Connection
Private oRS As ADODB.Recordset
```

```
Private ActiveTool As Integer
Private ExtX As Double
Private ExtY As Double
Private OldX As Double
Private OldY As Double
Private InitArea As Double
Private TotalVertices As Long
Private TotalDistance As Double

Private Sub cmdDefs_Click()
  On Error GoTo ErrHandler
  txtDimensions.Text = 10000
  txtCenX.Text = 0
  txtCenY.Text = 0
  txtGridInc.Text = 1000

  picDisplay.ScaleTop = Val(txtCenY.Text) + Val(txtDimensions.Text) / 2
  picDisplay.ScaleLeft = Val(txtCenX.Text) - Val(txtDimensions.Text) / 2
  picDisplay.ScaleHeight = Val(txtDimensions.Text) * -1
  picDisplay.ScaleWidth = Val(txtDimensions.Text)
  'Calculate position for cave center
  CenterX = picDisplay.ScaleLeft + picDisplay.ScaleWidth / 2
  CenterY = picDisplay.ScaleTop + (-1 * picDisplay.ScaleHeight / 2)

  'Calculate Grid
  picDisplay.Cls
  DispGrid.XDiv = Val(txtGridInc.Text)
  DispGrid.YDiv = Val(txtGridInc.Text)
  DispGrid.DrawGrid picDisplay, chkShowGrid, chkShowLabels, &HDCDCDC
  DrawCaveWalls
  Exit Sub
ErrHandler:
  MsgBox "Error in cmdDefs_Click: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Sub cmdEnterVertices_Click()
  On Error GoTo ErrHandler
  picDisplay.Cls
  DispGrid.DrawGrid picDisplay, chkShowGrid, chkShowLabels, &HDCDCDC
  NumVertices = 0
  Erase Vertice()
  Erase CaveWall()
  Erase CaveWalls()
  CaveExists = False
```

```
  frmEnterVertices.Show vbModal
  Exit Sub
ErrHandler:
  MsgBox "Error in cmdAddVertex_Click: " & Err.Number & vbNewLine &
Err.Description
  Resume Next
End Sub

Private Sub cmdFullExtent_Click()
  On Error GoTo ErrHandler
  Dim i As Long
  Dim lX As Double
  Dim hX As Double
  Dim lY As Double
  Dim hY As Double

  If CaveExists Then
    lX = Vertice(0).X
    hX = Vertice(0).X
    lY = Vertice(0).Y
    hY = Vertice(0).Y
    For i = 1 To NumVertices - 1
      If Vertice(i).X < lX Then lX = Vertice(i).X
      If Vertice(i).X > hX Then hX = Vertice(i).X
      If Vertice(i).Y < lY Then lY = Vertice(i).Y
      If Vertice(i).Y > hY Then hY = Vertice(i).Y
    Next i
    If hX - lX > hY - lY Then
      txtDimensions.Text = (hX - lX) * 1.1
    Else
      txtDimensions.Text = (hY - lY) * 1.1
    End If
    txtCenX.Text = lX + (hX - lX) / 2
    txtCenY.Text = lY + (hY - lY) / 2
    picDisplay.ScaleTop = txtCenY.Text + txtDimensions.Text / 2
    picDisplay.ScaleLeft = txtCenX.Text - txtDimensions.Text / 2
    picDisplay.ScaleHeight = txtDimensions.Text * -1
    picDisplay.ScaleWidth = txtDimensions.Text
    'Calculate Grid
    picDisplay.Cls
    DispGrid.XDiv = Val(txtGridInc.Text)
    DispGrid.YDiv = Val(txtGridInc.Text)
    DispGrid.DrawGrid picDisplay, chkShowGrid, chkShowLabels, &HDCDCDC
    DrawCaveWalls
  Else
    MsgBox "No Cave Exists Yet", , "Undefined Cave"
```

```
  End If
  Exit Sub
ErrHandler:
  MsgBox "Error in cmdFullExtent_Click: " & Err.Number & vbNewLine &
Err.Description
  Resume Next
End Sub

Private Sub cmdLoad_Click()
  On Error GoTo ErrHandler
  Dim CGWname As String
  Dim FileNum As Integer
  Dim NumCaves As Long
  Dim NumVs As Long
  Dim InX As Double
  Dim InY As Double
  Dim i As Long
  Dim j As Long

  picDisplay.Cls
  DispGrid.DrawGrid picDisplay, chkShowGrid, chkShowLabels, &HDCDCDC
  NumVertices = 0
  Erase Vertice()
  Erase CaveWall()
  Erase CaveWalls()
  CaveExists = False
  cdg1.DialogTitle = "Load Cave File"
  cdg1.Filter = "*.cgw|*.cgw"
  cdg1.FileName = ""
  cdg1.ShowOpen
  CGWname = cdg1.FileName
  FileNum = FreeFile
  If CGWname = "" Then Exit Sub
  Open CGWname For Input As #FileNum
  Input #FileNum, NumCaves, NumVs
  ReDim CaveWalls(0 To NumCaves)
  NumVertices = NumVs
  Do While Not EOF(FileNum)   ' Check for end of file.
    For i = 0 To NumCaves - 1
      ReDim CaveWall(0 To NumVs - 1, 1 To 2)
      For j = 0 To NumVs - 1
        Input #FileNum, InX, InY
        CaveWall(j, 1) = InX
        CaveWall(j, 2) = InY
      Next j
      CaveWalls(i) = CaveWall
```

```
    Next
  Loop
  Close #FileNum

  ReDim Vertice(0 To NumVertices - 1)
  For i = 0 To NumVertices - 1
    Set Vertice(i) = New CaveVertex
    Vertice(i).X = CaveWalls(0)(i, 1)
    Vertice(i).Y = CaveWalls(0)(i, 2)
  Next
  InitArea = PolyArea(Vertice())
  ReDim Vertice(0 To NumVertices - 1)
  For i = 0 To NumVertices - 1
    Set Vertice(i) = New CaveVertex
    Vertice(i).X = CaveWalls(UBound(CaveWalls) - 1)(i, 1)
    Vertice(i).Y = CaveWalls(UBound(CaveWalls) - 1)(i, 2)
  Next
  CaveExists = True
  DrawCaveWalls
  txtArea.Text = "Area: " & FormatNumber(PolyArea(Vertice()) / 1000000, 3) & " m2"
  txtDiffArea.Text = "Growth: " & FormatNumber((PolyArea(Vertice()) - InitArea) /
1000000, 6) & " m2"
  Exit Sub
ErrHandler:
  MsgBox "Error in cmdLoad_Click: " & Err.Number & vbNewLine & Err.Description
  Close #FileNum
End Sub

Private Sub cmdProcess_Click()
  On Error GoTo ErrHandler
  Dim i As Long
  Dim j As Long
  Dim JDay As Double 'current julian date in decimal days
  Dim JDay1 As Double 'initial record's julian date stored so that intervals are right from
the start of each loop (may not start at zero)
  Dim PreviousJDay As Double 'variable to store the last julian date processed
  Dim JYrs As Double 'accumulated interval of time that has been processed in decimal
days
  Dim dInterval As Double 'interval of time the current record represents
  Dim JInterval As Double 'inteval of time in decimal days since the last record(s)
were/was processed
  Dim WallNum As Integer
  Dim TempWaterArea As Double
  Dim TempDist As Double
  Dim IntervalCounter As Long
  Dim DRate As Double
```

```
  Dim WArea As Double
  Const MinsInDay As Double = 1440

  TotalVertices = 0
  TotalDistance = 0
  If tabShape.SelectedItem.Index = 1 Or tabShape.SelectedItem.Index = 2 Then 'if circle
or rectangle are chosen
    If txtNumVertices(tabShape.SelectedItem.Index - 1).Text < 4 Then
      MsgBox "A minimum of four vertices must be specified for this program to run.",
vbCritical, "Insufficient Number of Vertices"
      Exit Sub
    End If
  End If

  ReDim CaveWalls(0 To 100 / Val(cbxDraw.Text)) 'set up array to store cave walls
dependent on how many are to be drawn

  tbr1.Buttons(5).Enabled = False 'disable the add vertex button
  frmCaveGrowth.MousePointer = vbHourglass
  txtArea.Text = ""
  txtArea.Refresh
  txtDiffArea.Text = ""
  txtDiffArea.Refresh
  txtTotals.Text = ""
  txtTotals.Refresh
  pbr1.Visible = True
  pbr1.Value = 0
  WaterLevel = 0

  'Display Grid
  picDisplay.Cls
  DispGrid.DrawGrid picDisplay, chkShowGrid, chkShowLabels, &HDCDCDC

  'Calculate and display cave center
  CenterX = picDisplay.ScaleLeft + picDisplay.ScaleWidth / 2
  CenterY = picDisplay.ScaleTop + picDisplay.ScaleHeight / 2
  'picDisplay.PSet (CenterX, CenterY), vbGreen

  'Generate cave vertices
  Select Case tabShape.SelectedItem.Index
    Case 1
      VerticesFromCircle
    Case 2
      VerticesFromRect
    Case 3
  End Select
```

```
'store first cave wall
ReDim CaveWall(0 To NumVertices - 1, 1 To 2) 'set up array to hold X,Y pairs
For j = 0 To NumVertices - 1 'store vertice locations
  CaveWall(j, 1) = Vertice(j).X
  CaveWall(j, 2) = Vertice(j).Y
Next j
CaveWalls(0) = CaveWall 'store the cave wall into an array

'Display initial cave
DrawVertices
picDisplay.Refresh
InitArea = PolyArea(Vertice())

If fraInput(0).Visible Then 'simulation
  'Simulate dissolution events
  WaterArea = txtWaterLevel.Text
  For i = 0 To txtNumEvents.Text - 1
    TErosionDist = txtDistance 'distance to move vertices
    'determine which vertices are underwater and thus get more erosion
    'Debug.Print "Event # " & i
    CalcWaterLevel
'     SelectUnderWater
    'move vertices to new positions
    CalcCoords

    AddNewVertices (txtMaxDist.Text)

    If i <> 0 Then
      If (txtNumEvents.Text - 1) / (100 / cbxDraw.Text) >= 1 Then
        If i Mod (txtNumEvents.Text - 1) / (100 / cbxDraw.Text) = 0 Then
          DrawVertices
          WallNum = i / ((txtNumEvents.Text - 1) / (100 / cbxDraw.Text))
          ReDim CaveWall(0 To NumVertices - 1, 1 To 2)
          For j = 0 To NumVertices - 1
            CaveWall(j, 1) = Vertice(j).X
            CaveWall(j, 2) = Vertice(j).Y
          Next j
          CaveWalls(WallNum) = CaveWall
'           DrawUnderWater
          picDisplay.Refresh
          If i <> 0 Then
            pbr1.Value = i / (txtNumEvents.Text - 1) * 100
            pbr1.Refresh
          End If
        End If
```

```
    End If
  End If
 Next

Else 'real data
 If Not oRS Is Nothing Then 'make sure the recordset exists
  NumRecs = oRS.RecordCount
  'oRS.MoveFirst 'move to start of recordset
  'initialize counters
  i = 0
  JYrs = 0
  JInterval = 0
  Do While Round(JYrs, 8) < Val(txtYrs.Text) * 365 'repeat for the number of years
specified by the user
    oRS.MoveFirst 'move to start of recordset
    Do While Not oRS.EOF And (Round(JYrs, 8) <= Val(txtYrs.Text) * 365) 'read in
data until end of file
      'exit if a null record is encountered - user needs to fix data
      If IsNull(oRS.Fields(0).Value) Or IsNull(oRS.Fields(1).Value) Or
IsNull(oRS.Fields(2).Value) Or IsEmpty(oRS.Fields(0).Value) Or
IsEmpty(oRS.Fields(1).Value) Or IsEmpty(oRS.Fields(2).Value) Then
        MsgBox "Missing or null value encountered in data file at record " & i + 1 &
vbNewLine & "Please correct and process again", vbCritical, "Errors Occurred"
        Exit Sub
      Else
        'read values from current dataset
        JDay = oRS.Fields(0).Value 'Decimal Julian Date
        DRate = oRS.Fields(1).Value 'mm/yr
        WArea = oRS.Fields(2).Value * 1000000 'm2 (therefore convert to mm2)
        If i = 0 Then 'if the first record in the file
          PreviousJDay = JDay
          AddNewVertices (txtMaxDist.Text) 'Fill gaps with new vertices
          JDay1 = JDay 'record of start date in file (may not be zero)
        End If
        'due to the following line the first record is not evaluated so must be a row of
zeros
        dInterval = JDay - PreviousJDay 'interval represented by the record being read
        'next line seems odd - idea of multiplying area by time is to later divide by time to
determine an average and then determine area for the interval being processed
        TempWaterArea = TempWaterArea + (WArea * dInterval) 'units of area*time =
mm2*days
        TempDist = TempDist + (DRate * dInterval / 365) '(mm/yr)*(days/365)=mm

        JYrs = JYrs + dInterval 'Julian Years Counter stored in decimal days used to
know when to stop running
```

```
        JInterval = JInterval + dInterval 'keep tally of Julian time interval until it reaches
user specified interval
        If Round(JInterval, 8) >= Round((txtTimeInterval.Text / MinsInDay), 8) Then
'only process if time passed is greater than user defined interval
          Do While Round(JInterval, 8) >= Round((txtTimeInterval.Text / MinsInDay), 8)
'repeat as last record added may represent time greater than one user defined interval
            WaterArea = (TempWaterArea / JInterval) 'average area over interval
            TErosionDist = TempDist * ((txtTimeInterval.Text / MinsInDay) / JInterval)
'avg distance rate over interval multiplied by time =mm
            'Need to reduce TempWaterArea, TempDist, and JInterval to carry over unused
portions of both into the next equation
            TempWaterArea = TempWaterArea - (WaterArea * (txtTimeInterval.Text /
MinsInDay))
            TempDist = TempDist - TErosionDist
            JInterval = JInterval - (txtTimeInterval.Text / MinsInDay)
            'Calculate Water Levels
            CalcWaterLevel
            'select underwater
            SelectUnderWater
            'move vertices
            CalcCoords
            'add new vertices
            AddNewVertices (txtMaxDist.Text)
          Loop
        Else 'user defined interval was not yet reached so don't process - accumulate
values
        End If

        'determine if it is necessary to draw the cave walls
        If i <> 0 Then
          If NumRecs / (100 / cbxDraw.Text) >= 1 Then
            If i Mod NumRecs / (100 / cbxDraw.Text) = 0 Then
              DrawVertices
              WallNum = i / (NumRecs / (100 / cbxDraw.Text))
              ReDim CaveWall(0 To NumVertices - 1, 1 To 2)
              For j = 0 To NumVertices - 1
                CaveWall(j, 1) = Vertice(j).X
                CaveWall(j, 2) = Vertice(j).Y
              Next j
              CaveWalls(WallNum) = CaveWall
'             DrawUnderWater
              picDisplay.Refresh
              If i <> 0 Then
                pbr1.Value = (i / NumRecs * 100) Mod 100
                pbr1.Refresh
              End If
```

```
          End If
         End If
        End If
       End If
       PreviousJDay = JDay 'previous record's timestamp is incremented to the current
record's timestamp (processing complete)
       oRS.MoveNext 'move cursor to next record
       If Not oRS.EOF Then
        i = i + 1 'increment record counter
        txtRec.Text = i '+ 1 'display record number
        txtRec.Refresh
       Else
       i = i
       End If
      Loop
      PreviousJDay = JDay1 'reset previous each loop to make sure that intervals are
correct
      lblYrs.Caption = FormatNumber(JYrs / 365, 1) & " Year(s)" 'display number of
years that have passed
      frmCaveGrowth.Refresh 'refresh the form
     Loop

     'Process any left over portion of time smaller than the specified increment
     If Round(JInterval, 8) > 0 Then
      WaterArea = TempWaterArea / JInterval
      TErosionDist = TempDist / JInterval
      CalcWaterLevel 'Calculate Water Levels
      SelectUnderWater 'select underwater
      CalcCoords 'move vertices
      AddNewVertices (txtMaxDist.Text) 'add new vertices
     End If
     txtRec.Text = ""
     lblYrs.Caption = "-"
    Else
     MsgBox "Please select a data file", vbOKOnly, "No Data Selected"
    End If
   End If

   'Display final cave shape and water level
   DrawVertices

   txtArea.Text = "Area: " & FormatNumber(PolyArea(Vertice()) / 1000000, 3) & " m2"
   txtDiffArea.Text = "Growth: " & FormatNumber((PolyArea(Vertice()) - InitArea) /
1000000, 6) & " m2"
   txtTotals.Text = TotalVertices & " V Events, Avg Dis Rate = " & TotalDistance /
TotalVertices * txtYrs * 525600 / txtTimeInterval.Text
```

```
  frmCaveGrowth.MousePointer = vbDefault
  pbr1.Visible = False
  CaveExists = True
  Exit Sub
ErrHandler:
  MsgBox "Error in cmdProcess_Click: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Sub cmdSave_Click()
  On Error GoTo ErrHandler
    Dim CGWname As String
    Dim FileNum As Integer
    Dim OutX As Double
    Dim OutY As Double
    Dim i As Long
    Dim j As Long

    cdg1.DialogTitle = "Save Cave File"
    cdg1.Filter = "*.cgw|*.cgw"
    cdg1.ShowSave
    CGWname = cdg1.FileName
    FileNum = FreeFile
    Open CGWname For Output As #FileNum
    Write #FileNum, UBound(CaveWalls), NumVertices
    For i = 0 To UBound(CaveWalls) - 1
      On Error Resume Next 'this accounts for earlier caves having less vertices
      For j = 0 To NumVertices - 1
        OutX = CaveWalls(i)(j, 1)
        OutY = CaveWalls(i)(j, 2)
        Write #FileNum, OutX, OutY
      Next j
      On Error GoTo ErrHandler
    Next
    Close #FileNum
  Exit Sub
ErrHandler:
  MsgBox "Error in cmdSave_Click: " & Err.Number & vbNewLine & Err.Description
  Close #FileNum
End Sub

Private Sub cmdSelectData_Click()
  On Error GoTo ErrHandler
    Dim cat As New ADOX.Catalog
    If Not oRS Is Nothing Then
      oRS.Close
```

```
      oConn.Close
      DataFile = ""
      txtDataFile.Text = ""
    End If
    picDisplay.Cls
    DispGrid.DrawGrid picDisplay, chkShowGrid, chkShowLabels, &HDCDCDC

    cdg1.DialogTitle = "Select Input Data"
    cdg1.FileName = ""
    cdg1.Filter = "*.xls|*.xls"
    cdg1.ShowOpen
    DataFile = cdg1.FileName
    If DataFile = "" Then Exit Sub
    Set oConn = New ADODB.Connection
    Set oRS = New ADODB.Recordset
    oConn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & "Data Source=" & DataFile &
";" & _
             "Extended Properties=""Excel 8.0;HDR=YES;"""

    Set cat.ActiveConnection = oConn
    'oRS.Open "[Sheet1$]", oConn, adOpenStatic, adLockOptimistic
    'oRS.Open "[" & cat.Tables(0).Name & "]", oConn, adOpenStatic, adLockOptimistic
    oRS.Open "[SimData$]", oConn, adOpenStatic, adLockOptimistic
    txtDataFile.Text = DataFile
    txtDataFile.SetFocus
    txtDataFile.SelStart = 0
    txtDataFile.SelLength = Len(txtDataFile)
    Set cat = Nothing
  Exit Sub
ErrHandler:
  Set cat = Nothing
  MsgBox "Error in cmdSelectData_Click: " & Err.Number & vbNewLine &
Err.Description
  Resume Next
End Sub

Private Sub cmdSet_Click()
  On Error GoTo ErrHandler
  picDisplay.ScaleTop = Val(txtCenY.Text) + Val(txtDimensions.Text) / 2
  picDisplay.ScaleLeft = Val(txtCenX.Text) - Val(txtDimensions.Text) / 2
  picDisplay.ScaleHeight = Val(txtDimensions.Text) * -1
  picDisplay.ScaleWidth = Val(txtDimensions.Text)
  'Calculate position for cave center
  CenterX = picDisplay.ScaleLeft + picDisplay.ScaleWidth / 2
  CenterY = picDisplay.ScaleTop + (-1 * picDisplay.ScaleHeight / 2)
```

```
  'Calculate Grid
  picDisplay.Cls
  DispGrid.XDiv = Val(txtGridInc.Text)
  DispGrid.YDiv = Val(txtGridInc.Text)
  DispGrid.DrawGrid picDisplay, chkShowGrid, chkShowLabels, &HDCDCDC
  DrawCaveWalls
  Exit Sub
ErrHandler:
  MsgBox "Error in cmdSet_Click: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Sub Command1_Click()
  AddNewVertices (txtDistance.Text)
End Sub

Private Sub cmdCLS_Click()
  On Error GoTo ErrHandler
  picDisplay.Cls
  DispGrid.DrawGrid picDisplay, chkShowGrid, chkShowLabels, &HDCDCDC
  Exit Sub
ErrHandler:
  MsgBox "Error in cmdCLS_Click: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Sub Form_Load()
  On Error GoTo ErrHandler
  'Pi = 3.141596
  ActiveTool = "1"
  Pi = 4 * Atn(1)
  picDisplay.ScaleTop = Val(txtCenY.Text) + Val(txtDimensions.Text) / 2
  picDisplay.ScaleLeft = txtCenX - Val(txtDimensions.Text) / 2
  picDisplay.ScaleHeight = Val(txtDimensions.Text) * -1
  picDisplay.ScaleWidth = Val(txtDimensions.Text)
  Set DispGrid = New GridLines
  DispGrid.XDiv = Val(txtGridInc.Text)
  DispGrid.YDiv = Val(txtGridInc.Text)
  DispGrid.DrawGrid picDisplay, chkShowGrid, chkShowLabels, &HDCDCDC
  cbxDraw.AddItem 1
  cbxDraw.AddItem 5
  cbxDraw.AddItem 10
  cbxDraw.AddItem 20
  cbxDraw.AddItem 25
  cbxDraw.AddItem 50
  cbxDraw.AddItem 100
```

```
  cbxDraw.Text = 5
  CaveExists = False
  Exit Sub
ErrHandler:
  MsgBox "Error in Form_Load: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Sub picDisplay_MouseDown(Button As Integer, Shift As Integer, X As Single, Y
As Single)
  On Error GoTo ErrHandler
  If ActiveTool = 1 Or ActiveTool = 2 Or ActiveTool = 3 Then
    ExtX = X
    ExtY = Y
    picDisplay.DrawMode = vbInvert
  End If
  Exit Sub
ErrHandler:
  MsgBox "Error in picDisplay_MouseDown: " & Err.Number & vbNewLine &
Err.Description
  Resume Next
End Sub

Private Sub picDisplay_MouseMove(Button As Integer, Shift As Integer, X As Single, Y
As Single)
  On Error GoTo ErrHandler
  txtXY.Text = X & "," & Y
  CurX = X
  CurY = Y
  tmrDisp.Enabled = True
  tmrTip.Enabled = False
  lblMapTip.Visible = False
  If picDisplay.DrawMode = vbInvert Then
    If ActiveTool = 3 Then 'Pan
        picDisplay.Line (ExtX, ExtY)-(OldX, OldY)
        picDisplay.Line (ExtX, ExtY)-(X, Y)
    Else
      If ActiveTool = 1 Or ActiveTool = 2 Then
        picDisplay.Line (ExtX, ExtY)-(OldX, OldY), , B
        picDisplay.Line (ExtX, ExtY)-(X, Y), , B
      End If
    End If
  End If
  OldX = X
  OldY = Y
  Exit Sub
```

```
ErrHandler:
  MsgBox "Error in picDisplay_MouseMove: " & Err.Number & vbNewLine &
Err.Description
  Resume Next
End Sub

Private Sub picDisplay_MouseUp(Button As Integer, Shift As Integer, X As Single, Y
As Single)
  On Error GoTo ErrHandler
  Dim IDList() As String
  Dim i As Long
  Dim Message As String
  Dim lX As Single
  Dim hX As Single
  Dim lY As Single
  Dim hY As Single
  Dim g_Dim As Single
  Dim PanX As Single
  Dim PanY As Single

  Select Case ActiveTool
    Case 1 'Zoom In
      picDisplay.DrawMode = vbCopyPen
      If ExtY > Y Then
        hY = ExtY
        lY = Y
      Else
        lY = ExtY
        hY = Y
      End If
      If ExtX > X Then
        hX = ExtX
        lX = X
      Else
        lX = ExtX
        hX = X
      End If
      If hY - lY > hX - lX Then 'height of zoombox greater than its width
        g_Dim = hY - lY
        picDisplay.ScaleTop = hY
        picDisplay.ScaleLeft = (lX + hX) / 2 - g_Dim / 2
      Else 'width of zoombox greater than its height
        g_Dim = hX - lX
        picDisplay.ScaleLeft = lX
        picDisplay.ScaleTop = (lY + hY) / 2 + g_Dim / 2
      End If
```

```
    If g_Dim > 0 Then 'make sure the user did not just click on the map without dragging
      picDisplay.ScaleHeight = g_Dim * -1
      picDisplay.ScaleWidth = g_Dim

      'Calculate Grid
      picDisplay.Cls
      DispGrid.XDiv = Val(txtGridInc.Text)
      DispGrid.YDiv = Val(txtGridInc.Text)
      DispGrid.DrawGrid picDisplay, chkShowGrid, chkShowLabels, &HDCDCDC
      DrawCaveWalls
      'DrawVertices
      txtDimensions.Text = g_Dim
      txtCenX.Text = picDisplay.ScaleLeft + g_Dim / 2
      txtCenY.Text = picDisplay.ScaleTop - g_Dim / 2
    End If

  Case 2 'Zoom Out
    picDisplay.DrawMode = vbCopyPen
    If ExtY > Y Then
      hY = ExtY
      lY = Y
    Else
      lY = ExtY
      hY = Y
    End If
    If ExtX > X Then
      hX = ExtX
      lX = X
    Else
      lX = ExtX
      hX = X
    End If
    If hY - lY <> 0 Or hX - lX <> 0 Then
      If (hY - lY) > (hX - lX) Then
        'diff between zoombox height and the extent height is smallest (short wide zbox)
        g_Dim = picDisplay.ScaleHeight / (hY - lY) * picDisplay.ScaleHeight
      Else
        'diff between zoombox width and the extent width is smallest (short wide zbox)
        g_Dim = picDisplay.ScaleWidth / (hX - lX) * picDisplay.ScaleWidth
      End If
      picDisplay.ScaleTop = lY + (hY - lY) / 2 + g_Dim / 2 'centerY of zbox - half the
new extent
      picDisplay.ScaleLeft = lX + (hX - lX) / 2 - g_Dim / 2 'centerX of zbox - half the
new extent
      If g_Dim > 0 Then 'make sure the user did not just click on the map without
dragging
```

```
        picDisplay.ScaleHeight = g_Dim * -1
        picDisplay.ScaleWidth = g_Dim

        'Calculate Grid
        picDisplay.Cls
        DispGrid.XDiv = Val(txtGridInc.Text)
        DispGrid.YDiv = Val(txtGridInc.Text)
        DispGrid.DrawGrid picDisplay, chkShowGrid, chkShowLabels, &HDCDCDC
        DrawCaveWalls
        txtDimensions.Text = g_Dim
        txtCenX.Text = picDisplay.ScaleLeft + g_Dim / 2
        txtCenY.Text = picDisplay.ScaleTop - g_Dim / 2
      End If
    End If

  Case 3 'Pan
    picDisplay.DrawMode = vbCopyPen
    PanX = picDisplay.ScaleLeft + ExtX - X
    PanY = picDisplay.ScaleTop + ExtY - Y
    picDisplay.ScaleTop = PanY
    picDisplay.ScaleLeft = PanX
    txtCenX.Text = PanX
    txtCenY.Text = PanY

    'Calculate Grid
    picDisplay.Cls
    DispGrid.XDiv = Val(txtGridInc.Text)
    DispGrid.YDiv = Val(txtGridInc.Text)
    DispGrid.DrawGrid picDisplay, chkShowGrid, chkShowLabels, &HDCDCDC
    DrawCaveWalls
  Case 4 'ID
    If SelectVertex(X, Y) <> "" Then
      IDList = Split(SelectVertex(X, Y), ",")
      For i = 0 To UBound(IDList)
        Message = Message & "Vertex: " & IDList(i) & vbNewLine
        Message = Message & "X: " & Vertice(Val(IDList(i))).X & vbNewLine
        Message = Message & "Y: " & Vertice(Val(IDList(i))).Y & vbNewLine
        Message = Message & "Underwater: " & Vertice(Val(IDList(i))).Underwater &
vbNewLine
        Message = Message & vbNewLine
      Next i
      MsgBox Message, vbOKOnly, "Vertex Info"
    End If

  Case 5 'Add Vertex
    If fraShape(2).Visible Then
```

```
        picDisplay.Cls
        DispGrid.DrawGrid picDisplay, chkShowGrid, chkShowLabels, &HDCDCDC
        ReDim Preserve Vertice(0 To NumVertices)
        Set Vertice(NumVertices) = New CaveVertex
        Vertice(NumVertices).X = X
        Vertice(NumVertices).Y = Y
        NumVertices = NumVertices + 1
        DrawVertices
      End If
    Case Else
  End Select
  Exit Sub
ErrHandler:
  MsgBox "Error in picDisplay_MouseUp: " & Err.Number & vbNewLine &
Err.Description
  Resume Next
End Sub

Public Sub DrawVertices()
  Dim i As Long

  For i = 0 To NumVertices - 1
    picDisplay.PSet (Vertice(i).X, Vertice(i).Y), vbRed
    'picDisplay.Print Vertice(i).X & ", " & Vertice(i).Y
    picDisplay.DrawWidth = 1
    If i > 0 Then
      picDisplay.Line (Vertice(i - 1).X, Vertice(i - 1).Y)-(Vertice(i).X, Vertice(i).Y)
    End If
    If i = NumVertices - 1 Then picDisplay.Line (Vertice(0).X, Vertice(0).Y)-
(Vertice(i).X, Vertice(i).Y)
    picDisplay.DrawWidth = 3
  Next i
  Exit Sub
ErrHandler:
  MsgBox "Error in DrawVertices: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Sub DrawWaterLevel()
  On Error GoTo ErrHandler
  picDisplay.DrawWidth = 4
  picDisplay.Line (picDisplay.ScaleLeft, Vertice(WaterLevel).Y)-(picDisplay.ScaleLeft +
picDisplay.ScaleWidth, Vertice(WaterLevel).Y), vbCyan
  picDisplay.DrawWidth = 3
  Exit Sub
ErrHandler:
```

```
  MsgBox "Error in DrawWaterLevel: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Sub DrawUnderWater()
  On Error GoTo ErrHandler
  Dim i As Long
  For i = 0 To NumVertices - 1
    If Vertice(i).Underwater = True Then
      picDisplay.PSet (Vertice(i).X, Vertice(i).Y), vbBlue
    End If
  Next
  Exit Sub
ErrHandler:
  MsgBox "Error in DrawUnderWater: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Function SelectUnderWater() As String
  On Error GoTo ErrHandler
  Dim i As Long
  Dim VList As String

  VList = ""
  For i = 0 To NumVertices - 1
    'If Vertice(i).Y <= Vertice(WaterLevel).Y Then
    If Round(Vertice(i).Y, 6) <= Round(Vertice(WaterLevel).Y, 6) Then
      Vertice(i).Underwater = True
      If VList = "" Then
        VList = i
      Else
        VList = VList & "," & i
      End If
    Else
      Vertice(i).Underwater = False
    End If
  Next
  SelectUnderWater = VList
  Exit Function
ErrHandler:
  MsgBox "Error in SelectUnderWater: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Function

Private Function PolyArea(Point() As CaveVertex) As Double
'  On Error GoTo ErrHandler
```

```
  Dim i As Long
  Dim s1 As Double, s2 As Double
  Dim Area As Double
  Dim VCount As Integer

  VCount = UBound(Point)
  s1 = 0
  s2 = 0
  For i = 0 To (VCount - 1)
    s1 = s1 + (Point(i).X * Point(i + 1).Y)
    s2 = s2 + (Point(i).Y * Point(i + 1).X)
  Next
  s1 = s1 + Point(VCount).X * Point(0).Y
  s2 = s2 + Point(VCount).Y * Point(0).X
  PolyArea = 0.5 * Abs(s1 - s2)
  Exit Function
ErrHandler:
  MsgBox "Error in PolyArea: " & Err.Number & vbNewLine & Err.Description
End Function

Private Sub CalcCoords()
  Dim i As Long
  Dim Slope As Double
  Dim Bearing As Double
  Dim PosX As Boolean
  Dim PosY As Boolean
  Dim Max As Integer
  Dim Distance As Double
  Dim Undefined As Boolean
  Dim Xarray() As Double 'arrays to hold values while originals are being use in
calculations
  Dim Yarray() As Double

  ReDim Xarray(0 To NumVertices - 1)
  ReDim Yarray(0 To NumVertices - 1)
  Max = NumVertices - 1
  For i = 0 To Max
    Undefined = False
    If Not i = 0 Then
      If Not i = Max Then
        If Round(Vertice(i + 1).X, 6) <> Round(Vertice(i - 1).X, 6) Then
          Slope = (Vertice(i + 1).Y - Vertice(i - 1).Y) / (Vertice(i + 1).X - Vertice(i - 1).X)
        Else
          Undefined = True
        End If
        'PosX = IIf(Vertice(i + 1).X >= Vertice(i - 1).X, True, False)
```

```
    PosX = IIf(Round(Vertice(i + 1).X, 6) >= Round(Vertice(i - 1).X, 6), True, False)
  Else
    If Round(Vertice(0).X, 6) <> Round(Vertice(Max - 1).X, 6) Then
      Slope = (Vertice(0).Y - Vertice(Max - 1).Y) / (Vertice(0).X - Vertice(Max - 1).X)
    Else
      Undefined = True
    End If
    'PosX = IIf(Vertice(0).X >= Vertice(Max - 1).X, True, False)
    PosX = IIf(Round(Vertice(0).X, 6) >= Round(Vertice(Max - 1).X, 6), True, False)
  End If
Else
  If Round(Vertice(1).X, 6) <> Round(Vertice(Max).X, 6) Then
    Slope = (Vertice(1).Y - Vertice(Max).Y) / (Vertice(1).X - Vertice(Max).X)
  Else
    Undefined = True
  End If
  'PosX = IIf(Vertice(1).X >= Vertice(Max).X, True, False)
  PosX = IIf(Round(Vertice(1).X, 6) >= Round(Vertice(Max).X, 6), True, False)
End If
If Undefined = False Then
  Bearing = Atn(Slope) * 180 / Pi
  If Bearing < 0 Then
    If PosX = True Then
      Bearing = Bearing * -1
    Else
      Bearing = Bearing * -1 + 180
    End If
  Else
    If PosX = True Then
      Bearing = 360 - Bearing
    Else
      Bearing = 180 - Bearing
    End If
  End If
Else
  If i = 0 Then
    'PosY = IIf(Vertice(1).Y >= Vertice(Max).Y, True, False)
    PosY = IIf(Round(Vertice(1).Y, 6) >= Round(Vertice(Max).Y, 6), True, False)
  ElseIf i = Max Then
    'PosY = IIf(Vertice(0).Y >= Vertice(Max - 1).Y, True, False)
    PosY = IIf(Round(Vertice(0).Y, 6) >= Round(Vertice(Max - 1).Y, 6), True, False)
  Else
    'PosY = IIf(Vertice(i + 1).Y >= Vertice(i - 1).Y, True, False)
    PosY = IIf(Round(Vertice(i + 1).Y, 6) >= Round(Vertice(i - 1).Y, 6), True, False)
  End If
  If PosY = True Then
```

```
      Bearing = 270
    Else
      Bearing = 90
    End If
  End If
  If Vertice(i).Underwater = True Then
    Distance = TErosionDist '+ WErosion
    TotalVertices = TotalVertices + 1 'keep track of how many vertices are moved
    TotalDistance = TotalDistance + TErosionDist 'keep track of the total combined
distance moved
  Else
    Distance = 0 'TErosionDist
  End If
  Xarray(i) = Vertice(i).X + Distance * Sin(Pi / 180 * Bearing)
  Yarray(i) = Vertice(i).Y + Distance * Cos(Pi / 180 * Bearing)
 Next
 For i = 0 To NumVertices - 1
  Vertice(i).X = Xarray(i)
  Vertice(i).Y = Yarray(i)
 Next i
 Exit Sub
ErrHandler:
 MsgBox "Error in CalcCoords: " & Err.Number & vbNewLine & Err.Description
 Resume Next
End Sub

Private Sub CalcWaterLevel()
 On Error GoTo ErrHandler
 Dim i As Long
 Dim UW() As String
 Dim UWVertex() As CaveVertex
 Dim Increasing As Boolean
 Dim ChangeDir As Boolean
 Dim UWArea As Double
 Dim sUWList As String
 Dim dCaveArea As Double

 dCaveArea = PolyArea(Vertice())
 If WaterArea > dCaveArea Then
  'all vertices are wet
  For i = 0 To NumVertices - 1
   If Round(Vertice(i).Y, 6) > Round(Vertice(WaterLevel).Y, 6) Then WaterLevel = i
  Next i
  SelectUnderWater
  DrawPoly Vertice()
 Else
```

```
ChangeDir = False
Do While ChangeDir = False
  sUWList = SelectUnderWater
  If sUWList <> "" Then
    UW = Split(sUWList, ",")
    'could check for consecutive here
    ReDim UWVertex(0 To UBound(UW))
    For i = 0 To UBound(UW)
      Set UWVertex(i) = Vertice(Val(UW(i)))
    Next
    If UniqueVertexY Then
      If NextHighest <> -1 And NextLowest <> -1 Then
        'another vertex needs to be added as this one does not have a pair
        ReDim Preserve UWVertex(0 To UBound(UW) + 1)
        Set UWVertex(UBound(UWVertex)) = New CaveVertex
        UWVertex(UBound(UWVertex)).Y = Vertice(WaterLevel).Y
        UWVertex(UBound(UWVertex)).X = GetIntersectX
      End If
    End If

    UWArea = PolyArea(UWVertex())
    If WaterArea > UWArea Then
      'raise level - find next vertex with a greater y value
      If Increasing = False Then
        ChangeDir = True
      Else
        Increasing = True
        If NextHighest <> -1 Then
          WaterLevel = NextHighest
        Else
          ChangeDir = True
        End If
      End If
    ElseIf WaterArea = UWArea Then
      ChangeDir = True
    Else
      'drop level
      If Increasing = True Then ChangeDir = True
      Increasing = False
      If NextLowest <> -1 Then
        WaterLevel = NextLowest
      Else
        ChangeDir = True
      End If
    End If
  End If
```

```
    Loop

    'Debug.Print "Water Level: " & WaterLevel

    DrawPoly UWVertex() 'draw the water level
  End If
  Exit Sub
ErrHandler:
  MsgBox "Error in CalcWaterLevel: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Sub TabInput_Click()
  On Error GoTo ErrHandler
    Dim i As Integer
    For i = 1 To 2
      If TabInput.SelectedItem.Index = i Then
        fraInput(i - 1).Visible = True
      Else
        fraInput(i - 1).Visible = False
      End If
    Next
  Exit Sub
ErrHandler:
  MsgBox "Error in TabInput_Click: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Sub tabShape_Click()
  On Error GoTo ErrHandler
    Dim i As Integer
    For i = 1 To 4
      If tabShape.SelectedItem.Index = i Then
        fraShape(i - 1).Visible = True
      Else
        fraShape(i - 1).Visible = False
      End If
    Next
  Exit Sub
ErrHandler:
  MsgBox "Error in tabShape_Click: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Sub tbr1_ButtonClick(ByVal Button As MSComctlLib.Button)
  On Error GoTo ErrHandler
```

```
  If ActiveTool > 0 Then
    tbr1.Buttons(ActiveTool).Value = tbrUnpressed
  End If
  Button.Value = tbrPressed
  ActiveTool = Button.Index
  tbr1.Refresh
  Exit Sub
ErrHandler:
  MsgBox "Error in tbr1_ButtonClick: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Sub tmrDisp_Timer()
  On Error GoTo ErrHandler
  Dim IDList() As String
  Dim i As Long
  Dim Message As String

  If TimerX = CurX And TimerY = CurY Then 'no movement
    If SelectVertex(CurX, CurY) <> "" Then
      IDList = Split(SelectVertex(CurX, CurY), ",")
      For i = 0 To UBound(IDList)
        Message = Message & "Vertex: " & IDList(i) & vbNewLine
        Message = Message & "X: " & Vertice(Val(IDList(i))).X & vbNewLine
        Message = Message & "Y: " & Vertice(Val(IDList(i))).Y & vbNewLine
        Message = Message & "Underwater: " & Vertice(Val(IDList(i))).Underwater &
vbNewLine
      Next i
      lblMapTip.Caption = Message
      If CurX - lblMapTip.Width < picDisplay.ScaleLeft Then
        lblMapTip.Left = picDisplay.ScaleLeft
      Else
        lblMapTip.Left = CurX - lblMapTip.Width
      End If
      If CurY + lblMapTip.Height > picDisplay.ScaleTop Then
        lblMapTip.Top = picDisplay.ScaleTop
      Else
        lblMapTip.Top = CurY + lblMapTip.Height
      End If
      lblMapTip.Visible = True
      tmrTip.Enabled = True
    End If
  End If
  TimerX = CurX
  TimerY = CurY
  Exit Sub
```

```
ErrHandler:
  MsgBox "Error in tmrDisp_Timer: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Sub tmrTip_Timer()
  On Error GoTo ErrHandler
  lblMapTip.Visible = False
  tmrDisp.Enabled = False
  Exit Sub
ErrHandler:
  MsgBox "Error: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Sub txtDimensions_Change()
  On Error GoTo ErrHandler
  lblDimensions2.Caption = "x " & FormatNumber(txtDimensions.Text, 2)
  Exit Sub
ErrHandler:
  MsgBox "Error in txtDimensions_Change: " & Err.Number & vbNewLine &
Err.Description
  Resume Next
End Sub

Private Sub MaxDistCirc()
  On Error GoTo ErrHandler
  Dim Bearing As Double
  Dim Hypot As Double
  Dim Vertex1 As CaveVertex
  Dim Vertex2 As CaveVertex
  Dim NumVerts As Long

  If Not IsNumeric(txtNumVertices(0).Text) Then Exit Sub

  Radius = txtRadius.Text
  NumVerts = txtNumVertices(0).Text
  Bearing = 360 / NumVerts * 0
  Set Vertex1 = New CaveVertex
  Vertex1.X = CenterX + Radius * Sin(Pi / 180 * Bearing)
  Vertex1.Y = CenterY + Radius * Cos(Pi / 180 * Bearing)
  Bearing = 360 / NumVerts * 1
  Set Vertex2 = New CaveVertex
  Vertex2.X = CenterX + Radius * Sin(Pi / 180 * Bearing)
  Vertex2.Y = CenterY + Radius * Cos(Pi / 180 * Bearing)
```

```
  Hypot = Sqr((Vertex1.Y - Vertex2.Y) ^ 2 + (Vertex1.X - Vertex2.X) ^ 2)

  Set Vertex1 = Nothing
  Set Vertex2 = Nothing
  txtMaxDist.Text = Round(Hypot, 1)
  Exit Sub
ErrHandler:
  MsgBox "Error: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Sub VerticesFromCircle()
  On Error GoTo ErrHandler
  Dim Bearing As Double
  Dim i As Long

  Radius = txtRadius.Text
  NumVertices = txtNumVertices(0).Text
  ReDim Vertice(0 To NumVertices - 1)
  For i = 0 To NumVertices - 1
    Bearing = 360 / NumVertices * i
    Set Vertice(i) = New CaveVertex
    Vertice(i).X = CenterX + Radius * Sin(Pi / 180 * Bearing)
    Vertice(i).Y = CenterY + Radius * Cos(Pi / 180 * Bearing)
  Next
  Exit Sub
ErrHandler:
  MsgBox "Error: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Sub cmdAddVertex_Click()
  On Error GoTo ErrHandler
  picDisplay.Cls
  DispGrid.DrawGrid picDisplay, chkShowGrid, chkShowLabels, &HDCDCDC
  NumVertices = 0
  Erase Vertice()
  Erase CaveWall()
  Erase CaveWalls()
  CaveExists = False
  tbr1.Buttons(ActiveTool).Value = tbrUnpressed
  tbr1.Buttons(5).Value = tbrPressed
  tbr1.Buttons(5).Enabled = True
  ActiveTool = 5
  MsgBox "Add vertices by clicking on the display in a clockwise order.", vbOKOnly,
"Add Vertices"
```

```
  Exit Sub
ErrHandler:
  MsgBox "Error in cmdAddVertex_Click: " & Err.Number & vbNewLine &
Err.Description
  Resume Next
End Sub

Function NextHighest() As Integer
  On Error GoTo ErrHandler
  Dim i As Long
  Dim gap As Double
  Dim NextNum As Integer

  gap = -1
  For i = 0 To NumVertices - 1
    If Round(Vertice(i).Y, 6) > Round(Vertice(WaterLevel).Y, 6) Then
      If gap = -1 Then
        gap = Vertice(i).Y - Vertice(WaterLevel).Y
        NextNum = i
      Else
        If Round(Vertice(i).Y - Vertice(WaterLevel).Y, 6) < Round(gap, 6) Then
          gap = Vertice(i).Y - Vertice(WaterLevel).Y
          NextNum = i
        End If
      End If
    End If
  Next i
  If gap = -1 Then NextNum = -1
  NextHighest = NextNum
  Exit Function
ErrHandler:
  MsgBox "Error in NextHighest: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Function

Function NextLowest() As Integer
  On Error GoTo ErrHandler
  Dim i As Long
  Dim gap As Double
  Dim NextNum As Integer

  gap = -1
  For i = 0 To NumVertices - 1
    If Round(Vertice(i).Y, 6) < Round(Vertice(WaterLevel).Y, 6) Then
      If gap = -1 Then
        gap = Vertice(WaterLevel).Y - Vertice(i).Y
```

```
        NextNum = i
      Else
        If Round(Vertice(WaterLevel).Y - Vertice(i).Y, 6) < Round(gap, 6) Then
          gap = Vertice(WaterLevel).Y - Vertice(i).Y
          NextNum = i
        End If
      End If
    End If
  Next i
  If gap = -1 Then NextNum = -1
  NextLowest = NextNum
  Exit Function
ErrHandler:
  MsgBox "Error in NextLowest: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Function

Private Function SelectVertex(X As Single, Y As Single) As String
  On Error GoTo ErrHandler
  Dim i As Long
  Dim tolerance As Double
  Dim ReturnString As String
  Dim ExtMinX As Double
  Dim ExtMaxX As Double
  Dim ExtMinY As Double
  Dim ExtMaxY As Double

  tolerance = picDisplay.ScaleX(3, vbPixels, vbUser)
  ExtMinX = X - tolerance
  ExtMaxX = X + tolerance
  ExtMinY = Y - tolerance
  ExtMaxY = Y + tolerance
  For i = 0 To NumVertices - 1
    If Vertice(i).X < ExtMaxX And Vertice(i).X > ExtMinX And Vertice(i).Y < ExtMaxY
And Vertice(i).Y > ExtMinY Then
      If ReturnString = "" Then
        ReturnString = i
      Else
        ReturnString = ReturnString & "," & i
      End If
    End If
  Next i
  SelectVertex = ReturnString
  Exit Function
ErrHandler:
  MsgBox "Error in SelectVertex: " & Err.Number & vbNewLine & Err.Description
```

```
End Function

Private Function LargeGapList(Dist As Double) As String
  On Error GoTo ErrHandler
  Dim i As Long
  Dim Hypot As Double
  Dim SpaceList As String

  For i = 0 To NumVertices - 2
    Hypot = Sqr((Vertice(i).Y - Vertice(i + 1).Y) ^ 2 + (Vertice(i).X - Vertice(i + 1).X) ^
2)
    If Hypot > Dist Then
      SpaceList = SpaceList & i + 1 & ","
    End If
  Next i
  Hypot = Sqr((Vertice(NumVertices - 1).Y - Vertice(0).Y) ^ 2 + (Vertice(NumVertices -
1).X - Vertice(0).X) ^ 2)
  If Hypot > Dist Then
    SpaceList = SpaceList & NumVertices & ","
  End If
  LargeGapList = SpaceList
  Exit Function
ErrHandler:
  MsgBox "Error in LargeGapList: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Function

Private Sub DrawCaveWalls()
  On Error GoTo ErrHandler
  Dim i As Long
  Dim j As Long

  If CaveExists Then
    For j = 0 To UBound(CaveWalls) - 1
      For i = 0 To UBound(CaveWalls(j))
        picDisplay.PSet (CaveWalls(j)(i, 1), CaveWalls(j)(i, 2)), vbRed
        picDisplay.DrawWidth = 1
        If i > 0 Then
          picDisplay.Line (CaveWalls(j)(i - 1, 1), CaveWalls(j)(i - 1, 2))-(CaveWalls(j)(i, 1),
CaveWalls(j)(i, 2))
        End If
        If i = UBound(CaveWalls(j)) Then picDisplay.Line (CaveWalls(j)(0, 1),
CaveWalls(j)(0, 2))-(CaveWalls(j)(i, 1), CaveWalls(j)(i, 2))
        picDisplay.DrawWidth = 3
      Next i
    Next j
```

```
  End If
  Exit Sub
ErrHandler:
  MsgBox "Error in DrawCaveWalls: " & Err.Number & vbNewLine & Err.Description
'  Resume Next
End Sub

Private Sub DrawPoly(Point() As CaveVertex)
  Dim i As Long

  For i = 0 To UBound(Point)
    picDisplay.DrawWidth = 6
    picDisplay.PSet (Point(i).X, Point(i).Y), vbCyan
    picDisplay.DrawWidth = 1
    If i > 0 Then
      picDisplay.Line (Point(i - 1).X, Point(i - 1).Y)-(Point(i).X, Point(i).Y), vbBlue
    End If
    If i = UBound(Point) Then picDisplay.Line (Point(0).X, Point(0).Y)-(Point(i).X,
Point(i).Y), vbBlue
    picDisplay.DrawWidth = 3
  Next i
  Exit Sub
ErrHandler:
  MsgBox "Error in DrawPoly: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Sub

Private Function UniqueVertexY() As Boolean
  On Error GoTo ErrHandler

  Dim i As Long
  Dim VCount As Long
  Dim WL As Double

  WL = Vertice(WaterLevel).Y
  For i = 0 To NumVertices - 1
    If Vertice(i).Y = WL Then VCount = VCount + 1
  Next i
  If VCount > 1 Then
    UniqueVertexY = False
  Else
    UniqueVertexY = True
  End If
  Exit Function
ErrHandler:
  MsgBox "Error in UniqueVertexY: " & Err.Number & vbNewLine & Err.Description
```

```
  Resume Next
End Function

Private Function GetIntersectX()
'  On Error GoTo ErrHandler

  Dim i As Long
  Dim WL As Double
  Dim Vs As String
  Dim V() As String
  Dim Slope As Double
  Dim X As Double
  Dim P() As String

  Vs = ""
  WL = Vertice(WaterLevel).Y

  'Find vertice pairs containing the waterlevel Y value
  If WaterLevel <> 0 And WaterLevel <> NumVertices - 1 Then
    If Vertice(0).Y <= WL And Vertice(NumVertices - 1).Y >= WL Or _
      Vertice(0).Y >= WL And Vertice(NumVertices - 1).Y <= WL Then
      Vs = NumVertices - 1 & ",0 "
    End If
  End If
  For i = 0 To NumVertices - 2
    If i <> WaterLevel And i + 1 <> WaterLevel Then
      If Round(Vertice(i).Y, 6) <= Round(WL, 6) And Round(Vertice(i + 1).Y, 6) >=
Round(WL, 6) Or _
        Round(Vertice(i).Y, 6) >= Round(WL, 6) And Round(Vertice(i + 1).Y, 6) <=
Round(WL, 6) Then
        Vs = Vs & i & "," & i + 1 & " "
      End If
    End If
  Next i
  Vs = Trim(Vs)
  If Vs <> "" Then
    V = Split(Vs, " ")
    For i = 0 To UBound(V)
      P = Split(V(i), ",")
      'Point Slope Formula..... y-y1 = m(x-x1)
      If Round(Vertice(P(0)).X, 6) = Round(Vertice(P(1)).X, 6) Then
        X = Vertice(P(0)).X
      Else
        Slope = (Vertice(P(0)).Y - Vertice(P(1)).Y) / (Vertice(P(0)).X - Vertice(P(1)).X)
        X = (Vertice(WaterLevel).Y - Vertice(P(1)).Y) / Slope + Vertice(P(1)).X
      End If
```

```
    Next i
  End If
  GetIntersectX = X
  Exit Function
ErrHandler:
  MsgBox "Error in GetIntersectX: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Function

Private Sub AddNewVertex(Position As Long)
  Dim i As Long

  ReDim TempVertice(NumVertices)
  If Position = 0 Then
    TempVertice(Position) = New CaveVertex
    'Populate new vertex
    For i = 1 To NumVertices - 1
      Set TempVertice(i) = Vertice(i - 1)
    Next i
  ElseIf Position > 0 Then
    For i = 0 To Position - 1
      Set TempVertice(i) = Vertice(i)
    Next i
    Set TempVertice(Position) = New CaveVertex
    'Populate new vertex
    TempVertice(Position).X = (Vertice(Position - 1).X + Vertice(Position).X) / 2
    TempVertice(Position).Y = (Vertice(Position - 1).Y + Vertice(Position).Y) / 2
    For i = Position + 1 To NumVertices
      Set TempVertice(i) = Vertice(i - 1)
    Next i
  Else
    MsgBox "Error - new vertex position cannot be negative", vbCritical, "Error in
AddNewVertex"
  End If
  Vertice = TempVertice
  Erase TempVertice
  DrawVertices
End Sub

Private Function AddNewVertices(MaxDist As Double)
  On Error GoTo ErrHandler
  Dim i As Long
  Dim j As Long
  Dim Hypot As Double
  Dim NumVs As Long
  Dim PosDist As Double
```

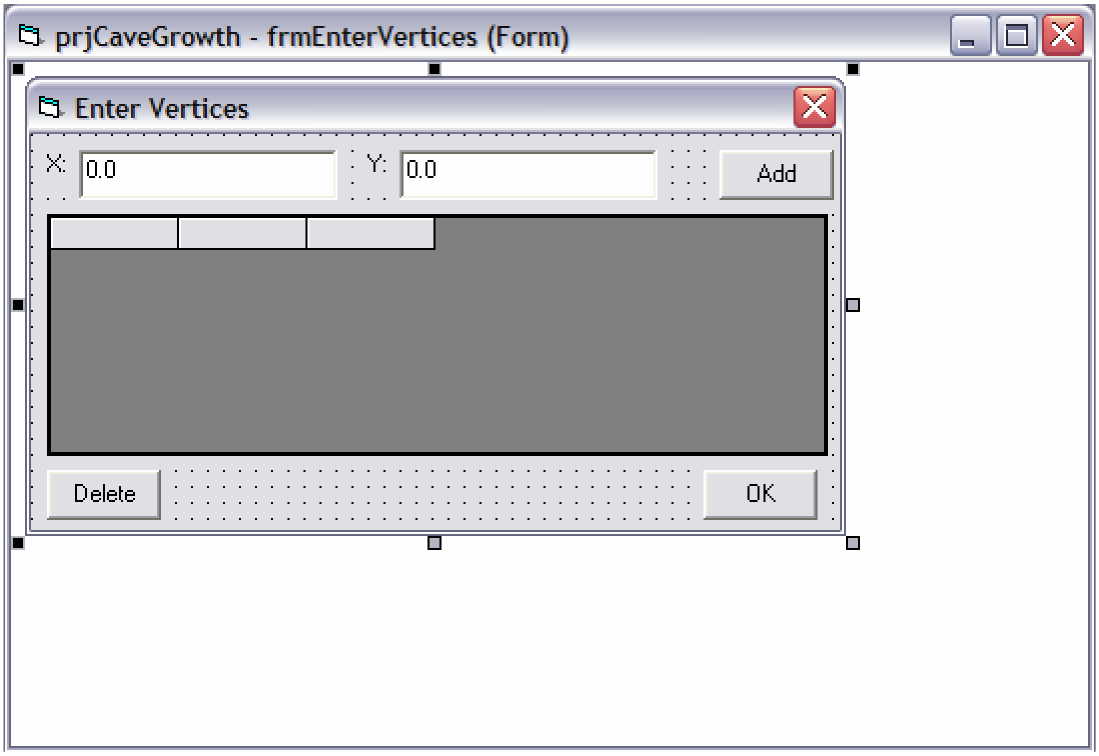
```
  Dim NewVertex As CaveVertex
  Dim NumNewVs As Long

  'Debug.Print "START"
  NumNewVs = 0
  For i = 0 To NumVertices - 2

    ReDim Preserve TempVertice(i + NumNewVs) 'increment the temparray appropriately

    Set TempVertice(i + NumNewVs) = Vertice(i)
    Hypot = Sqr((Vertice(i).Y - Vertice(i + 1).Y) ^ 2 + (Vertice(i).X - Vertice(i + 1).X) ^
2)
    'Debug.Print Hypot
    If Hypot > MaxDist Then 'if the distance is greater than the maximum then split it by
adding vertices
      NumVs = Int(Hypot / MaxDist) 'divide the distance by the max distance to determine
the number of vertices to add

      For j = 1 To NumVs
        NumNewVs = NumNewVs + 1 'increment number of new vertices
        ReDim Preserve TempVertice(i + NumNewVs)
        Set NewVertex = New CaveVertex
        NewVertex.X = Vertice(i).X + ((Vertice(i + 1).X - Vertice(i).X) / (NumVs + 1)) * j
        NewVertex.Y = Vertice(i).Y + ((Vertice(i + 1).Y - Vertice(i).Y) / (NumVs + 1)) * j
        Set TempVertice(i + NumNewVs) = NewVertex
      Next j
    End If
  Next i
  'deal with distance between penultimate and last vertex
  ReDim Preserve TempVertice(i + NumNewVs)
  Set TempVertice(i + NumNewVs) = Vertice(i)
  Hypot = Sqr((Vertice(i).Y - Vertice(0).Y) ^ 2 + (Vertice(i).X - Vertice(0).X) ^ 2)
  If Hypot > MaxDist Then
      NumVs = Int(Hypot / MaxDist) 'divide the distance by the max distance to determine
the number of vertices to add
      For j = 1 To NumVs
        NumNewVs = NumNewVs + 1 'increment number of new vertices
        ReDim Preserve TempVertice(i + NumNewVs)
        Set NewVertex = New CaveVertex
        NewVertex.X = Vertice(i).X + ((Vertice(0).X - Vertice(i).X) / (NumVs + 1)) * j
        NewVertex.Y = Vertice(i).Y + ((Vertice(0).Y - Vertice(i).Y) / (NumVs + 1)) * j
        Set TempVertice(i + NumNewVs) = NewVertex
      Next j
  End If
  Vertice = TempVertice
  NumVertices = UBound(Vertice) + 1
```

```
  Erase TempVertice
  DrawVertices
  'Debug.Print "END"
  Exit Function
ErrHandler:
  MsgBox "Error in AddNewVertices: " & Err.Number & vbNewLine & Err.Description
  Resume Next
End Function

Private Sub txtNumVertices_Change(Index As Integer)
  If Index = 0 Then
    MaxDistCirc
  Else
    MaxDistRect
  End If
End Sub

Private Sub VerticesFromRect()
  On Error GoTo ErrHandler
  Dim Bearing As Double
  Dim i As Long
  Dim RectWidth As Double
  Dim RectHeight As Double
  Dim Perimiter As Double
  Dim AvSpace As Double
  Dim VSpace As Double
  Dim HSpace As Double
  Dim VNum As Long
  Dim HNum As Long
  Dim MinX As Double
  Dim MinY As Double
  Dim MaxX As Double
  Dim MaxY As Double
  Dim OddFlag As Boolean
  Dim OddSpace As Double

  OddFlag = False
  RectWidth = txtWidth.Text
  RectHeight = txtHeight.Text
  NumVertices = txtNumVertices(1).Text
  ReDim Vertice(0 To NumVertices - 1)
  If NumVertices Mod 2 = 1 Then OddFlag = True 'is the number of vertices odd?

  Perimiter = RectWidth * 2 + RectHeight * 2
  If OddFlag = False Then
    AvSpace = Perimiter / NumVertices
```

```
Else
  AvSpace = Perimiter / (NumVertices - 1) 'extra vertice will be added to the bottom
End If
'HNum and Vnum represent the number of spaces
If Int(RectHeight / AvSpace) < RectHeight / AvSpace Then
  If Int(RectWidth / AvSpace) > 0 Then 'width smaller than average spacing
  'If Int(RectWidth / AvSpace) Mod 2 > 0 Then
    VNum = Int(RectHeight / AvSpace) + 1  'height gets extra vertices
  Else
    VNum = Int(RectHeight / AvSpace) 'detract one for the width
  End If
Else 'evenly divisible by the average spacing
  If Int(RectWidth / AvSpace) > 0 Then
    VNum = Int(RectHeight / AvSpace)
  End If
End If
HNum = Int(RectWidth / AvSpace)

If VNum = 0 Then '(height is less than an average space)
  VSpace = RectHeight
Else
  VSpace = RectHeight / VNum
End If
If HNum = 0 Then '(width is less than an average space)
  HSpace = RectWidth
  OddSpace = RectWidth / 2 'if odd, the base needs to accomodate an extra vertice
Else
  HSpace = RectWidth / HNum
  OddSpace = RectWidth / (HNum + 1) 'if odd, the base needs to accomodate an extra
vertice
End If

'determine rectangle bounds
MinX = CenterX - RectWidth / 2
MinY = CenterY - RectHeight / 2
MaxX = CenterX + RectWidth / 2
MaxY = CenterY + RectHeight / 2

Set Vertice(0) = New CaveVertex
Vertice(0).X = MinX
Vertice(0).Y = MaxY
For i = 1 To NumVertices - 1
  Set Vertice(i) = New CaveVertex
  If Round(Vertice(i - 1).X, 6) = Round(MinX, 6) Then 'Left side
    If Round(Vertice(i - 1).Y, 6) < Round(MaxY, 6) Then
      Vertice(i).X = MinX
```

```
      Vertice(i).Y = Vertice(i - 1).Y + VSpace
    Else 'Start along top
      Vertice(i).X = MinX + HSpace
      Vertice(i).Y = MaxY
    End If
  ElseIf Round(Vertice(i - 1).Y, 6) = Round(MinY, 6) Then 'Bottom
    If OddFlag = False Then
      If Round(Vertice(i - 1).X, 6) > Round(MinX, 6) Then
        Vertice(i).X = Vertice(i - 1).X - HSpace
        Vertice(i).Y = MinY
      End If
    Else 'odd number of vertices so add one to bottom
      If Round(Vertice(i - 1).X, 6) > Round(MinX, 6) Then
        Vertice(i).X = Vertice(i - 1).X - OddSpace
        Vertice(i).Y = MinY
      End If
    End If
  ElseIf Round(Vertice(i - 1).X, 6) = Round(MaxX, 6) Then 'Right side
    If Round(Vertice(i - 1).Y, 6) > Round(MinY, 6) Then
      Vertice(i).X = MaxX
      Vertice(i).Y = Vertice(i - 1).Y - VSpace
    Else 'Start along bottom
      Vertice(i).X = Vertice(i - 1).X - HSpace
      Vertice(i).Y = MinY
    End If
  ElseIf Round(Vertice(i - 1).Y, 6) = Round(MaxY, 6) Then 'Top
    If Round(Vertice(i - 1).X, 6) < Round(MaxX, 6) Then
      Vertice(i).X = Vertice(i - 1).X + HSpace
      Vertice(i).Y = MaxY
    End If
  End If
 Next
 Exit Sub
ErrHandler:
 MsgBox "Error in VerticesFromCircle: " & Err.Number & vbNewLine &
Err.Description
 Resume Next
End Sub
```

VB6 Project: prjCaveGrowth
Form: frmEnterVertices




```
Option Explicit

Private Sub cmdAdd_Click()
  frmCaveGrowth.picDisplay.Cls
  DispGrid.DrawGrid frmCaveGrowth.picDisplay, frmCaveGrowth.chkShowGrid,
frmCaveGrowth.chkShowLabels, &HDCDCDC
  ReDim Preserve Vertice(0 To NumVertices)
  Set Vertice(NumVertices) = New CaveVertex
  Vertice(NumVertices).X = txtXCoord
  Vertice(NumVertices).Y = txtYCoord
  'lstVertices.AddItem "Vertice " & NumVertices & ": " & Vertice(NumVertices).X & ",
" & Vertice(NumVertices).Y
  flx1.AddItem NumVertices + 1 & vbTab & Vertice(NumVertices).X & vbTab &
Vertice(NumVertices).Y
  NumVertices = NumVertices + 1
  frmCaveGrowth.DrawVertices
End Sub

Private Sub cmdDelete_Click()
  Dim SelRow As Long
  Dim i As Long
  Dim NewVertex As CaveVertex
```

```
  SelRow = flx1.RowSel
  flx1.Row = SelRow
  ReDim TempVertice(UBound(Vertice) - 1) 'increment the temparray appropriately
  If SelRow > 1 Then
    For i = 0 To SelRow - 2 'vertice() is zero based and row number includes title
      Set TempVertice(i) = Vertice(i)
    Next i
    For i = SelRow - 1 To UBound(TempVertice)
      Set TempVertice(i) = Vertice(i + 1)
    Next i
  Else
    For i = 0 To UBound(TempVertice)
      Set TempVertice(i) = Vertice(i + 1)
    Next i
  End If
  Vertice = TempVertice
  NumVertices = UBound(Vertice) + 1
  Erase TempVertice
  PopulateFlexGrid
  frmCaveGrowth.picDisplay.Cls
  DispGrid.DrawGrid frmCaveGrowth.picDisplay, frmCaveGrowth.chkShowGrid,
frmCaveGrowth.chkShowLabels, &HDCDCDC
  frmCaveGrowth.DrawVertices
End Sub

Private Sub cmdOK_Click()
  Dim l As Long
  ReDim CaveWalls(0 To 1)
  'store first cave wall
  ReDim CaveWall(0 To NumVertices - 1, 1 To 2)
  For l = 0 To NumVertices - 1
    CaveWall(l, 1) = Vertice(l).X
    CaveWall(l, 2) = Vertice(l).Y
  Next l
  CaveWalls(0) = CaveWall
  CaveExists = True

  Unload Me
End Sub

Private Sub Form_Load()
  flx1.Row = 0
  flx1.Col = 0
  flx1.Text = "Vertex"
  flx1.Col = 1
```

```
  flx1.Text = "X"
  flx1.Col = 2
  flx1.Text = "Y"
  flx1.ColWidth(0) = 700
  flx1.ColWidth(1) = 2500
  flx1.ColWidth(2) = 2500
End Sub

Private Sub PopulateFlexGrid()
  Dim l As Long
  flx1.Rows = 1
  For l = 0 To UBound(Vertice)
    flx1.AddItem l + 1 & vbTab & Vertice(l).X & vbTab & Vertice(l).Y, l + 1
  Next l
End Sub
```

VB6 Project: prjCaveGrowth
Module: basGlobals

```
Option Explicit

Public Vertice() As CaveVertex
Public TempVertice() As CaveVertex
Public NumVertices As Long
Public DispGrid As GridLines

Public CaveExists As Boolean
Public CaveWalls() As Variant 'array of CaveWall arrays
Public CaveWall() As Double '2 dimensional array of cave vertices (X,Y)
```

VB6 Project: prjCaveGrowth
Class Module: CaveVertex

```
Option Explicit
Private m_X As Double
Private m_Y As Double
Private m_UnderWater As Boolean

Public Property Get X() As Double
  X = m_X
End Property

Public Property Let X(ByVal NewX As Double)
  m_X = NewX
End Property

Public Property Get Y() As Double
  Y = m_Y
End Property

Public Property Let Y(ByVal NewY As Double)
  m_Y = NewY
End Property

Public Property Get Underwater() As Boolean
  Underwater = m_UnderWater
End Property

Public Property Let Underwater(ByVal NewUnderwater As Boolean)
  m_UnderWater = NewUnderwater
End Property

Private Sub Class_Initialize()
End Sub
```

VB6 Project: prjCaveGrowth
Class Module: GridLines

```
Option Explicit
Private m_Xmin As Double
Private m_Xmax As Double
Private m_Ymin As Double
Private m_Ymax As Double
Private m_XDiv As Double
Private m_YDiv As Double

Public Property Get XDiv() As Double
  XDiv = m_XDiv
End Property

Public Property Let XDiv(ByVal Div As Double)
  m_XDiv = Div
End Property

Public Property Get YDiv() As Double
  YDiv = m_YDiv
End Property

Public Property Let YDiv(ByVal Div As Double)
  m_YDiv = Div
End Property

Public Sub DrawGrid(PicBox As PictureBox, grid As Boolean, labels As Boolean,
LineColor As Long)
  Dim Xmin As Double
  Dim Xmax As Double
  Dim Ymin As Double
  Dim Ymax As Double
  Dim startX As Double
  Dim startY As Double
  Dim CurrentX As Double
  Dim CurrentY As Double
  Dim OldWidth As Integer

  If m_XDiv = 0 Then
    MsgBox "XDiv is zero", vbCritical, "Invalid Grid Increment Value"
    Exit Sub
  End If
  OldWidth = PicBox.DrawWidth
  PicBox.DrawWidth = 1
  Xmin = PicBox.ScaleLeft
```

```
  Xmax = PicBox.ScaleLeft + PicBox.ScaleWidth
  Ymin = PicBox.ScaleTop + PicBox.ScaleHeight
  Ymax = PicBox.ScaleTop

  'PicBox.ForeColor = vbBlack
  startX = (Xmin \ m_XDiv) * m_XDiv '\ returns an integer
  CurrentX = startX
  Do While CurrentX < Xmax
    If grid Then
      PicBox.Line (CurrentX, Ymin)-(CurrentX, Ymax), LineColor
    Else
      PicBox.CurrentX = CurrentX
      PicBox.CurrentY = Ymax
    End If
    If labels Then PicBox.Print CurrentX
    CurrentX = CurrentX + XDiv
  Loop
  startY = (Ymin \ m_YDiv) * m_YDiv  '\ returns an integer
  CurrentY = startY
  Do While CurrentY < Ymax
    If grid Then
      PicBox.Line (Xmax, CurrentY)-(Xmin, CurrentY), LineColor
    Else
      PicBox.CurrentX = Xmin
      PicBox.CurrentY = CurrentY
    End If
    If labels Then PicBox.Print CurrentY
    CurrentY = CurrentY + YDiv
  Loop

  PicBox.DrawWidth = OldWidth
End Sub
```

APPENDIX II

S-PLUS Gamma Function Code

SPLUS Script: CaveDataGamma.ssc

Function:

```
CaveDataGamma<-function(numrecs, alpha, beta){
    df1<-data.frame()
    df1$FLOW<-rgamma(numrecs,alpha,beta)
    df1$FLOW2<-df1$FLOW*(12255649896/sum(df1$FLOW))
    df1
        }
```

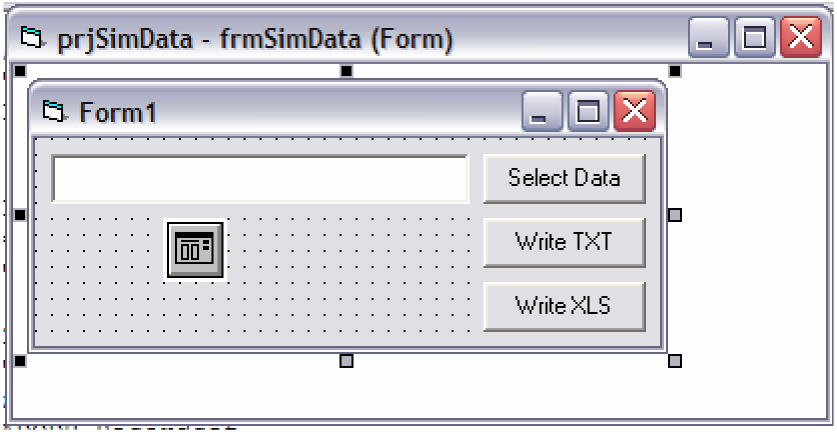
Commands Window Usage Example:

```
GA001B1 <- CaveDataGamma(8760, 0.01, 1)
```

APPENDIX IIII

Simulation Data Generation Code

VB6 Project: prjSimData
Form: FrmSimData




```
Option Explicit

Private oConn As ADODB.Connection
Private oRS As ADODB.Recordset
Private DataFile As String

Private Sub cmdGenerate_Click()
  Dim FileName As String
  Dim i As Long
  Dim JDay As Double 'Julian Days
  Dim Flow As Double 'Flow in Liters
  Dim Disch As Double 'Discharge in Liters per Second
  Dim Diss As Double 'Dissolution Rate in mm per year
  Dim XArea As Double
  Const FullPassage = 25 'Full Passage is about 24.75

  cdg1.DialogTitle = "Save Output Textfile"
  cdg1.FileName = Mid(DataFile, InStrRev(DataFile, "\") + 1, Len(DataFile) -
InStrRev(DataFile, "\") - 4) & ".txt"
  cdg1.InitDir = App.Path
  cdg1.Filter = "Text (*.txt)|*.txt"
  cdg1.ShowSave
  FileName = cdg1.FileName

  Open FileName For Output As #1
  i = 0
  If Not oRS Is Nothing Then 'make sure the recordset exists
    Print #1, "Julian Day, Dissolution Rate (mm/yr), Wetted X-Secton (m2), Flow (L),
Discharge (L/s)"
    Print #1, "0, 0, 0, 0" 'Line of zeros so that the first real time increment is not ignored in
the program
```

```
    oRS.MoveFirst 'move to start of recordset
    Do While Not oRS.EOF 'read in data until end of file
      i = i + 1
      JDay = i / 24 'decimal days - presumes records are in hours
      Flow = oRS.Fields(1).Value 'l
      Disch = Flow / 3600 'assumption that # of records is in hours
      Diss = 0.3412 * (1 - Exp(-0.0001168 * Disch))
      XArea = IIf(Disch <= 3500, 0.009284 * Disch + 1.867, FullPassage)
      Print #1, JDay & ", " & Diss & ", " & XArea & ", " & Flow & ", " & Disch
      oRS.MoveNext
    Loop
    Close #1
  End If
End Sub

Private Sub cmdSelect_Click()
  On Error GoTo ErrHandler
    Dim cat As New ADOX.Catalog

    If Not oRS Is Nothing Then
      oRS.Close
      oConn.Close
      DataFile = ""
      txtDataFile.Text = ""
    End If

    cdg1.DialogTitle = "Select Input Data"
    cdg1.FileName = ""
    cdg1.Filter = "*.xls|*.xls"
    cdg1.ShowOpen
    DataFile = cdg1.FileName
    If DataFile = "" Then Exit Sub
    Set oConn = New ADODB.Connection
    Set oRS = New ADODB.Recordset
    oConn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & "Data Source=" & DataFile &
";" & _
            "Extended Properties=""Excel 8.0;HDR=YES;"""

    Set cat.ActiveConnection = oConn

    oRS.Open "[" & cat.Tables(0).Name & "]", oConn, adOpenStatic, adLockOptimistic
    txtDataFile.Text = DataFile
    Set cat = Nothing
  Exit Sub
ErrHandler:
```

```
  MsgBox "Error in cmdSelectData_Click: " & Err.Number & vbNewLine &
Err.Description
  Resume Next
End Sub

Private Sub cmdXLS_Click()

  Dim NewRS As ADODB.Recordset
  Dim cat As New ADOX.Catalog
  Dim NewTable As ADOX.Table
  Dim NewCol As ADOX.Column
  Dim i As Long
  Dim JDay As Double 'Julian Days
  Dim Flow As Double 'Flow in Liters
  Dim Disch As Double 'Discharge in Liters per Second
  Dim Diss As Double 'Dissolution Rate in mm per year
  Dim XArea As Double
  Const FullPassage = 25 'Full Passage is about 24.75

'"Julian Day (Days), Dissolution Rate (mm/yr), Wetted X-Secton (m2), Flow (L),
Discharge (L/s)"
  i = 0
  If DataFile = "" Then Exit Sub
  cat.ActiveConnection = "Provider=Microsoft.Jet.OLEDB.4.0;" & "Data Source=" &
DataFile & ";Extended Properties=""Excel 8.0;HDR=YES;"""
  Set NewTable = New ADOX.Table
  NewTable.Name = "SimData"
  Set NewCol = New ADOX.Column
  NewCol.Name = "Julian Day (Days)"
  NewCol.Type = adDouble
  NewTable.Columns.Append NewCol
  Set NewCol = Nothing
  Set NewCol = New ADOX.Column
  NewCol.Name = "Dissolution Rate (mm/yr)"
  NewCol.Type = adDouble
  NewTable.Columns.Append NewCol
  Set NewCol = Nothing
  Set NewCol = New ADOX.Column
  NewCol.Name = "Wetted X-Secton (m2)"
  NewCol.Type = adDouble
  NewTable.Columns.Append NewCol
  Set NewCol = Nothing
  Set NewCol = New ADOX.Column
  NewCol.Name = "Flow (L)"
  NewCol.Type = adDouble
  NewTable.Columns.Append NewCol
```

```
  Set NewCol = Nothing
  Set NewCol = New ADOX.Column
  NewCol.Name = "Discharge (L/s)"
  NewCol.Type = adDouble
  NewTable.Columns.Append NewCol
  Set NewCol = Nothing
  cat.Tables.Append NewTable
  Set NewRS = New ADODB.Recordset
  NewRS.Open "[SimData]", oConn, adOpenStatic, adLockOptimistic
  If Not oRS Is Nothing Then 'make sure the recordset exists
    oRS.MoveFirst 'move to start of recordset
    NewRS.AddNew 'Line of zeros so that the first real time increment is not ignored in
the program
    NewRS.Fields(0).Value = 0
    NewRS.Fields(1).Value = 0
    NewRS.Fields(2).Value = 0
    NewRS.Fields(3).Value = 0
    NewRS.Fields(4).Value = 0
    NewRS.Update 'save record
    Do While Not oRS.EOF 'read in data until end of file
      i = i + 1
      JDay = i / 24 'decimal days - presumes records are in hours
      Flow = oRS.Fields(1).Value 'l
      Disch = Flow / 3600 'assumption that # of records is in hours
      Diss = 0.3412 * (1 - Exp(-0.0001168 * Disch))
      XArea = IIf(Disch <= 3500, 0.009284 * Disch + 1.867, FullPassage)
      NewRS.AddNew
      NewRS.Fields(0).Value = JDay
      NewRS.Fields(1).Value = Diss
      NewRS.Fields(2).Value = XArea
      NewRS.Fields(3).Value = Flow
      NewRS.Fields(4).Value = Disch
      NewRS.Update 'save record

      oRS.MoveNext
    Loop
    MsgBox "Done"
    Set oRS = Nothing
    Set oConn = Nothing
    Set cat = Nothing
    Set NewRS = Nothing
  End If
End Sub
```