An Open-Source Software for Interactive Visualization Using C++ and OpenGL: Applications to Stochastic Theory Education in Water Resources Engineering

ROBBY FLORENCE,[1] FAISAL HOSSAIN,[2] DAVID HUDDLESTON[2]

[1]*Department of Computer Science, Tennessee Technological University, Cookeville, Tennessee* 38505-0001

[2]*Department of Civil and Environmental Engineering, Tennessee Technological University, 1020 Stadium Drive, Prescott Hall, Cookeville, Tennessee* 38505-0001

Received 24 July 2008; accepted 29 September 2008

ABSTRACT: The purpose of this article is to explain the design and implementation of an open-source engineering education software called Stochastic Theory Education through Visualization Environment (STEVE), version 2.0. In an earlier article, a proof-of-concept for a computer-aided visualization tool (also named STEVE, version 1.0) for stochastic theory education in water resources engineering was articulated[Q2] [see, Schwenk et al. Comput. Appl. Eng. Educ., 2008, in press). Using Java Native Interfacing, it was shown that STEVE 1.0 could wrap a space–time stochastic model written in any computer language and be independent of any specific language compiler during tool usage. This article describes the general philosophy, software design, and classroom usage for STEVE with significant improvements on visualization and user-friendliness (hence, rightfully called version 2.0). The software was created using the C++ programming language with the Microsoft Windows Applications Programming Interface (API). OpenGL was used for the visualization display, and the OpenGL Utility Toolkit (GLUT) was used to visualize text inside the OpenGL window. The instructor-specified simulation program on stochastic theory was written in Fortran 77. The application has user-friendly options for modifying input data and parameter specifications as desired by the instructor or the student user. STEVE 2.0 has been tested with the Windows XP and Windows Vista operating systems. For the benefit of interested users and software makers, we

Correspondence to F. Hossain (fhossain@tntech.edu).
Contract grant sponsor: Engineering Development Friends Endowment of Tennessee Technological University.

also provide the software application, a short tutorial and all pertinent source codes as freeware for download on our STEVE homepage at http://iweb.tntech.edu/saswe/steve.html.
© 2009 Wiley Periodicals, Inc. Comput Appl Eng Educ 17: 1–10, 2009; Published online in Wiley InterScience (www.interscience.wiley.com); DOI 10.1002/cae.20288

Keywords: water resources engineering; stochastic theory; curriculum; computer-aided visualization. OpenGL; C++

INTRODUCTION: MOTIVATION FOR STOCHASTIC THEORY VISUALIZATION

In an earlier article, Schwenk et al. [1] commented on the importance of stochastic theory visualization for water resources engineering education as follows:

> . . . most engineering university baccalaureate programs introduce students to these concepts only in the graduate level. Our recently concluded survey of curriculum on stochastic theory in water resources engineering education indicate that 84% of all courses in nation are graduate level. This means that the diverse but foundational concepts making up stochastic theory, such as random variables and processes, probability density functions, moments, geostatistics, autocorrelation, random field generation, time-series analysis etc., can overburden freshmen graduate students unless particular care is taken in demonstrating these concepts via real-world examples . . . Conventional teaching paradigm for delivering stochastic . . . continues to rest mostly on text-based pedagogy involving comprehensive stochastic theory books. While the traditional method is still needed, there is scope to make the subject matter more exciting and 'learner-friendly' by leveraging visualization technology.

For such a visualization system to be effective for stochastic theory education, Schwenk et al. [1] further reported that the visualization scheme should have the following features: (1) real-world application of a wide range of concepts of stochastic theory via a practical tool that allows convenient computational modeling of the variability of natural phenomena; (2) full interactive control to students over the tool to allow them to conveniently and rapidly modify concepts, parameter values through add/remove options, observe corresponding effect and thereby foster inductive learning and generate research curiosity; (3) multimedia and a computer-assisted technology, such as a graphical user interface (GUI), that combines (1) and (2) and further enhances the user-friendliness of the modular modeling system. Although there is no any educational software, to the best of our knowledge, tailored for stochastic theory education in water resources engineering, the interested reader can refer to some examples on visualization tools for environmental education from Lai and Wang [2], Valocchi and Werth [3], Li and Liu [4], and Rivvas et al. [5].

The purpose of this article is to explain the enhancement of an open-source engineering education software called *S*tochastic *T*heory *E*ducation through *V*isualization *E*nvironment (STEVE), version 2.0. In an earlier article appearing in the same journal, a proof-of-concept of an earlier version for STEVE (named STEVE, version 1.0) was articulated (see, Schwenk et al. [1]). Therein, Schwenk et al. [1] provided justification for the development of the visualization software on stochastic theory through survey of graduate and undergraduate curriculum across the nation and the perception of classroom instructors willing to use such a free software.

While the general concept embedded in STEVE (version 1.0) and its potential for classroom usage that can be afforded was described in that article of Schwenk et al. [1], specific software building issues were absent for interested software users and makers. This article therefore addresses the software design and implementation aspects along with significant improvements on visualization and user-friendliness (hence, justifiably called version 2.0). In essence, this article is a sequel to Schwenk et al. [1] as the second part of a two part series. Our motivation for such a design and implementation document is to encourage users, specifically software makers, to apply and modify the tool for continual improvement in an open-source manner.

Hereafter, we provide the details of the software design issues in a step-by-step manner. Second section describes the general philosophy of STEVE, while third section elaborates the software design aspects. Fourth section dwells on the classroom usage of STEVE 2.0. Fifth section describes the possible ways of improving initial understanding of difficult stochastic theory concepts using STEVE 2.0. Finally, conclusions are presented in the last section. We also provide the software application, user manual, a short tutorial, and all pertinent source codes as freeware for

download on our STEVE homepage at http://iweb.tntech.edu/saswe/steve.html.

GENERAL PHILOSOPHY OF STEVE 2.0

STEVE 2.0 can essentially embed any stochastic theory model and visualize its output. Typically, such a stochastic theory model manifests several different concepts (such as spatial statistics, temporal statistics, probability density functions, random fields, etc.) wherein the dominance of each concept can be controlled quantitatively through user-defined set of inputs. In STEVE 2.0, a stochastic theory model called "SREM2D" (two-dimensional satellite rainfall error model) developed by Hossain and Anagnostou [6]. This model employs a stochastic theory code written in Fortran 77, which corrupts a time series of rainfall fields in space and time as per user-specified error parameters. Users do not require a background on computing to use STEVE 2.0. The general flowchart for STEVE 2.0 is shown below:

General Folder and Data Organization of STEVE 2.0

We encourage that readers download our STEVE 2.0 application package that is provided as a freeware at http://iweb.tntech.edu/saswe/steve.html. Examination of the source codes and folders will better facilitate understanding of the STEVE software making process

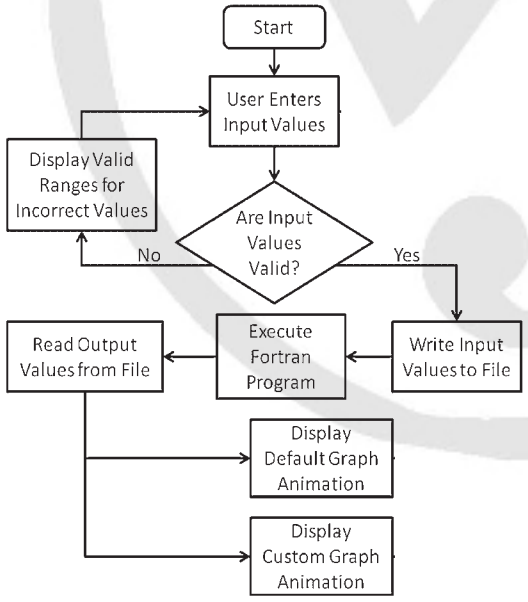


Figure 1 General flow-chart of STEVE 2.0 that visualizes the output of the Fortran-coded SREM2D against user-specified input.

described in this article. There are three folders, one readme file, and one executable (on STEVE GUI). The folders are:

- "**doc**": Contains all the necessary help and documentation literature for the user to access when needed from the GUI help menu. The user need not do anything to this folder.
- "**img**": Contains iconic images for the STEVE GUI. The user need not do anything to this folder.
- "**simul**": Contains the SREM2D Fortran code, the SREM2D Fortran code executable, user-specified input parameter file, user-specified input parameter range file, input data, and output data. It is basically this folder that the user needs to manipulate for STEVE 2.0 usage.

Starting STEVE 2.0

STEVE 2.0 opens the visualization window by clicking on the executable file STEVE.exe that is shown as an icon in the package (Fig. 2).

SOFTWARE DESIGN ASPECTS

There are six major software design aspects of the STEVE 2.0 program: (1) Stochastic Theory Simulation Program; (2) Program Window; (3) Input Form; (4) Visualization Process; (5) Visualization Color Scheme; and (6) Configuration Parameters. Hereafter, we describe the details of each of these six design aspects (note that we use the terms as proper nouns, and hence the capitalization of the first letter of each word).

Stochastic Theory Simulation Program

The Simulation Program (SREM2D, in this case) is separate from the STEVE program ("simul/simulation_fast.exe"). It executes a simulation with the input values from an Input Form, and its output is read by STEVE program. The Simulation Program is executed by the STEVE Window (Fig. 2). It reads the list of input values from the "simul/params.dat" file written by the Input Form. After the simulation is complete, the output file ("simul/output.dat") is read by the custom Visualization Process (described in detail in Visualization Process Section as aspect #4).

The detailed processing of the Simulation Program is not relevant to the development of STEVE 2.0. It is a "black box" entity, so any variation of the simulation program can be substituted by the user or

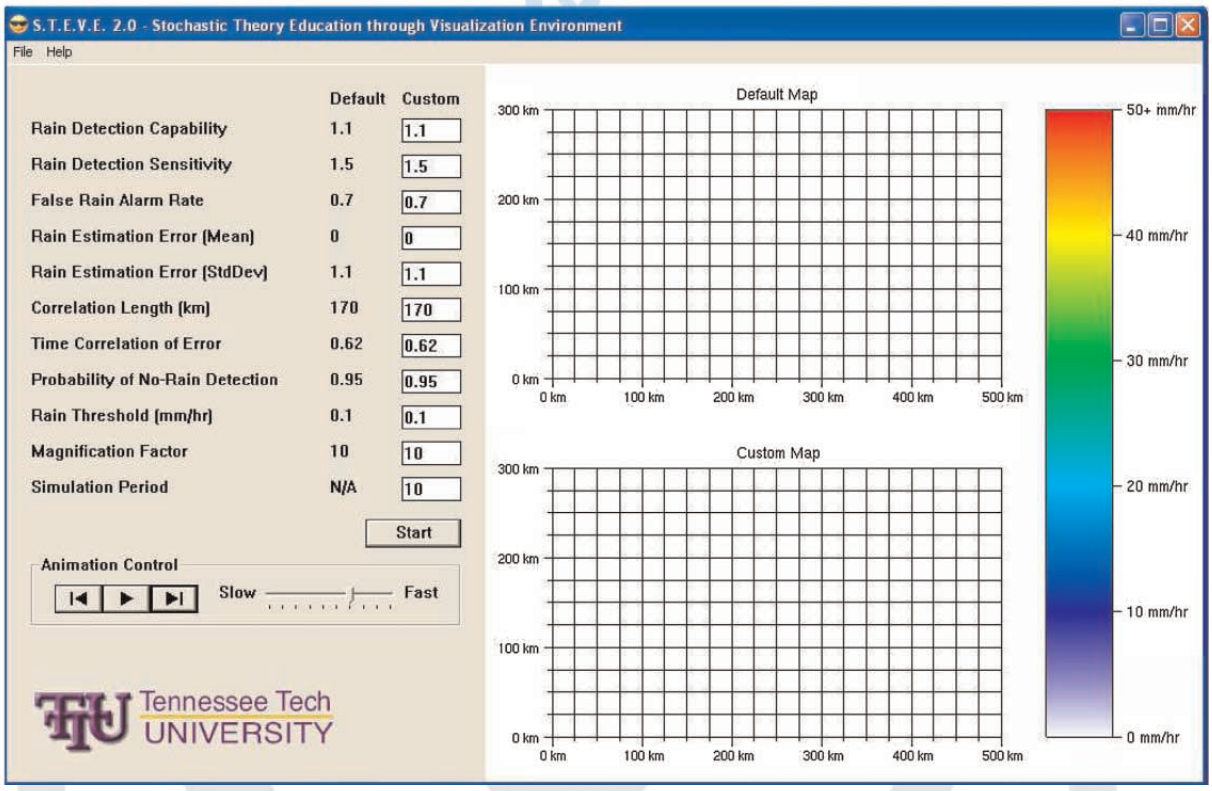


Figure 2 Screen-shot of STEVE 2.0. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

instructor in its place as long as it uses the same input and output formats. Although our software visualization package can be applied to many tasks, we have developed and implemented it using a stochastic theory simulation program unique to modeling satellite rainfall data because of our strong interest in water resources engineering.

The Simulation Program reads the parameters from ''simul/params.dat'' generated by the Input Form as well as simulation input from ''simul/input.dat.'' The latter file is not used in any way by the STEVE program. STEVE then writes the output of the simulation to ''simul/output.dat'' (Fig. 1). Also, in order to render the default visualizations, the file ''simul/default.dat'' must be created by executing the Simulation Program with the default parameter values and the maximum simulation period. The resulting output file should be renamed to ''default.dat.'' This only needs to be done once when a new Simulation Program is used in the project.

Program Window

The Program Window is the main part of the STEVE program. STEVE creates the window, menu, Input Form, and Visualizations (see Fig. 2). It handles the Windows message loop and messages for itself and the Input Form. The STEVE Program Window manages communication between the Input Form, the Simulation Program, and the Visualizations. It also enables OpenGL for the visualizations and controls their animation (Fig. 1). In essence, this window is part of the GUI that is manifested through the Input Form (described in Input Form Section).

STEVE creates an Input Form at startup and two Visualizations after a simulation run have been executed (e.g., notice the two maps on the right-hand side of Fig. 2). It handles window messages for the Input Form and calls input form functions when the corresponding messages are received. The Program Window is responsible for executing the Simulation Program when the ''Start'' button of the Input Form is pressed. Lastly, it draws the visualizations as well as the map axes and color bar.

When the ''Start'' button is pressed, the Program Window creates and registers the window class (Windows API, not this design entity). It then creates the main window for the program, where the Visualizations will be drawn, and the parent window for the input form. The Program Window also creates a shell

execution information structure to run the Simulation Program.

The Program Window enables OpenGL in its main window for the Visualizations by getting a handle to a device context, a pixel format that is appropriate for both the graphics being drawn and the monitor, and creating a rendering context. OpenGL commands can then be executed and will be drawn inside the main window. OpenGL is disabled when the program exits by deleting the rendering context.

The STEVE Program Window also handles the message loop for itself and the Input Form. The loop continues until the program quits. If there is a message waiting in the message queue, it is sent to the WndProc function. Otherwise, Window draws the visualizations. The WndProc function handles any Windows messages, including when a button in the Input Form is pressed, when a menu item is selected, when the program window is resized, and when the program window is closed. When the Start button in the Input Form is pressed, Window tells the Input Form to check the user input values. If the test passes, the Input Form writes the user input values to the Simulation Program's input file (''simul/params.dat'').

A small ''Please Wait'' dialog box is displayed while the simulation program is running. The Simulation Program is then executed using the shell execution info created in the WinMain function. The program waits until the Simulation Program finishes. This is accomplished by a loop that checks for any Window messages to the ''Please Wait'' dialog box, handles the messages if there are any available, and then waits until either the Simulation Program finishes or a new Window message is added to the message queue. The only message handled by this loop is the message generated when the user clicks the ''Cancel'' button on the dialog box. The loop checks if the ''Cancel'' button has been pressed and if so, the Simulation Program is terminated and the program refreshes to its original execution. After the simulation finishes (if it was not canceled), the time it took to run is displayed by the Input Form, and two new visualizations are created. The default visualization is created from a pre-made output file (''simul/default.dat'') generated by the Simulation Program with the default input values, and the custom visualization is created from the new output file of the Simulation Program (''simul/output.dat'').

The WndProc function also handles messages from the program's menu by calling the appropriate Input Form functions or opening help documents. It creates dialog boxes for the ''Report a Bug'' and ''Acknowledgements'' menu items. These dialogs, along with the ''Please Wait'' dialog box displayed while the simulation is running and the menu itself, are resources created in ''resource.rc.'' The ''Report a Bug'' and ''Acknowledgements'' dialog boxes have separate message handler functions called RepBugDlgProc and AckDlgProc, respectively. Both functions handle the message to remove the dialog box when it is closed. AckDlgProc additionally loads the institution logo (Tennessee Technological University) from ''img/TTULogoSm.bmp'' when the dialog box is created and launches the default web browser to the project's website when the URL is clicked.

When the main Program Window is resized, WndProc handles the message and calls the resizeWnd function. This function extends the Input Form to the bottom of the resized window and resizes the OpenGL viewport to the new dimensions of the window minus the space taken up by the Input Form. It also sets the OpenGL orthographic projection, allowing the Visualizations to be drawn in two dimensions instead of three.

The Program Window draws the visualizations in the message loop and controls their animation. The map axes and color bar are always drawn, and both Visualizations are drawn if they have been created. If the Back button is pressed in the animation controls of the Input Form, the frame of both visualizations is decremented. If the Forward button is pressed or the animation delay time has passed, the frame of both Visualizations is incremented. The animation delay time is measured in clock ticks since the program started. If the animation is not paused, the animation delay is retrieved from the Input Form in seconds and converted to the next number of clock ticks to advance the frame.

Input Form

The Input Form class creates a GUI for the user to enter input values to send to the Simulation Program, start the Simulation Program, and control the animation of the Visualizations. It also writes the parameters to the Simulation Program's input file. Input Form can load or save the user's parameters to a user-defined file.

Window creates an Input Form and handles all window messages sent to the Input Form. The Input Form class creates a list of Parameters for all input values needed by the Simulation Program. Input Form's constructor sets the parent window of the form elements and creates the list of Parameters, giving each Parameter its name and other values. The parent

window must be created before the Input Form. The list of Parameters is read from the file ''simul/paramInfo.dat.'' The first line of this file must always be the number of Parameters in the list. The following line is the column headers for each Parameter's name, minimum, maximum, and default. This line is ignored. Input Form reads each remaining line and creates a Parameter with the information from the line. The name of the Parameter must be separated from the minimum value by at least one tab. All characters up to the first tab in the line are stored as the Parameter's name. By using this file to create the list of Parameters, the number and type of stochastic theory concepts manifested by parameters can be changed to allow changes to the Simulation Program.

The createWindows function is called by Window after the Input Form is instantiated. This function displays the list of Parameters in the parent window, each with a name (static), default value (static), and user input value (edit). A ''Start'' button is created below the list. A box to control the animation of the visualizations is created below the Start button, with three buttons to move back one frame, play/pause, and move forward one frame (Fig. 2). A track bar is created to control the speed of the animation. The images for the animation buttons are loaded from the corresponding files in the ''img'' folder. An empty static field is created to display the simulation generation time after a simulation has been completed. The institution logo (Tennessee Technological University in this case) is loaded from ''img/TTULogo.bmp'' and displayed on the bottom. When the Input Form is created or when it is reset through the program's menu, all user input values are set to the default value of the respective Parameter.

Before the Simulation Program is executed, all input values must be checked to make sure they are between the parameters' minimum and maximum values. This ensures that the Simulation Program does not crash or produce an unrealistic output.dat file. If one or more of the input values are invalid, an error message will appear listing all invalid values and Window will not execute the Simulation Program. The Input Form then writes the user input values to a file, which will be read by the Simulation Program (''simul/params.dat''). Once the Simulation Program completes, Window will calculate the time it took to run, and Input Form will display the time in ''mm:ss'' format.

The Input Form class also has capabilities to load and save the user's list of input values for later use. These functions open a standard Windows ''Open'' or ''Save As'' dialog box, and either set the user input values in the GUI to the values in the file or write the user input values to the file. In the animation controls, Input Form alternates between play and pause when the play/pause button is pressed and changes the image displayed in the button accordingly. Input Form also calculates the time between each frame of the Visualizations based on the position of the animation speed track bar. There are 10 positions on the track bar, with the right position representing 0.33 s per frame and each additional position to the left adds 0.33 s to the time between each frame. Input Form stores the handle to its parent window (HWND). This window is created before the Input Form and cannot be changed after the Input Form is created.

The Input Form class also stores handles to the play and pause images (HGDIOBJ), which are needed when the play/pause button is pressed to alternate images. Lastly, this class stores the number of Parameters (int) and the list of Parameters (Parameter) for the required input values of the Simulation Program. This array is dynamically allocated in the Input Form constructor and cannot be changed after the Input Form is instantiated.

Visualization Process

The Visualization class draws all OpenGL elements in the program, including the map axes, the color bar, and the output map of the Simulation Program. It reads the output file of the Simulation Program and draws a map of the simulation at each time step.

The Window class creates two visualizations for the default and custom maps. It also controls the animation of the maps. The output file of the Simulation Program (''simul/output.dat'') is read for the custom visualization, and the default output of the Simulation Program (''simul/default.dat'') is read for the default visualization. It uses the Color class to store the color for each grid in the map.

When a visualization is created, it reads the output file of the Simulation Program and stores the values in a three-dimensional array of floats. The first dimension of the array is the time step, which is given as a parameter to the constructor, followed by the row and column of each value. The array is dynamically allocated and is deleted when the visualization is deleted.

The Visualization Process draws the map at the current time step with the draw function. The top left corner of the area to draw the map is given to draw the map in the top or bottom map area. These parameters should always be the predefined constants MAP1_LEFT, MAP1_TOP or MAP2_LEFT, MAP2_TOP. The current time period of the Visualization is displayed in the center of the GL window. To make

the gradients smooth, each grid is divided into four triangles. The color for each value in the array is set at the vertex in the center of the grid. Looping through all but the last row and column, the values in the map at (row + 1, col), (row, col + 1), and (row + 1, col + 1) form a square. The colors for these four values are averaged, and a fifth vertex is created in the center of the square, where the grid lines intersect. The five vertexes are then connected by triangles. A total of 16 triangles are drawn for all interior grids, connecting them with all 8 adjacent grids. When this loop finishes, a small border (0.05 GL units) still remains undrawn in the map.

Because there are not values outside the map to get four color values, the outer edges cannot be drawn in the same way as the center of the Visualization. To draw the vertical edges, rectangles are drawn connecting each grid with the grid below it. The top two vertices are set to the color of the upper grid, and the bottom two vertexes are set to the color of the lower grid. The horizontal edges are drawn in the same way. This still leaves an undrawn square in each corner with a side length of 0.05 GL units. These squares are filled with the color of adjacent corner grid of the map.

The functions to draw the map axes and the color bar are static and can be called without an instantiated Visualization class. To draw the map axes, the drawAxes function receives the top left corner of the area to draw the axes in the same way as the draw function. The axes are labeled ''Default Map'' or ''Custom Map'' depending on the given coordinates. A grid line is drawn every 0.1 OpenGL units, and every four grid lines extends a little farther out of the map and is labeled. Each grid represents 25 km on the map. To draw the color bar, a large rectangle is drawn from the top of the top map to the bottom of the bottom map. The rectangle is divided into five smaller rectangles, with the color gradient from red at the top, to yellow, green, cyan, blue, and finally white at the bottom. Each interval is labeled and represents 10 mm/h. The top red interval is labeled ''50+ mm/h'' to show that all values greater than 50 are colored red.

When the draw function needs to determine the color of a value in the map, the getColor function interpolates the color based on the color bar. The color bar of white, blue, cyan, green, yellow, and red allows the RGB value of the color to be determined exactly because each red, green, and blue value is either 0.0 or 1.0 for these colors. For values in the range (0, 10], the blue value is always 1.0, and the red and green values are interpolated from 1.0, if the value is 0.0, to 0.0, if the value is 10.0. The equation ''1 − value/10.0'' gives this output. The value is divided by 10.0 because the interval between white and blue is 10.0 mm/h. For values in the range (10, 20], the red value is always 0.0, the blue value is always 1.0, and the green value is interpolated from 0.0 to 1.0. The equation ''(value − 10.0)/10.0'' gives this output. The subtraction is needed to get the percentage the value has passed the previous interval (blue, 10.0 mm/h). The color for values in the rest of the intervals are determined in the same way, in the range (20, 30] for cyan to green, (30, 40] for green to yellow, (40, 50] for yellow to red. Everything greater than 50.0 is red, and 0.0 is the default color of white.

The main data member of visualization is the three-dimensional map array (float***). This array is dynamically allocated to the correct size when the Visualization is created. Visualization also contains the simulation period of the map (int), which is used to read the correct amount of data from the Simulation Program's output file and to loop the animation. The map and simulation period cannot be changed after the Visualization is created. This class contains the current frame of the animation (int), which is incremented or decremented by the incFrame and decFrame functions. The frame is the time step to display when the draw function is called.

Visualization Color Scheme

The Color class contains a color value in RGB format. The Visualization class uses Color to store the RGB value for a point in the grid and update the OpenGL color. Color's default constructor sets its value to white (1.0, 1.0, 1.0). Another constructor is available to set the RGB values when the Color is created (not currently used in the program). The RGB values can be changed after the Color is created by the setRGB function. All color values are restricted between 0.0 and 1.0 (inclusive) when they are changed by the constructor or setRGB function. Color contains a color's red, green, and blue values (double). The data members are private and can be retrieved by public ''get'' functions.

Configuration Parameters

The Input Form creates a list of Configuration Parameters (hereafter called ''Parameters''). The Parameter class contains information about an input value for the Simulation Program, including its name, minimum value, maximum value, and default value. Parameters do not reference any other entities. A Parameter is given all of its values in its constructor. The values are checked to make sure the maximum is

greater than or equal to the minimum and the default is between the minimum and maximum.

Parameters contains the name (char[50]) of the input value, as well as its minimum, maximum, and default values (double). All of these data members are private and can be retrieved by public ''get'' functions. The data members cannot be changed after the Parameters is created.

CLASSROOM USE OF STEVE 2.0

After clicking on the STEVE.exe icon, the user can key in quantitative values in the input fields on the left-hand side of the GUI. If the values are beyond a realistic range specified in ''simul/paramInfo.dat,'' an error message will appear and the program will not execute. The user can also specify how many time steps the SREM2D code will run (in our case, the data allow a choice between 1 and 2,952 time steps).

Once all the fields are keyed in, the user hits the start button. The GUI transfers the user-specified parameters to SREM2D for execution and generation of ''output.dat.'' The total simulation time is printed when the execution is complete. Once simulation is complete, STEVE visualizes the output.dat (a space−time field of rainfall shown as ''Custom Map'') simultaneously with the ''default.dat'' (shown as ''Default Map'').

The simultaneous visualization allows the student user to observe visually the impact on space and time of the quantitative difference in input parameter value between the default setting and the custom (user-specified) setting (see Fig. 3). It is essentially this user-friendly and interactive feature of STEVE that we hypothesize as making the subject of stochastic theory more interesting than textbook pedagogy. (*Note*: In our earlier ensemble of works reported by Schwenk et al. [1] and Hossain and Huddleston [7], complete details on specific stochastic theory concepts are provided). User can stop the animation and observe one snapshot at a time for closer inspection. User may also rewind, forward one snap shot at a time, and also manipulate the animation speed. In addition to all this, the user can perform numerical analysis of the ''simul/output.dat'' and ''simul/input.dat'' or ''simul/default.dat'' to explore the stochastic concepts further.

Using the ''File'' menu of STEVE, user may also save the settings on input parameters. To save the

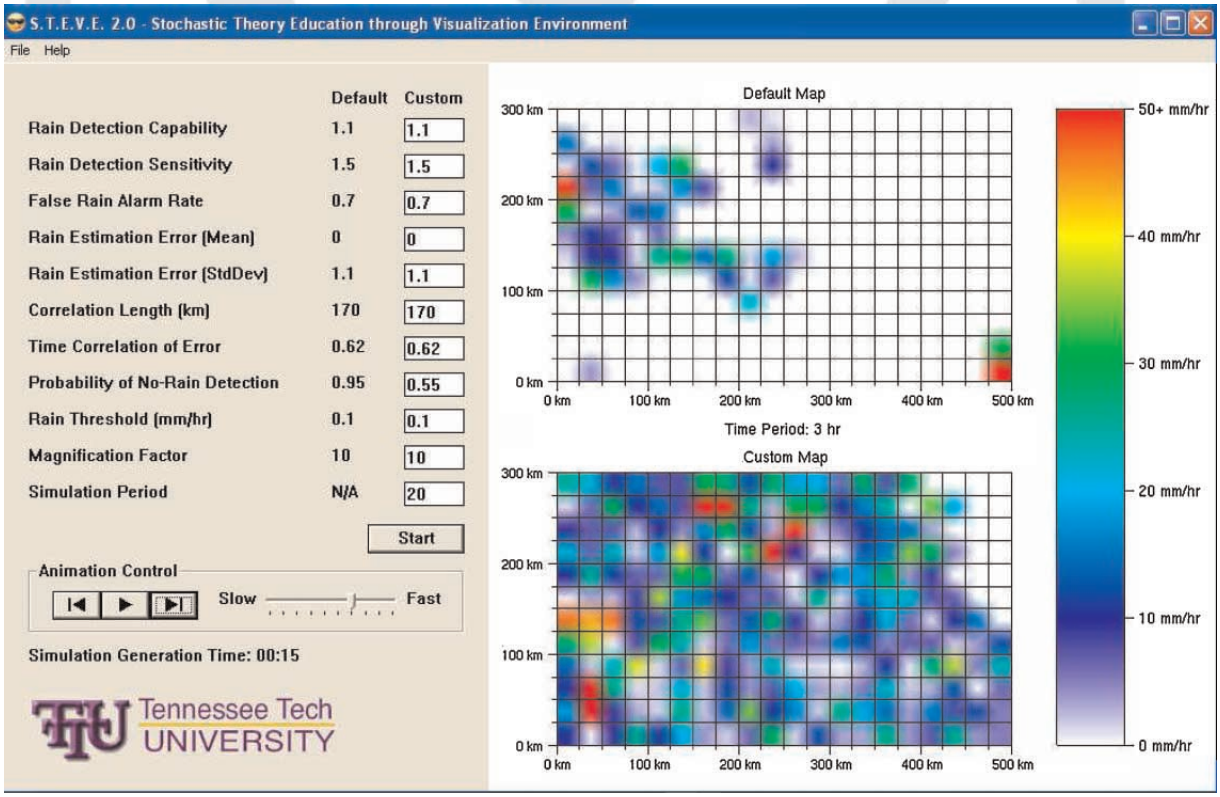


Figure 3 Executing STEVE as per user-specified input and observing the visual nuances between default and custom maps that can connect to a set of stochastic theory concept. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

''output.dat'' with a particular filename so that user can identify it in future against the input setting and/or perform statistical analysis, the user needs to use the usual file renaming using ''Explorer'' or ''My Computer.''

For convenience of the user, there exists a subfolder named ''standard'' under ''simul.'' In this subfolder, there are three files:

- ''inputstd.dat'': This file contains the input time series of rainfall fields. The user should treat this as a backup version of the ''input.dat'' that is in the main ''simul'' folder. If the ''input.dat'' file is accidentally deleted, the user should copy this ''inputstd.dat'' file to the ''simul'' folder and rename it ''input.dat.''
- ''inputstd_magnified10.dat'': This file contains the ''input.dat'' with values magnified by a factor of 10. This file can be used to visualize the input.dat in STEVE with magnified rainfall values if there is a need. The user needs to copy this file to ''simul'' folder and rename it as ''default.dat.'' The visualization then appears on the upper panel.
- ''default_magnified10.dat'': Same as ''input_magnified10.dat'' except that it is the output file generated with default SREM2D parameters with output values magnified by a factor of 10 (make the visualization color scheme prominent). The user needs to copy this file to ''simul'' folder and rename it ''default.dat'' (visualization will appear on the upper panel).

USING STEVE 2.0 FOR IMPROVING UNDERSTANDING OF STOCHASTIC THEORY CONCEPTS

As non-exhaustive set of examples, the following stochastic theory concepts can be interacted with in STEVE 2.0 (reader is encouraged to refer to [6]):

(1) Logistic regression: Rain detection capability and rain detection sensitivity.
(2) Probability density function: False alarm rain rates.
(3) First and second-order moments: Mean and standard deviation.
(4) Geostatistics, random fields, and variograms: Correlation length.
(5) Autocorrelation: temporal correlation.
(6) Bernoulli trials: probability of no-rain detection.

The user should either increase or decrease each parameter value from the default value and then compare the visualized output with the custom map. Subsequently, the user should try to reconcile the observed differences with the theory or initial understanding of the specific stochastic theory concept. For example, an increase in correlation length can mean that the rainfall structure may look ''stretched'' more. Similarly, if the probability of no-rain detection is reduced (from 0.95 to 0.55, Fig. 3), this means that 45% Bernoulli trials would be unsuccessful in detecting no-rain and hence, these events would then be subjected to false alarm rain rates (which can also be played with).

It is up to the user how he/she wants to use STEVE based on instruction provided by the instructor. It is recommended that the instructor provides some guidance and suggestions for setting up various hypothesis construction experiments before using STEVE 2.0. Understanding of the significance of each of these stochastic concepts is better appreciated if the general concepts of SREM2D error corruption are understood first. We encourage that instructor first explains the SREM2D concept (or any other stochastic model in general that STEVE can use as the black-box Simulation Program) through an introductory workshop prior to STEVE 2.0 usage.

CONCLUSIONS

This article explained the design and implementation of an engineering education software called STEVE, version 2.0. Readers were encouraged to download the freeware software package and source codes available at http://iweb.tntech.edu/saswe/steve.html and examine the contents and source codes. The motivation for such a design and implementation document was to encourage users specifically, software makers, to apply and modify the tool for continual improvement. The software was created using the easily available C++ programming language with the Microsoft Windows Applications Programming Interface (API). OpenGL was used for the visualization display, and the OpenGL Utility Toolkit (GLUT) is used to visualize text inside the OpenGL window. The instructor-specified stochastic theory simulation program was written in Fortran 77, although the simulation program itself is a ''black-box'' in STEVE 2.0. The application has user-friendly options for modifying input data and parameter specifications as desired by the instructor or student user. STEVE 2.0 has been tested with the Windows XP and Windows Vista operating systems. It is our hope that the open-source nature of STEVE 2.0 will prompt software makers to improve such educational

tools and make them available freely for the student and instructor community.

ACKNOWLEDGMENTS

The first author gratefully acknowledges the support received from the Engineering Development Friends Endowment of Tennessee Technological University in the form of a grant. Partial support from the NASA New Investigator Program (Hossain) is also acknowledged.

REFERENCES

[1] J. Schwenk, F. Hossain, and D. Huddleston, A[Q3] computer-aided visualization tool for stochastic theory education in water resources engineering, Comput Appl Eng Educ (2008) (in press).

[2] X. Lai and P. Wang, GeoSVG: A web based interactive plane geometry system for mathematics education, Proceedings of ICET 2006—Education and Technology, July 17–19, Alberta, Canada, 2005. (File last retrieved on July 24, 2008 from http://www.actapress.com/PaperInfo.aspx?PaperID=27538).

[3] A. J. Valocchi and C. J. Werth, Web-based interactive simulation of groundwater pollutant fate and transport, Comput Appl Eng Educ 12 (2004), 75–83.

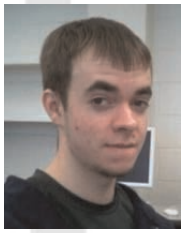
[4] S.-G. Li and Q. Liu, Interactive groundwater (IGW): An innovative digital laboratory for groundwater education and research, Comput Appl Eng Educ 11 (2003), 179–202.

[5] A. Rivvas, T. Gomez-Acebo, and J. C. Ramos, The application of spreadsheets to the analysis and optimization of systems and processes in the teaching of hydraulic and thermal engineering, Comput Appl Eng Educ 14 (2006), 256–268.

[6] F. Hossain and E. N. Anagnostou, A two-dimensional satellite rainfall error model, IEEE Trans Geosci Remote Sensing 44 (2006), 1511–1522 (10.1109/TGRS.2005.863866).

[7] F. Hossain and D. Huddleston, A proposed computer-assisted graphics-based instruction scheme for stochastic theory in hydrological sciences, Comput Educ J XVII (2007), 16–25.

BIOGRAPHIES



Robby Florence graduated in 2008 with a BS in Computer Science from Tennessee Technological University. He is currently pursuing a MS at University of North Carolina, Chapel Hill. His research interests span the development of user-friendly education software.

Faisal Hossain is an assistant professor in Civil Engineering at Tennessee Technological University. He obtained his PhD in Environmental Engineering from the University of Connecticut in 2004 and his BS from the Indian Institute of Technology, Varanasi (India) in 1996. His research interest in engineering education involves the development of visualization softwares for enhancing instruction in the classroom. He is the recipient of the NASA New Investigator Program Award in 2008. Since 2006, he has been the associate editor of the Journal of American Association of Water Association.

Dr. David H. Huddleston PhD, PE, was appointed Interim Dean, College of Engineering, at Tennessee Technological University (TTU), Cookeville, Tennessee, in July 2007. Previously, he served as professor and chair of the Department of Civil and Environmental Engineering at Tennessee Technological University (TTU), Cookeville, Tennessee, beginning in August 2004. He earned his BS in Engineering Science at TTU, his MS in Engineering Science and Mechanics from Virginia Polytechnic Institute and State University, and his PhD in Engineering Science from the University of Tennessee. His current research interests include computational hydrodynamics, computational fluid dynamics, computational design, and surface water quality modeling. He is a registered engineer in Mississippi and Tennessee. Prior to his appointment at TTU, Huddleston held faculty appointments in the Civil Engineering and Computational Engineering Departments at Mississippi State University. Before entering academia, Huddleston was employed by Sverdrup Technology, Inc. and Pan-Am World Services, Inc. at the USAF Arnold Engineering Development Center, and TRW, Inc.

Q1: Please check the suitability of the short title on the odd-numbered pages. It has been formatted to fit the journal's 45-character (including spaces) limit.

Q2: Please check the presentation of the reference citation in the abstract and also update.

Q3: Please update.