A COMPUTATIONAL CELLULAR AUTOMATON FOR MODELING SURFACE WATER FLOW IN ROCKY MOUNTAIN NATIONAL PARK AND THE WALNUT GULCH EXPERIMENTAL WATERSHED

THESIS

Presented to the Graduate Council of
Texas State University
in Partial Fulfillment of
the Requirements

For the Degree
Master of APPLIED GEOGRAPHY

By

Jay A. Parsons, B.S.

San Marcos, Texas
May 2004

ACKNOWLEDGEMENTS

I wish to thank my adviser, Dr. Mark Fonstad, for his enthusiasm and interest in my research and academic career.

I would also like to thank my committee members, Drs. Sharolyn Anderson and Deborah Bryan, for their patience and insight to help me throughs process.

I want to especially thank my wife, Viki. Her support and encouragement for my return to school to pursue an advanced degree were invaluable.

This manuscript was submitted on April 19, 2004

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

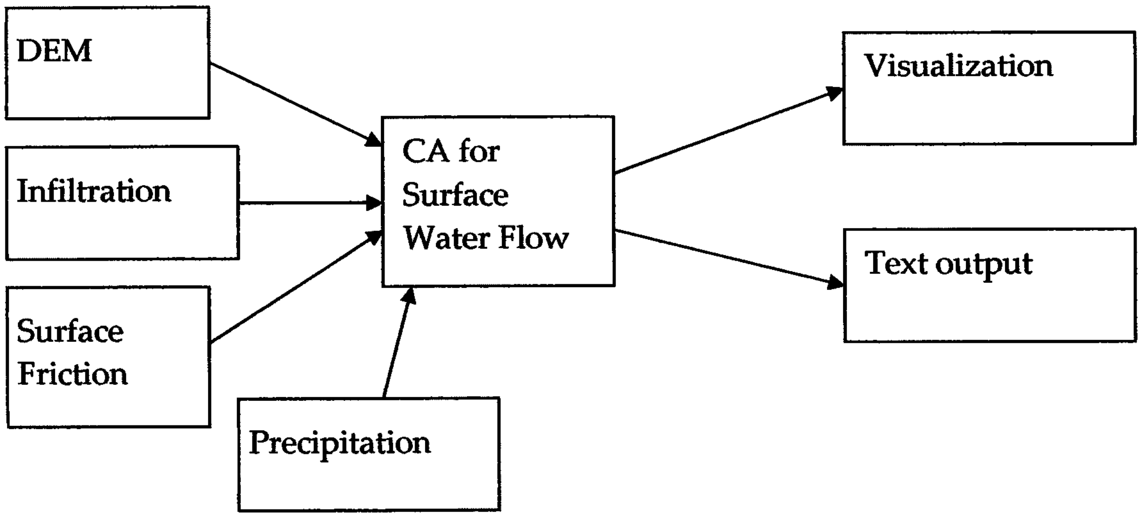
INTRODUCTION

Modeling surface water flow is an important part of hydrological studies. Understanding how water flows over a terrain will help predict floods and accurately predict and monitor the recharging of water reservoirs. This paper describes a computational model that answers the research question, "Can surface water flow be accurately modeled using Cellular Automata theory?"

Several techniques have been used to computationally model surface water flow. The two primary techniques are kinetic and dynamic wave theory (differential equations) and cellular automata(CA). Cellular automata models are appealing to geographers due to their inherent spatial properties. In CA models, the study area is divided into a grid or matrix of cells; each cell has its own spatial properties based on its position within the matrix. Cell behavior and state is dictated by a set of rules. Much like Waldo Tobler's First Law of Geography that states that all things are related, but nearby things are more related than distant things (Tobler 1970), a cell's state is determined by its neighborhood of nearby cells and the predetermined rules.

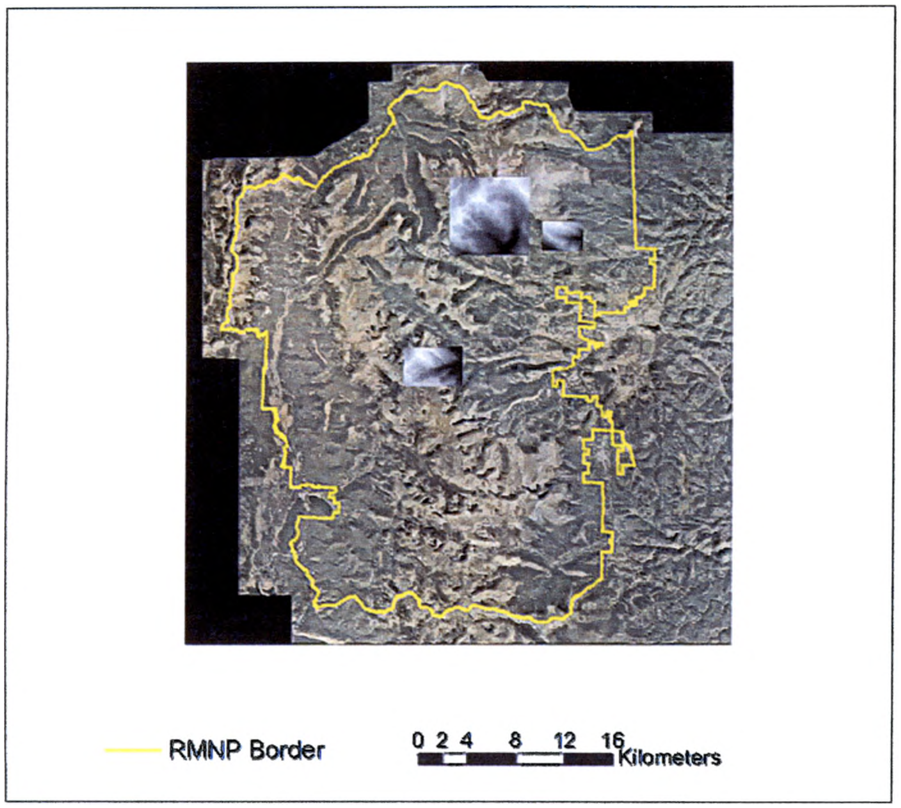
This paper describes the development and application of a computational model based on cellular automata theory to simulate surface water flow over a terrain for specific study areas. The computational model requires four data inputs: a digital elevation model of a study area, either separate layers or constant values for infiltration, surface roughness values and a precipitation event to accurately portray surface water flow. Unlike most current cellular automata models, this model is time explicit with attempts to model a watershed's unsteady response to a precipitation event. The computational CA model computes water flow and provides several means of visualization and text based output. Figure 1 shows the input layers (or values) and output options of the CA model.

Figure 1: Diagram of inputs and outputs for the surface water flow model.



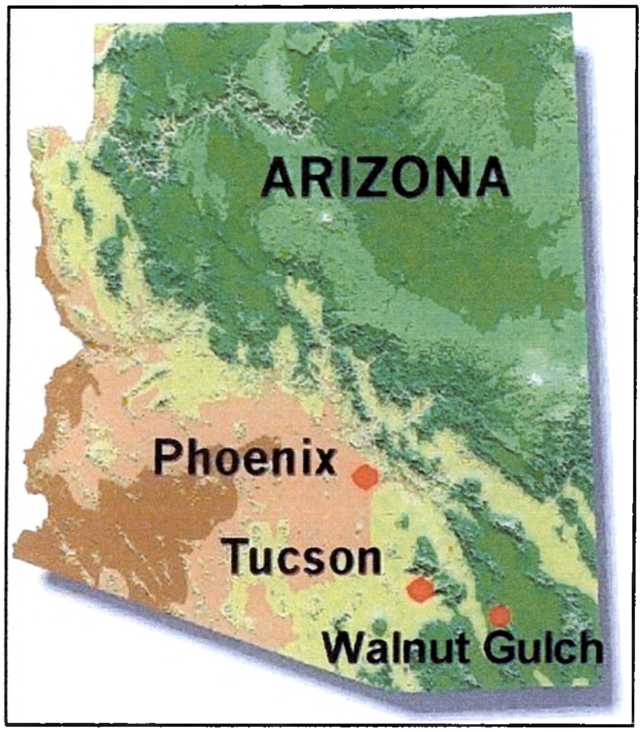
Two study areas were modeled for this research, the Rocky Mountain National Park (RMNP) in Colorado and the Walnut Gulch Experimental Watershed (WGEW) in Arizona. The park is a useful study area due to its limited development and availability of terrain data. In Figure 2, the Rocky Mountain National Park is outlined in yellow and the digital elevation models (DEMs) for the study sites are in grayscale. These sites were solely used for simulations because of the limited amount of precipitation and flow data that was available to validate the model for the area.

Figure 2: Rocky Mountain National Park, CO (outlined in yellow) and three study sites within the park (grayscale DEMs).



The second study area is at the Walnut Gulch Experimental Watershed (WGEW) in Arizona, shown in Figure 3. It is an excellent study site because researchers from the Southwest Watershed Research Center (SWRC) were able to provide storm event data to validate the model. This semi-arid watershed has limited elevation changes and provides a unique contrast to the high relief, forested watersheds in the RMNP.

Figure 3: Walnut Gulch Experimental Watershed is located in southeast Arizona (SWRC 2004).



CHAPTER II

LITERATURE REVIEW

The literature pertinent to the development of a CA model for surface water flow falls into several categories: cellular automata, CA and hydrological processes, and additional water flow models. The analysis of these previous works will demonstrate the importance of modeling tools.

Cellular Automata

Stephen Wolfram wrote "Cellular Automata" in 1983 to describe how CA models can be used to research and simulate complex systems. Wolfram explains that CAs "may be considered as discrete idealizations of the partial differential equations often used to describe natural systems. Their discrete nature also allows an important analogy with digital computers: cellular automata may be viewed as parallel-processing computers of simple construction" (Wolfram 1983, 2). He also describes CA as "discrete dynamical systems with simple construction but complex self-organizing behavior"

(Wolfram 1984, 1) much like natural systems such as surface water flow. Wolfram believes that cellular automata theory replaces differential equations as the basis for most current models of natural systems, "Cellular automata are mathematical models for complex natural systems containing large numbers of simple identical components with local interactions" (Wolfram 1984, 1). This statement characterizes the current state of hydrological modeling which is divided into CA and differential equation techniques. Wolfram has written many articles on cellular automata and their uses in modeling natural systems that lay the foundation of CA research.

CA Hydrological and Sediment transport models

In 1997, Brad Murray and Chris Paola describe cellular models as providing "the freedom for the system to evolve spatially and form complex patterns without constraint" (Murray and Paola 1997, 1002). This description emphasizes that CA models contain no rules to govern the whole environment, but the simple rules between a cell and its surrounding neighborhood dictate how that cell changes. When each cell is modified in this manor, complex systems may develop over the entire study area. Thus CA based models are useful for systems with spatially complex, self-organizing patterns (Murray and Paola 1997). Murray and Paola developed a cellular model to simulate the creation of braided rivers over time by modeling sediment transport with water flow (Murray and Paola 1994). Their model is essentially a "channel" model in that the water from one cell can flow only to the three lower cells.

R. Thomas and A. P. Nicholas (2001) expanded on the work begun by Murray and Paola. They sought to improve on previous work by "presenting a new scheme for routing flow in geomorphological models, and by assessing the performance of this approach using both field data and the predictions of a hydrologic model that solves the shallow water form of the Navier-Stokes equations" (Thomas and Nicholas 2001, 180). While they still only route discharge downstream, their model moves the discharge to five downstream cells as opposed to three in Murray's model. The authors also incorporate several techniques to correct errors in the cellular routing scheme. One technique is to modify how the local bed slope is used in determining direction of discharge through an additional modifier to the local bed slope. The authors believe that "use of the local bed slope is problematic since it may be either positive or negative, and is likely to provide only an estimate of the local energy slope" (Thomas and Nicholas 2001, 181). An additional correction is added to correct the situation where all of the discharge from the five upstream cells is routed to a single cell making that single cells water depth relatively large.

In an article published in 2001, Wei Luo describes LANDSAP, a cellular automata model for landscape evolution simulation. Luo explains that "the cellular automata approach is ideal for simulating long-term landscape evolution that involves multiple interacting processes" (Luo 2001, 363). LANDSAP uses the Gilbert model for surface runoff, originally developed by Chase in 1992. This model "applies simple rules iteratively to individual cells of a digital topographic grid after storm events (termed precipitons) randomly fall on the grid" (Luo 2001, 363). Luo added groundwater sapping rules to the model to add the effects of seepage to the erosion model.

Thomas Coulthard wrote his thesis on a steady-state cellular automata for water flow and sediment transport in 1999. The primary purpose of this CA was to model landscape evolution. To achieve this end, Coulthard had to model sediment flow and hydrological processes. Every CA must have rules to dictate cell state; Coulthard's rules included consideration for hydrologic routing, hydraulic routing, fluvial erosion, fluvial deposition, slope processes and mass movement (Coulthard 1999). He also required very high resolution DEMs to simulate channel erosion; this created a massively large matrix for the CA to process. To minimize processing time, Coulthard utilized a technique where only a portion of the cells within the matrix were processed at each iteration.

Hydrological Models

In 2003, Mark S. Johnson et al. compared two different hydrological models, Hydrological Simulation Program-FORTRAN (HSPF) and Soil Moisture Routing (SMR) models, for surface flow over time. Neither of the models discussed are CA based, but they provide interesting insight into runoff models. The authors describe the primary difference between the two models as how they define surface runoff, which can be modeled "as either (1) infiltration excess (or Hortonian) overland flow, or (2) saturation-excess overland flow" (Johnson et al. 2003, 57). Both models require a significant amount of calibration of parameters that "have a large effect on runoff volume" that require detailed knowledge of the study area. These parameters include: timing control, interflow, and surface and soil moisture storage and losses. In one model, "overland flow is treated as a turbulent flow process and is simulated by the Chezy-Manning equation and average values of roughness, length and slope for the overland flow" (Johnson et al. 2003, 61). The second model treats runoff as a "response to saturation excess-the moisture in excess of soil moisture in excess of the soil-moisture storage capacity after moisture is routed from each grid cell laterally by subsurface lateral flow and vertically by percolation and evapotranspiration" (Johnson et al. 2003, 61).

CHAPTER III

METHODOLOGY

The development of a time-based computational watershed model involves two primary components: the calculation of water movement and the visual presentation of the results. The water movement component was developed by creating a cellular automaton to calculate water velocities and depths and move the water through the study area. The computational model described below has several specific goals that dictate how it was developed. The first goal was the ability to model surface water flow at multiple study areas. This required minimizing the amount of site specific inputs that must be gathered. The second goal of this model was to achieve a real-time based model that could be compared to recorded precipitation events. This combination could enable many researchers and hydrologists to model their own watersheds with limited resources.

Required Inputs

The initial version of this modeling tool was developed based on a site within the Rocky Mountain National Park. However, it was developed such that surface water flow can be simulated for any study site with the required inputs. Only minimal data layers are required to model a site's surface water flow: a DEM, infiltration, and a Manning's n value for roughness. A digital elevation model (DEM) is the primary layer required to calculate slope and flow direction. An infiltration layer or a constant representing the average infiltration value for the entire site is required to determine the amount of surface water absorbed through the soil. A roughness layer or an average infiltration value for the entire site is needed for the calculation of water depth and velocity.

Digital elevation models (DEMs) are the primary data input for a study site. High resolution DEMs are necessary to accurately predict surface water flow over the given terrain. The minimal horizontal spatial resolution of the DEM must be 10 meters while the vertical elevation value (height) resolution must be 1 meter or less. DEMs for the study area can be cropped and corrected in any GIS or remote sensing software package. The DEM should then be exported into an ASCII file format. These tasks can be accomplished either with general GIS/remote sensing software or with the software tool, 3DEM from Visualization Software LLC, which is available for free on the internet.

Another required input is the hydraulic roughness coefficient for the land surface in the study area. One form of input is a standard (average) Manning's n value for the entire site. This should be accurate enough for research in most simple, uniform watersheds. However, more complicated research may require detailed roughness values for different sections of the study area. A "friction" data layer can be developed by classifying ground cover in the study area and associating a roughness value for each ground cover type. For example, areas with exposed rock will have a low friction value of 0.011 while areas with dense growth may have a friction value of 0.2. The ground cover layer should then be exported as an ASCII file.

The final required input is an infiltration value for the study site. This value describes the maximal amount of surface water lost to infiltration (soil absorption) per time unit. This value may be input as one average value for the entire study area or an "infiltration" layer may be created based on soil classification. Table 1 shows the varying infiltration rates for different soil types.

Table 1: Infiltration rates for various soil types. Rate is the depth of water infiltrated in one hour (FAO 2004).

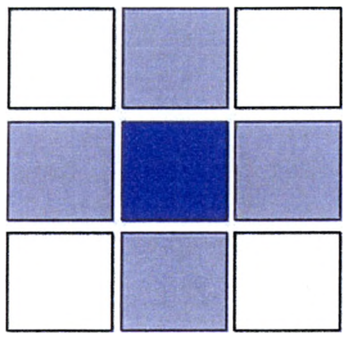
Soil type	Basic infiltration rate (mm/hour)
Sand	less than 30
sandy loam	20 – 30
loam	10 – 20
clay loam	5 – 10
clay	1 – 5

Cellular Automata Model

In 1984, Stephen Wolfram described cellular automata as "examples of mathematical systems constructed from many identical components, each simple, but together capable of complex behavior. From their analysis one may, on the one hand, develop specific models for particular systems, and, on the other hand, hope to abstract general principles applicable to a wide variety of complex systems." This statement describes how simple rules and cell-based actions of cellular automata can model complex surface water flow.

A cellular automata model consists of a two-dimensional matrix of cells, a neighborhood definition, and a set of rules. The CA described here contains a matrix of cells depicting a spatial "cell" or area of the terrain the size of the DEM cell resolution with properties pertaining to its elevation and water volume. The neighborhood selected for this model is the Von Nuemann neighborhood. The Von Neumann neighborhood consists of the cells directly above, below and to the sides, shown in gray, of the center cell in question as shown in Figure 4.

Figure 4: A Von Neumann neighborhood consists of the four cells (gray) directly above, below and to the sides of the center cell (blue).



While water could actually flow to the cells diagonal to the center cell, this simplification was necessary for the initial development of the surface water flow CA. Because of the dispersive nature of water, this simplification will likely not adversely affect the final results of the model. The model disperses water to all four neighbors to ensure proper water movement. The rules for water movement will depend on cell elevation, water depth, water velocity and the Manning's equation.

Most existing CA models for water flow are based on "steps" and "precipitons". In these models the "steps" are not actual time increments and the "precipitons" do not correlate to actual water volumes. Due to their abstract units, these models are difficult to compare to real-world watersheds and precipitation events. This model uses actual time in increments of 1 second, or fractions of a second, and calculates water volume based on realistic precipitation events. The selection of a 1 second (or less) increment is based on the availability of 10 meter spatial resolution digital elevations models and realistic time it takes to traverse a 10 meter cell.

The CA described in this paper was developed in the object oriented programming language Java. The primary object in the model is the cell object. The cell object depicts the state of a particular cell within the two-dimensional matrix. A description of the cell object and its fields are listed in Table 2.

Table 2: Fields in the Cell object that are used to determine surface water flow.

Field	Type	Description
nValue	float	Manning's n value for the cell
elevation	float	elevation from DEM
waterDepth	float	depth of the water (precipitation is added to this field)
surfaceElev	float	elevation + waterDepth
cellSlope	float	slope calculated using surfaceElev and neighborhood
vol	float	amount of water in cell
futureVol	float	storage field for water moving into this cell (acts as a buffer layer)
movabelVol	float	amount of water that can move
toCells	List	cells to receive water from this cell
releaseTime	integer	Number of iterations to hold the water before releasing it to neighboring cells
countTme	integer	Number of iterations since cell clock was reset. Compare to releasetime to determine if it is time to release water

Additional objects include: NeighborhoodOp, CellMatrix and WaterFlow. The NeighborhoodOp object contains static methods to perform neighborhood operations such as calculation of cell slope and determining flow direction. The CellMatrix object contains methods and fields that pertain to every cell in the CA matrix such as: printing various outputs, cell size, and spatial properties. Finally, the WaterFlow object is the object external applications will utilize to create and control the CA model.

Cellular Automaton Rules

The CA rules for water movement within the surface water flow model are based on notes written by Mark Fonstad in 2003 and are as follows: infiltration rule, water velocity, and water flow direction (Fonstad 2003). The infiltration rule is based in the infiltration rate that is either input for the entire watershed as a constant value or input as a layer so that each cell has its own infiltration rate.

$$vol = vol - \inf \qquad \text{(Equation 1)}$$

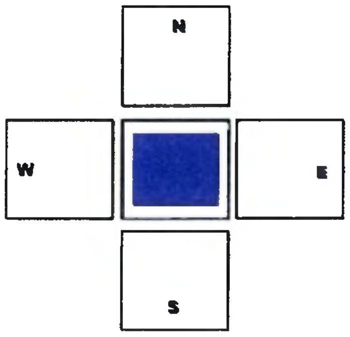
Equation 1 states that each cell loses a particular amount of surface water volume at each time increment due to infiltration.

Water depth is calculated using Equation 2.

$$depth = \frac{vol}{length * width} \qquad \text{(Equation 2)}$$

Cell slope is calculated using the Zevenbergen and Thorne's method in Equation 3:

Figure 5: The cells directly above (N), below (S) and to the sides (W, E) are used to calculate the slope of the center cell (blue).



The variables W, E, N and S represent the surface elevation values for the cells in Figure 5.

$$g = {(-W + E)}/{2 * width}$$

$$h = {(N - S)}/{2 * width}$$

$$slope(s) = \sqrt{g^2 + h^2} \qquad \text{(Equation 3)}$$

Water velocity is calculated using the Equation 4. Each variable is described in Table 3.

$$vel = \frac{depth^{2/3} * \sqrt{s}}{n} \qquad \text{(Equation 4)}$$

Table 3: Variables for slope and velocity equations

vel	velocity of water in the cell
width	width of cell
depth	water depth
s	cell slope
n	Manning's n value for surface friction

The alarm clock, or time needed to traverse the cell is determined with Equation 5. Each variable for Equation 5 is described in Table 4.

$$t = {}^{width}\!/_{vel} \qquad \text{(Equation 5)}$$

Table 4: Equation 5 variables

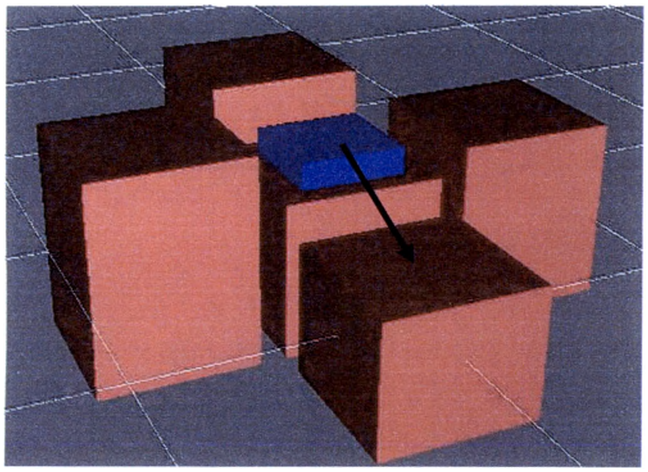
t	time in seconds
width	width of cell in meters
vel	velocity in meters per second

Water Flow Direction

The direction to move water volume once it has traversed the cell is determined by surface elevation values. Any neighboring cell with a surface elevation lower than that of the center cell will receive a portion of the water. Neighborhood cells with surface elevation values less than center cell's surface elevation receive a percentage of the volume based on their relative height difference. The "receiving" cells are stored in the toCells field of the cell object.

The example in Figure 6 shows that the entire water volume in the center cell will move to the lower cell once its release clock has triggered movement.

Figure 6: The entire water volume moves to the cell directly below the center cell.



Boundary Condition

The CA model utilizes an "absorbing" boundary to control flow. The outermost cells of the matrix will store any water moved into them. These boundary cells will store the water (excluding the amount infiltrated) leaving the study area matrix in the futureVol field. The water in these "border cells" is not allowed to affect the flow of water not in the border. Tracking the amount of water moved into the boundary cells will identify the location and quantity of water as it leaves the study area.

Development Tools

The CA for modeling surface water flow was developed in Java (see Appendix). The primary reason for selecting Java as the development language is the ability to utilize the same code base within a variety of stand alone applications on various operating systems. Developing the tool in Java also enables a user interface that can be launched from a web browser.

Model Output

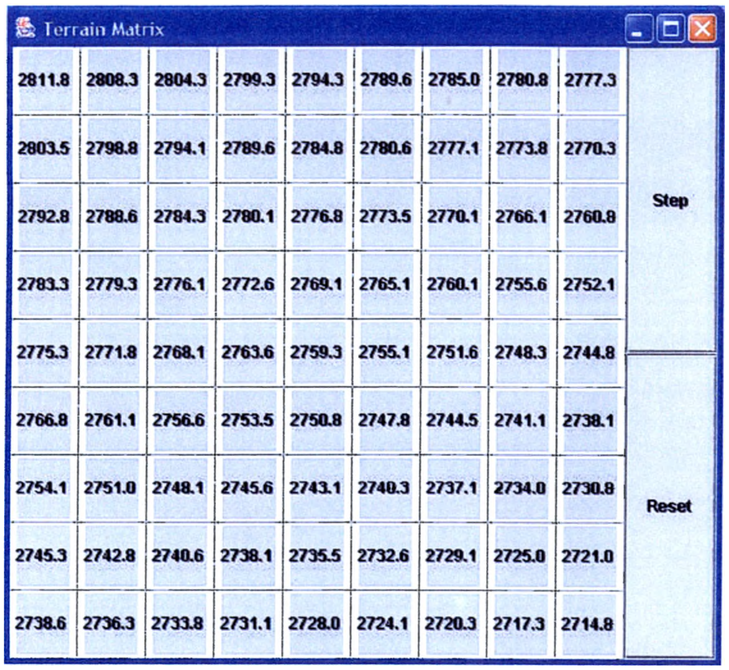
The computational model outputs the water levels at specified time intervals for further analysis. The model can save the water volume matrix as an ASCII file that could then be read into third-party visualization and analysis tools. Users of the model also have the ability to select a single cell to monitor flows, acting as a flow gauge in a stream. Discharge values for the gauge cell are then output to generate a simulated hydrograph.

Visualization

While the CA surface water flow component is separate from visualization, visualization is necessary to communicate the resulting surface water flow effectively. Several forms of visualization were developed based the size of the elevation matrix. These alternate forms of visualization serve a variety of purposes such as: explaining water movement with small matrices, 3D visualization of an entire watershed or drainage area, and hydrographs of flow generated from a "gauge cell".

Small sites, less than 11 by 11 cells square, can be displayed in a table form. This level of visualization is essential to describe and educate how water flow is determined. Figure 7 shows an example of a small site and the initial elevation values for each cell. The "Step" button will increment the model clock by one second.

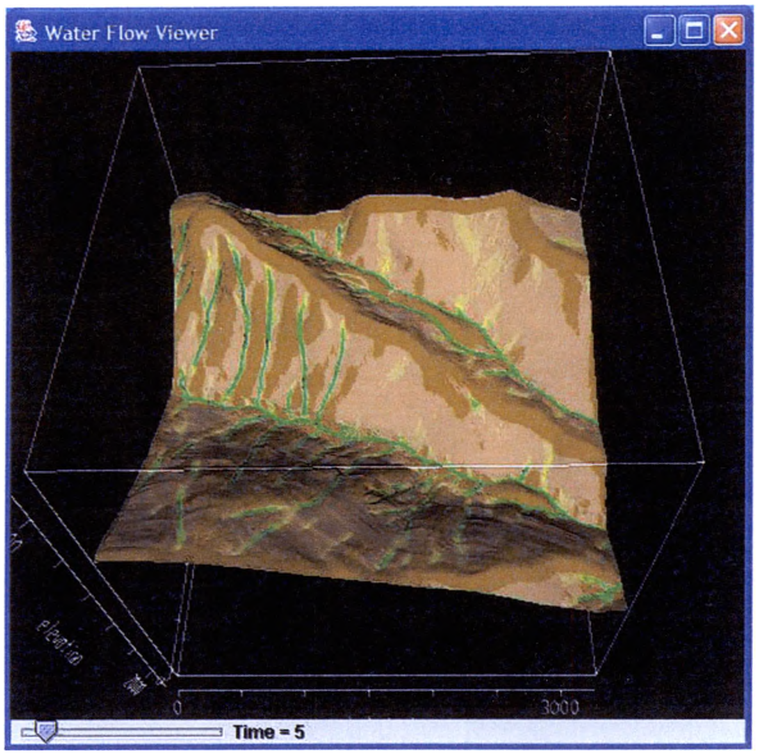
Figure 7: Small matrix visualization shows the actual elevation values for each cell. Pressing the "Step" button cycles the model through one iteration and updates the cell values (visible when the cell is clicked).



2811.8	2808.3	2804.3	2799.3	2794.3	2789.6	2785.0	2780.8	2777.3
2803.5	2798.8	2794.1	2789.6	2784.8	2780.6	2777.1	2773.8	2770.3
2792.8	2788.6	2784.3	2780.1	2776.8	2773.5	2770.1	2766.1	2760.8
2783.3	2779.3	2776.1	2772.6	2769.1	2765.1	2760.1	2755.6	2752.1
2775.3	2771.8	2768.1	2763.6	2759.3	2755.1	2751.6	2748.3	2744.8
2766.8	2761.1	2756.6	2753.5	2750.8	2747.8	2744.5	2741.1	2738.1
2754.1	2751.0	2748.1	2745.6	2743.1	2740.3	2737.1	2734.0	2730.8
2745.3	2742.8	2740.6	2738.1	2735.5	2732.6	2729.1	2725.0	2721.0
2738.6	2736.3	2733.8	2731.1	2728.0	2724.1	2720.3	2717.3	2714.8

Medium sized sites, up to 350 by 350 cells, can be displayed in a Java 3D application. This visualization, shown in Figure 8, is intended as the primary form of visualization; however, the computation performance limits it to the smaller sites. In this visualization, the different water depths are shown with differing colors, dark brown for minimal water depths and green for the greatest depths. The "3D" visualization was written in Java and utilizes VisAD; VisAD (Visualization and Algorithm Development) is an open-source Java component library for data visualization and analysis (VisAD 2004).

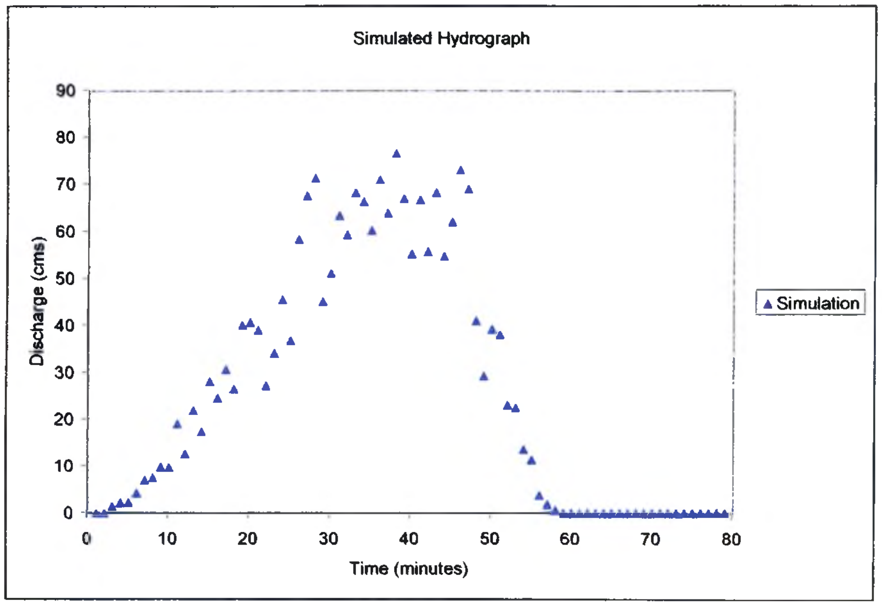
Figure 8: Java 3D visualization of surface water flow over a study area. Cells with no water are in brown and cells with the deepest water are bright green.



The water volume values for large sites, over 350 by 350 cells, must be read into third party applications. This is accomplished by saving each time-based water volume matrix as an image. The water volume matrix image can then be overlaid onto the study area's DEM. The resulting images could be combined into animated gifs to show water flow over time.

Another useful output from the model is a hydrograph. Users are able to designate a single cell as a "gauge cell"; water depths, water velocities and discharge rates are then output for that cell. Those values can then be pulled into a charting program to build a chart that shows the simulated hydrograph as seen in Figure 9.

Figure 9: Sample hydrograph generated by surface water flow model.



CA Flow

The logical flow of the surface water flow component is described in the following steps:

```
I.   Initialize Matrix from input values
II.  Loop through matrix with the following steps per cell
     1. Add precipitation
     2. Subtract infiltration
     3. Calculate water depth
     4. Calculate water velocity
     5. Calculate time required to traverse cell and set alarm
     6. Check alarm to see if time met
          a) Yes
               i.   determine flow direction
               ii.  determine movable volume
               iii. move water to receiving cells futureVol field
          b) no
               iv.  increment cell counter
     7. Add to vol from futureVol and clear futureVol
     8. Output CA state.
III. Check exit condition (time to model watershed response)
IV.  Output state of CA (for visualization or file storage)
```

CHAPTER IV

MODEL VALIDATION

Validating a surface water flow model is a difficult task due to the lack of detailed watershed information and the lack of storm-based, time-explicit reading from precipitation and flow gauges. Validation must be accomplished before any results from the model can be trusted to predict discharges from hypothetical precipitation events. The primary challenge with validating a surface water flow model is obtaining detailed observations for both precipitation and flow. The model described here works most efficiently for short, intense storm events which require high-resolution, time-explicit precipitation and discharge measurements. Understanding the time specific conditions (such as current amount of moisture in soil) at the time of the event may also be necessary to understand the observed watershed response. While 10 meter spatial resolution is available for many areas, it may not be accurate enough to show the elevation changes and define stream channels within a watershed. After a lengthy search, the required data was found at the Walnut

Gulch Experimental Watershed (WGEW) in Arizona. This section describes the watershed, the storm event used in validation and how the model results compare to the observed flows.

Study Site: Walnut Gulch Experimental Watershed (WGEW)

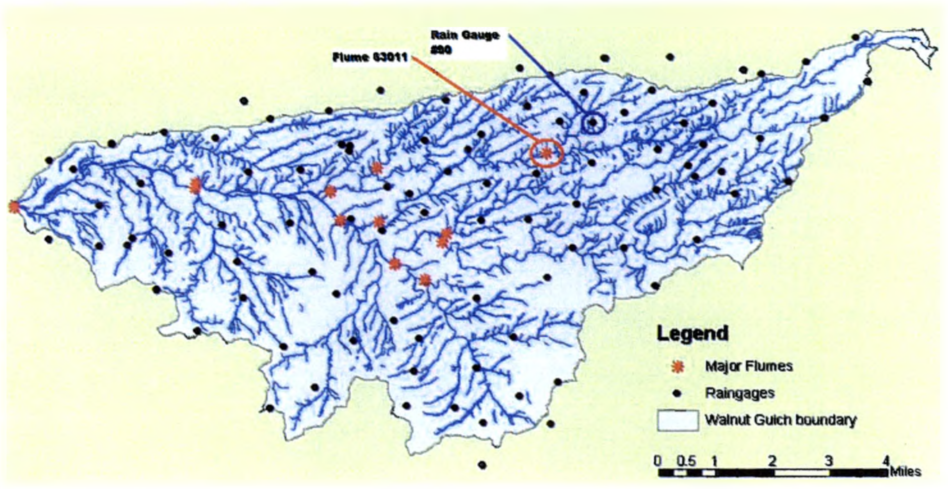
The Walnut Gulch Experimental Watershed in Arizona (encompassing Tombstone, AZ) is operated by the Southwest Watershed Research Center and is considered a large outdoor research laboratory. Research began at the experimental watershed in 1953 and that research has contributed much to the understanding of semi-arid watersheds. Facts about the watershed include:

- is the most densely gauged semi-arid watershed in the world
- is approximately 150 square kilometers
- contains nearly 100 recording precipitation gauges
- contains 25 flumes and weirs to measure flow

Sub Watershed

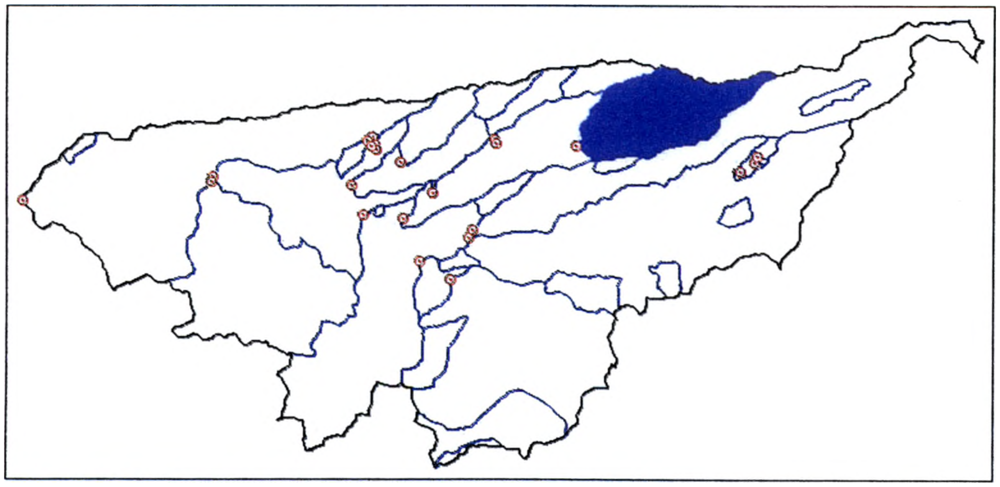
A sub-watershed within the WGEW was used to validate the model. The watershed above flume 63011 is approximately 8.24 square kilometers and contains precipitation gauge #90 near its center (Figure 11). The sub-watershed conditions are described as 20% covered by desert shrubs and 80% desert grasses(SWRC 2004).

Figure 11: Flume #63011 and gauge #90 locations within the Walnut Gulch Experimental Watershed (SWRC 2004).



The watershed that drains into flume 63011 is in blue in Figure 12.

Figure 12: Sub-watershed (blue) within the WGEW. Red circles are flume locations (SWRC 2004).



The elevations for the sub-watershed were cropped from a DEM covering the entire WGEW. Figure 13 shows a visualization of the sub-watershed. The elevations range from 1418 meters (shown in blue) to 1604 meters (shown in red).

Figure 13: 3D visualization of WGEW sub-watershed digital elevation model. Lowest elevations are depicted in blue and the highest elevations are in red.

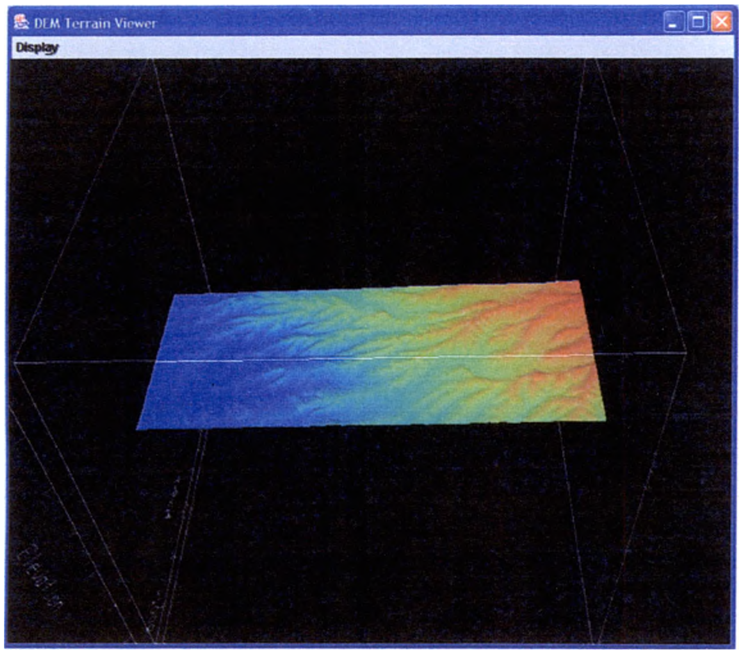
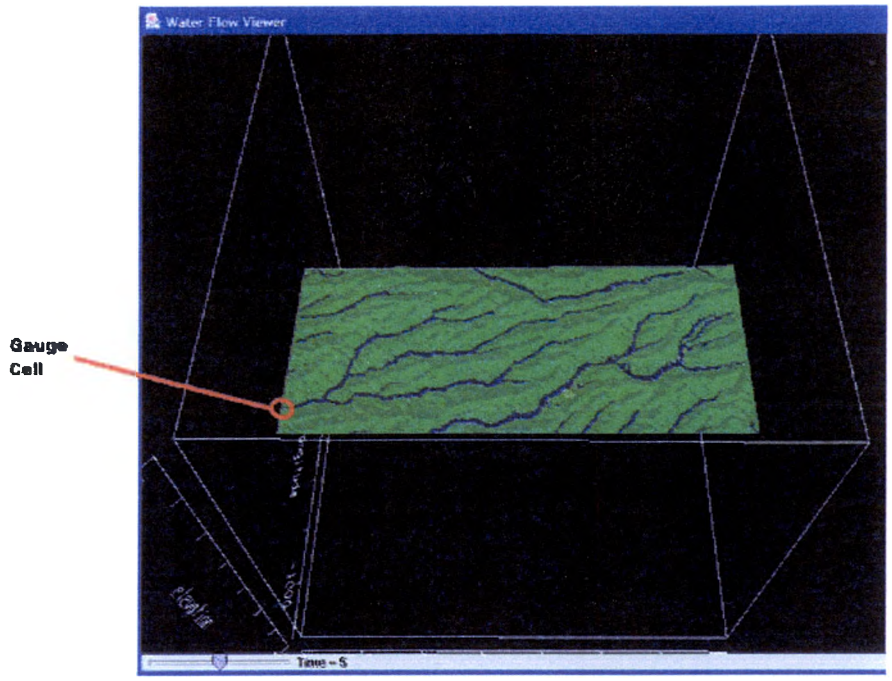


Figure 14 is a visualization of water flow through the study area. A "digital gauge" was placed at cell 45, 1 to produce the hydrographs seen in the simulations of the watersheds response to a storm event.

Figure 14: Water flow over WGEW sub-watershed. The water (blue) is in channels leading out of the watershed.



Storm Event

Researchers at the WGEW provided rain gauge data and discharge data for a specific storm event that occurred on August 4, 1980. The summer rain events that occur are typical of semi-arid areas; the storms are usually very intense and have a short duration. A storm of this type occurred on August 4, 1980, with 1.68 inches of precipitation over two hours and 44 minutes after the rain event started. The majority of the precipitation fell in the first 47 minutes of the event. Table 5 is the recording of the cumulative precipitation amount over time from gauge 90 near the center of the sub-watershed for the rain event.

Table 5: Recorded Rain Event on August 4, 1980 from precipitation gage #90. All times are relative to 12:35.

Time	Depth (inches)
0	0.00
25	0.00
28	0.05
31	0.13
32	0.20
33	0.34
35	0.45
36	0.59
39	0.72
41	0.86
44	0.99
45	1.05
49	1.20
51	1.24
60	1.48
64	1.52
67	1.57
72	1.61
81	1.65
93	1.66
169	1.67
189	1.68
370	1.68

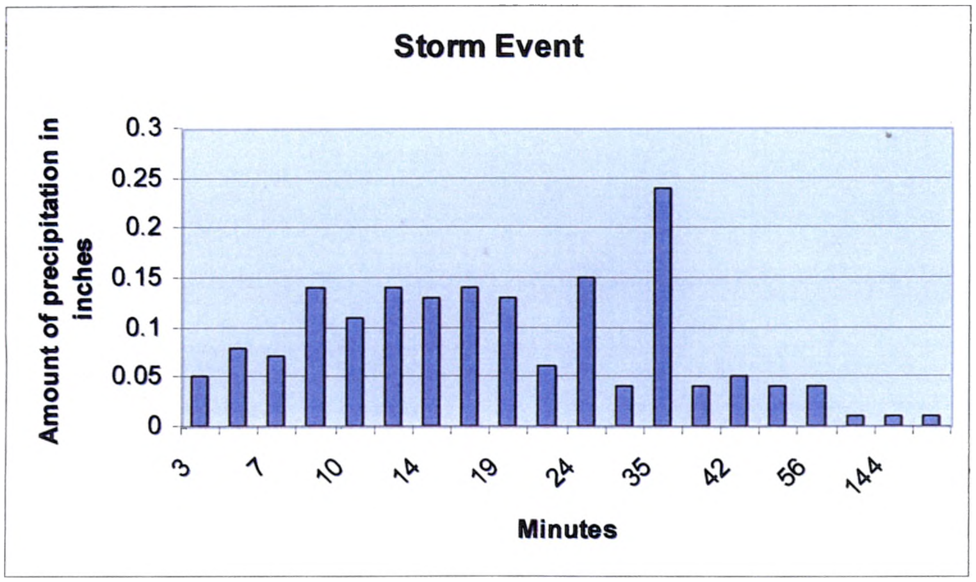
Modeling the Storm Event

Modeling a specific storm event over a watershed is a difficult challenge. Precipitation amounts vary spatially and temporally. The precipitation gauge for this watershed is near the center of the sub-watershed. This may not account for rain that does not fall equally over the study area and cases where the direction of the storm's movement may impact runoff rates.

Researchers from the WGEW provided a detailed recording of a storm event that occurred on August 4, 1980. The discharge data was obtained from flume #63011 and the precipitation data came from rain gauge #90.

A specific control program was created to model the exact precipitation event recorded on gauge 90. A series of if..then statements moderated the precipitation rate over time to control the storm's intensity. For example, 0.05 inches of rain fell between the times 25 and 28 minutes. Then another 0.08 fell between 28 and 31 minutes, this process continued until the storm was split into time periods and intensities. Figure 15 shows the duration and intensity of the event.

Figure 15: Intensity of the storm event that occurred on August 4, 1980.



Model Inputs

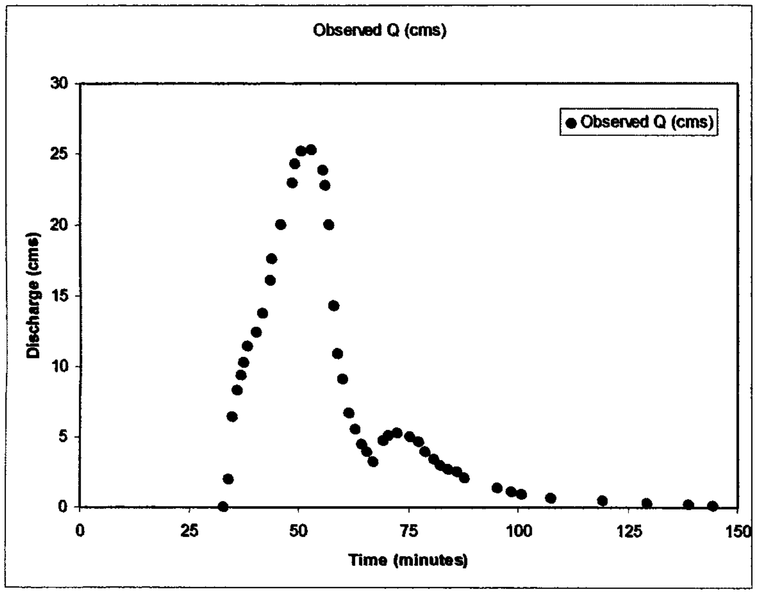
The WGEW provided a DEM of the entire watershed and locations of the flumes and gauges. The author cropped the DEM to the sub-watershed that drained into flume 63011. Since infiltration rates and roughness values were not explicitly known for the area, several simulations were run using various values for the unknown inputs. Infiltration rates used were based on known soil types for the watershed and the range of infiltration rates for the know types of soils.

Results

Multiple simulations were performed to determine the impact of varying the unknown parameters of roughness and infiltration sensitivity analysis. The

Manning's n value for roughness varied between 0.01 and 0.02 while the infiltration rate varied between 0 mm/hour and 10.8 mm/hour. The results from each simulation are then compared to the observed flow recorded from the flume. The observed flow is shown in Figure 16. A peak of 25.29 cubic meters per second (cms) occurred at 53 minutes after the start of the storm event.

Figure 16: Observed flow at flume #63011. The flow is first recorded at 34 minutes after the storm starts and the peak occurs at 53 minutes.

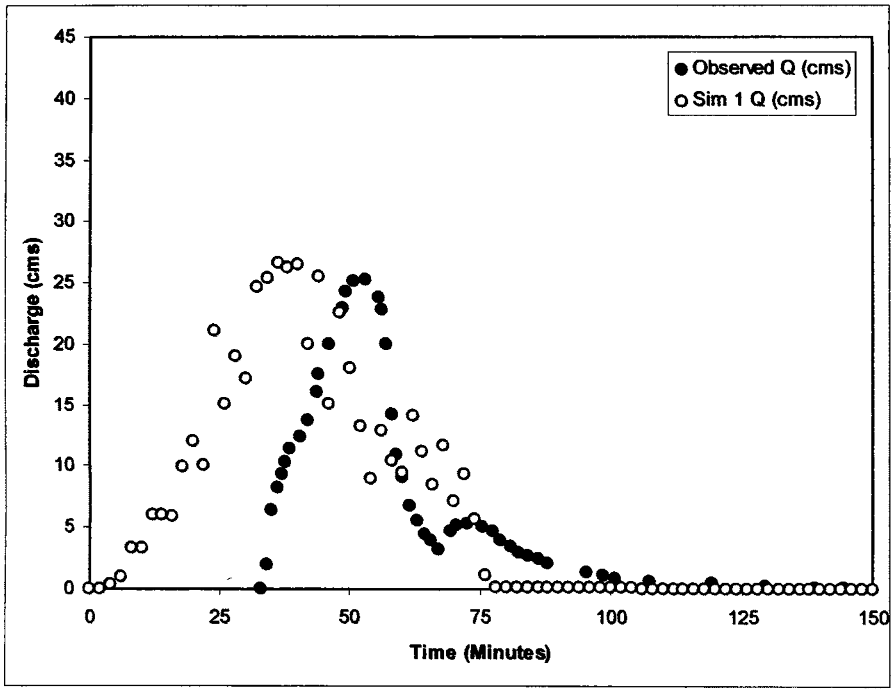


Simulation 1

The results of the first simulation (Sim 1), shown in Figure 17, demonstrate how the model reacts to a constant infiltration rate of 7.2 mm/hour and 0.01 for a Manning's n roughness value. In Sim 1, the peak discharge of

26.59 cubic meters per second (cms) occurred at a time of 36 minutes after the storm event began. The simulated hydrograph is similar to the observed hydrograph; the peak values are close but the peak in the simulation occurs 17 minutes earlier than in the observed.

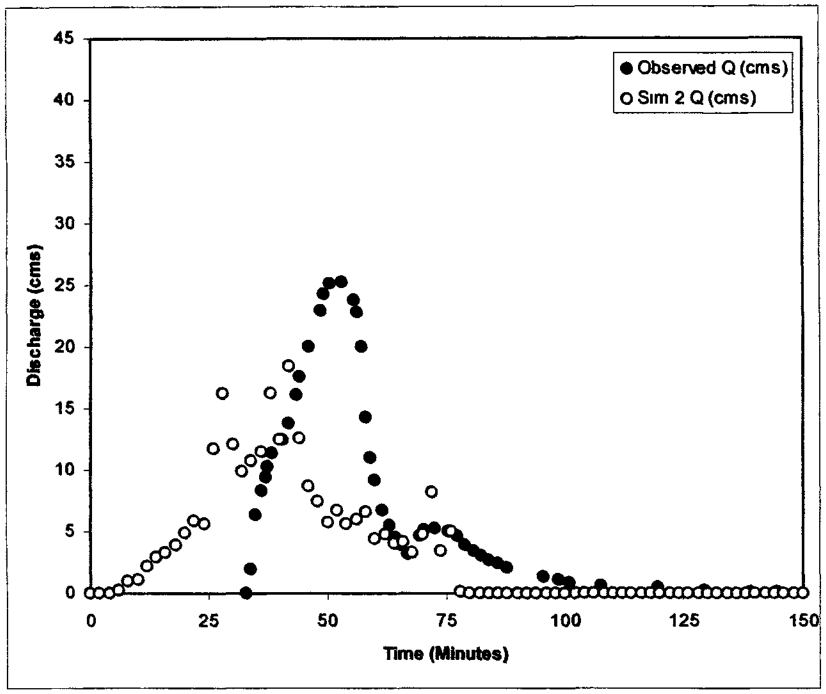
Figure 17: Comparison of Sim 1 hydrograph to observed hydrograph.



Simulation 2

Simulation 2 (Sim 2), shown in Figure 18, shows the result of a constant infiltration rate of 7.2 mm/hour and a Manning's n value of 0.02. In this simulation, a peak discharge of 18.46 cms occurred at 42 minutes after the event started. The hydrograph from this simulation is much "flatter" than the observed. The peak discharge of the simulation is 6.83 cms less than the observed.

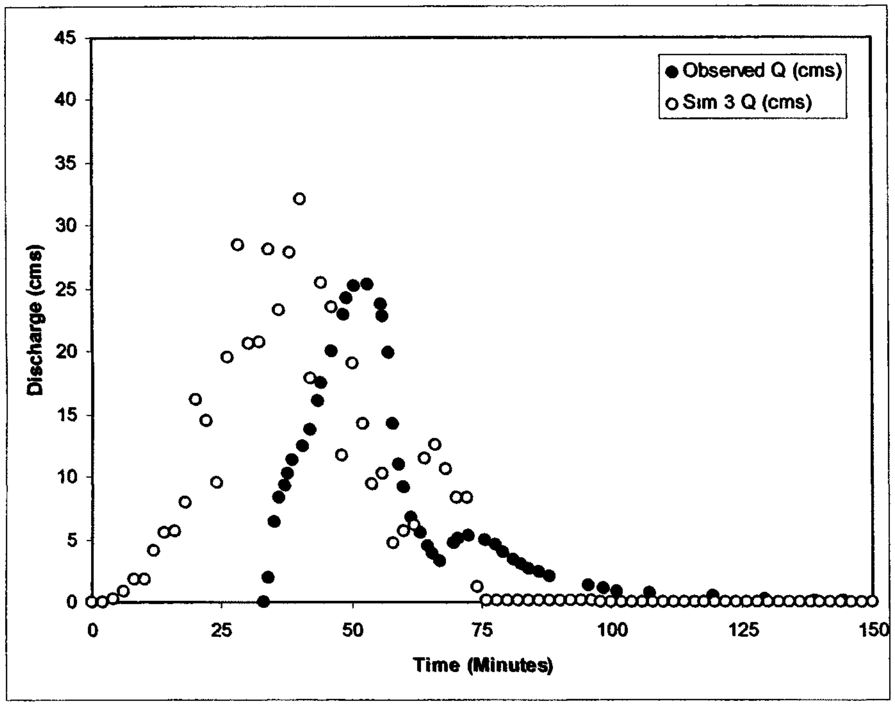
Figure 18: Comparison of Sim 2 hydrograph to observed hydrograph.



Simulation 3

Simulation 3, seen in Figure 19, shows the results of an infiltration rate of 10.8 mm/hour and a Manning's n value of 0.01, results depicted in Figure 19. In this simulation, a peak discharge of 32.13 cms occurred at 40 minutes. The hydrograph from this simulation maintains the shape of the observed hydrograph but peaks 13 minutes early for an additional 6.84 cms.

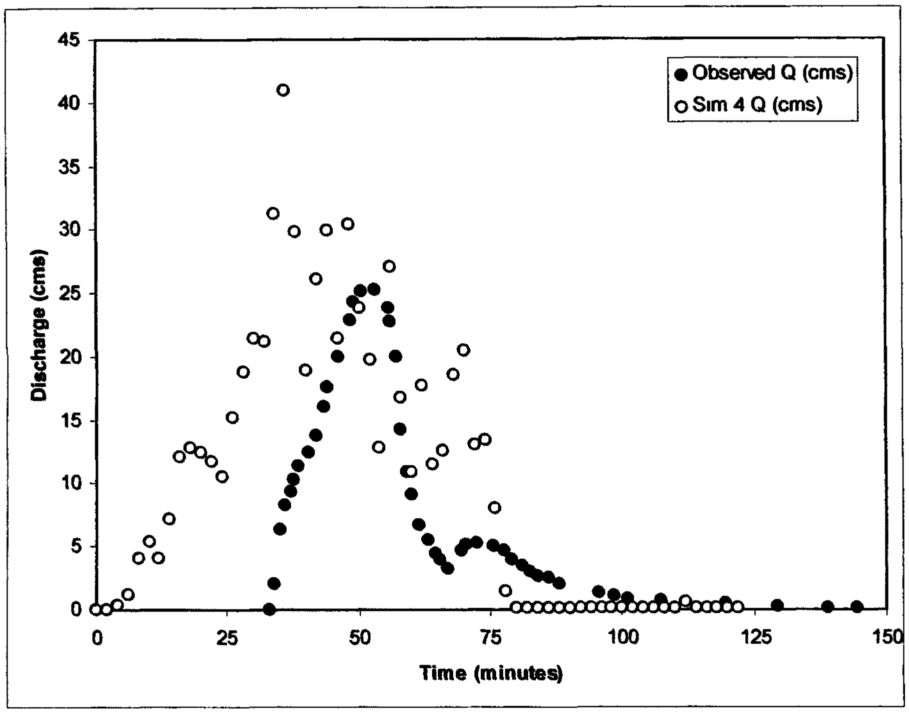
Figure 19: Sim 3 hydrograph and observed hydrograph.



Simulation 4

Simulation 4, shown in Figure 20, shows the result of no infiltration and a Manning's n value of 0.01. This simulation showed the highest peak value of 40.95 cms at 36 minutes. The general shape of the simulated hydrograph is consistent with the observed hydrograph but the peak is 17 minutes early with a high discharge rate 15.66 cms greater than the observed.

Figure 20: Sim 4 hydrograph relative to observed hydrograph.



Validation Discussion

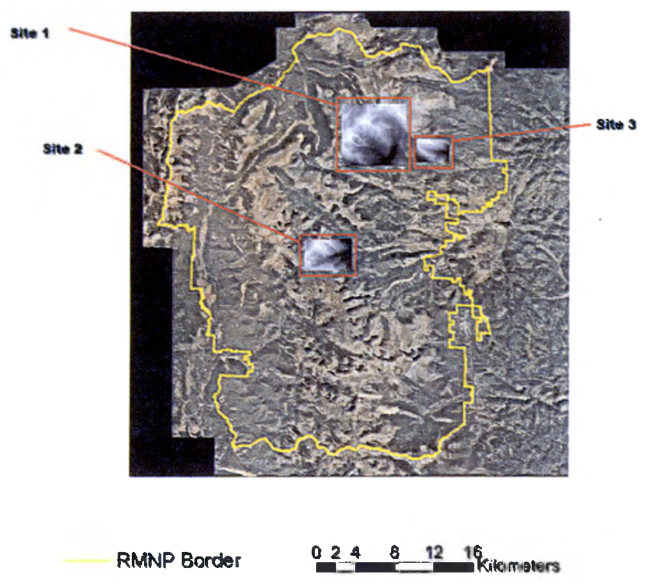
The results of the surface water flow model show that the model closely simulates the observed data. While not exactly precise, the model is highly accurate considering the limitations of the input data. 10 meter DEMs limit the accuracy of any water flow model and they cannot depict the spatial characteristics of a semi-arid watershed, such as precise channel hydraulics. The simulated hydrographs show peak earlier than the observed, this may be due to the spatial attributes of the storm. The lag between the start of the storm event and discharge flow at the flume might be due to the storm starting in the upper portions of the watershed. Simulation 1 appears to have the hydrograph most similar to the observed hydrograph with an infiltration rate of 7.2 mm/hour and a Manning's n roughness value of 0.01.

CHAPTER V

PREDICTING RUNOFF IN THE ROCKY MOUNTAIN NATIONAL PARK

The original purpose of the cellular automata based surface water flow model was to predict runoff in the Rocky Mountain National Park. Unlike the Walnut Gulch Experimental Watershed, storm event data is not available to validate the model for this study area. Until the model can be validated against historical precipitation and discharge data, the output described here can only be used for general forecasting purposes. The Rocky Mountain National Park is an interesting study area because of its diverse terrain. The park has elevations ranging from 8,000 feet to over 14,000 feet (RMNP 2004). The park also contains a variety of landscape ranging from flat grassland areas to steeply slope rock fields at the mountain peaks (RMNP 2004). Three drainage areas in the park were identified as study sites and used in separate simulations. The three sites are shown in Figure 21.

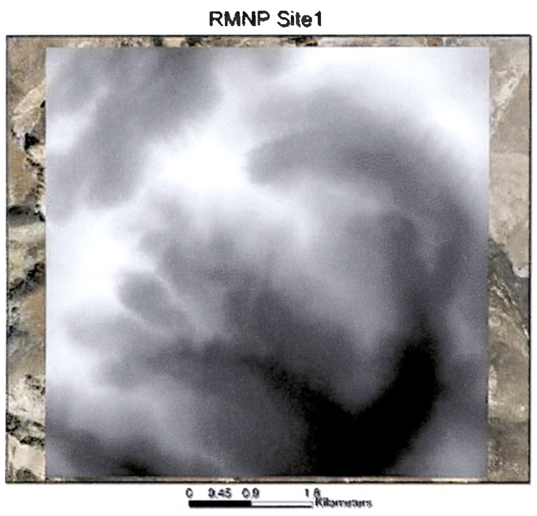
Figure 21: Three study sites located in the Rocky Mountain National Park.

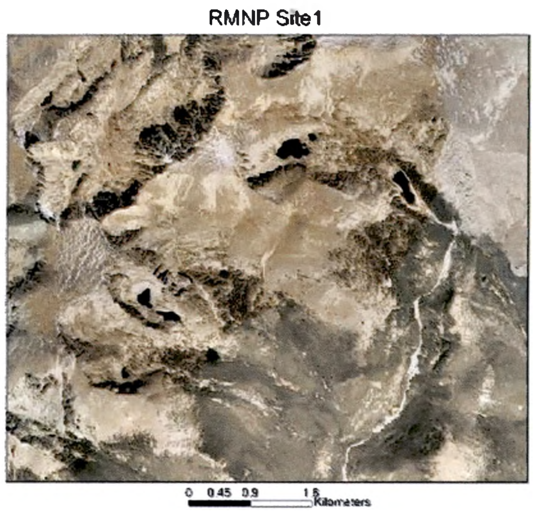


Site 1 is the largest site with a grid of 632 by 651 cells and an elevation range of 2869 meters to 4135 meters. Site 2 consists of a grid of 308 by 500 cells. The elevation ranges from 2698 meters to 3938 meters. Site 3 consists of a grid of 238 by 344 cells with an elevation ranging from 2846 meters to 3562 meters. The ground cover at the sites consists of barren rocks, tundra grasses, limber pine, lodgepole pine, Engleman spruce and field grasses.

Forecasting at Site 1

Figure 22: Site 1 with 10 meter DEM and aerial image.





Three simulations were performed at site 1 to demonstrate the effects of altering the infiltration rates on flow rates. The precipitation rate was held to 1.417 inches in 30 minutes and the Manning's n value for roughness was held constant at .03 for the three simulations. The infiltration rate increased with each simulation, as seen in Table 6.

Table 6: Infiltration rates for simulations at site 1.

Simulation	**Infiltration rate**
Sim 1	3.6 mm/hour
Sim 2	7.2 mm/hour
Sim 3	10.8 mm/hour

Figure 23 shows the resulting hydrograph from the three simulations at site 1. The increasing infiltration rate had a slight effect on the peak discharges. As the infiltration rate increased, the peak discharge value decreased. Figure 24 shows a 3D perspective of the study area and gauge location.

Figure 23: Hydrograph for the three simulations performed at Site 1.

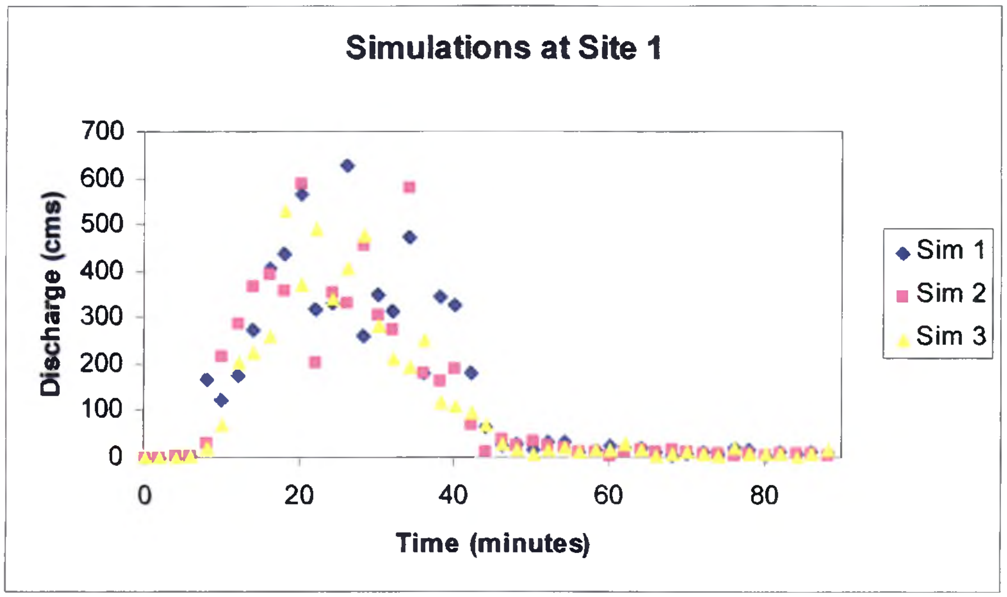
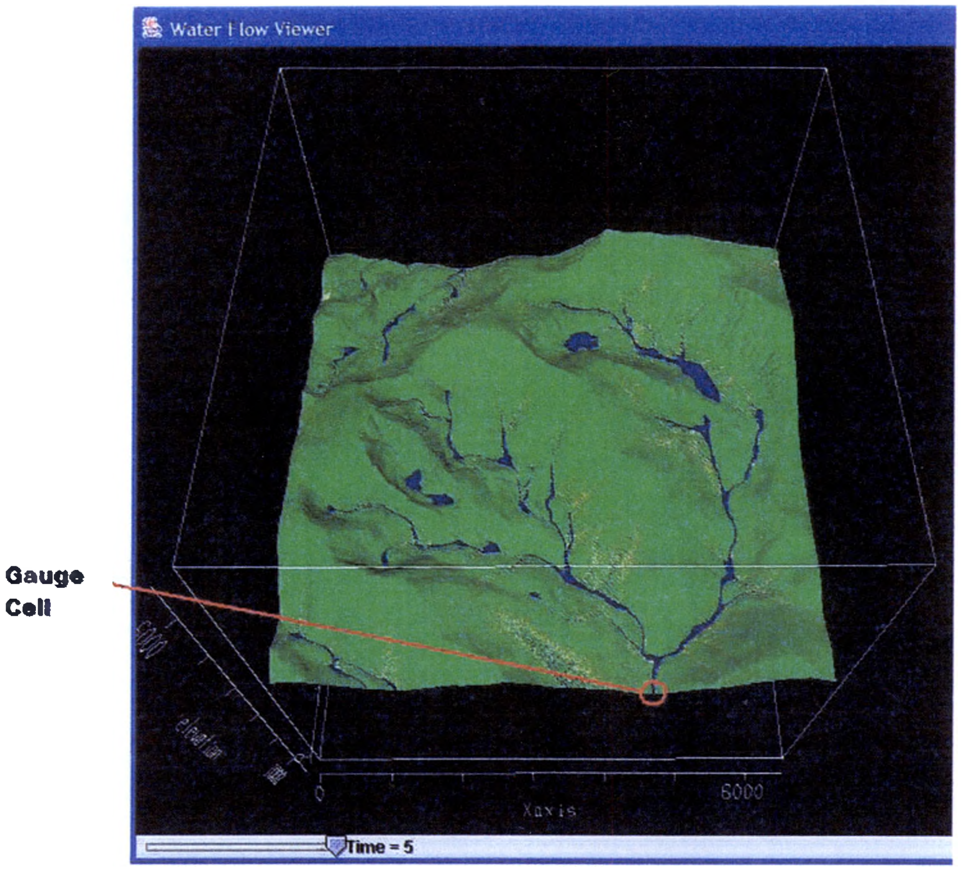
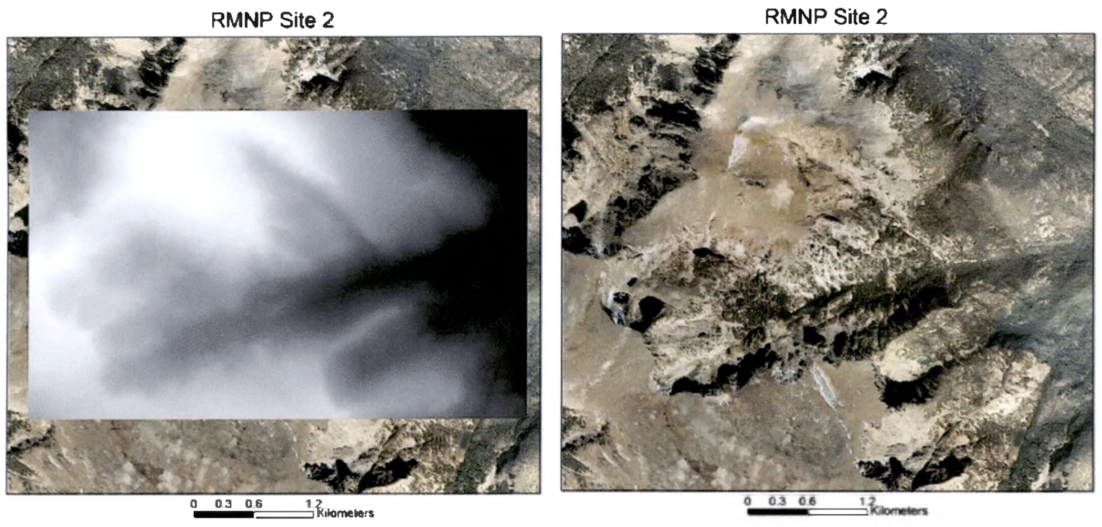


Figure 24: 3D visualization of surface water flow over site 1. Cells with the most water are blue, cells with no water are green and intermediate water depths are yellow.



Forecasting at Site 2

Figure 25: Site 2 with 10 meter DEM and aerial image.



Three simulations were performed at site 2 to get a range of possible flows over time for a single rain event. The simplified rain event simulated a constant intensity storm event that dropped 0.036 meters (1.417 inches) in 30 minutes. Infiltration was held at a constant 7.2 mm/hour, but the Manning's n value for roughness varied with each simulation.

Table 7: Manning's n value for simulations at site 2

Simulation	**n value**
Sim 1	.01
Sim 2	.02
Sim 3	.03

Figure 26 shows the resulting hydrographs for each simulation.

Figure 26: Hydrograph of three simulations at site 2.

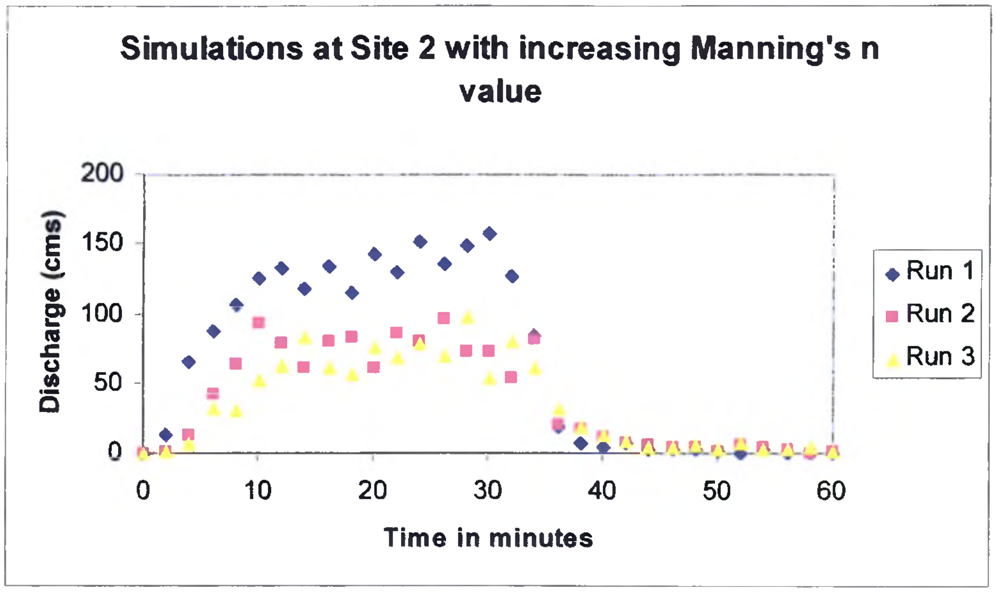
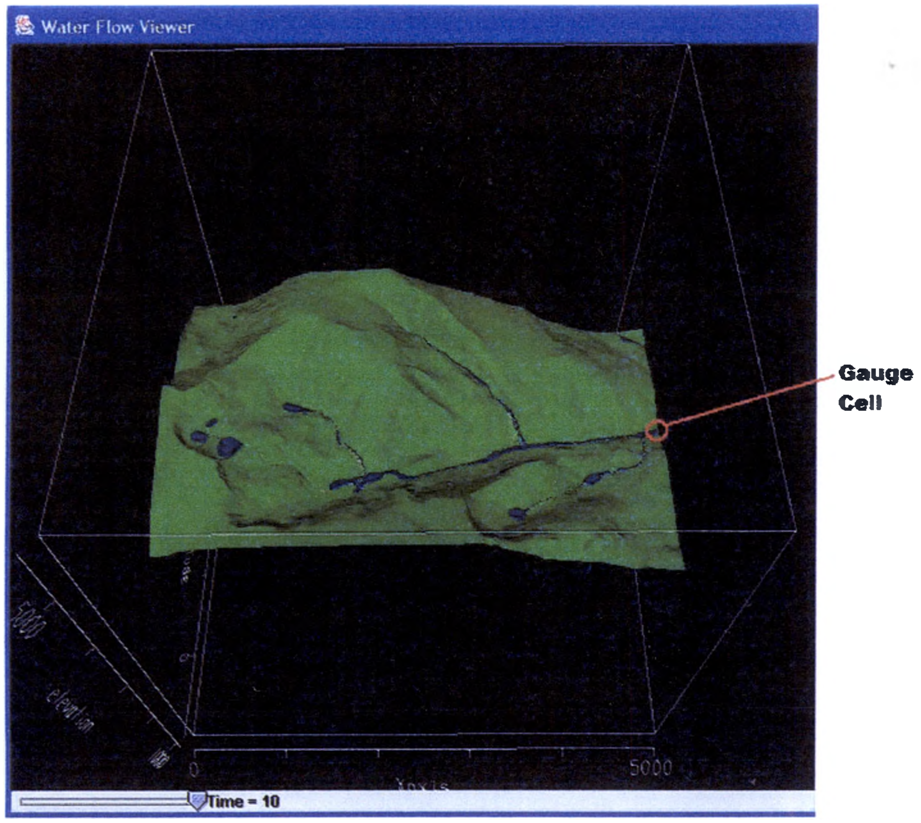


Figure 27 shows a "3D" visualization of surface water flow over site 2.

Figure 27: Visualization of surface water flow at site 2. Cells with the most water are blue while cells with no water are green.

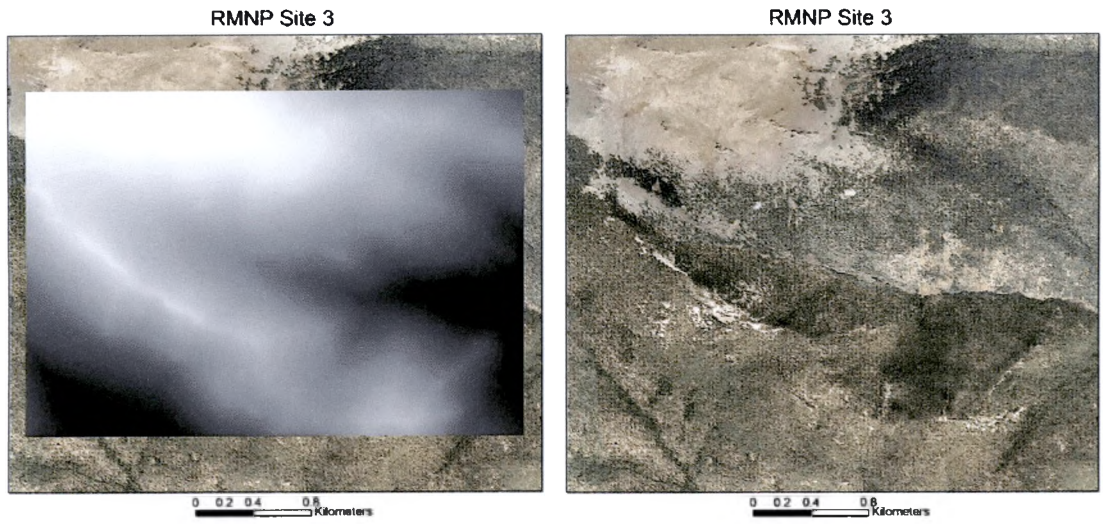


Discussion

The simulations at site 2 show the affect of altering the roughness value on the discharge values for the site. The increasing roughness value reduces the peak discharge value at the gauge cell. The peak discharges occurred at similar times, but for smaller values with the larger roughness values.

Forecasting at Site 3

Figure 28: Site 3 in the RMNP with 10 meter DEM and aerial image.



Three simulations were performed at site 3. Each simulation had a constant infiltration rate of 7.2 mm/hour and a Manning's n value of 0.03, but precipitation amounts for a 30 minutes storm were steadily increased. The precipitation amount for the 30 minute storms are shown in Table 8.

Table 8: Precipitation amounts of 30 minute storms for simulations at site 3.

Simulation	**Precipitation amount**
Sim 1	1.417 inches
Sim 2	1.772 inches
Sim 3	2.126 inches

Figure 29: Hydrographs for the three simulations at site 3.

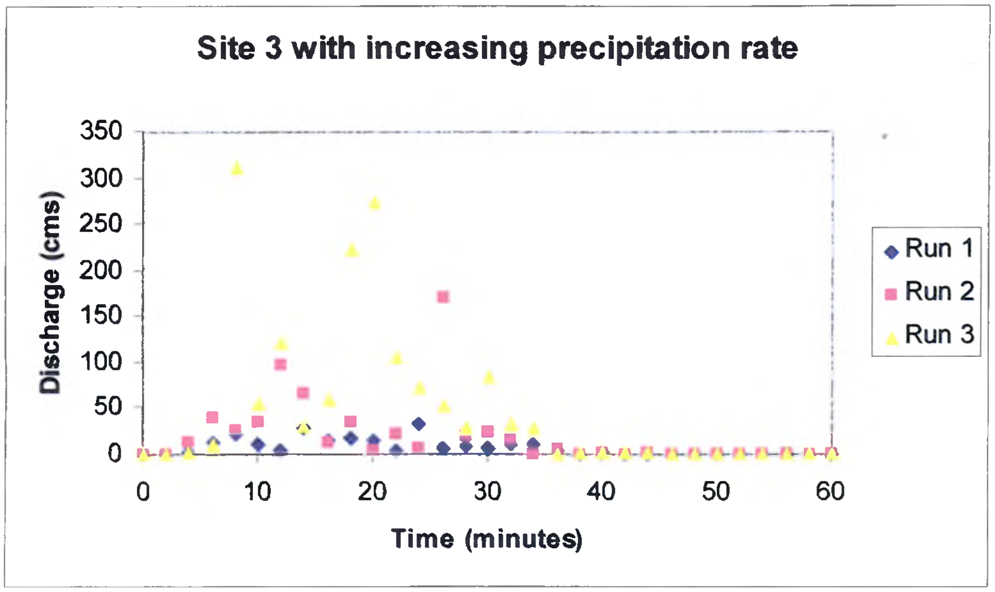
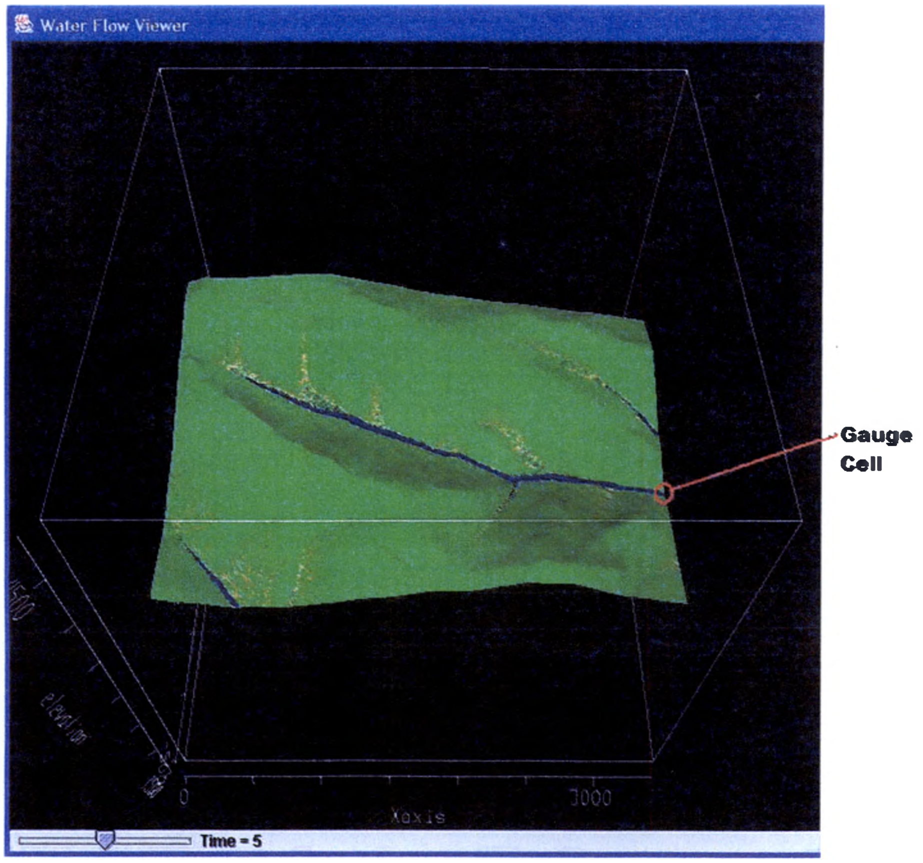


Figure 30: Visualization of surface water flow at site 3 in RMNP.



Discussion

The increased precipitation rate caused a dramatic increase in the peak discharge values. Table 9 shows the precipitation amounts and corresponding peak discharges. Simulation 2 shows that the increasing precipitation amount greatly increases peak flow but the timing of the flow is similar to the flow of Simulation 2. However, Simulation 3 shows that additional increases in precipitation amount eventually change the timing of peak flows. The peak flow for Simulation 3 occurs at 8 minutes after the storm begins as opposed to 26 minutes of Simulation 2.

Table 9: Peak flows for the simulations at site 3 in RMNP.

Simulation	**Precipitation**	**Peak Discharge (cms)**	**Peak Time**
Sim 1	1.417 inches	33.95	24
Sim 2	1.772 inches	170.22	26
Sim 3	2.126 inches	312.87	8

CHAPTER VI

DISCUSSION

The surface water flow model described in this paper was based on cellular automata theory and developed in Java. The validation of the model against the observed precipitation event and discharge values from the Walnut Gulch Experimental Watershed in Arizona proves that cellular automata is a viable theory for modeling surface water flow. Modeling in Rocky Mountain National Park shows that the CA approach is useful for hydrologic forecasting.

Model Improvements

The model described here was developed as a simplified modeling tool and several improvements would greatly improve the performance and accuracy of the model. Masking off non-watershed areas of the DEM would greatly improve processing time. Currently, every cell in the DEM is processed for water flow but only the cells within the watershed are necessary. Multithreading the program would improve the model's processing time on high performance machines. Allowing the user to select more than one cell as a "gauge cell" would give user the ability to observe the flow at various points within the study area.

Incorporating an additional set of cellular automata rules to model subsurface water movement would increase the current model's ability to simulate longer storm flows with a high baseflow component.

Additional Validation

Continued validation against various watersheds should be performed to ensure model accuracy. This paper describes validation against one storm event over one watershed. While this validation was very helpful, comparing against multiple types of watershed shapes, ground cover, sizes and soil types is necessary to completely validate the model.

Improved Watershed Representation

The current representation of watersheds leaves much to be desired for modeling purposes. Ten meter resolution DEMs are the current standard elevation source for watershed modeling. Higher resolution DEMs would improve model accuracy by detailing small elevation changes within the watersheds that are currently lost in the 10 meter DEMs. It is believed that channel definition would also improve model accuracy. Water would collect in channels and move rapidly through the system. Remote sensing may also provide spatial parameters such as the spatial distribution of infiltration and hydraulic roughness.

Conclusion

Cellular automata theory has proven to be a valid theory for surface water flow modeling. The time-explicit model described here uses inputs such as DEMs, infiltration rates, roughness values to model a watershed's response to a given precipitation event. Once validated for a watershed, the model can then be used to predict the flow and water height for possible precipitation events.

APPENDIX

JAVA CLASSES

```
package jap.ca.time;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.math.BigInteger;
import java.text.NumberFormat;
import java.util.Calendar;

/**
 * @author jparsons
 *
 * This control progam is specific to the walnut gulch watershed
 * It is an atempt to mimic the precipittion event exactly.
 *
 * ! Event of 8/4/80
   ! All gage elapsed times relative to 12:35
 *
 * BEGIN GAGE 90
  SAT = 0.124
  X = 558018.9, Y = 272479.4
  N = 23
  TIME      DEPTH !(in)          min from start|depth diff in m| m per sec this
    0      0.00
   25       0.00          0
   28       0.05          3
   31       0.13          6
   32       0.20          7
   33       0.34          8
   35       0.45          10
   36       0.59          11
   39       0.72          14
   41       0.86          16
   44       0.99          19
   45       1.05          20
   49       1.20          24
   51       1.24          26
   60       1.48          35
   64       1.52          39
   67       1.57          42
   72       1.61          47
   81       1.65          56
   93       1.66
   169       1.67
   189       1.68
   370       1.68
 END
 *
 *
 */
public class WalnutGulch {

    private AsciiReader reader;
    private ConfigReader cfgReader;
    private DEMHeader demHeader;
    private TCell[][] hood;
    private String fileName, cfgName;
    private WaterFlow wf;
    private Calendar start,finish;
    private double precipRate,manningsN=0.0f, cellSize=0.0f;
    private BigInteger hydroInterval=BigInteger.valueOf(120);
    private double infilRate=0.0f;
    private int secRain, secRun, gaugeRow, gaugeCol;
    private int iterRun,iterRain, iterInterval, factor;


    public static void main(String[] args)
    {
        WalnutGulch t=null;
        if(args.length<1)
        {
            System.out.println("Need <config file>");
            System.exit(0);
        }

        try{
            t=new WalnutGulch(args);
            t.initMatrix();
        }catch(Exception ex)
        {
            System.out.println("Error initializing "+ex.toString());
            System.exit(0);
        }
        t.runModel();
    }

    public WalnutGulch(String[] args)
    {
        cfgName=args[0];
    }
```

```
public void initMatrix()
{
    readConfig(cfgName);
    //Store config values locally
    fileName=cfgReader.getDEMName();
    precipRate=cfgReader.getPrecipRate();
    infilRate=cfgReader.getInfilRate();
    gaugeRow=cfgReader.getGaugeRow();
    gaugeCol=cfgReader.getGaugeCol();
    secRun=cfgReader.getSecsRun();
    secRain=cfgReader.getSecsRain();
    manningsN=cfgReader.getManningsN();
    cellSize=cfgReader.getCellSize();
    factor=cfgReader.getFactor();
    hydroInterval=BigInteger.valueOf(cfgReader.getInterval());

    System.out.println("iteration per sec = "+factor);
    iterRun=secRun*factor;
    iterRain=secRain*factor;
    hydroInterval=BigInteger.valueOf(cfgReader.getInterval()*factor);
    precipRate=precipRate/factor;
    infilRate=infilRate/factor;

demHeader=new DEMHeader();
    readDEM(fileName);
    if (manningsN > 0.0f)
        wf= new WaterFlow(manningsN, cellSize);
    else
    {
        System.out.println("Invalid Manning's n value");
        System.exit(1);
    }
    System.out.println("    Precip rate:"+precipRate+"    Infiltration rate:"+infilRate);
    hood=wf.initData(reader.getDemArray());
}

private void runModel()
{
    start=Calendar.getInstance();
    System.out.println("Starting model for "+secRun+" seconds.  Rain for "+secRain+"
secs");
    System.out.println("    model iteration "+iterRun+".    Rain iteration "+iterRain);
    System.out.println("Precipitation rate:"+precipRate+"    infiltration rate:"+infilRate);
    System.out.println("    discharge every "+hydroInterval+" iterations");
    NumberFormat nf=NumberFormat.getInstance();
nf.setMaximumFractionDigits(6);

double velocity=0.0f, vol=0.0f;
int time=0;
double slope=0.0f;
double rainDepth=0.0f;
for(int i=0; i<iterRun; i++)
{
    //Is this control program, rain is more explicitly controlled
    //iterRain is the entire storm event
    if(i < iterRain)
    {
        if (i<(181*factor))
        {
            wf.addPrecipitationDepth(.00000706f/factor);
            rainDepth=rainDepth+.00000706f/factor;
        }else if(i<(361*factor))
        {
            wf.addPrecipitationDepth(.00001129f/factor);
            rainDepth=rainDepth+.00001129f/factor;
        }else if(i<(421*factor))
        {
            wf.addPrecipitationDepth(.00002963f/factor);
            rainDepth=rainDepth+.00002963f/factor;
        }else if(i<(481*factor))
        {
            wf.addPrecipitationDepth(.00005927f/factor);
            rainDepth=rainDepth+.00005927f/factor;
        }else if(i<(601*factor))
        {
            wf.addPrecipitationDepth(.00002328f/factor);
            rainDepth=rainDepth+.00002328f/factor;
        }else if(i<(661*factor))
        {
            wf.addPrecipitationDepth(.00005927f/factor);
            rainDepth=rainDepth+.00005927f/factor;
        }else if(i<(841*factor))
        {
            wf.addPrecipitationDepth(.00001834f/factor);
            rainDepth=rainDepth+.00001834f/factor;
        }else if(i<(961*factor))
        {
            wf.addPrecipitationDepth(.00002963f/factor);
            rainDepth=rainDepth+.00002963f/factor;
        }else if(i<(1141*factor))
        {
```

```
        wf.addPrecipitationDepth(.00001834f/factor);
        rainDepth=rainDepth+.00001834f/factor;
    }else if(i<(1201*factor))
    {
        wf.addPrecipitationDepth(.00002540f/factor);
        rainDepth=rainDepth+.00002540f/factor;
    }else if(i<(1441*factor))
    {
        wf.addPrecipitationDepth(.00001588f/factor);
        rainDepth=rainDepth+.00001588f/factor;
    }else if(i<(1561*factor))
    {
        wf.addPrecipitationDepth(.00000847f/factor);
        rainDepth=rainDepth+.00000847f/factor;
    }else if(i<(2101*factor))
    {
        wf.addPrecipitationDepth(.00001129f/factor);
        rainDepth=rainDepth+.00001129f/factor;
    }else if(i<(2341*factor))
    {
        wf.addPrecipitationDepth(.00000423f/factor);
        rainDepth=rainDepth+.00000423f/factor;
    }else if(i<(2521*factor))
    {
        wf.addPrecipitationDepth(.00000706f/factor);
        rainDepth=rainDepth+.00000706f/factor;
    }else if(i<(2821*factor))
    {
        wf.addPrecipitationDepth(.00000339f/factor);
        rainDepth=rainDepth+.00000339f/factor;
    }else if(i<(3361*factor))
    {
        wf.addPrecipitationDepth(.0000188f/factor);
        rainDepth=rainDepth+.00000188f/factor;
    }else if(i<(4081*factor))
    {
        wf.addPrecipitationDepth(.00000035f/factor);
        rainDepth=rainDepth+.00000035f/factor;
    }else if(i<(8641*factor))
    {
        wf.addPrecipitationDepth(.00000006f/factor);
        rainDepth=rainDepth+.00000006f/factor;
    }else if(i<(9841*factor))
    {
        wf.addPrecipitationDepth(.00000021f/factor);
        rainDepth=rainDepth+.00000021f/factor;
```

```
                }
            }

            wf.removeInfiltrationDepth(infilRate);
            wf.run();

            BigInteger bT=BigInteger.valueOf(i);

            if(bT.mod(hydroInterval).intValue() < 30)
            {
                vol=vol+wf.getCellsVol(gaugeRow,gaugeCol);
                velocity=velocity+wf.getCellsVelocity(gaugeRow,gaugeCol);
            }else if (bT.mod(hydroInterval).intValue()==30)
            {
                System.out.println(i+"  Avg " +bT.divide(hydroInterval)+" "+ nf.format(vol/30)+"
velocity: "+nf.format(velocity/30));
                vol=0.0f;
                velocity=0.0f;
                time=0;
                slope=0.0f;
            }
        }

        finish=Calendar.getInstance();
        long millis=finish.getTimeInMillis()-start.getTimeInMillis();
        System.out.println();
        System.out.println("---------------");

        System.out.println("Time to complete for "+secRun+" sec model: "+millis/1000+"
seconds");
        System.out.println("A total of "+rainDepth+" fell over "+ secRain+" seconds");
        wf.printFinal();
    }

    public void writeVolumeMatrix()
    {
        String filename=new String("flow/FlowGrid.txt");
        double min=0.0f;
        double max =1.0f;
        double value;
        //Loop to find max and min
        for ( int row = 0; row <hood.length ; row++ )
            for(int col=0; col<hood[0].length; col++)
            {
                min=Math.min(min,hood[row][col].getVol());
                max=Math.max(max,hood[row][col].getVol());
            }

          System.out.println("Max volume:"+max+"        Min volume:"+min);
          try{
             BufferedWriter file=new BufferedWriter(new FileWriter(filename));
              for ( int ii = 0; ii <hood.length ; ii++ )
              {
                for(int col=0; col<hood[0].length; col++)
                {
                   value=hood[ii][col].getVol();
                   file.write(Double.toString(value));
                   file.write("::");
                }
                file.write("\r\n");
              }
             file.flush();
             file.close();
          }catch (Exception ex)
          {
             System.out.println("Error writing file"+ex.toString());
          }
   }

  private void readDEM(String filename)
  {
    reader = new AsciiReader(fileName);
   reader.readHeader();
   reader.readElevations();
   demHeader.LengthX =  reader.getRasterSize().width;
   demHeader.LengthY =  reader.getRasterSize().height;
   demHeader.missing = (double) reader.getMissingValue();
   demHeader.min = (double) reader.getMinimum();
    demHeader.max = (double) reader.getMaximum();
   System.out.println("  rows x cols\t{min, max}\tres in m     missing value");
   System.out.println("  "+demHeader.LengthY + " x " + demHeader.LengthX + "     {"+
demHeader.min+ ",  "+ demHeader.max + "}\t"+ demHeader.missing);
  }

  private void readConfig(String name)
  {
    cfgReader=new ConfigReader(name);
  }

}
```

```java
package jap.ca.time;

import java.util.Iterator;

/**
 * @author jparsons
 *
 *  Test Cell properties and Neighboorhood Operator
 *
 *
 *
 * General Process
 *
 * 1. Create matrix of Cells with elevation  <initData>
 * <Loop>
 * 2. Insert initial amount of precip
 * 2.5 Subtract infiltration
 * 3. Intial data
 *      calc waterdepth
 *      calc discharge direction & amounts
 * 4. Move current Cell's discharge to movableQ
 * 5. Move discharge from movableQ to destination cell's futureQ
 * 6. Write sum of futureQ to cellQ
 * <End Loop>
 *
 *  */

public class WaterFlow
{

    TCell[][] matrix;
    double cellWidth=0.0f;
    boolean notFirst=false;
    long timeCount=0;
    double manningsN=0.05f;
    double totalFallenVol=0.0d;
    double totInfil=0.0d;

    public WaterFlow(double friction, double cellsize)
    {
        manningsN=friction;
        cellWidth=cellsize;
        System.out.println("***Setting Mannings N:"+manningsN+"    Setting
CellSize:"+cellWidth);
    }

    public TCell[][] initData(double[][] elevations)
    {
        System.out.println("Entering WaterFlow.initData()    setting N
value:"+manningsN);
        //Create 2D matrix of TCells
        matrix=new TCell[elevations.length][elevations[0].length];
        for (int row=0; row<elevations.length; row++)
          for (int col=0; col<elevations[0].length; col++)
          {
              matrix[row][col]=new TCell(elevations[row][col]);
              matrix[row][col].setManningsN(manningsN);
              matrix[row][col].row=row;
              matrix[row][col].col=col;
              matrix[row][col].setCellSize(cellWidth);
          }
        return matrix;
    }

    public void run()
    {
        process();
        moveDischarge();
    }

    public double getCellsVol(int row,int col)
    {
        return matrix[row][col].getVol();
    }

    public double getCellsDepth(int row,int col)
    {
        return matrix[row][col].getWaterDepth();
    }
```

```
public double getCellsVelocity(int row,int col)
{
   double f=matrix[row][col].getVelocity();
   return f;
}

private void process()
{
   timeCount++;
   TCell[][] matrix2=new TCell[3][3];
   for (int row=1;row<matrix.length-1; row++)
      for (int col=1; col<matrix[0].length-1; col++)
      {
         //Currently Building a Moore neighborhood,
         //This is ok for now, will only use subset (Von Nuemann)
         if(matrix[row][col].getVol() > 0.0f)
         {
               matrix2[0][1]=matrix[row-1][col];
               matrix2[1][0]=matrix[row][col-1];
               matrix2[1][1]=matrix[row][col];
               matrix2[1][2]=matrix[row][col+1];
               matrix2[2][1]=matrix[row+1][col];

            if(matrix[row][col].newVolume())
            {
               TNeighborhoodOp.cellSlope(matrix2, cellWidth);
               matrix[row][col].newVolume=false;
            }
            matrix[row][col].incrementTime();
            if (matrix[row][col].alarm())
            {
               TNeighborhoodOp.dischargeDirectionMin(matrix2, cellWidth);
            }
         }
      }
   matrix2=null;
   notFirst=true;
}
```

```
private void moveDischarge()
{
   for (int row=1;row<matrix.length-1; row++)
      for (int col=1; col<matrix[0].length-1; col++)
      {
         if (matrix[row][col].getVol() > 0.0f)
         {
            if (matrix[row][col].getQtoCells() !=null)
            {
               Iterator iter=matrix[row][col].getQtoCells().iterator();
               if (iter != null)
               {
                  double cellDischarge=0.0f;
                  while(iter.hasNext())
                  {
                     DischargeTCell dc=(DischargeTCell)iter.next();
                     double f=(dc.getQFract()*matrix[row][col].getMovableVol());
                     dc.getCell().setFutureVol(f);
                  }
                  matrix[row][col].resetMovableQ();
                  matrix[row][col].QtoCells=null;
                  //NOTE added Mar 22
                  if(matrix[row][col].getVol()<=0.0f)
                  {
                     matrix[row][col].resetTime();
                     matrix[row][col].releaseTime=0;
                  }


               }
            }
         }
      }

//This moves from the futureVol to actual Volume.
//This step is used to replace access every neighbor to get vol.
//Without futureVol, each cell must be "touch" many more times

//Not moving water in border cells to regular volume
//that was cause high surface elevations...essentially a dam
      for (int row=1;row<matrix.length-1; row++)
```

```
        for (int col=1; col<matrix[0].length-1; col++)
        {
            if (matrix[row][col].getFutureVol() > 0.0f)
            {

matrix[row][col].setVol(matrix[row][col].getVol()+matrix[row][col].getFutureVol());
                matrix[row][col].resetFutureVol();
            }
        }
}

public void borderDischarge()
{
    double totalDischarge=0.0f;
    for (int col=0; col<matrix[0].length; col++)
    {
        totalDischarge=totalDischarge+matrix[0][col].getVol();
        totalDischarge=totalDischarge+matrix[matrix.length-1][col].getVol();
    }
    for (int row=0; row<matrix.length; row++)
    {
        totalDischarge=totalDischarge+matrix[row][0].getVol();
        totalDischarge=totalDischarge+matrix[row][matrix[0].length-1].getVol();
    }
}

public void addPrecipitation(double precip)
{
    for (int row=1; row<matrix.length-1; row++)
        for (int col=1; col<matrix[0].length-1; col++)
        {
            totalFallenVol=totalFallenVol+precip;
            matrix[row][col].setVol(matrix[row][col].getVol()+precip);
        }
}

//switched to this method for adding precip,
double tempVol=0.0f;
double totPrecip=0.0f;
public void addPrecipitationDepth(double precip)
{
    totPrecip=totPrecip+precip;
    tempVol=precip*(cellWidth*cellWidth);
    for (int row=1; row<matrix.length-1; row++)
        for (int col=1; col<matrix[0].length-1; col++)
        {
            totalFallenVol=totalFallenVol+tempVol;
            matrix[row][col].setVolNoCalc(matrix[row][col].getVol()+tempVol);
        }
}

public void addCellPrecipitation(double precip)
{
    totalFallenVol=totalFallenVol+precip;
matrix[35][35].setVol((matrix[35][35].getVol())+precip);
}

/*
 * Subtrac infiltration rate from all cells,
 * if those cells have < infiltration rate
 * set Vol to 0 and add only that amount to totalInfiltrated
 */
private void removeInfiltration(double infil)
{
    for (int row=0; row<matrix.length; row++)
        for (int col=0; col<matrix[0].length; col++)
        {
            if(matrix[row][col].getVol() > infil)
            {
                totInfil=totInfil+infil;
                matrix[row][col].addInfiltratedVol(infil);
                matrix[row][col].setVol(matrix[row][col].getVol()-infil);

            }else
            {
                totInfil=totInfil+matrix[row][col].getVol();
                matrix[row][col].addInfiltratedVol(matrix[row][col].getVol());
                matrix[row][col].setVol(0.0f);
            }
        }
    }

    public void removeInfiltrationDepth(double infil)
    {
        tempVol=infil*(cellWidth*cellWidth);
        for (int row=0; row<matrix.length; row++)
            for (int col=0; col<matrix[0].length; col++)
            {
                if(matrix[row][col].getVol() > tempVol)
                {
                    totInfil=totInfil+tempVol;
                    matrix[row][col].addInfiltratedVol(tempVol);
                    matrix[row][col].setVol(matrix[row][col].getVol()-tempVol);
                }else
                {
                    totInfil=totInfil+matrix[row][col].getVol();
                    matrix[row][col].addInfiltratedVol(matrix[row][col].getVol());
                    matrix[row][col].setVol(0.0f);
                }
            }
    }

     public double getMatrixVol()
     {
         double dvol=0.0d;
         for ( int ii = 0; ii <matrix.length ; ii++ )
             for ( int jj = 0;jj<matrix[0].length; jj++ )
             dvol = dvol+matrix[ii][jj].getVol();

     return  dvol;
     }

     public double getInfiltrationVol()
     {
         return totInfil;
     }

     public double getInteriorVol()
     {
         double dvol=0.0d;
         for ( int ii = 1; ii <matrix.length-1 ; ii++ )
             for ( int jj = 1;jj<matrix[0].length-1; jj++ )
             dvol = dvol+matrix[ii][jj].getVol();

     return  dvol;
     }

     public double getBorderVol()
     {
         double dvol=0.0d;
         for ( int jj = 0;jj<matrix[0].length-1; jj++ )
         {
             dvol = dvol+matrix[0][jj].getVol();//+matrix[0][jj].getFutureVol();
             dvol = dvol+matrix[matrix.length-1][jj].getVol();//+matrix[matrix.length-
1][jj].getFutureVol();
         }

         for ( int ii = 0; ii <matrix.length-1 ; ii++ )
         {
         dvol = dvol+matrix[ii][0].getVol();//+matrix[ii][0].getFutureVol();
             dvol = dvol+matrix[ii][matrix[0].length-
1].getVol();//+matrix[ii][matrix[0].length-1].getFutureVol();
         }

     return  dvol;
     }

     public double getMovedBorderVol()
     {
         double dvol=0.0d;
         for ( int jj = 0;jj<matrix[0].length-1; jj++ )
         {
             dvol = dvol+matrix[0][jj].getFutureVol();
             dvol = dvol+matrix[matrix.length-1][jj].getFutureVol();
         }

         for ( int ii = 0; ii <matrix.length-1 ; ii++ )
         {
         dvol = dvol+matrix[ii][0].getFutureVol();
        dvol = dvol+matrix[ii][matrix[0].length-1].getFutureVol();
    }

return  dvol;
}

public void printMaxCellBorderVol()
{
    double max=0.0d;
    TCell tc=null;
    for ( int jj = 0;jj<matrix[0].length-1; jj++ )
    {
        if(matrix[0][jj].getFutureVol() > max)
        {
            max=matrix[0][jj].getFutureVol();
            tc=matrix[0][jj];
        }

        if(matrix[matrix.length-1][jj].getFutureVol() > max)
        {
            max=matrix[matrix.length-1][jj].getFutureVol();
            tc=matrix[matrix.length-1][jj];
        }

    }

    for ( int ii = 0; ii <matrix.length-1 ; ii++ )
    {
        if(matrix[ii][0].getFutureVol() > max)
        {
            max=matrix[ii][0].getFutureVol();
            tc=matrix[ii][0];
        }

        if(matrix[ii][matrix[0].length-1].getFutureVol() > max)
        {
            max=matrix[ii][matrix[0].length-1].getFutureVol();
            tc=matrix[ii][matrix[0].length-1];
        }
```

```
    }

    System.out.println("Border Cell with maximum volume: "+ max+" cell:
"+tc.row+","+tc.col);

 }

 public void printMaxCellInteriorVol()
 {
    double max=0.0d;
    TCell tc=null;
    for ( int ii = 1; ii <matrix.length-1 ; ii++ )
    for ( int jj = 1;jj<matrix[0].length-1; jj++ )
    {
        if(matrix[ii][jj].getVol() > max)
        {
            max=matrix[ii][jj].getVol();
            tc=matrix[ii][jj];
        }
    }

    if(tc != null)
        System.out.println("Interior Cell with maximum volume: "+ max+" cell:
"+tc.row+","+tc.col);

 }

 public float[] getDepthArray()
 {
    float[]  samples = new float[matrix.length*matrix[0].length];
    int index = 0;
    for ( int ii = 0; ii <matrix.length ; ii++ )
    {
        for ( int jj = 0;jj<matrix[0].length; jj++ )
        {
            samples[index] = (float)matrix[ii][jj].getWaterDepth();
            index++;
        }
    }
    return  samples;
```

```
}

public double[] getInfiltrationArray()
{
    double[]  samples = new double[matrix.length*matrix[0].length];
    int index = 0;
    for ( int ii = 0; ii <matrix.length ; ii++ )
    {
        for ( int jj = 0;jj<matrix[0].length; jj++ )
        {
            samples[index] = matrix[ii][jj].getInfiltratedVol()/(cellWidth*cellWidth);
            index++;
        }
    }
    return  samples;
}

public double[] getDisplayArray()
{
    double[]  samples = new double[matrix.length*matrix[0].length];
    int index = 0;
    for ( int ii = 0; ii <matrix.length ; ii++ )
    {
        for ( int jj = 0;jj<matrix[0].length; jj++ )
    {
            samples[index] = matrix[ii][jj].getVol();
        index++;
    }
    }

return  samples;
}
}
```

```
package jap.ca.time;

import java.util.ArrayList;
import java.util.Iterator;
/**
 * @author jparsons
 * This class contains static methods to perform neighborhood operations
 *
 */

public class TNeighborhoodOp {

    /*
     * cellSlope(...)
     * Cell Slope is calculated using Zevenbergen and Thornes method
     */
    public static double cellSlope(TCell[][] neighborhood, double cellWidth)
    {
        double slope=0.0f;
        try{
            double g= -1*neighborhood[1][0].getSurfaceElev() +
neighborhood[1][2].getSurfaceElev();
            g=g/(2*cellWidth);

            double h= (neighborhood[0][1].getSurfaceElev() -
neighborhood[2][1].getSurfaceElev()) / (2*cellWidth);
            slope=(double)Math.sqrt(Math.pow(g,2) +Math.pow(h,2));
            if (0.0f==slope)
                slope=0.0001f; //set a default slope
        }catch(Exception ex)
        {
            System.out.println("Error calculating cell slope.");
            System.out.println(ex.toString());
        }
        //Go ahead and store slope here as well as return it
        neighborhood[1][1].setCellSlope(slope);
        return slope;
    }
```

```
//*****************************************************
//Modify This method for Von Nueman neighborhood and time based movement
//Will determine the direction of flow and amount to move.
//Stage 1.  Move to one cell
//          Determine cell with greatest slope
//          Store as the DischargeCell in ArrayList..may add new more cells


//Stage 2.  Diffuse all cells to lower than center
//          Add all cells lower than center to arraylist
    public static ArrayList dischargeDirectionMin(TCell[][] hood, double cellWidth)
    {
        ArrayList al=new ArrayList();
        TCell centerCell=hood[1][1];
        double diff =0.0f;
        double diffMin=0.0f;  //needed to calculate amount to move
        double totSlope=0.0f;
        double sumElevDiff=0;

        //pick direction
        TCell receiver=null;
        diffMin=centerCell.getSurfaceElev() - hood[0][1].getSurfaceElev() ;
        if (diffMin > 0)
        {
            al.add( new DischargeTCell(hood[0][1], diffMin));
            sumElevDiff=diffMin;
        }else
            diffMin=Float.MAX_VALUE;  //set high

        diff=centerCell.getSurfaceElev() - hood[1][0].getSurfaceElev();
        if (diff > 0)
        {
            diffMin=Math.min(diffMin,diff);
            sumElevDiff=sumElevDiff+diff;
            al.add( new DischargeTCell(hood[1][0], diff));
        }

        diff=centerCell.getSurfaceElev() - hood[1][2].getSurfaceElev();
        if (diff > 0)
        {
```

```
      diffMin=Math.min(diffMin,diff);
      sumElevDiff=sumElevDiff+diff;
      al.add( new DischargeTCell(hood[1][2], diff));
   }

   diff=centerCell.getSurfaceElev() - hood[2][1].getSurfaceElev();
   if (diff > 0)
   {
      diffMin=Math.min(diffMin,diff);
      sumElevDiff=sumElevDiff+diff;
      al.add( new DischargeTCell(hood[2][1], diff));
   }

   //If there is only one discharge cell
   if(al.size() == 1)
   {
      DischargeTCell dc=(DischargeTCell)al.get(0);
      dc.qFract=1;
               double depthDiff=0.0f;
      if(dc.getCell().getIsBorderCell())
      {
         depthDiff=(centerCell.getSurfaceElev()-dc.getCell().getSurfaceElev());
      }else
         depthDiff=diffMin;

      double
volume=cellWidth*cellWidth*Math.min(depthDiff,centerCell.getWaterDepth());
      centerCell.setMovableVol(volume);
      centerCell.QtoCells=al;
   }else if (!al.isEmpty())
   {
      //If >1 receiving (discharge) cells
      Iterator iter=al.iterator();
      while(iter.hasNext())
      {
         DischargeTCell dc=(DischargeTCell)iter.next();
         dc.qFract=dc.diff/sumElevDiff;
      }
      double depthDiff=diffMin;
      double
volume=cellWidth*cellWidth*Math.min(depthDiff,centerCell.getWaterDepth());
      centerCell.setMovableVol(volume);
      centerCell.QtoCells=al;
   }else if (al.isEmpty())
   {
      al=null;
   }

   return al;
  }

}
```

```
package jap.ca.time;

import java.util.*;
/**
 * @author jparsons
 *
 */

public class TCell {

    static int factor = 1;      //default number of iterations per sec
    double N;                   //default Manning's N value
    double CellWidth = 0.0f; //default cell size in meter
    double elevation;           //value from DEM
    double cellSlope = .0001f;    //default slope value
    double waterDepth;            //calculated water depth
    double surfaceElev;         //elev+waterdepth
    private double vol;       //amount of water currently in cell in cubic m
    double futureVol;           //amount of water moving into the cell
    double movableVol;            //amount of water that will move
    public ArrayList QtoCells = null; //list of cells to receive water (and amount)
    double velocity;            //calculated velocity
    double totInfil;            //total amount of infiltrated water
    boolean borderCell = false; //set to true if it is a border cell
    boolean newVolume = false;   //set to true if the cell get more water
    private boolean wasDry = false; //for continuous code
    public int releaseTime = 1; //number of iterations to hold water in a cell
    public int countTime = 0;     //count iterations for this cell

    //DEBUG only....store row and col within matrix
    public int row;
    public int col;
    //DEBUG

    public TCell(double e) {
        elevation = e;
        surfaceElev = e;
    }

    static public void setFactor(int f) {
        factor = f;
    }

    /*
     * Needed for continuous flow
     */
    public void setWasDry(boolean b) {
        wasDry = b;
    }

    /*
     * Needed for continuous flow
     */
    public boolean getWasDry() {
        return wasDry;
    }

    public double getElevation() {
        return elevation;
    }

    public double getManningsN() {
        return N;
    }

    public void setManningsN(double friction) {
        N = friction;
    }

    /*  Uncomment if watershed mask is ever implemented
        public boolean getInWatershed()
        {
            return inWatershed;
        }

        public void setInWatershed(boolean w)
        {
            inWatershed=w;
        }
    */
```

```
public double getCellSope() {
    return cellSlope;
}

public void setCellSlope(double s) {
    cellSlope = s;
}

public double getWaterDepth() {
    return waterDepth;
}

public void setWaterDepth(double w) {
    waterDepth = w;
}

public double getSurfaceElev() {
    return surfaceElev;
}

public void setSurfaceElev(double s) {
    surfaceElev = s;
}

public double getVol() {
    return vol;
}

/*
 * setVol()
 * set the amount of water volume in the cell
 * perform calculations
 *  waterdepth
 *  velocity
 * set newVolume to true
 */
public void setVol(double s) {
    if (s < 0.0f)
        s = 0.0f; //catch rounding errors
    vol = s;
    calcWaterDepth();
    setVelocity();
    newVolume = true;
}

/*
 * setVolNoCalc()
 * reduce the number of calculations per iteration with this version
 * must be used carefully
 *
 */
public void setVolNoCalc(double s) {
    if (s < 0.0f)
        s = 0.0f; //catch rounding errors
    vol = s;

    newVolume = true;
}

public boolean newVolume() {
    return newVolume;
}

public double getMovableVol() {
    return movableVol;
}

public void setMovableVol(double f) {
    movableVol = f;
}

/*
 * resetMovableQ()
 * set how much volume is left in cell after water movement
 */
public void resetMovableQ() {
    double volf = vol - movableVol;
    if (volf < 0.0f)
        setVol(0.0f);
    else
        setVol(volf);
    movableVol = 0.0f;
    //Note: Could reset count here if too fast
}

public double getFutureVol() {
    return futureVol;
}

public void resetFutureVol() {
    futureVol = 0.0f;
}

public void setFutureVol(double s) {
    futureVol = futureVol + s;
}

public ArrayList getQtoCells() {
    return QtoCells;
}

public void printCell() {
    System.out.println("Cell position: " + row + "," + col);
    System.out.println("  Elevation   : " + elevation);
    System.out.println("  Cell slope  : " + cellSlope);
    System.out.println("  Water Depth : " + waterDepth);
    System.out.println("  Surface Elev: " + surfaceElev);
    System.out.println("  Velocity    : " + velocity);
    System.out.println("  Release time: " + releaseTime);
}

/*
 * Calculate the water depth in the cell
 * add to elevation for surfaceElev
 * Depth=volume/(length*width)
 */
public void calcWaterDepth() {
    if (vol > 0.0f) {
        waterDepth = vol / (CellWidth * CellWidth);
        surfaceElev = elevation + waterDepth;
    } else {
        waterDepth = 0.0f;
        surfaceElev = elevation;
    }
}

public void incrementTime() {
    countTime++;
}

/*
 * alarm()
 * check to see if it is time to move water
 */
public boolean alarm() {
    if (countTime >= releaseTime)
        return true;
    else
        return false;
}

public void resetTime() {
    countTime = 0;
}

//Velocity
private double hydrR;
private double vel;

/*
 * setVelocity()
 * Calculate velocity
 * and set release time
 */
public void setVelocity() {
    //double velocity=0.0f;
    if (vol == 0.0f) {
        velocity = 0.0f;
        releaseTime = 1;
        } else {
            hydrR = Math.pow(waterDepth, .66666667);
            vel = hydrR * Math.pow(cellSlope, .5f);

            //vel is in meters per sec
            vel = vel / N;
            velocity = (double) vel;
            //this is relase time in fractions of a second
            releaseTime = (int) Math.round((CellWidth / vel) * factor);
            //(CELLWIDTH_1Sec/CellWidth));
            if (releaseTime < 1)
                releaseTime = 1;
        }
    }

    public int getReleaseTime() {
        return releaseTime;
    }

    public double getVelocity() {
        return velocity;
    }

    public void addInfiltratedVol(double v) {
        totInfil = totInfil + v;
    }

    public double getInfiltratedVol() {
        return totInfil;
    }

    public void setIsBorderCell(boolean b) {
        borderCell = b;
    }
    public boolean getIsBorderCell() {
        return borderCell;
    }

    public double getCellSize() {
        return CellWidth;
    }

    public void setCellSize(double f) {
        CellWidth = f;
    }
}
```

REFERENCES

Coulthard, T. J. 1999. Modelling Upland Catchment Response to Holocene Environmental Change. *Unpublished Phd Thesis*. University of Leeds.

Crave, A. and P. Davy. 2001. A stochastic "precipiton" model for simulating erosion/sedimentation dynamics. *Computers & Geosciences* 27, no. 7 : 815-827.

Fonstad, Mark. 2003. A Time-based CA: Notes from Moscow. Personal Communication.

Food and Agriculture Organization of the United Nations (FAO). *Irrigation Water Management: Irrigation Methods.* Available on-line. http://www.fao.org/docrep/S8684E/s8684e00.htm.

Johnson, Mark, William F. Coon, Vishal K. Metha, Tammo S. Steenhuis, Erin S. Brooks and Jan Boll. 2003. Application of two hydrologic models with different runoff mechanisms to a hillslope dominated watershed in the northeastern US: a comparison of HSPF and SMR. *Journal of Hydrology* 284: 57-76.

Luo, Wei. 2001. LANDSAP: a coupled surface and subsurface cellular automata model for landform simulation. *Computers & Geosciences* 27, no. 3: 363-367.

Murray, A. Brad and Chris Paola. 1994. A cellular model of braided rivers. *Nature* 371, no. 1: 54-59.

Murray, A. Brad and Chris Paola. 1997. Properties of a cellular braided-stream model. *Earth Surface Processes and Landforms* 22: 1001-1025.

Rocky Mountain National Park (RMNP). 2004. Available on-line. http://www.nps.gov/romo/index.html.

Rocky Mountain National Park GIS Data (RMNP GIS). 2004. Data acquired from Rocky Mountain National Park GIS Program. Estes Park, Colorado.

Southwest Watershed Research Center (SWRC). Walnut Gulch Experimental Watershed. [Online]. Available from http://www.tucson.ars.ag.gov/unit/Watersheds/WGEW.htm.

Thomas, R. and A.P. Nicholas. 2002. Simulation of a braided river flow using a new cellular routing scheme. *Geomorphology* 43: 179-195.

Tobler, Waldo. 1970. A computer movie simulating urban growth in the Detroit region. *Economic Geography* 46, no. 2 : 234-240.

VisAD Home Page. 2004. VisAD. Available on-line. http://www.ssec.wisc.edu/~billh/visad.html.

Wolfram, Stephen. 1983. Cellular Automata. *Los Alamos Science*. 9: 2-21.

Wolfram, Stephen. 1984. Universality and Complexity in Cellular Automata. *Physica D*. 10: 1-35.

VITA

Jay Alan Parsons was born in San Angelo, Texas, on January 17, 1972. He graduated from Midland High School in Midland, Texas. He received a Bachelor of Science in Computer Science from Texas A&M University in May 1995. After completing his undergraduate degree, he worked as a software developer in Lewisville and Austin, Texas. In January, 2001, he entered the Graduate College at Texas State University-San Marcos.

Permanent Address:	3009 Overland Street
	Round Rock, Texas 78681

This thesis was typed by the author.