Fall 2014

Watershed Delineation in the Field: A New Approach for Mobile Applications Using LiDAR Elevation Data

Samuel Adam Noel
Purdue University

Graduate School Form 30
(Updated 11/20/2014)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Samuel Adam Noel

Entitled
Watershed Delineation in the Field: A New Approach for Mobile Applications Using LiDAR Elevation Data

For the degree of Master of Science in Agricultural and Biological Engineering

Is approved by the final examining committee:

Dennis Buckaster

Bernard Engel

Jane Frankenberger

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification/Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Dennis Buckaster
Bernard Engel

Approved by: Bernard Engel 11/25/2014

Head of the Department Graduate Program Date

WATERSHED DELINEATION IN THE FIELD: A NEW APPROACH FOR MOBILE APPLICATIONS USING LIDAR ELEVATION DATA

A Thesis

Submitted to the Faculty

of

Purdue University

by

Samuel Adam Noel

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Agricultural and Biological Engineering

December 2014

Purdue University

West Lafayette, Indiana

ACKNOWLEDGEMENTS

First, I would like to acknowledge my committee for their support and guidance throughout this undertaking: Dr. Dennis Buckmaster, Dr. Bernard Engel, and Dr. Jane Frankenberger. Without their belief in me, I would not have been able to see through this truly transformative learning opportunity. I would also like to thank Aaron Ault, who introduced me to programming and also let me ride around with him in his combine. To that same point I would also like to thank Dr. Jim Krogmeier as well as Andrew Balmos and Alex Layton for their part in showing me how to think like a computer engineer (at least as best as I can).

It must also be stated that I could not have done this without the love and support of my family and friends. I would especially like to thank my parents Millie and Steve for teaching me the value of hard work and an education. Their unwavering support has been vital to my commitment to a lifetime of learning. Finally, I cannot thank enough my future wife Kristen for being there when times were tough and for helping me find my way through it.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

NOMENCLATURE

Catchment – In the Watershed Delineation App, catchments refer to the overall state of connectivity across the entire DEM area. Each unique set of cells that drain to a common point is a different catchment. They may be either depressions or areas that drain to and terminate at the edge of the DEM area.

Curve Number – An empirical parameter that describes the runoff potential of a given area based on hydrologic soil group, cover type (land use), treatment, hydrologic condition, and antecedent runoff condition.

Depression – An area in which water is retained due to a basin-like shape in the DEM. The entire area/set of cells that drain into the basin make up the confines of the depression feature.

DEM – Digital Elevation Model. A digital depiction of the earth's surface elevations. Common forms of DEMs include contours, a "point cloud" of individual point elevation measurements, and a rectangular grid of elevations. The usage of 'DEM' in the methods of this work refers to grid-based models.

Hydrologic Connectivity – The extent to which the landscape impedes or allows water to be conveyed across the ground surface. In terms of modelling hydrologic connectivity, it is the number of grid cells or the proportion of the DEM area that flows freely to and off of the DEM edges.

Pit Cell – The particular cell at the bottom of a depression where all eight neighboring cells are of a higher elevation, meaning water has nowhere to travel from this particular cell. It is also common for these cells to be called sink cells.

Puddles – Puddles are an additional visual layer supplied in the Watershed Delineation App that may be used for troubleshooting and validation of results. The Puddles layer is generated by comparing the DEM after elevations have been raised in the filling process with the original DEM. Cells that have been altered should coincide with puddles on the ground after rainfall.

Watershed – The area draining to a particular point of interest.

ABSTRACT

Noel, Samuel A. M.S.A.B.E., Purdue University, December 2014. Watershed Delineation in the Field: A New Approach Using LiDAR Elevation Data. Major Professor: Dennis Buckmaster.

With the advancement of mobile devices, opportunities to take watershed management tasks out of the office and into the field can be realized. In turn, field workers can utilize these technologies to expedite the decision-making process so that they may focus on meeting with clients and addressing agricultural watershed management issues. High-resolution (~1.5 m post-spacing) elevation data gathered by light detection and ranging (LiDAR) provides the topographic detail necessary to model hydrology at the field-scale (~1 km2).

Non-artifactual surface depressions lead to erroneous surface flow patterns when using existing algorithms. So a sequential depression-filling algorithm (SDFA) has been developed to address topographies that contain these types of features. Given a rainfall amount, water distributed across the landscape accumulates and fills only those depressions as necessary, halting the filling process when the only depressions that remain require additional rainfall. After the filling process is completed, the watershed contributing area draining to any particular point of interest may be identified and in the future this may be used as

input to hydrologic models. Methods have also been developed to implement subsurface drainage features such as culverts and tile-inlets as well as soil infiltration such that the dynamics of how water is shed from a given landscape can be better represented. Tile inlets and drainage features may be identified via user input and assigned a drainage rate while infiltration may be implemented by assigning a drainage rate to each grid cell in the DEM based on their soil-type. The combination of the sequential depression-filling algorithm and this drainage feature implementation provides the tools to model localized drainage patterns that will match user's field observations at the scale of hundreds of hectares.

The flow routing, depression identification, and filling procedures of the SDFA were compared to similar functions in the ArcGIS Hydrology Toolset under conditions where all depressions were filled in order to validate that those components of the algorithm are identical as intended. Furthermore, several digital elevation models (DEMs) were analyzed to determine the variability in hydrologic connectivity across these landscapes as a function of rainfall and as a function of DEM size. In addition to depression storage, the impacts of infiltration on hydrologic connectivity over these landscapes were also analyzed using the SCS Curve Number Method. The assumptions made by existing algorithms that require complete hydrologic connectivity do not hold up in all landscapes, even more so when considering the effects of infiltration. In these landscapes, surface hydrologic connectivity varies noticeably with rainfall excess and it is inaccurate to assume that the watershed should be modeled as a monotonically descending surface. In an applicability study of DEM size, depression features began to be

captured around the 1 km^2 scale while it is recommended to use DEMs larger than 2 km^2 to ensure that the depressional features and their contributing areas are completely captured within the DEM extent so that the SDFA may account for those features correctly.

The SDFA algorithm was ported from Matlab to an Android application for mobile phones and tablets. The Watershed Delineation app is free and publicly available through the Google Play Store. Users may view DEMs on a Google Map, use the sequential depression-filling algorithm to fill depressions, and delineate watersheds. It was found that the performance of this algorithm is a function of the number of depressions in the DEM which increases with DEM resolution (due to signal-noise effects). At a 3-meter resolution, the ideal DEM dimensions suitable for use of the SDFA on a Google Nexus 4 phone are about 500 x 500 (225 hectares), which took 68 seconds to run. At DEM sizes much greater than this, performance is drastically reduced. As DEM resolution increases, noise effects in the data (which vary based on the raw LiDAR data) result in a high amount of depression features causing an excessive number of iterations of the filling procedure within the algorithm.

CHAPTER 1. INTRODUCTION

1.1 Motivation

As the global population increases, farmers must work to increase food production while the number of farmable acres remains relatively constant. To address this challenge, inputs such as fertilizers, pesticides, herbicides, and water are being used at increasing rates to improve productivity of the existing land. These input-intensive farming systems require proper management in order to maximize productivity while minimizing environmental impacts. Surface runoff resulting from excessive rainfall events are a major mobilizer of nutrients, pesticides, and detached sediment. These constituents may be and often are carried to streams and waterways, leading to eutrophication and dead zones in streams and coastal waters.

Tools and approaches to manage agricultural runoff at the field scale must be developed in order to mitigate risks of impairment to the natural resources that farmers depend on and the environment as a whole. Water issues are local; they are assessed by observing the specific situation at the location of the problem in the field, and decisions to manage these issues should be tailored to fit this situation. For farmers with agricultural water issues, field-scale solutions currently exist in the form of conservation practices such as grassed waterways.

Tools such the National Agricultural Pesticide Risk Analysis (NAPRA) tool exist to evaluate the consequences of various management options to aide in the decision-making process (Antony and Engel, 2009).

The most basic element utilized in the implementation of these tools and solutions is the watershed. Most water issues typically demand an inquiry as to one or more of the following: a) How does the water flow through a given landscape? b) How much water moves through a specific point? and c) What area is contributing to some observable adverse effect at a given location? Correspondingly, a watershed indeed offers some indication as to the general patterns of water movement throughout an area, the magnitude of water (volume/flow) at a given outlet point of interest, and the area footprint that must be managed to produce changes at the outlet point. A tool to delineate field-scale watersheds seems to be the logical first step to providing management solutions on the ground at the source of the issue. However, current means of watershed delineation are not properly equipped to perform well at field-scales. Traditional watershed algorithms neglect the effects of ponding by assuming that all topography from ridges to valleys are monotonically descending, providing a direct route for runoff into waterways. In fact, this is rarely the case due to natural depressions, road embankments, and berms constructed to prevent erosion and pollutant-loaded runoff direct access to streams.

1.2 Enabling Technologies

The recent widespread availability of high-resolution Light Detection and Ranging (LiDAR) data has been one of the major enablers to automated

watershed delineation at field scales. Additionally, advances in mobile devices such as cell phones and tablets have effectively given individuals the processing power and visualization capacity of a small PC in the palm of their hand. With this, automated watershed delineation can be taken out of offices and into mobile settings, providing users with a tool to aid their observations, perform hydrologic computations, and expedite the decision-making process. Furthermore, the ability to visually communicate information while on site can provide the all-important “teachable moment” when changes should be made.

1.3 Background, Problem, and Proposition

Current automated watershed delineation practice is based on a set of functions first described by O’Callaghan and Mark (1984) for the purpose of digital terrain modeling and stream network extraction. These operations utilized digital elevation models (DEMs) containing a regularly spaced grid of elevation values. For each cell in a DEM, the direction of water flow out of that cell is computed by looking at the 8 surrounding cells to find the one with the steepest downward slope away from the current cell. The direction to the cell with steepest descent is assigned as the *flow direction* for the cell in question. Given a fully computed set of flow directions, the path of water through a DEM can be followed. All of those cells upstream and directed toward a cell may be delineated as the contributing area, or *watershed*.

However, the application of existing watershed delineation algorithms developed for low resolution data to high resolution DEMs can produce results which conflict with in-person observations in the field. These algorithms fail to

properly account for depression storage as it pertains to generation of surface flow (Chu et al., 2010; Appels et al., 2011; and Yang and Chu, 2012). Additionally, localized drainage features such as tile inlets and culverts are not easily identified by DEMs and dealt with by automated methods.

The glaciated landscapes of the Midwestern United States present additional complications in the form of enclosed natural depressions that rarely become hydrologically connected via surface flow. Because standard, large-scale watershed delineations require hydrologic surface connectivity, areas which do not naturally drain over the surface such as these depressions become key problem areas. Existing algorithms artificially alter either water flow direction or the DEM itself by filling or breaching depressions until complete connectivity is obtained (O'Callaghan and Mark, 1984; Martz and Garbrecht, 1999; Lindsay and Creed, 2005). This guarantees that the contributing area to a downstream point is continuous. However, sometimes these alterations can be quite dramatic, creating unrealistic scenarios and flow predictions that may not agree with in-person observations at line-of-sight scales.

Field-scale watershed delineations should not require hydrologic connectivity. Interesting questions at a field scale are generally related to actual rainfall events: e.g. the size of tile riser needed to drain a large area in a typical growing season, or the type and size of grassed waterway needed to reduce erosion at a particular location. In addition, much of the rainfall on active farmland drains through infiltration or underground structures such as tiles and

culverts rather than over the surface. A person in the field can easily identify manmade structures, real stream locations and flow directions, changes to the landscape over time, or other conditions that may not be accurately captured by existing automated watershed delineation algorithms.

1.4 Objectives

Toward the ultimate goal of accurately visualizing water movement while in person to manage runoff at field scales, the specific objectives of this work were to:

1) **develop an automated watershed delineation algorithm**: The algorithm should account for field-scale topography, match field-level observations, and take into account field-scale drainage features (i.e. tile inlets and culverts) and modifications (alteration of elevations).
2) **verify and analyze the algorithm**: The algorithm will be evaluated by comparing the outputs of our algorithm to the outputs of traditional pit-filling algorithms. An applicability study will be performed to evaluate to what extent the algorithm developed differs from existing methods and any differences across landscapes.
3) **implement delineation into a mobile application:** The algorithm should be accessible and useful in the field. The implementation of the algorithm into an application for mobile devices and the methods of development should be described.

CHAPTER 2. BACKGROUND

2.1 Elevation Data

2.1.1 Digital Elevation Model (DEM) Data Structures

Topographic surface data is typically represented by one of three data structures called digital elevation models: triangular irregular networks (TINs), contours, and gridded (or rasterized) data. A DEM contains only ground features while a digital surface model (DSM) may capture buildings, trees, and other non-ground-level features. Each data structure has its own advantages, disadvantages, and best-usage situations.

A Triangular Irregular Network (TIN) is a data structure comprised of non-uniformly spaced point data (x, y, z coordinates) connected by lines to form a mesh of triangular planes, called facets, that produce a three-dimensional surface. Areas with low topographic relief do not require the same density of point data as a more highly variable surface; in this way TINs are a more efficient data structure than others, often reducing the file size necessary to represent a surface. However, the irregularity of the data requires more complicated algorithms and computations for applications such as the water of flow (Moore et al. 1991).

Contours are a set of isolines drawn through a landscape where each line represents a line of constant elevation. The tighter the lines are together, the more rapid the change in elevation (i.e. the steeper the slope) and vice versa. Attribution (e.g. slope, specific catchment area, curvature, etc.) of contour data is generally an order of magnitude higher data size than grid-based data. However, contours are an appropriate data structure when calculating water flow—lines drawn orthogonally to the contours represent flow lines across the surface (Moore et al. 1988).

Grid-based, or rasterized, data is the most commonly used DEM due to its simple data structure, lending itself to computationally efficient algorithms and attribution. In comparison to other data structures, grid-based data can become particularly inefficient in terms of data sizes due to the potential redundancy of information in low-relief landscapes. Additionally, grid-based data is unable to handle abrupt changes in topography such as cliffs or undercut embankments (Moore et al. 1991). Although several data structures for digital elevation models have just been described, the term *DEM* is frequently used to refer to grid-based elevation data and will be used this way in the remainder of the document.

2.1.2 Light Detection and Ranging (LiDAR) Data

Light detection and ranging (LiDAR) is a remote sensing technique used to collect surface data. In airborne LiDAR, pulses of infrared light are emitted from an aircraft toward the ground surface, and the reflected light return information is then recorded. Using high accuracy GPS and Inertial Measuring Units (IMUs) to triangulate the plane's position, the location of each surface point may be

determined given the angle of the pulse and the time elapsed between the pulse emission and return. Pulses are capable of producing as many as 5 returns should the pulse get partially intercepted by vegetation. Last returns are those of those of the lowest elevation, typically the ground surface. Such information is used to produce classifications for each point such as vegetation, water, ground, and buildings. The pulses are emitted at high frequencies (150,000 pulses per second), resulting in non-uniform 'point clouds' with an average point spacing between one to four meters. LiDAR-based DEMs are capable of providing high levels of surface detail including roads, ditches, streams, and buildings. According to the USGS, 38% of the lower 49 states have LiDAR coverage as of August 2013 (USGS, 2014).

However, several disadvantages accompany this increased topographic detail. Immediately apparent are the large data sizes associated with the amount of detail inherent in LiDAR-based DEMs (1.7 megabytes per square kilometer for 1.5 meter resolution data). Additionally, with LiDAR's high point densities, the signal to noise ratio in the data can become an issue, particularly in low-relief landscapes where the change in elevation between adjacent cells is below the vertical accuracy of the dataset which typically ranges from 15 to 30 centimeters (NOAA 2012). The added detail gained with high resolution data often leads to additional complications in many applications. For example, hydrologic models rely on terrain data to route water. If an anthropogenic structure such as a bridge is captured in a DEM because it is unable to detect the channel underneath, an algorithm that routes water will have difficulty negotiating this obstruction, leading

to improper flow routing, streams that flow in the wrong direction, etc. Bridges are typically removed in the standard hydro-flattening process used by USGS for all data it distributes (Heidemann, 2014), but additional hydro-conditioning of the DEM would be required to remove culverts.

2.2 Current Watershed Delineation Practice

Current watershed delineation practice as implemented by GIS software and tools utilize the steps described in this section.

2.2.1 Flow Direction

The single direction flow direction method was originally described in Marks et al. (1984) and O'Callaghan and Mark (1984). A 3 x 3 window traverses the DEM, and the eight neighbors of the center cell are examined to determine the direction to which the center cell will flow (one of eight possible directions, hence D8). The slope between the center cell and each of the neighbors is calculated, and the flow direction will be directed toward the neighbor with the steepest downslope. From the center cell, the neighbors in the four cardinal directions have a distance of one while those neighbors in the four diagonal directions will have a distance of $\sqrt{2}$, meaning that the flow direction must be calculated using slope rather than by taking the minimum elevation of the

32	64	128
16		1
8	4	2

Figure 1. D8 single direction flow direction definition. Each direction is defined as a unique power of two starting center right and increasing clockwise.

neighbors. The flow direction of a cell was encoded as a power of two beginning with the neighbor to the right and increasing clockwise (Figure 1). Despite a lack of precision, this method to derive flow direction is quite reliable for many uses, even using LiDAR data, according to Dhun (2011, pg 21). Flow directions crossing ridges will not be assigned, allowing major topographic features to persist in the derivative datasets, regardless of the specific flow direction method.

Special cases, such as multiple equivalent flow directions and flat areas (discussed later) were not initially addressed by O'Callaghan and Mark's (1984) flow direction method. They were later addressed by Jenson and Domingue (1988) and implemented by several other algorithms (Tarboton, 1997; Pan et al., 2011; Tribe 1992). When multiple neighbors are of equivalent greatest distance-weighted drop, a look-up table is used to resolves the flow direction. For example, when two neighbors are of equivalent, one is chosen arbitrarily, and when three adjacent neighbors are equivalent, the center cell is chosen (Jenson and Domingue, 1988).

2.2.2 Pit Filling

If no neighboring cells are found to be downslope when assigning flow directions, the cell is declared a pit cell (also commonly called a sink cell). O'Callaghan and Mark (1984) devised an approach that attempts to fill them as they would be filled by water. The minimum boundary elevation enclosing the depression is identified as the overflow point, and the flow is redirected out through this point. This is implemented by reversing the flow of the cells between the depression bottom and the overflow point in a linear path. While this may be

a way to quickly provide connectivity between all of the cells in a depression and the downstream areas, the routing of flow within the pit may not be reliable and it may struggle to handle complicated pit structures (Dhun, 2011). O'Callaghan and Mark (1984) note that their adjustments to drainage direction to resolve interior pits 'were not apparent in the results of the data sets,' and that 'more sophisticated techniques would attempt to modify the drainage directions of other points in the basin and perhaps treat the flooded area as a special feature.'

Jenson and Domingue (1988) revised how flow direction within a filled pit is resolved. First, all cells within the pit that are below the overflow elevation are raised to the overflow point's elevation. The cells that make up the flat raised area were encoded with a flow direction equal to the sum of those neighbors' flow directions which were of equal elevation (e.g. all eight neighbors having the same elevation would result in a value of 255). Then, the perimeter was examined to find the outlet point with the largest distance-weighted drop. Starting with this cell, adjacent cells of the flat area were resolved by directing them toward this outlet cell. Iteratively, unresolved flat cells were directed toward cells with known, valid flow directions until all flat cells were resolved. This approach attempts to respect the change in processes as flow would be routed more uniformly across the filled "puddle" areas. Each pit is iteratively resolved in this manner until all sink/pit cells are removed from the DEM. Due to its straightforwardness, this method of filling has seen widespread implementation (Jenson and Domingue, 1988; Tarboton, 1997; Martz and Garbrecht, 1999).

2.2.3 Flow Accumulation

Following the flow direction operation, the number of cells flowing through each cell, known as flow accumulation, is calculated. By iteratively looking at each cell and tracing the flow paths directed toward that cell, the number of cells, or contributing area, may be found for each cell (O'Callaghan and Mark, 1984). Band (1989) implemented this concept recursively. Specifically, each cell in the flow direction matrix is traversed, and each of the eight neighbors are checked to determine whether it points toward the center cell. If so, then the flow accumulation of the target cell is incremented and the function is recursively called upon that neighbor cell and checked for neighbors pointing to it in a similar fashion. During recursion, if a cell has no neighboring cells directed to it, the function returns, falling back to the downstream cell where it resumes looking for any unchecked neighbors.

To increase the efficiency of flow accumulation computations, the matrix may be updated while operating in these recursive calls, allowing for each cell to be visited only once (Tribe, 1992). In future accumulation inquiries, it may first be determined whether a cell value has already been calculated before the recursive function is called to ensure each cell is visited only once. This decreases the algorithm complexity, making for more time-efficient computations.

2.2.4 Flow Accumulation Thresholds and Stream Network Extraction

Mark (1984) proposed that stream networks identified by these algorithms should represent areas where flow is concentrated to the extent that fluvial processes dominate hillslope processes. The flow accumulation operation

attempts to represent this runoff flow concentration spatially. In other words, once flow accumulates beyond a specified threshold then the above-mentioned transition in geomorphological processes takes place (Mark, 1984). A spatial stream network dataset consists of those cells which exceed this threshold.

Speight (1968) first performed this by hand using a contour map. A square grid of points were placed over a contour map and a set of slope lines perpendicular to the contours were drawn downslope from each point on the grid to the edge of the map. Then, a set of line segments parallel to the contours were drawn on the square grid. If more than 100 slope lines crossed a contour segment then that point was considered on a 'water-course' (Speight, 1968; as obtained by Mark, 1984).

2.2.5 Stream Network Segmentation and Watershed Delineation

Once a stream network is established, stream links are defined as the unbranched segments between junctions in the stream. Interior links have a junction at each end and exterior links, or first order streams, occur where a stream is initiated at the upstream end and a junction is at the downstream end (O'Callaghan and Mark, 1984). A watershed may then be delineated as the area draining to a given stream link or set of adjacent stream links.

Tarboton and Rodrigues-Iturbe (1991) explored these thresholds more closely. They examined the relationship of scaling laws that reflected the above-mentioned transition between hillslope and channel erosion and transport mechanisms. According to Broscoe (1959), the average drop in elevation from end to end of a Strahler stream segment is roughly constant, regardless of the

stream order. By weighting cells that are upwardly concave and utilizing this 'constant drop law,' a threshold in weighted contributing area exists where the difference in mean stream drop between first and higher order streams is not significantly different. This threshold corresponds to the initiation of the stream network (Tarboton and Rodrigues-Iturbe, 1991).

2.3 Alternative Flow Direction Methods

The routing of water across a surface is a critical computation in providing a model that reliably represents natural phenomenon. In terms of watershed delineation, a reliable flow direction method may be backtracked from the outlet point of interest to determine the contributing area, essentially finding the ridges enclosing that outlet point. Flow direction methods may also play a role in compensating for the data structure of the terrain dataset. For example, in grid-based DEMs, the basic element is the grid cell, which have some inherent area, and naturally they are simplified into a set of square planes where elevations are known only at the center of each plane. This makes assigning flow directions across these planes—specifically ridge cells—a particularly ambiguous task. The following flow direction methods focus on routing water in grid-based elevation datasets.

2.3.1 Probability-Based and Multiple Outflow Methods

Single direction flow methods are challenged by divergent (convex) and convergent (concave) topography. On convex surfaces, grid bias in the eight major directions in the flow direction data results in the concentration of flow

when, in fact, flow should be dispersed more evenly across the surface (Costa-Cabral and Burges, 1994; Tarboton 1997; Pan et al., 2011).

Several solutions were devised to accommodate situations when multiple downslope neighbors are encountered. Fairfield and Leymarie's Rho8 model (1991) introduced a stochastic element by randomly assigning flow direction to one of the eligible neighbors with the probability of choosing that neighbor proportional to slope. Multiple outflow methods such as those developed by Quinn (1991) and Freeman (1991) proportioned flow to all of the eligible neighbors. While Quinn proportioned flow based on slope, Freeman proportioned flow based on slope raised to an exponent. Lea (1992) and Costa-Cabral and Burges (1994) fit local planes to each pixel and determined flow direction as the aspect of that plane expressed as an angle.

2.3.2 Plane-Fitting Methods

Lea (1992) routes flow as a ball released from the center of pixel, based on a plane created by averaging elevations at pixel corners from the adjacent cells. This method provides a more precise (0 - 2π) direction without dispersion. Costa-Cabral and Burges' (1994) DEMON fitted planes to the grid by taking pixel values as the corners of each plane, and routed water in two-dimensional flow strips originating over the entire pixel area rather than from the center of one pixel to another as in Lea's method. Costa-Cabral and Burges (1994) were concerned with the ability of flow direction methods to represent divergent and convergent flow as it relates to the calculation of specific contributing area (SCA)

and specific dispersal area (SDA) where understanding the area over which flow is spread is important.

Specific contributing area is defined as the total contributing area (TCA), A, also known as upslope area, divided by the length of the contour of which the upslope area was inquired. This produces an area per unit width of contour which has applications in geomorphology and hydrology such as hillslope hydrologic response, landslide risk, soil water content, vulnerability to pollution, and long-term basin evolution (Costa-Cabral and Burges, 1994). Similarly, specific dispersal area is related to the downslope area over which the contour may potentially disperse and may be used to determine areas prone to pollution given an upstream source or as an indicator of soil drainage rate (Costa-Cabral and Burges, 1994; Speight, 1975).

2.3.3 Facet-Based Methods

Tarboton's (1997) D-Infinity method operates by traversing the DEM with a 3 x 3 window in a manner similar to single direction methods. Eight triangular facets are developed between the center cell and pairs of adjacent neighbors as in Figure 2. The steepest downslope vector is then taken amongst the eight facets, producing an angle between 0 and 2 π (hence, an infinite number of potential directions). If this angle falls on a cardinal or diagonal direction then flow is apportioned to the appropriate neighbor. If the angle falls between two neighbors, flow is distributed between the neighbors given the proportion of the angles between each neighbor and the direction vector. Flats and pits are

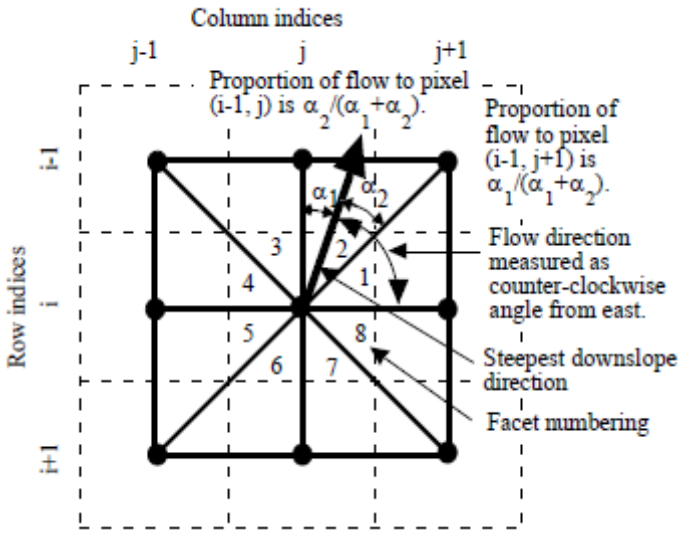


Figure 2. D-Infinity flow direction method. Flow direction is defined as the steepest downward slope on planar triangular facets on a block centered grid (Tarboton, 1997).

resolved using D8 methods to ensure that infinite loops in flow direction are removed (Tarboton, 1997).

Similar in the usage of triangular facets, Zhou et al. (2011) developed a flow routing method that attempts to model flow convergence and divergence appropriately. For each set of four adjacent points in a square orientation, there are two triangular facets formed in one of two possible configurations by either drawing a diagonal from one set of opposite corners or the other. The slope and aspect of each facet are calculated to determine a continuous flow direction from 0 to 2 π. By originating flow at the center of each facet, the flow from each facet is tracked across the facet network to create a network of flow lines. At this point, the amount of accumulated flow can be calculated for a line segment of interest as the number of flow lines crossing that line segment. As such, this method provides accumulation values that are more consistent with the SCA definition.

Dispersion is avoided by tracing the linear flow path from each facet to its ultimate destination (pit, outlet of interest, or DEM edge) rather than distributing flow to multiple neighboring cells. Additionally, by tracing the flow of each facet individually, the propagation of error in flow accumulation data from error-prone upstream cell accumulation values is also avoided.

2.3.4 Flat Areas

Flow direction methods are challenged by flat areas in the DEM because of their localized nature; if a valid downslope flow direction cannot be resolved from the eight adjacent neighbors, the cell is deemed a pit. A method to handle these flat areas becomes important because not only do they occur naturally in DEMs as a result of limited vertical precision or from landscapes with low relief, but also because the pit filling approach alleviates pits by raising groups of cells, creating large areas of constant elevation. Several efforts have been made to resolve flow direction across these flat areas.

Garbrecht and Martz (1997) developed a method in which a gradient is applied across flat areas. Flat cells adjacent to areas of higher terrain are raised infinitesimally such that a gradient is formed in a direction away from areas of higher elevation. Incrementally, the gradient is grown outward from higher terrain and toward one or more outlets of lower elevation.

Pan et al. (2011) proposed a similar method that utilizes linear interpolation across elevations within flat areas. Rather than raise elevations by arbitrary, infinitesimal amounts, elevations are raised by linearly interpolating between high terrain and low terrain outlet elevations along the perimeter of the

flat region, scaling elevations between these areas more naturally. This way, flow is routed in a direct line from each point within the flat region to the outlet where possible. Irregularly-shaped flat areas are navigated by iteratively resolving cells that have a direct route available, grouping any remaining unresolved cells, and similarly routing them in direct lines toward cells that have already been resolved.

2.3.5 Flow Direction Review

Tarboton (1997) offers a review of many of these flow direction methods as well as his own D-Infinity flow direction method. In devising his D-Infinity method, Tarboton (1997) targets five ways in which grid-based (DEM) flow direction methods may be evaluated:

1) The need to avoid or minimize dispersion
2) The need to avoid grid bias, due to orientation of the numerical grid
3) The precision with which flow directions are resolved
4) A simple and efficient grid based matrix storage structure.
5) Robustness. The ability to cope with “difficult” data such as saddle-shaped topographic features, but also including pits and flat areas.

It is noted that single flow direction methods (D8) handle points 1, 4, and 5, but lack precision in resolving flow direction, introducing grid-bias as discussed above. In regard to Fairfield and Leymarie’s (1991) probability-based method, algorithmically unpredictable or unrepeatable solutions are not desirable, specifically as it concerns deterministic values such as SCA. The Quinn (1991) and Freeman (1991) multiple outflow methods disperse flow significantly, to as

many as all eight neighboring pixels. Tarboton (1997) argues that although SCA may be used as a surrogate for a physical quantity that is affected by dispersion, it is inconsistent with the physical definition of upslope area and SCA. Dispersion should be minimized in the calculation of SCA and may then be accounted for independently. In addition, these multiple outflow methods require complicated data storage structure because flow to each neighbor must be stored (point 4). Finally, the local plane-fitting methods (Lea, 1992 and Costa-Cabral and Burges, 1994) result in surfaces that are not continuous from one pixel to another because the planes cannot be fitted to the four pixel corners. As a result, numerous special cases are required to handle complex topographic features (point 5).

2.4 Alternative Pit-Resolving Methods

As empirical datasets, most DEMs contain pit-like depressions that terminate flow paths because no downhill neighboring flow paths exist. If many pits exist, particularly in low-relief landscapes, then most flow direction approaches provide little useful information since the resulting flow information will be largely disconnected and segmented. As a result, several methods including the previously-mentioned pit-filling method were devised.

2.4.1 Binomial Smoothing

In the early stages of automated stream network delineation, Mark (1984) implemented binomial smoothing as developed by Tobler (1966) to remove pit cells from DEMs. Within the first or second pass of this smoothing operator, most if not all pit cells were eliminated from the DEMs. However, this smoothing

operation would alter not only the elevations of pit cells, but also the elevations of the remaining DEM including natural pits and stream valleys. It was observed that additional passes would begin to produce obstructions by smoothing over stream valleys, resulting in discontinuities in the stream network. For this reason, Collins' (1975) comments are appropriate regarding the preservation of DEM data: 'all available information about terrain elevation resides in the raw data of the DEM…it should not be diluted or falsified by any smoothing or averaging technique.' Such adjustments to the DEM have the potential to introduce significant error in flow direction (Zhou and Liu, 2004).

2.4.2 Breaching

As DEM resolutions improved to the point that small-scale obstacles such as roads and ditches became discernable, it was necessary to handle the man-made structures which route water through them. Martz (1998) identified elevated cells at the perimeter of depressions as obstructions to continuous flow. By lowering these few cells in the DEM at the depression edges (known as *breaching*), more appropriate flow paths could be computed.

While filling assumes depressions are artifacts created by underestimation of the surface when the data was collected, breaching assumes the opposite. Breaching recognizes that overestimation errors also exist as raised obstructions to flow and lowers these values in an effort to leave a smaller footprint on the DEM (Martz and Garbrecht, 1998). Martz and Garbrecht (1999) proposed a breaching algorithm which looks at all of those cells within the contributing area of a closed depression and determines if an adjacent cell exists which is outside

of the contributing area and below the “overflow” outlet elevation of that closed depression. As many as two cells may be lowered to this elevation (a distance of 60 meters given the 30-meter resolution DEMs used)—the cell outside of the closed depression perimeter, satisfying the above-mentioned conditions, and a second cell just inside of the depression. If the criteria for breaching were not met, the depression was filled instead. Breaching is particularly effective to resolve pits that are the result of roads that cross a stream or ditch. DEMs are only able to detect the top surface elevation of the bridge or road, resulting in obstructions to flow when in fact water should be able to pass underneath through bridges and culverts.

Lindsay and Creed (2005) developed a method that fills and breaches depressions such that the impact on the DEM is minimized. The number of modified cells (NMC) and the extent to which the elevations of cells are modified, called the mean absolute difference (MAD) are assessed for each depression via both breaching and filling, and the one which impacts the DEM least is chosen. Depressions which are tightly associated with one another (cascading depressions are the example given) must be considered together, and the least-cost method of either filling or breaching is performed on the group as a whole.

2.5 Hydrologic Connectivity and Appropriateness

Filling and breaching, together, are known as *hydrologic conditioning*; they are methods for altering a DEM or flow directions in order to make hydrologic analyses (e.g. stream network extraction, watershed delineation) more effective. In each method, pits are removed in order to guarantee complete *surface*

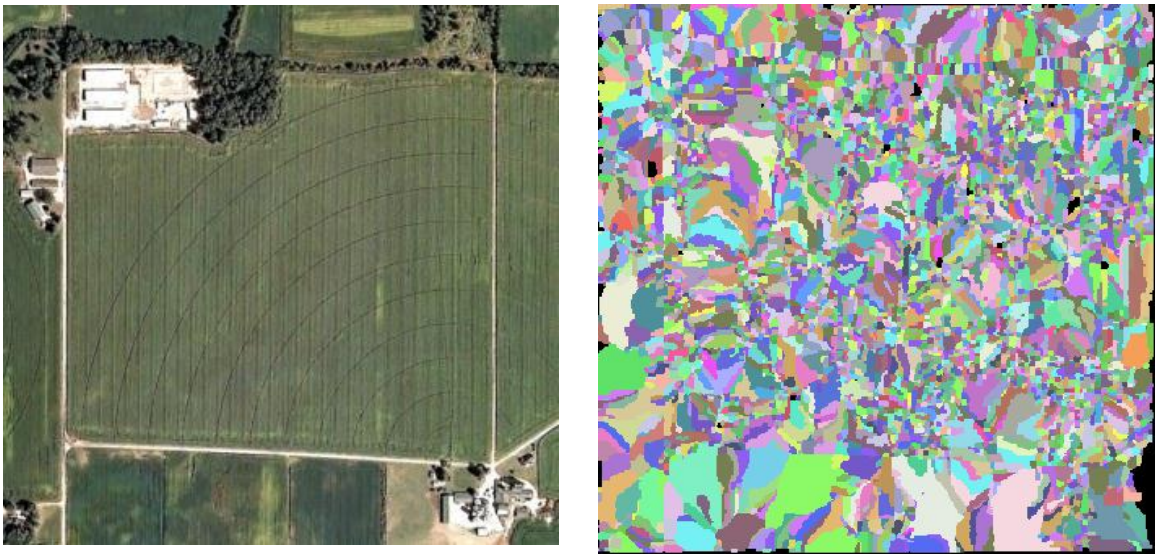
hydrologic connectivity: i.e. yield a depressionless DEM where each cell has an unbroken, monotonically descending path to the DEM edges. This requirement ends up modeling unrealistic scenarios. For example, some depressions would require a major flood in order to overflow, but these depressions are filled the same as small depressions which overflow after a relatively small rainfall event.

There is considerable debate on the appropriateness of requiring hydrologic connectivity given that surface depressions widely exist in nature and are readily apparent in higher resolution DEMs (Martz and DeJong, 1988; Tribe, 1992; MacMillan et al., 1993; McCormack et al., 1993; Burrough and McDonnell, 1998; Metcalf & Buttle, 1999). Lindsay and Creed (2006) summarize the reasons that most depressions were previously assumed to be artifactual and fit for removal:

1. Natural depressions with an extent equal to or greater than that of the DEM resolutions (30-meter or 90-meter resolution at that time) are generally non-existent with exception to some land features such as rock quarries (Lindsay and Creed 2006, Tribe 1992).
2. Only specific terrain types such as glacial, karst, and limestone are acknowledged as areas containing natural depressions (Lindsay & Creed, 2006; Tribe, 1992; Muehrcke & Muehrcke, 1998; Mark, 1988; O'Callaghan & Mark, 1984).
3. Finally, it could be assumed that these depressions have minimal impact on hydrogeomorphic processes as these depressions are likely to overflow and or take subsurface pathways closely approximated by surface topography (Lindsay & Creed, 2006).

Given new acquisition techniques and the resulting accessibility of high-resolution DEMs, these assumptions are no longer as reliable (Lindsay and Creed 2006). It may be argued that the significance of natural depressions may be greater and the occurrence more frequent than has been acknowledged in related literature. High-resolution DEMs are capable of resolving finer topographic details including natural depressions which before were too small for the course-resolution DEMs. Moore (1991) notes that the hydrologic significance of surface depressions may vary between terrains; the hydrologic response of some areas such the prairie pot-hole region in the Midwestern US are heavily influenced by surface depressions.

Additionally, at finer scales in flat areas, the landscape relief may be less than the vertical accuracy of the acquisition method, leading to a greater number of depressions in the data of these surfaces. Figure 3b shows the drainage patterns of the 100-hectare area shown in Figure 3a when flow direction is derived from a LiDAR-based DEM at 3-meter resolution before filling any depressions. Each uniquely colored polygon flows to a common destination and is a unique depression. Cells that are colored black are not part of a depression and are edge effects: if the dataset extent were slightly larger, they would also belong to a depression. According to MacMillan et al. (2003), the number of depressions increases exponentially with increasing DEM resolution. Because these depressions may become so numerous, it is no longer appropriate to assume all depressions are artifacts and that it is suitable to fill them indiscriminately. Moreover, because of the interdependence of small pits



A B

Figure 3. An agricultural field and surrounding area in Fulton County, Indiana, USA (approximately 100 hectares, 3 meter resolution): a) reference image, and b) connectivity map. Each polygon is a collection of cells that flow to a common location. Black polygons along the perimeter flow off the edge of the DEM.

(potentially noise in the data) nested within larger pits (likely natural depressions), filling such 'first-order' pits may not remove the larger second-order pits; these relationships must be understood.

Lindsay and Creed (2006) present methods to distinguish actual depressions from artifactual depressions in a DEM. By considering several approaches including ground inspection, modeling, classification systems, knowledge-based approaches, and examination of the source data, four methods were devised to determine artifactual pits from actual pits. Given that ground inspection should provide the most reliable results, the four methods developed are approaches that may be automated:

1) A discriminant analysis in which a classification model is calibrated given a representative DEM where natural and artifactual pits may be confirmed.

This method explicitly requires ground inspection on the representative DEM before it may be applied to other DEMs. The calibration is based on depression properties such as depth, volume, and location.

2) A heuristic rule whereby all depressions with 'minor extent' (less than two cells large) are removed.
3) A heuristic rule whereby all depressions with 'minor extent' and depth (less than 0.3 meters deep) are removed.
4) A Monte Carlo-based modeling approach, referred to as stochastic simulation modeling (SSM).

After finding the Gaussian error probability distribution function for the elevation data, values from this PDF may be applied at random to each cell in the DEM. By repeating this several times for the entire DEM, the probability of a depression being natural may be calculated as the number of outcomes where the depression existed divided by the total number of times the process was repeated. A threshold probability may be determined such that those depressions that have a probability below this threshold are removed Lindsay and Creed (2006).

2.6 Problems with Current Methods

The application of existing algorithms to LiDAR-based DEMs may produce several effects which would conflict with field observations and reality. The first conflict is that surface flow connections are observed in locations that rarely see surface flow, and this occurs at multiple locations as the result of several different reasons in this particular DEM. Occasionally, berms are established along

streams in order to prevent erosion and loss of nutrients directly into streams. By obstructing flow, these berms create areas that allow water to collect and puddle. Subsurface drainage is typically relied on to remove this excess water while an inlet may be used if standing water becomes an issue. Traditional algorithms do not have this information and instead enforce complete hydrologic connectivity by filling these depressions until they flow over the berms and into the streams. The downstream hydrologic responses resulting from the implementation of such drainage features will vary from those produced from actual overland flow.

In Figure 4, this phenomenon can be observed as connections B, and D are shown to flow directly into the ditch. In connections E, I, J, and K, water is shown to flow over road embankments, and the marked location G is a surface flow connection made after a natural depression was filled. Locations D, E, and G are drained by tile risers while connections B, I, J, and K are drained by culverts.

Another conflict observed was that streams were shown to be flowing in the opposite direction of reality. This is because bridges and other obstructions prevent water from flowing downhill. At location C, a section of the stream flows off of the northern edge of the DEM when, in reality, water travels in the opposite direction. Because the DEM data captured the elevation of the bridge at location A rather than the channel elevation below, this obstruction prevents flow to continue downstream (westward) through the ditch. Instead, a pit cell was located at the lowest channel elevation just before the raised obstruction, and the associated depression was eventually filled. Filling continued working upstream

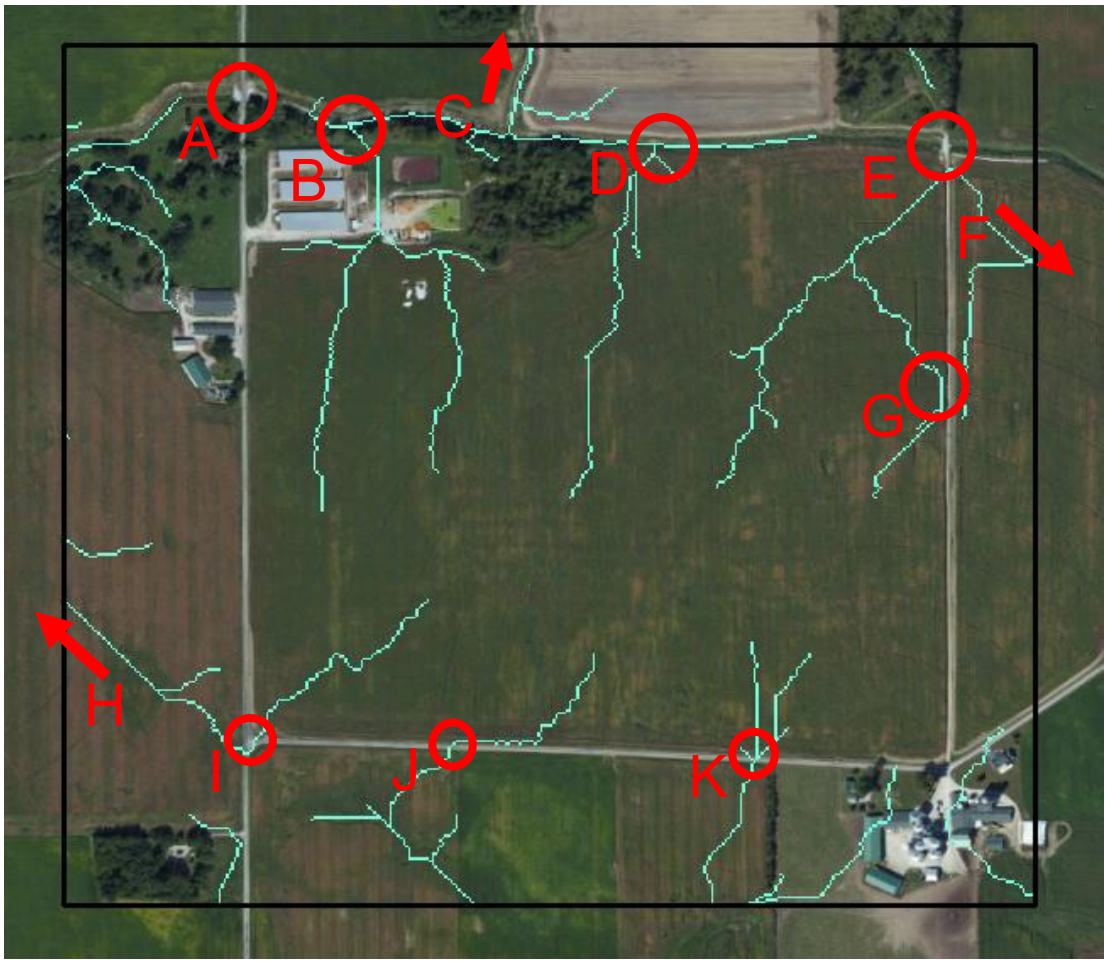


Figure 4. Aerial imagery and view showing ArcGIS flow accumulation and points where complete connectivity (pit filling) is not realistic at this 1 km^2 scale

through the ditch until the “path of least resistance” is found to be out the northern edge of the DEM as shown (and before a path was found over the berms or the bridge). By mapping the filled areas, it can be observed that filling took place all along the low-elevation stream channel cells until a path was made available which exited the DEM.

If the DEM were extended northward then the ‘path of least resistance’ would likely be over or around the bridge. Although they are not stream locations, this also occurs at the marked locations F and H. For location F, the raised berm

and lane elevations obstructed flow, and flow was rerouted back uphill to the east because this was the path of least resistance. This serves to emphasize two points: a) the extent of the DEM is an important consideration before watershed delineations take place and b) tile and culvert drainage connections affect flow patterns.

2.6.1 A Potential Solution: Sequential Depression-Filling Algorithms (SDFAs)

Depressions play an important role in the field hydrology as well as sediment and chemical movement. Surface flow is a primary means of transport for pesticides and phosphorus to surface water bodies and aquatic ecosystems (Blanchoud et al, 2007; Louchart et al., 2001; Probst, 1985; Turtola & Jaakkola, 1995; Simard et al, 2000; Heathwaite et al., 2005); typically, surface flow concentrations of phosphorus are ten times greater than groundwater or tile-drained effluent concentrations (Rozemeijer, 2010). However, surface depressions act as detention basins, providing short-term storage for water, sediment, and nutrients while allowing water to evaporate and infiltrate (Lindsay & Creed, 2006; Hubbard & Linder, 1986; Rosenberry & Winter, 1997; Hayashi & van der Kamp, 2000; Antonic et al., 2001).

Enforcing hydrologic connectivity (i.e. performing hydrologic conditioning), which assumes that all water flows across the surface of the ground rather than being held in surface depressions, could therefore result in very different estimates of the transport of pesticides, nutrients, and sediment when estimating water quality. The effects of hydrologic connectivity on infiltration, runoff generation, and evaporation have been studied recently with the use of

sequential depression-filling algorithms (Darboux et al., 2001; Chu et al., 2010; Chu et al., 2013; Antoine et al., 2009; Zinn and Harvey, 2003). By understanding the dimensions of each depression in the DEM, the capacity to retain water in these depressions may be modeled and the order in which these depressions overflow may be determined. In doing this, hydrologic responses may be understood as connectivity varies throughout the merging process.

The current sequential depression-filling algorithms focus on depressions at microtopographic scales. Darboux et al. (2001) analyzed the effects of soil surface roughness on hydrologic connectivity; the DEMs used were acquired using a laser profiler on 2.4 meter × 2.4 meter lab soil boxes. A model was developed to fill microtopographic depressions using a condition-walker method given randomly distributed water over the entire surface. Chu et al. (2010) developed a Windows-based software package that delineates depressions, their retention volumes, and their hierarchy given a filling-merging-spilling process. Higher-order depressions are created as a pair of lower-order depressions merge (i.e. when one depression overflows into a second). The study was performed on a DEM acquired from a lab soil box using a laser profiler (0.98-mm resolution) as well as on a 30-meter resolution watershed-scale DEM. Chu et al. (2013) further examines the hierarchy of these puddle units (i.e. depressions), specifically addressing the possibility of a high-order puddle splitting back into two lower-order puddles should water levels recede due to infiltration or evaporation as rainfall decreases.

Antoine et al. (2009) made use of three statistically generated surfaces (river, random, and crater) of varying hydrologic connectivity developed by Zinn and Harvey (2003) to study indicators of runoff connectivity properties. These connectivity indicators were divided in terms of *structural* variables such as elevation and soil properties that can be studied without specifying boundary conditions and *functional* variables that are process-based and reflect the propensity of the system to respond to a boundary stimulus (e.g. the ability for water to move). A filling algorithm similar to Darboux et al. (2001) was implemented to produce simplified hydrographs that show the effect of retention storage on runoff triggering. Appels et al. (2011) again made use of the statistically generated surfaces from Zinn and Harvey (2003) to understand hydrologic connectivity of microtopography using a ponding and redistribution model that also integrates Philip's infiltration model. The model performs a water balance for each depression at each time step in order to calculate water levels. Hydrographs similar to those of Antoine et al. (2009) were used, and a dimensionless analysis was performed to evaluate the development of surface runoff in relation to fields of varying size and statistical structure.

Currently, watershed delineation algorithms assume complete hydrologic connectivity. The effects of topography are neglected in that ponding in surface depressions is unaccounted for. All pits are filled to provide a monotonically descending path from each cell to the edges of the area of interest, resulting in surface flow connections that may be rarely observed in the field; the conflicts typically occur because some depressions have sufficient retention capacity that

they require a substantial amount of rainfall to overflow and because there are man-made connections such as tiles or culverts that are not represented in the digital elevation data. The development of sequential depression filling algorithms have enabled studies of hydrologic connectivity as it pertains to field-scale depressions and the resulting impact on watershed delineations. Depressions within field-scale topography may be quantified, the fill-spill-merge hierarchy of these depressions may be developed, an appropriate extent of connectivity may be determined, and watersheds may be delineated under a given state of connectivity.

CHAPTER 3. METHODS

A sequential depression-filling algorithm (SDFA) was developed to handle field-scale topography (i.e. natural depressions) and to allow for the adjustment of the extent of hydrologic surface connectivity to match field observations. The algorithm can account for drainage features such as tile inlets and culverts. Because it is aimed at field-scale delineations while on location, optimizations were made to facilitate implementation of the algorithm on mobile devices in resource-constrained environments.

To evaluate the algorithm, watersheds were delineated and compared to traditional algorithms which fill all depressions in order to determine whether ridges are located properly and the correct spatial area is shown to drain to the desired outlet points. A suitability study was also performed on the algorithm in order to determine the implications of assuming all depressions should be filled versus filling depressions sequentially across various landscapes. For several DEMs throughout Indiana, USA, the variation in hydrologic connectivity was plotted and compared to the levels at which complete depression filling occurs, and it will be determined whether this difference is significant.

3.1 Algorithm Description

3.1.1 Flow Routing

Given the initial landscape DEM, a system for routing flow must first be devised. Single direction (D8) flow routing, as described in O'Callaghan and Mark (1984), was selected due to its simplicity and robustness. For each cell, the neighboring eight cells are searched, to determine the maximum distance-weighted drop. Cells which have a maximum distance-weighted drop less than or equal to zero are denoted as pit cells and assigned a flow direction value of -1. If multiple neighbors have the same maximum distance-weighted drop, the first one encountered is taken. Neighbors are traversed beginning with the first column (far left) and looping through all rows (top to bottom) before moving onto the next column. In this way, flow is routed across the landscape, downhill from one grid cell to another until it either runs off of the edge of the DEM or it reaches a local minima, or pit cell, where no further downhill route is available. At these minima locations, water must accumulate and overflow before it may continue to flow further downstream.

However, an alternative approach was taken in the way the method was implemented. A tree structure was developed by representing flow direction as the index of the cell to which it is directed (i.e. its 'child'). The cells pointing into the current cell may also be associated as another attribute (i.e. a list of 'parents'), making the tree structure doubly-linked. Figure 5 demonstrates this implementation. By associating the indices of parents and children directly to each cell, the speed at which flow paths are traversed may be increased when

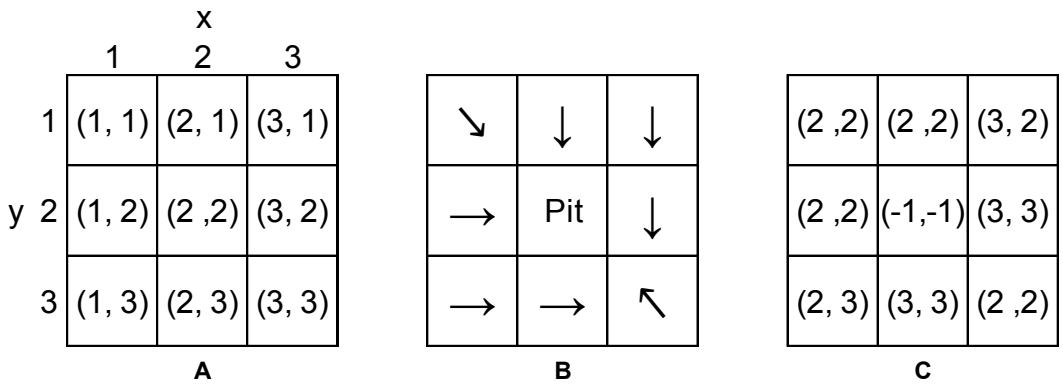


Figure 5. An example of flow direction as implemented in the algorithm developed. A) Matrix indexing definition. B) Flow direction indicated with arrows. C) Flow direction expressed as destination cell index. For example, the flow direction of cell (1, 1) is expressed as (2, 2), the cell to which it is directed according to (B). Cell (2, 2) is a pit cell, has no flow direction, and is assigned a value of (-1, -1).

compared to methods where flow direction must be translated from a power of two to a particular neighbor. Given the tree structure outlined, flow paths are simple to traverse iteratively to perform computations such as contributing area (Figure 6). For the purpose of determining the cells that drain to a pit cell, this becomes advantageous because a WHILE loop may be utilized instead of recursion; the order in which the cells are traversed is not important as long as

```
Add original outlet cell to the list of cells to check for parents
WHILE the list of cells to check is not empty
        FOR each parent of the first cell in list of cells to check
                Increment contributing area
                Add parent cell to list of cells to check
        ENDFOR
        Remove first cell in list of cells to check
ENDWHILE
```

Figure 6. Pseudocode for finding contributing area.

each parent is traversed and subsequent additional parents of that cell are added to the list of connected cells. While this doubly-linked tree structure can multiply the memory usage, the cost of the increase can be mitigated through caching of large flash storage that is generally available on mobile devices.

3.1.2 Excess Rainfall

Abstractions such as the infiltration of water into the soil are a critical component that affects how well the model represents reality. Vegetative interception, root uptake, infiltration, and evapotranspiration take place, reducing the *rainfall excess*, or *effective rainfall*, which is runoff-ready and may be simulated in the depression-filling model. Without these abstractions, the ground is modeled as an impervious surface where all rainfall is applied to the depression-filling process, resulting in an overestimation of hydrologic connectivity. Without abstractions, the algorithm would represent conditions when the soil is frozen, saturated, compacted, or when the rainfall rate is much greater than the infiltration rate, but, in most scenarios, losses should be considered before runoff begins.

The SCS Curve Number Method was utilized to maintain a computationally efficient algorithm while adjusting the simulated rainfall to account for infiltration and other abstractions. It is a runoff equation that makes use of an empirical *curve number* which represents the runoff potential of various surfaces. Given the input rainfall event and the land characteristics, rainfall excess may be calculated, providing a means to scale rainfall events simulated in

the model to more closely match field observations. Curve numbers are published in the SCS TR-55 Manual for various land use, condition, and soil type combinations (USDA 1986). Equation 1 may be used to calculate the potential maximum retention after runoff begins:

$$S = \frac{25400}{CN} - 254 \quad (1)$$

Where:

S = maximum soil water retention, mm

CN = runoff curve number

Knowing this and the input precipitation amount, Equation 2 may be used to calculate rainfall excess:

$$Q = \frac{(P-0.2S)^2}{P+0.8S} \quad (2)$$

Where:

Q = direct runoff, effective rainfall, or rainfall excess, mm

P = precipitation, mm

A curve number must be selected which best represents the overall conditions of a given DEM. Spatial land use and soil type information may be used to calculate an area weighted average curve number, but accounting for infiltration on a cell-by-cell basis would put the computational complexity beyond the scope of a watershed delineation algorithm. The calculated excess rainfall is

then applied to the SDFA, which accounts for topographical retention (as compared to microtopographical retention at the soil surface).

3.1.3 Drainage Features

Sub-surface drainage tiles are a common practice across the United States Midwest to increase agricultural trafficability and yields. While these tiles are typically laid horizontally approximately one meter below the ground surface, a *tile inlet* may be run up to the ground surface to drain puddles and concentrated surface flows. Oftentimes, natural depressions in the Midwest are tile drained because they are so prone to puddle following rainfall. Once tile drained, a depression is much less likely to overflow and drain over land into adjacent areas during typical rainfall.

The algorithm will handle drainage features such as agricultural drainage tile risers and road culverts that effectively reduce the rate at which a depression fills. If the algorithm did not account for drainage features, depressions may overflow and surface flow may occur in locations where that would only occur in the most extreme cases. Drainage rate is controlled by the pipe diameter and slope (as opposed to an absolute volume value), necessitating a time-based SDFA in order to properly compute accumulation rates for each depression. For example, if a given amount of rain falls over the course of several days (low intensity), it is less likely to exceed the drainage capacity of a tile drain than the same amount of rainfall spread over 10 minutes (high intensity). As a result, in the former case, the depression is less likely to overflow while in the latter case it is more likely to accumulate rainfall in excess of the drainage rate, fill the

depression (assuming the riser is located in a depression), overflow, and contribute surface flow downstream.

In terms of the algorithm developed, drainage features are implemented by assigning a drainage rate on a cell-by-cell basis. Through user input (because this is intended for field workers), drainage features and drainage rates may be specified. During the assignment of flow directions, the incoming rainfall rate (intensity) must exceed the drainage rate on a cell-by-cell basis before a flow direction is assigned. If the rainfall rate does not exceed the drainage rate, the cell is considered a pit cell in the flow direction matrix.

3.1.4 Sequential Depression Filling

A sequential depression filling algorithm is key to simulating physical processes and enabling the visualized output to match real-world situations with varied rainfall events. Specifically, as shown in Figure 7, a single pit cell and all the cells which flow to it will be referred to as a *depression*. Each depression is given a unique, positive identification number and all cells that are part of that depression are marked in a depression identification matrix.

For each depression, there is a minimum ridge elevation along the perimeter whereby if the pit were filled with water, it would begin to spill over at this *spillover elevation*. After finding the spillover elevation, the volume of a depression may be determined. For each cell in the depression with an elevation below this elevation, the volume may be computed as the difference between the spillover elevation and the DEM surface elevation of that cell multiplied by the

grid cell area. Equation 3 is used to calculate the volume of depressions which have not been filled, or *first-order depressions*:

$$V_d^1 = \sum_{i=1}^{n}(E_s - E_i) \times A_i \quad (3)$$

Where:

V_d^1 = total volume of first-order depression, m^3

E_s = spillover elevation, m

E_i = cell elevation

A_i = cell area, m^2

n = number of cells identified within the depression

The retention volume of the depression is important in determining how much water it may take before a depression overflows, but equally important is the contributing area. Because the accumulation of water in a depression is

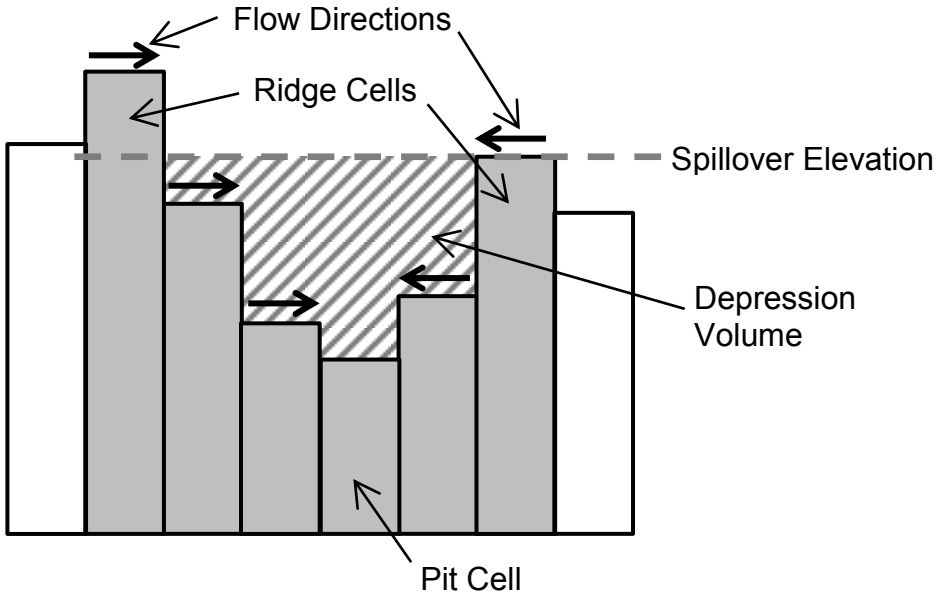


Figure 7. A 2-D depression annotated to illustrate terminology.

dependent on rainfall, the contributing area determines the area over which water is funneled into the depression. For example, given two depressions with the same volume, the one with a greater contributing area will have a larger area over which to funnel water into the depression, and so it will overflow more rapidly. Furthermore, the volume to contributing area ratio may be used to quantify the relative rates at which a depression will overflow. In fact, this value is equivalent to the excess rainfall amount (depth) necessary to fill it. Equation 4 calculates this value:

$$P = \frac{V_d}{A_c} \tag{4}$$

Where:

P = rainfall amount necessary to fill a depression, or volume to contributing area ratio, m

V_d = Depression volume, m^3

A_c= depression contributing area m^2

This description of relative overflow sequence is satisfactory in the absence of drainage features. However, in the presence of drainage features, the rate at which water accumulates in a depression is no longer purely a function of contributing area. Drainage features act to reduce the rate of water accumulation in a depression, necessitating a time-based computation to determine when each depression will overflow relative to each other. The rate of accumulation for a given depression may be computed using Equation 5:

$$R_a = \sum_{i=1}^{n}(i_e - R_{d,i}) \times A_i \quad (5)$$

Where:

R_a = Rate of accumulation of the depression, $\frac{m^3}{h}$

i_e = rainfall excess intensity, $\frac{m}{h}$

$R_{d,i}$ = drainage rate of cell i, $\frac{m}{h}$

The rainfall excess intensity may be computed using Equation 6:

$$i_e = \frac{P_e}{T_d} \quad (6)$$

Where:

P_e = rainfall excess amount, m

T_d = rainfall duration, h

The length of time it will take for a first-order depression to fill can be computed using Equation 7:

$$T_o^1 = \frac{V_d^1}{R_a} \quad (7)$$

Where:

T_o^1 = Time to overflow of a first-order depression, h

And T_o^1 = ∞ if $R_a \leq 0$

By definition, cells with drainage rates that exceed the excess rainfall intensity may only occur at the depression bottom (i.e. pit cell). However, if the pit cell's

drainage rate is excessive, this may result in a negative overall depression accumulation rate. Consequently, the time to overflow will be a negative value. When filling begins, the depressions that require the least amount of time to fill will overflow first; depressions with a negative overflow time will erroneously be filled first in the filling sequence. To avoid this, any depressions with a negative drainage rate are assigned an infinite overflow time. These depressions are incapable of overflowing, but other depressions may overflow into them. They will behave as an infinitely deep sink until sufficient cells begin to flow into the depression causing the rate of accumulation to become a positive value. Depressions which require more time than the storm duration will not overflow; they will remain their own hydrologically common *subcatchment*, and do not flow and contribute to downstream watersheds.

3.1.5 Handling Edge Effects

Because cells along the edge of the DEM lack all eight neighboring cells to determine a valid flow direction, no flow direction is assigned. However, it is possible that other cells within the area of interest may flow toward one of these boundary cells. To denote hydrologically common areas that have begun to flow off of the edge of the area of interest, each cell along the DEM perimeter is identified as a unique pit cell and any cells initially flowing into these edge pit cells are considered a hydrologically common *edge depression*. They are each given a unique negative ID number to distinguish them in the pit identification matrix.

Depressions identified along the perimeter have no retention capacity, they are not involved in the filling sequence, and it is unnecessary to store many of the parameters for these depressions. *Internal depressions* that must be filled and are not connected to the DEM edges may overflow into these edge depressions, whereby the aggregate merged depression would no longer be "live," is furthermore identified as an edge depression, and given a negative ID number accordingly.

3.1.6 The Merging/Filling Process

Once the parameters for each depression have been computed, the pits are sorted in ascending order based on the time required to overflow each of them. At this point, the set of depressions are prepared to be filled in sequence. The depression filling "rainfall simulation" starts by overflowing the first depression in the list, and continues in order until either all depressions are filled or until the rainfall duration expires and any depressions that are still on the list remain unfilled.

When a depression overflows, it will either overflow into another depression or it will begin to flow off of the edges of the area of interest. If it overflows into another depression, the new spillover location and time to overflow are calculated for the aggregate depression. The DEM is adjusted, raising those cells of the depression that are below the overflow elevation up to the overflow elevation (i.e. fill the depression). Flow direction within this depression is then resolved using an iterative process similar to the Jenson and Domingue (1988) method described in Section 2.1. Beginning with the outlet cell, neighboring cells

that are part of the flat region are iteratively resolved by directing them back toward cells with known, valid flow directions. Figure 8 presents pseudocode for this process.

In this way, the SDFA is able to keep track of each change in the state of hydrologic connectivity by performing computations only on those depressions involved in the overflow event. The loop runs once per depression rather than using any particular time step. Depression relationships and hierarchy are *causal* in that relationships between low-order depressions and high-order depressions are realized only after each spillover event. When a depression overflows, it affects one other depression (the one into which it overflows), and only at that moment are the higher-order, aggregate depression parameters computed. Because of this, the hierarchy of multiple depressions cannot be determined ahead of time, but only one depression is handled at a time.

Because of the possible orientation of two merging depressions and

```
Add outlet cell to the list of cells to check for flat neighbors
WHILE list of cells to check for flat neighbors is not empty
        FOR each neighbor of the first cell in the list of cells to check
                IF neighbor is part of the flat area and hasn't already been resolved
                        Direct that neighbor cell back to the current cell
                        Add neighbor cell to list of cells to check
                ENDIF
        ENDFOR
        Remove first cell in list of cells to check
ENDWHILE
```

Figure 8. Pseudocode for resolving flow direction of cells that have been filled.

because the elevations of some cells are raised in the filling process, Equation 3 may not be used to calculate the total depression volume of depressions which are comprised of one or more depressions that have been filled. To illustrate this, Figure 9 depicts two possible orientations of a pair of depressions as they merge.

In the first scenario (Figure 9a), two first-order depressions are initially defined as spilling over into one another. After one overflows into the other (based on time to overflow), the new, combined depression's spillover elevation will be different than the spillover elevations of either of the two lower-order depressions. Equation 3 will not account for the previously filled volume due to the raised elevations of filled depressions. A revised equation is necessary and will be presented below.

In the second scenario (Figure 9b), one of the lower-order depressions fills, cascading down into the other. The application of Equation 3 to this scenario would result in a miscalculation of the total volume of the merged depression because entire depression that was filled exists above the aggregate depression's new spillover elevation.

Given these examples, it is necessary to record the volumes of any lower-order depressions that have previously been filled for each depression in the database. As any two depressions merge, the total volume of the depression that is being filled is summed into the previously filled volume of the depression with which it merged. The previously filled volume of the merged, higher-order pit may be calculated using Equation 8:

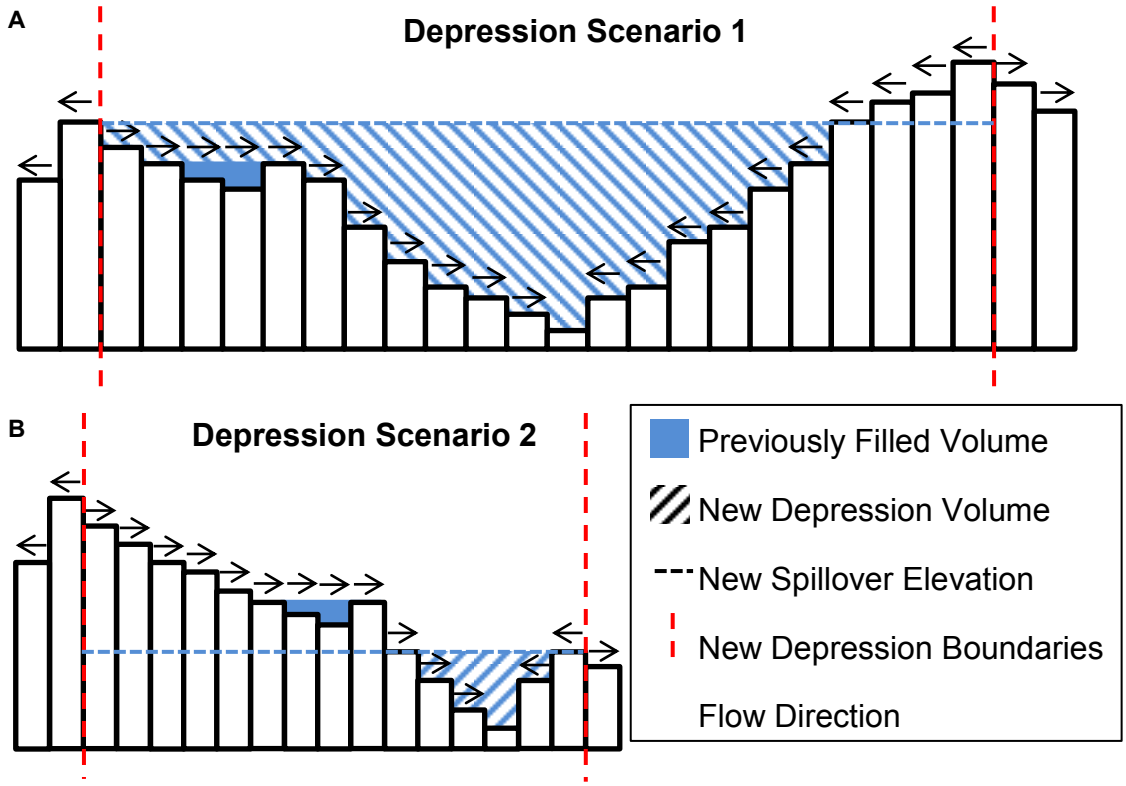


Figure 9. Comparison of depression scenarios: A) the total volume of the new depression is the sum of the differences between cell elevations and the new spillover elevation. B) the previously filled depression retains water above the spillover elevation which necessitates recording any previously filled volumes as any two depressions merge.

$$V_{pf}^{h} = V_{d,1} + V_{pf,2} \quad (8)$$

Where:

V_{pf}^{h} = previously filled volume of higher-order depression, m^3

$V_{d,1}$ = total volume of the depression that overflows, m^3

V_{pf2} = previously filled volume of the depression that is overflowed into, m^3

The total volume of higher-order depressions may now be calculated using Equation 9:

$$V_d^h = \sum_{i=1}^{n}(E_s - E_i) \times A_i + V_{pf}^h \quad (9)$$

Where:

V_d^h = total volume of higher-order (non-first-order) depressions, m^3

Accordingly, the overflow time of higher-order depression may be computed with Equation 10:

$$T_o^h = \frac{V_d^h}{R_a} \quad (10)$$

Where:

T_o^h = Time to overflow of a higher-order (non-first-order) depression, h

And T_o^h = ∞ if $R_a \leq 0$

By precomputing the time to overflow each depression and sorting the overflow times, algorithmic complexity is improved. The algorithm does not waste time looping through time steps where no spillover events occur. The computationally intensive maintenance of dynamic water levels in each pit during simulation is avoided. After one depressions overflows into another, this relationship is realized and, thereafter, the aggregate depression is considered a single unit and the time to overflow this aggregate is computed. This new depression is then placed back into the sorted list of depressions based on the overflow time. Since the list is already sorted, inserting the new merged pit into the list is algorithmically more efficient. This makes the algorithmic complexity

dependent solely on the number of pits that need to be filled rather than the number of time steps in the simulation.

3.1.7 Identifying Proper Spillover Locations

Because flow direction is defined as the direction of steepest descent, the maximum elevation cell at a ridge peak will belong to only one of the two adjacent depressions that meet at that ridge feature. As such, when inspecting the ridges of a given depression to find the spillover location, a cell of greater elevation may lay one cell beyond the edge cells of the depression. Looking at Figure 10, the minimum boundary elevation of Depression 2's ridge cells is at Elevation B. However, it is apparent that Depression 2 will not overflow until filled to Elevation A. This distinction is important with respect to the calculation of each depression's retention volume, time to fill, and, consequently, ordinal position in the filling sequence. It is therefore important when gathering

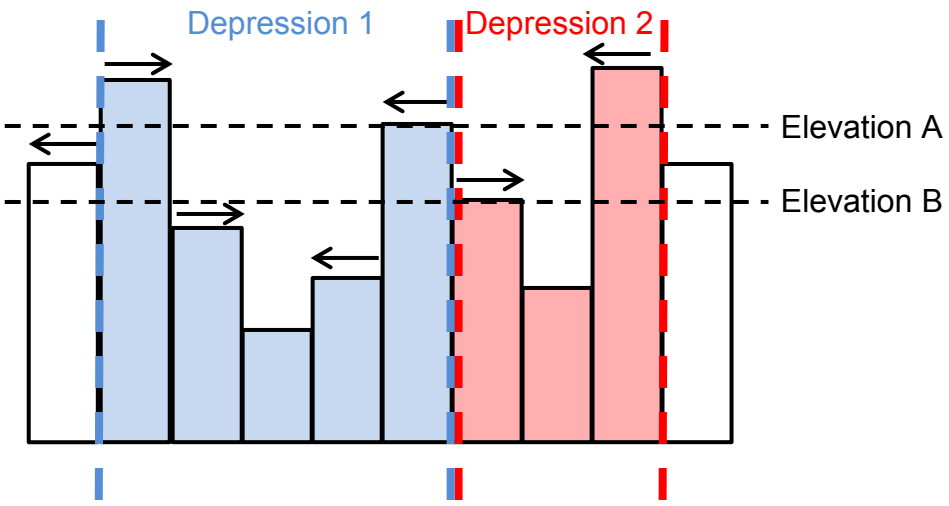


Figure 10. This figure illustrates the importance of inspecting one cell beyond the pit boundaries to find the true minimum spillover elevation: the minimum boundary elevation for Depression 2 is at Elevation B, but it will not overflow until filled to Elevation A.

information about each depression to inspect one cell beyond the depression boundaries. Treating cells in this manner also properly handles single-cell depressions whereby no adjacent cells flow into the pit cell.

3.2 Algorithm Example

Figure 11 shows a small (5 x 8 cell) contrived DEM containing two internal depressions. Each step in the filling process is displayed as a column of datasets including the DEM, flow direction, and pit identification matrices corresponding to that stage in the filling process. Internal depressions are color coded for convenience. Notice, the initial depressions are defined by the flow direction of the raw DEM. Pit cells are then identified and the set of cells contributing to each pit cell makes up each unique depression. Each cell along the border is given a unique, negative ID number and no flow direction is defined.

Given one centimeter of rainfall excess over a duration of one hour, a rainfall intensity of 1 $\frac{cm}{h}$ is computed using Equation 6. Using this rainfall event and assuming a one meter DEM resolution, the calculated values of volume, contributing area, and rainfall excess to fill each depression are listed in Table 1. For the first order depressions (Depressions 1 and 2), volume is calculated using

Table 1. Pit parameters for each internal depression identified in Figure 11. This assumes a 1 cm, 1-hour rainfall event (1 cm/h rainfall excess intensity).

Pit ID	Volume (m^3)	Area (m^2)	Rainfall to Fill (mm)
1	2	6	333
2	6	12	500
3	8	18	444

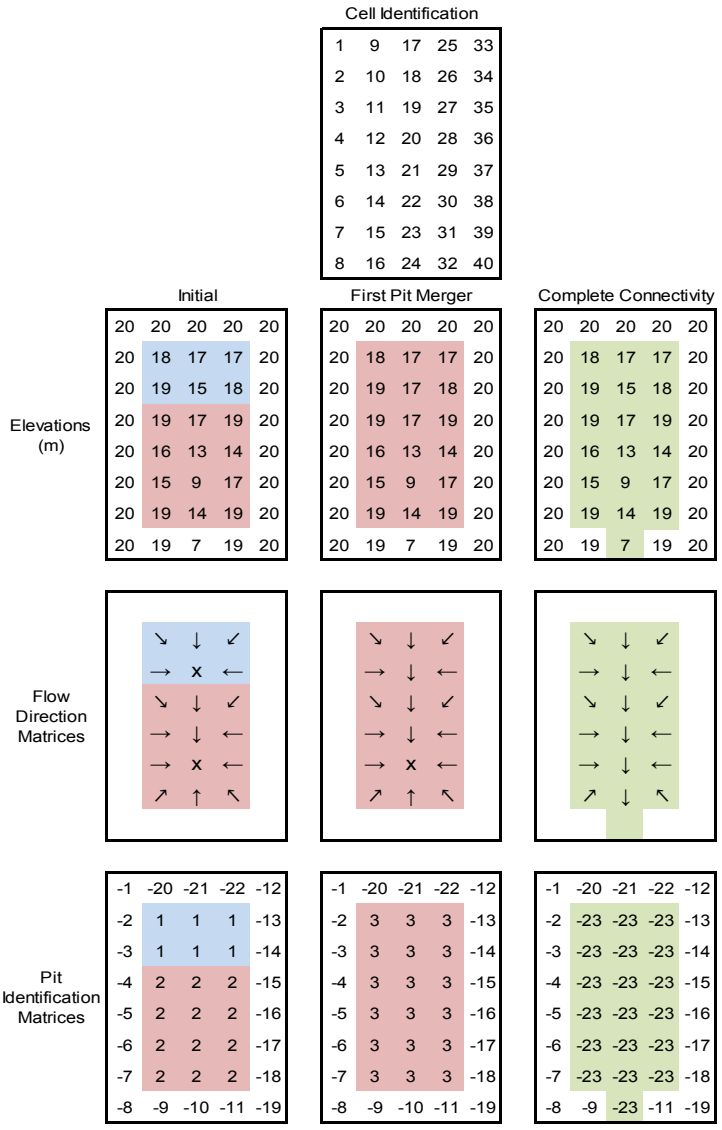


Figure 11. An Illustrative Example of Sequential Depression Filling. From top to bottom: cell identification by numbering, the DEM (meters), flow direction matrix, and pit identification matrix. From left to right: two internal depressions are initially identified, the two internal depressions merge into a single internal depression, and the merged internal depression begins to run off of the DEM after merging with a border depression.

Equation 3, area is calculated as the number of cells that make up that depression, and the rainfall excess amount to fill that depression is calculated using Equation 4.

Given this initial configuration, Depression 1 will overflow before Depression 2. Depression 1 will overflow into Depression 2, forming Depression 3. The elevations of Depression 1 are raised to simulate filling at the time of this merger. As a result, Equation 8 must be used to keep track of this filled volume, while Equation 9 is then used to sum it with the unfilled portion of Depression 3. Finally, the overflow time of this aggregate depression is calculated using Equation 10. Notice that parameters are not maintained for negative

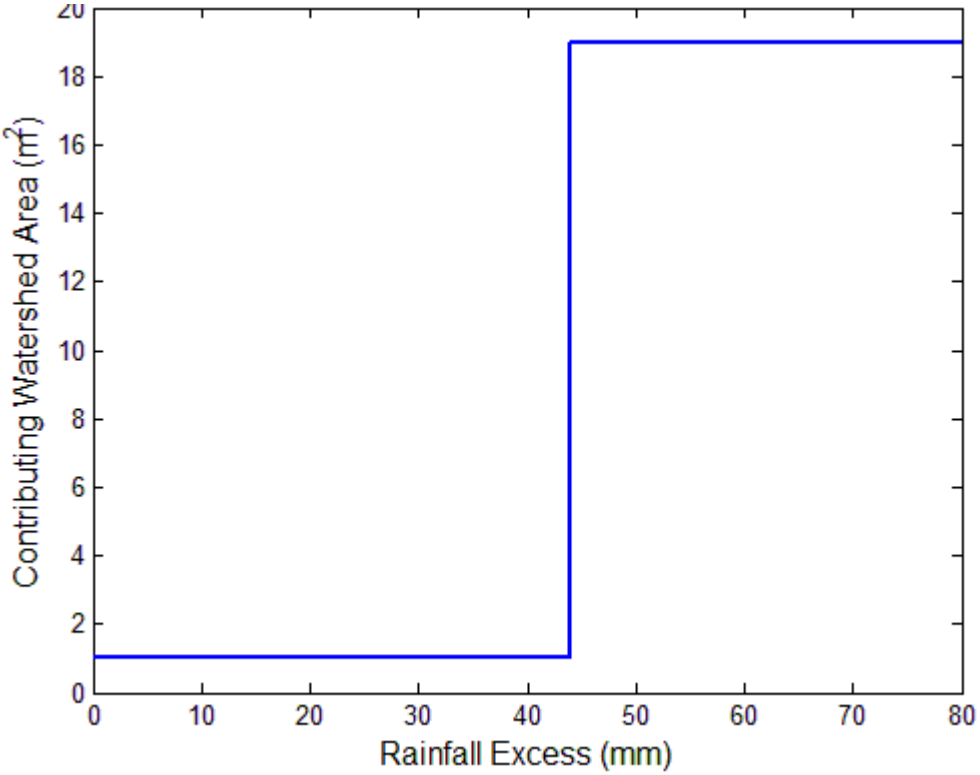


Figure 5. Watershed contributing area corresponding to a watershed delineation performed at cell 24 of Figure 11. Initially, only one cell (itself) drains to this location, but once the depressions overflow, a jump in contributing area occurs.

depressions (i.e. those that flow off of the edges). Figure 12 shows how these depressions can affect a watershed delineation; the contributing area of cell 24 (of Figure 10) versus rainfall excess will yield a jump in contributing area as the spillover event occurs and Depression 3 begins to flow through this downstream location.

3.3 Validation and Analysis

For those steps of the algorithm that were an implementation of existing methods (i.e. flow direction, filling and flow rerouting procedures), a set of validation procedures were developed to ensure the algorithm functions as intended. The primary distinction between the algorithm developed and existing methods is that all depressions are not required to be filled. However, it must be verified that flow between cells is routed properly, ridges are recognized, and the contributing area resulting from watershed delineations be evaluated. Several points were delineated after filling all depressions using the SDFA and the resulting watersheds were compared with watersheds generated by processing the same DEM and performing delineations at the same set of locations using the ArcGIS Hydrology toolset (ESRI, 2013).

The resulting watersheds were then compared on a cell-by-cell basis and the percent error was computed using Equation 11:

$$\epsilon = \frac{\sum_{i=1}^{n} A_i \times |\delta_{i,A} - \delta_{i,S}|}{A_i \times \delta_{i,A}} \times 100 \quad (11)$$

Where:

ϵ = Percent Error (%)

$\delta_{i,A}$ = 1 if cell i is in the ArcGIS Hydrology toolset watershed, otherwise 0

$\delta_{i,S}$ = 1 if cell i is in the SDFA watershed, otherwise 0

As shown in Figure 13, error may occur because a) cells are missing from the SDFA watershed that are present in the ArcGIS watershed and b) cells exist which are part of the ArcGIS watershed that are not part of the SDFA watershed. The relative contributions of these two components to the error term may be described by rewriting Equation 11 as Equation 12:

$$\epsilon = \frac{(\sum_{i=1}^{n} A_i \times \delta_{i,S} \times |\delta_{i,A} - \delta_{i,S}|) + (\sum_{i=1}^{n} A_i \times \delta_{i,A} \times |\delta_{i,A} - \delta_{i,S}|)}{A_i \times \delta_{i,A}} \times 100 \quad (12)$$

Where:

Error A = $\sum_{i=1}^{n} A_i \times \delta_{i,S} \times |\delta_{i,A} - \delta_{i,S}|$ = number of SDFA cells outside of the ArcGIS watershed

Error B = z$\sum_{i=1}^{n} A_i \times \delta_{i,A} \times |\delta_{i,A} - \delta_{i,S}|$ = number of SDFA cells missing from the ArcGIS watershed

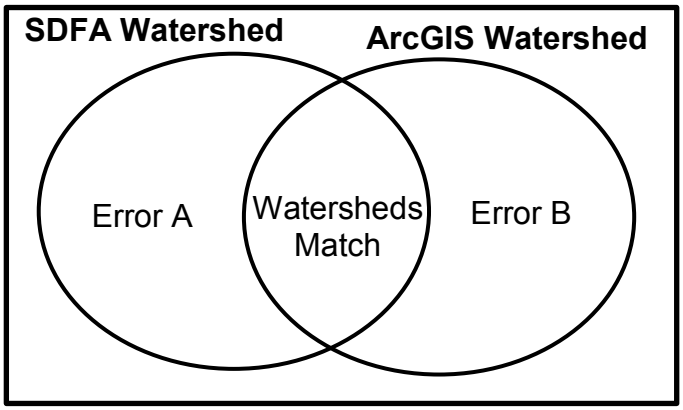


Figure 63. Illustration of two error types in watershed comparisons.

In ArcMap, watersheds were created by starting with a newly created point-type feature class which may be edited to create the desired watershed outlet locations. This layer was then converted to a raster by using the Feature to Raster tool. Then, the DEM was processed by running the Fill (Tarboton 1991), Flow Direction (Greenlee 1987, Jenson and Domingue 1988), Flow Accumulation (Jenson and Domingue 1988, Tarboton 1991), and Watershed tools in that order. The same "pour point" watershed outlets were then imported into MATLAB (The Mathworks, Inc. 2013) before watersheds were delineated using the SDFA. DEMs were acquired from OpenTopography for various locations at a 3-meter DEM resolution (OpenTopography 2012); this source provides on-the-fly gridding of Indiana's LiDAR point data into DEMs (IndianaMap 2013). For the gridding process, inverse distance weighting with a search radius of 3 meters was used while cells lacking LiDAR point data were filled with a given "Null Filling" window size parameter of 7.

3.3.1 Sequential Depression Filling Applicability Study

While the development of the SDFA was initially motivated by the presence of observable surface depressions and inconsistencies in the outputs of existing algorithms as described in Section 2.6, it remains necessary to understand how surface flow connectivity varies in different landscapes to determine the applicability and necessity of the SDFA developed over existing methods. Those landscapes that have many depressional features and experience a greater variation in connectivity may necessitate usage of the SDFA to provide the appropriate extent of connectivity before delineations take

place. For landscapes that lack natural depressions, filling all depressions may be more fitting.

The area draining off of the edge of the DEM may be analyzed as an indicator of hydrologic connectivity and will also enable comparisons of the SDFA to current methods. By definition, hydrologic connectivity increases as any depression overflows and begins to contribute elsewhere (whether they begin to flow off of the DEM or not); however, this analysis observes only those depressions that overflow and begin to drain off of the edge of the DEM because that is the goal sought by the current methods and so this methodology would enable comparison to that standard.

Plotting the area draining off of the edge of the DEM versus rainfall excess produces stepwise function that shows a step up each time a depression overflows and begins to run off of the area of interest. Such an analysis communicates the effects of depression storage as a function of rainfall. The plots generated may be normalized to the total DEM area by expressing runoff area as a percentage of the total area. Figure 14 shows this demonstrated for the example outlined in Section 3.3. Notice the cells along the edge of the DEM produce an initial level of 55% flowing off of the DEM edges. When the depression overflows after 44 millimeters of rainfall, there is a step to 100% of the DEM area running off.

If complete connectivity is approached rapidly, it may be assumed that the effects of depressions are negligible in that terrain. However, if several distinct

steps are apparent in the response and connectivity is reached only after a significant amount of precipitation, the assumptions of complete hydrologic connectivity may be inappropriate and having this insight can be valuable. To that end, SCS return-period storm events also may be plotted with the watershed response to determine the extent to which complete connectivity contrasts with the SDFA response. If a 100-year rainfall event is required to fill several depressions in the area of interest, then filling all depressions may be a radical assumption to make for some applications. For example, using a watershed that is generated only by a 100-year rainfall event may be excessive for a grassed

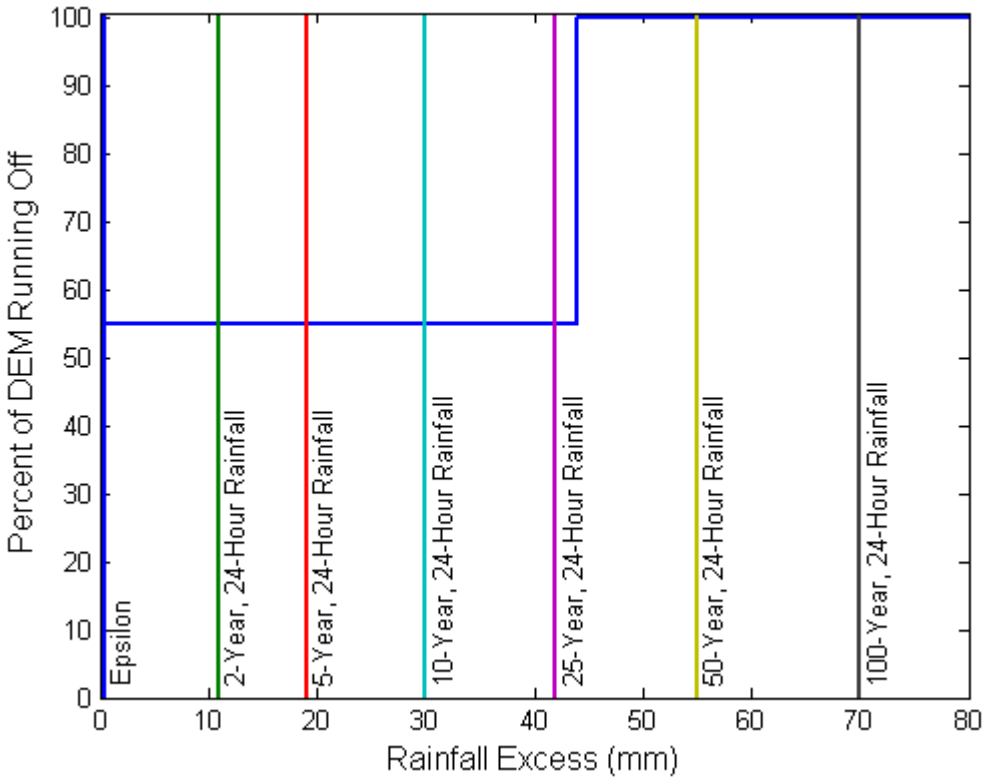


Figure 7. Percent area running off versus rainfall excess corresponding to the example described in Section 3.3.3. The vertical lines denote X-Year, 24-hour SCS rainfall events.

waterway design. For example, in Figure 14, a rainfall event in excess of the 25-year, 24-hour rainfall amount is necessary to achieve complete connectivity.

3.3.2 Applicability as a Function of Scale

Scale is another important consideration when trying to understand the variation of hydrologic connectivity for a given DEM. Depressions may be encompassed only at a particular DEM size or the interplay of depressions filling and overflowing into one another may need to be evaluated at a particular scale in order to be represented correctly. From a single DEM tile, non-overlapping subsets will be taken at a range of DEM sizes. For each non-overlapping sub-DEM iteration at a given DEM size, the percent of the DEM running off of the edges will be recorded as a function of rainfall excess. Additionally, as described in the previous section, SCS return-period storms were plotted for reference; the points where the curve intersects each SCS return period storm may then be plotted for each DEM size. Also included will be an epsilon rainfall excess value of 5 mm as a baseline minimum level so as to definite the initial extent of connectivity.

At small scales (the hectare scale), it is unlikely that significant depressions will be captured in DEM subsections, and the entire DEM subsection area is likely to run off after only a small amount of precipitation. At medium scales (tens of hectares), field-scale depressions should be present in the subsets and perhaps occupy a large portion of the DEM area. At large scales (hundreds of hectares), the rainfall amount required to fill all depressions will indicate the presence of significant depressions which affect hydrologic

connectivity while the percentage of the DEM running off after various rainfall levels may indicate the occurrence of depressions at that scale; it could be determined whether depressions are throughout the DEM or whether a single depression with small area footprint exists. If full connectivity is approached at minimal levels of rainfall for all scales, then depressions play little role in that landscape as it pertains to hydrologic connectivity.

To perform these analyses, several LiDAR-based DEMs were acquired. Because this application is aimed toward the management of agricultural runoff, agricultural areas were targeted for available LiDAR DEMs. DEM datasets supplied by Indiana's LiDAR vendor were sought out to avoid the steps of filtering ground points and gridding the LiDAR point data. These DEMs are often delivered in small tiles due to the large file sizes associated with this high-resolution data. Tile sizes and DEM resolutions vary by state (typically 100-250 hectares, 1000 x 1000 to 3000 x 3000 cells). This study utilized DEMs from Fulton, Clinton, and Pulaski County, Indiana. The DEM data for these areas are at 1.524-meter (5-feet) resolution and provided in 1000 x 1000 cell tiles via the Indiana Spatial Data Portal (Indiana University 2014). Table 2 presents the DEM subset sizes to be analyzed and the corresponding percent of the full 1000 x 1000 tile covered by the non-overlapping subsets.

The percent of the DEM running off at each of the return-period rainfall amounts was determined for each DEM subset, and for each DEM size these values were averaged. A plot of percent of the DEM running off versus DEM size

was produced for each return period rainfall event. Using this plot, trends in hydrologic connectivity as a function of DEM scale could be observed.

Table 2. DEM subset sizes, areas, the number of non-overlapping iterations fit into the 1000 x 1000 cell tile, and the percent of the DEM covered by these non-overlapping subsets.

DEM Size	DEM Area (hectares)	Number of Subsets	Percent Coverage
100 x 100	2.32	100	100
200 x 200	9.29	25	100
300 x 300	20.9	9	81
400 x 400	37.2	4	64
500 x 500	58.1	4	100
750 x 750	131	1	56
1000 x 1000	232	1	100

CHAPTER 4. RESULTS

4.1 Overview

An algorithm has been developed that enables field-scale watershed delineations on LiDAR-based DEMs. Depressions are filled sequentially in order to match surface flow conditions as observed in the field, enabling that filling may be stopped. First, to establish that the algorithm developed can properly delineate watersheds, several point delineations were performed after all depressions are filled sequentially and the results were compared with those produced using the ArcGIS Hydrology Toolset. Next, the functionality of the algorithm was demonstrated through the identification of depressions and the possible variation in the extent of depressions filled (i.e. hydrologic connectivity). An example will illustrate the implementation of drainage features into the algorithm and how these features can alter field-scale hydrologic surface connectivity. Finally, the implications of complete filling were evaluated using contributing area versus rainfall plots on several DEMs. The effects of scale on these hydrologic connectivity responses were analyzed on these DEMs.

4.2 Validation

After preparing two DEMs using both the SDFA as well as ArcGIS Hydrology Toolset, several watershed delineations were performed to validate that the

algorithm routes flow properly, recognizes ridge features as hydrologic boundaries, and delineates upslope contributing areas in a similar manner. The percent difference in watershed area when comparing the two methods was computed using Equation 10.

Figures 15 and 16 show the watershed delineations produced using each method on two fields in Fulton County, Indiana. Parts A and B of these figures represent the watersheds delineated after filling depressions using the ArcGIS and SDFA methods, respectively. Appendix B shows two additional DEMs in which an additional 12 watersheds were validated. Table 3 presents the resulting percent difference when comparing the watersheds produced from the SDFA developed to the existing ArcGIS methods. The error may result from cells missing from the SDFA watershed that are present in the ArcGIS watershed or cells in the SDFA watershed that are not part of the ArcGIS watershed.

Figure 15. For an agricultural field in Fulton County, Indiana, USA (lower left corner at -86.187 degrees west longitude, 40.974 degrees north latitude): A) watersheds based on ArcGIS Hydrology toolset and B) watersheds based on the SDFA.

Based on the percent difference, the delineations give similar results. However, there are situations in which differences may occur. Flow accumulation rasters (Figure 17) may be used as a guide to determine locations where watershed delineations are likely to yield a considerable contributing area. Delineations performed along high flow accumulation "stream" lines will produce large watersheds, while delineations performed even one cell off of these high flow accumulation cells will result in watersheds dramatically smaller in size. As a result, delineations performed in locations where the high flow accumulation cells do not align between the two methods will produce watersheds which conflict most. For example, looking at a zoomed-in area of Figure 15 shown in Figure 17, it is apparent that there are some differences where high flow accumulation cells do not align between the two methods. This is a result of

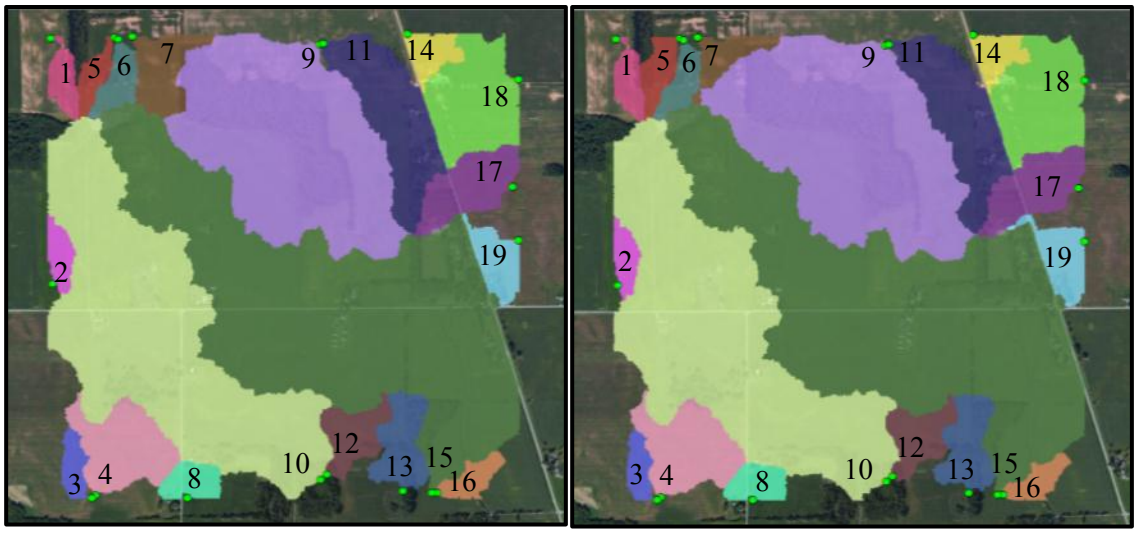


Figure 86. For an agricultural field in Fulton County, Indiana, USA (lower left corner at -86.183 degrees west longitude, 40.990 degrees north latitude): A) watershed based on ArcGIS Hydrology toolset. B) watersheds based on SDFA.

differences in the underlying flow direction information, and, as a result, delineations performed in the conflicting cells will produce watersheds with a high percent difference. Despite the differences about to be explained, it is apparent that flow is routed in a similar manner and that depressions have been identified, filled, and flow rerouted in ways that are similar. The same ridgelines that separate watersheds have been identified in each method as expected. Therefore, the behavior of the filling process implemented in the SDFA developed is roughly equivalent to those comparable existing ArcGIS processes.

Conflicts in flow routing and the subsequent flow accumulation data are caused by differences in how flat areas are handled. Specifically, challenges occur when 1) the flow direction of a single cell is to be assigned, but more than one neighbor has the same maximum distance-weighted drop, 2) one or more cells are raised to the same elevation as a result of the filling process and flow direction across this filled area must be resolved, and 3) a spillover location is to

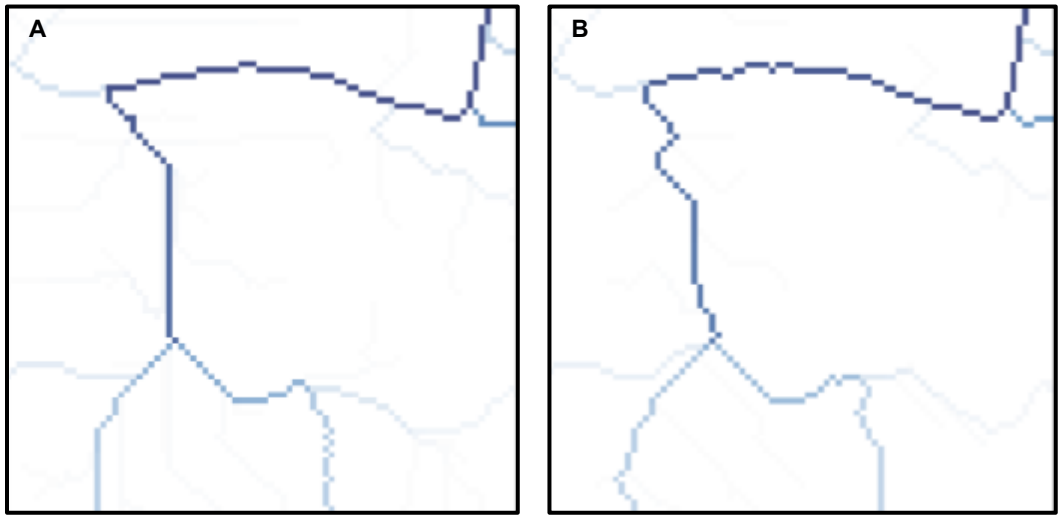


Figure 97. Flow accumulation raster produced using: A) ArcGIS algorithm and B) the SDFA developed. Darker lines indicate higher flow accumulation.

Table 3. Percent difference between watersheds delineated using the ArcGIS Hydrology Toolkit and the SDFA.

DEM	Watershed	Error A, Cells in SDFA, not ArcGIS	Error B, Cells in ArcGIS, not SDFA	Total ArcGIS Cell Count	Percent Difference
Figure 15	1	65	49	2414	4.70%
Figure 15	2	90	113	2116	9.60%
Figure 15	3	12	41	2063	2.60%
Figure 15	4	122	30	9833	1.50%
Figure 15	5	768	48	2054	39.70%
Figure 15	6	28	271	2923	10.20%
Figure 15	7	317	2852	5822	54.40%
Figure 15	8	20	25	2818	1.60%
Figure 15	9	3514	220	50503	7.40%
Figure 15	10	265	373	65185	1.00%
Figure 15	11	119	617	14869	4.90%
Figure 15	12	69	65	4725	2.80%
Figure 15	13	28	81	5738	1.90%
Figure 15	14	60	96	2821	5.50%
Figure 15	15	263	470	91011	0.80%
Figure 15	16	10	71	2379	3.40%
Figure 15	17	130	83	7561	2.80%
Figure 15	18	588	295	12758	6.90%
Figure 15	19	593	71	3896	17.00%
Figure 16	1	74	52	4679	2.70%
Figure 16	2	57	52	16700	0.70%
Figure 16	3	113	32	4678	3.10%
Figure 16	4	50	723	8528	9.10%
Figure 16	5	72	56	6592	1.90%
Figure 16	6	678	99	17036	4.60%
Figure B1	1	70	70	15799	0.89%
Figure B1	2	232	543	86607	0.89%
Figure B1	3	45	300	5914	5.83%
Figure B1	4	740	167	32228	2.81%
Figure B1	5	3	12	3767	0.40%
Figure B2	1	28	88	12025	0.96%
Figure B2	2	44	93	18551	0.74%
Figure B2	3	20	12	3387	0.94%
Figure B2	4	6	9	2847	0.53%
Figure B2	5	39	28	21701	0.31%
Figure B2	6	171	70	16603	1.45%
Figure B2	7	3	106	5516	1.98%
TOTALS		**9506**	**8383**	**574647**	**3.11%**

be identified, but multiple cells along the perimeter are of the same minimum elevation. The first challenge arises when flow direction of a single cell is to be assigned and multiple neighbors have the same maximum distance-weighted drop. In the SDFA developed, the maximum distance-weighted drop is updated only as a new maximum is encountered. As a result, when multiple neighbors have an equivalent maximum distance-weighted drop, the last one encountered is taken as the flow direction. However, in the Jenson and Domingue (1988) method employed by ArcGIS, a lookup table is used to decide flow direction based on the orientation of the neighbors with equivalent maximum distance-weighted drops.

Figure 18 shows how this distinction may affect how flow is routed in each of the two methods. When identifying the flow direction of cell 5, cells 2 and 8 are of an equivalent maximum distance-weighted drop. The SDFA developed directs flow toward cell 2 because it is encountered first and a new maximum is not found thereafter. However, because cells 2 and 8 are on opposite sides, the look-up table utilized by the ArcGIS method assigns flow toward cell 8.

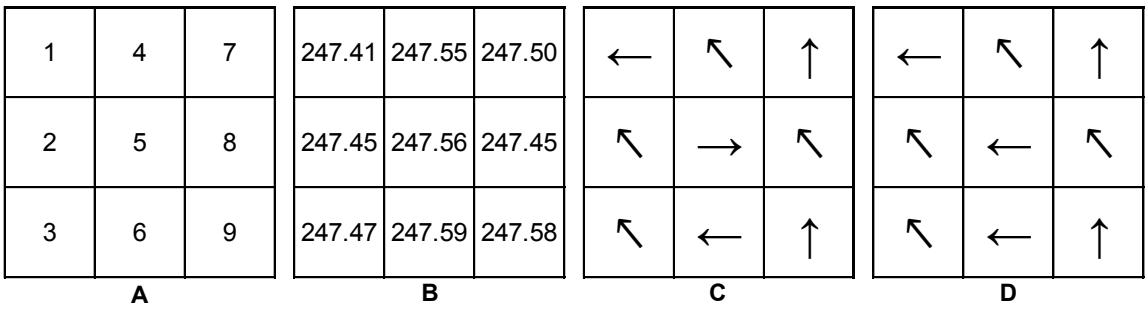


Figure 10. A) Reference cell indexing. B) Elevations. C) ArcGIS flow direction. D) SDFA flow direction. Although cell 1 is the neighbor with the lowest elevation, cells 2 and 8 have the greatest distance-weighted drop.

In the second situation, both methods resolve flow within a region that has been filled by iteratively working from the outlet, directing cells in the flat area toward cells with a known flow direction. As a result, a similar diagonal pattern is formed in the flow direction data (Figure 19), separating two contiguous areas of the same flow direction value. However, the specific order in which neighboring cells are traversed differs between each method, resulting in different flow direction values and flow accumulation datasets. In each method, the solution is arbitrary in that they are imperfect representations of the physical process of water flow through a puddle. Instead, the patterns in flow direction are the result of methods developed to assure that each cell in the filled region will reach the outlet.

The third situation occurs when a spillover location is to be identified for a given depression, but multiple cells along the perimeter are of the same minimum elevation. The true overflow location is obscured by limitations in the vertical resolution of the elevation data. Because the algorithms may check the

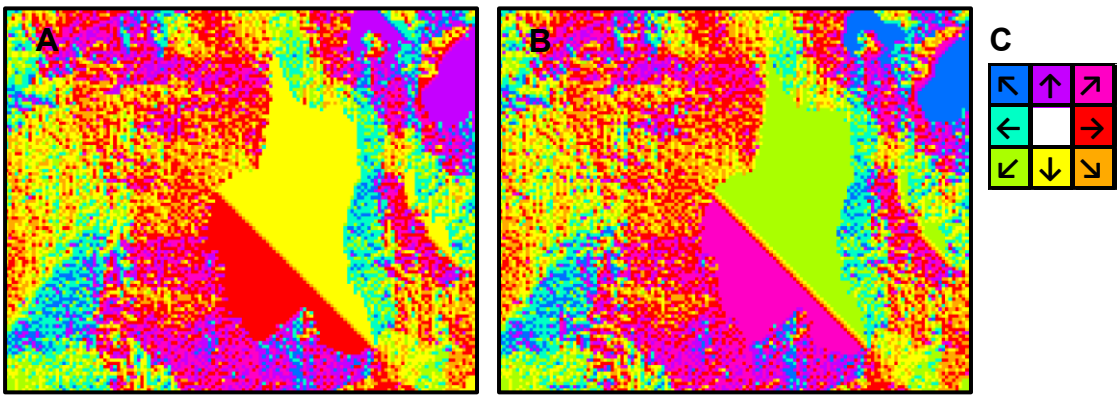


Figure 119. Comparison of how flow direction is resolved in depressions that have been filled. A) ArcGIS solution. B) SDFA solution. C) color/direction key.

boundary cells in a different, arbitrary order, contradictory overflow points may be chosen. Should the depression overflow frequently, the true overflow point may eventually erode over time and become a more dominant spillover location. Figure 20 shows a large region which conflicted in the DEM analyzed in Figure 15. From the underlying flow accumulation data shown in Figure 20, parts B and C, it can be seen that each method had routed flow from the orange area in opposite directions. Upon further investigation, this region was indeed a depression which had the same minimum spillover elevation on either side of the depression. While it was filled to the same elevation in each algorithm, the selection of the spillover location resulted in conflicting results. This particular example demonstrates the consequences that may occur as a result of differences in the fine details of each algorithm (combined with limited vertical data resolution).

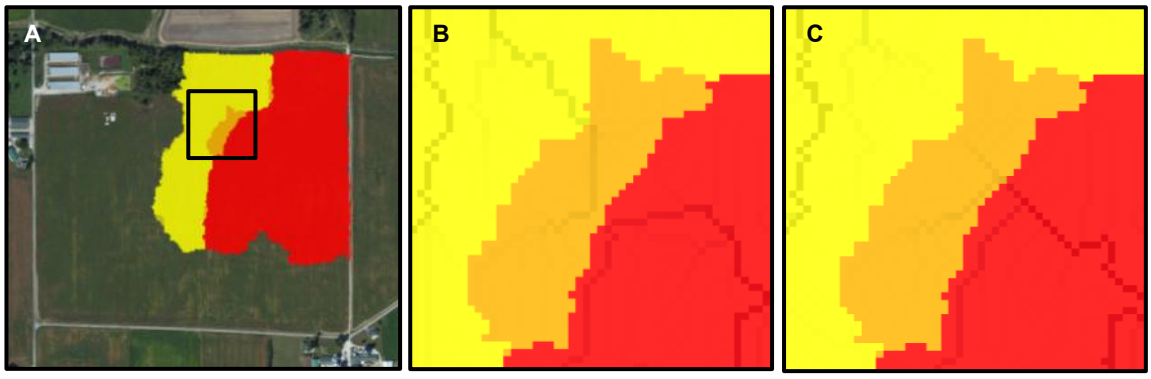


Figure 20. A) Two watersheds delineated using ArcGIS Hydrology Toolset and the SDFA. An area that conflicts between methods is shown in orange. B) the area of difference with underlying flow accumulation data produced from the ArcGIS methods. C) the area of conflict with underlying flow accumulation data produced from the SDFA.

4.3 Sequential Depression Filling: Effects on Watershed Delineation

While it is valuable to know that the SDFA developed mimics the flow direction and filling procedures of existing algorithms (when all depressions are filled), the utility of the SDFA lies its ability to fill only those depressions that will be filled in a specified rainfall event in an automated fashion. Figure 21 shows a location for which an inlet tile or culvert may need to be sized; watersheds are delineated with increasing rainfall using the developed SDFA and the resulting polygons are

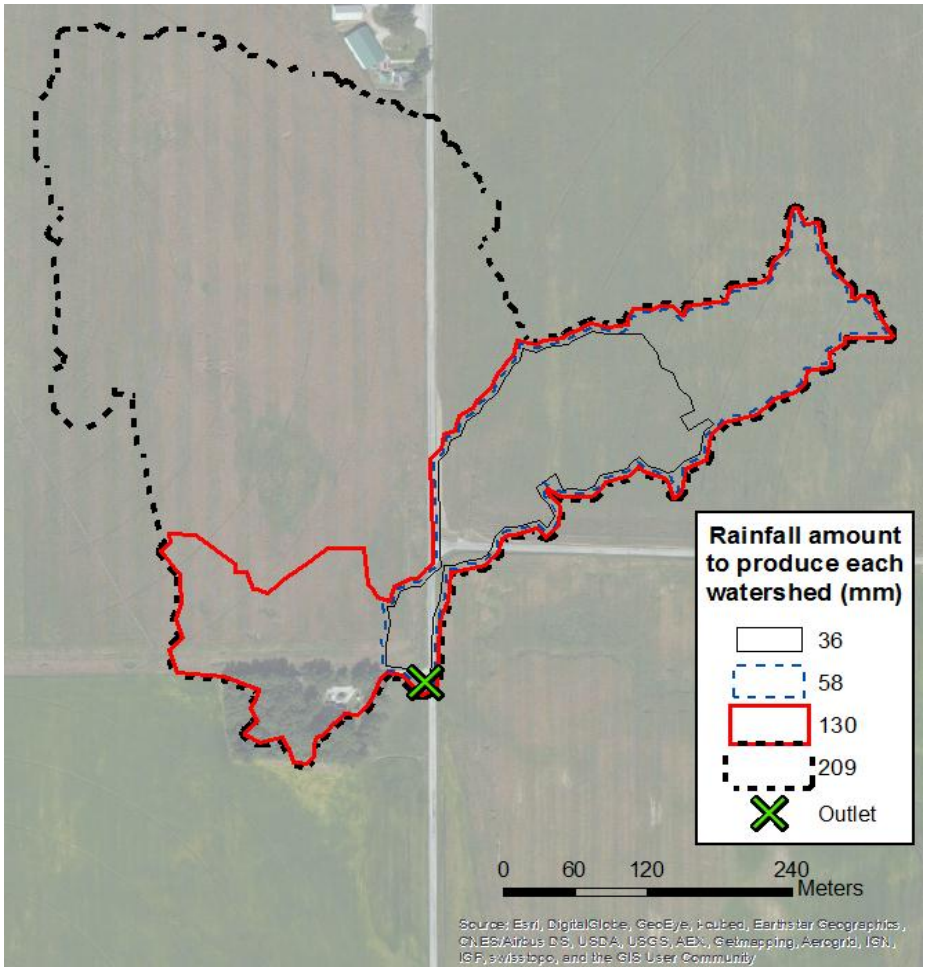


Figure 21. Watershed delineation in Fulton, County, Indiana (lower left corner at -86.194 degrees west longitude, 40.974 degrees north latitude). The progression of a watershed delineated at the marked outlet point with increasing rainfall. Losses have been accounted for using the SCS Curve Number Method with a curve number of 75.

shown. Figure 22 shows the associated contributing area plot for this selected outlet point as a function of rainfall applied. As the retention capacity of each surface depression is reached, a spillover event occurs, resulting in a step-wise increase in the watershed contributing area. To assume complete hydrologic connectivity would be to assume that the largest (outermost) polygon represents the appropriate watershed for all usage scenarios. However, the largest polygon in Figure 21 would occur only after 137 mm of rainfall, nearly equivalent to the 100-year, 24-hour storm event on a lossless surface. For analyses of lesser storm severity, this assumption is an error and leads to an overestimation of contributing area.

While rainfall excess is provided as a result of the depression storage and

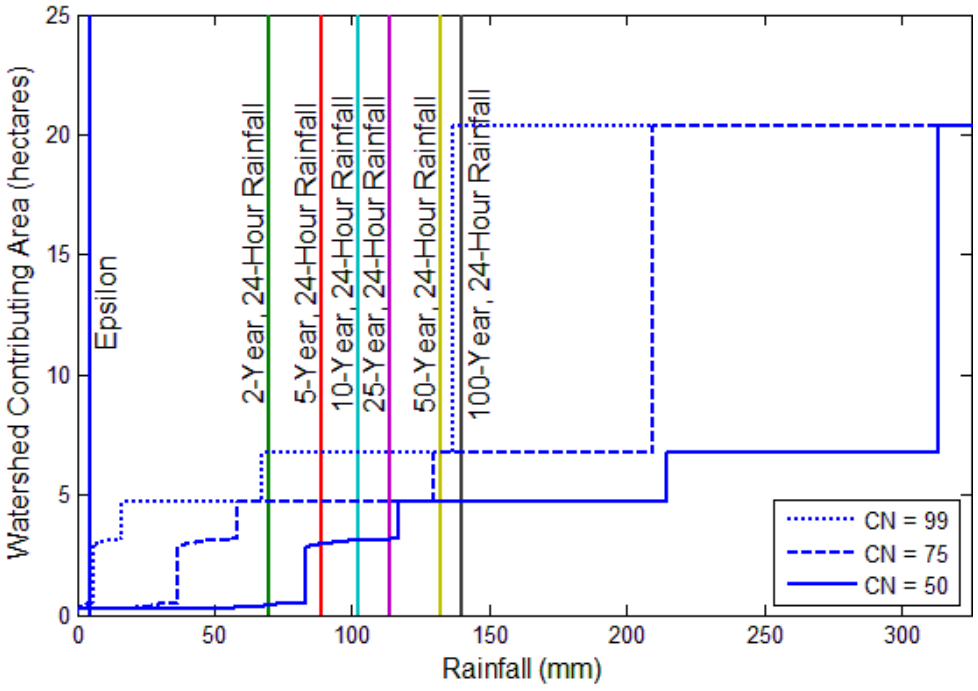


Figure 12. Watershed contributing area versus rainfall for the watershed delineations in Figure 21. Also demonstrated is how losses with three different curve numbers to convert excess rainfall to rainfall.

sequential filling components of the algorithm, rainfall is instead plotted which allows for increased interpretive value as it is now possible to compare directly to return period design storms. Using the SCS Curve Number Method, plots of precipitation were determined from the given rainfall excess values by calculating infiltration and losses u sing curve numbers of 50, 75, and 99. These values span a wide range of ground conditions including all of those associated with agricultural land usages. A curve number of 50 represents the best-case (high-infiltration) scenario for most agricultural land uses while a curve number of 99 (the maximum curve number) converts nearly all rainfall to runoff-ready rainfall excess as a nearly lossless surface. With lower curve numbers (i.e. more liberal assumptions of infiltration and losses), the plot is shifted further to the right. It takes more rainfall to generate the same contributing area. The calculation of precipitation values from rainfall excess is unconventional. However, the SDFA developed handles surface flow and depression storage which occur after infiltration and other losses takes place.

Having the inputs in terms of precipitation leads to more intuitive output such as “the contributing area resulting from 50 mm of rainfall is 300 square meters.” In the end, the inclusion of losses furthers the point that surface flow connectivity and watershed delineations are a function of rainfall in many landscapes; with more losses, additional rainfall is necessary before runoff begins and more rainfall is necessary to fill depressions and produce a hydrologically connected landscape. This goes against watershed model structures that apply hydrologic estimates of infiltration and other processes to a

given, predetermined watershed boundary; at large scales, the effects of such assumptions may not affect the model overall.

Additional examples of how watershed contributing area may vary with rainfall are provided in Appendix A (as Figures A1-A4). Losses in each of these watersheds have been accounted for by using a curve number of 75. Figure A1 shows a small watershed (taken from an agricultural field in Fulton County, IN) which includes a known natural depression which the algorithm estimates to require 113 mm of rainfall to overflow. Figure A2 demonstrates another watershed in Fulton County, Indiana which was found to experience two distinct jumps in the rainfall versus contributing area plot (Figure A2a) that occur as a result of natural features; the first is a natural depression which is estimated to require 142 mm of rainfall while the other (more on this in Figure 23). The second, however, is a pond which requires an extraordinary amount of rainfall to overflow. Such features should be omitted from downstream analyses because of they are so unlikely to overflow. From the associated plot of contributing area versus rainfall this can be easily observed (over 1000 mm of rainfall for this last step). It can then be excluded from analysis or incorporated in a retention pond model if it is determined to drain into this watershed. However, there are other landscapes which lack natural depressions and as a result require very little rainfall to achieve their maximum potential watershed size (and complete hydrologic connectivity). Figures A3 and A4 (though they each have a small natural depression in the forested areas) demonstrate this. It should be noted that, because there are no macrotopographic features in these watersheds, the

plots associated with Figures A3 and A4 more clearly show individual, small steps caused by depressions produced from noise in the LiDAR-derived DEMs. In fact, small exclusions can be observed that appear like holes in the watershed polygons.

4.4 Drainage Feature Implementation – Tile Inlet

Tile inlets or risers act to drain areas that are hydrologically isolated and frequently inundated if not remedied by these subsurface conduits. Because these features go undetected in the DEM, the depressions that are drained by these tile risers will be filled and rerouted over the ground surface. As a result, watershed delineations will be affected by such alterations to surface flow patterns.

Figure 23 shows the implementation of a tile drainage inlet using the SDFA on one of the DEMs used in Section 5.2 (validation). Figure 23a shows the location of a tile riser (R) located at the base of a natural depression that requires drainage. The 20-cm inlet at this location is capable of a draining around 1500 cubic meters per hour (assuming typical materials and 1% grade, Panuska, 2012). Figure 23b shows a watershed delineated downstream of this depression after all depressions have been filled, yielding 54 hectares. As the 1500 m^3/h are being drained, 17000 m^3 must accumulate in order to overflow the basin (which at 10 mm/h would take 6.8 hours, for example). In Figure 23c a 14-hectare watershed is delineated downstream of the depression while accounting for this drainage feature. Such considerations are important if watershed

delineations are to match on-site observations and reality. In this scenario, the riser actually diverts a significant (in this case 75%) of the watershed area.

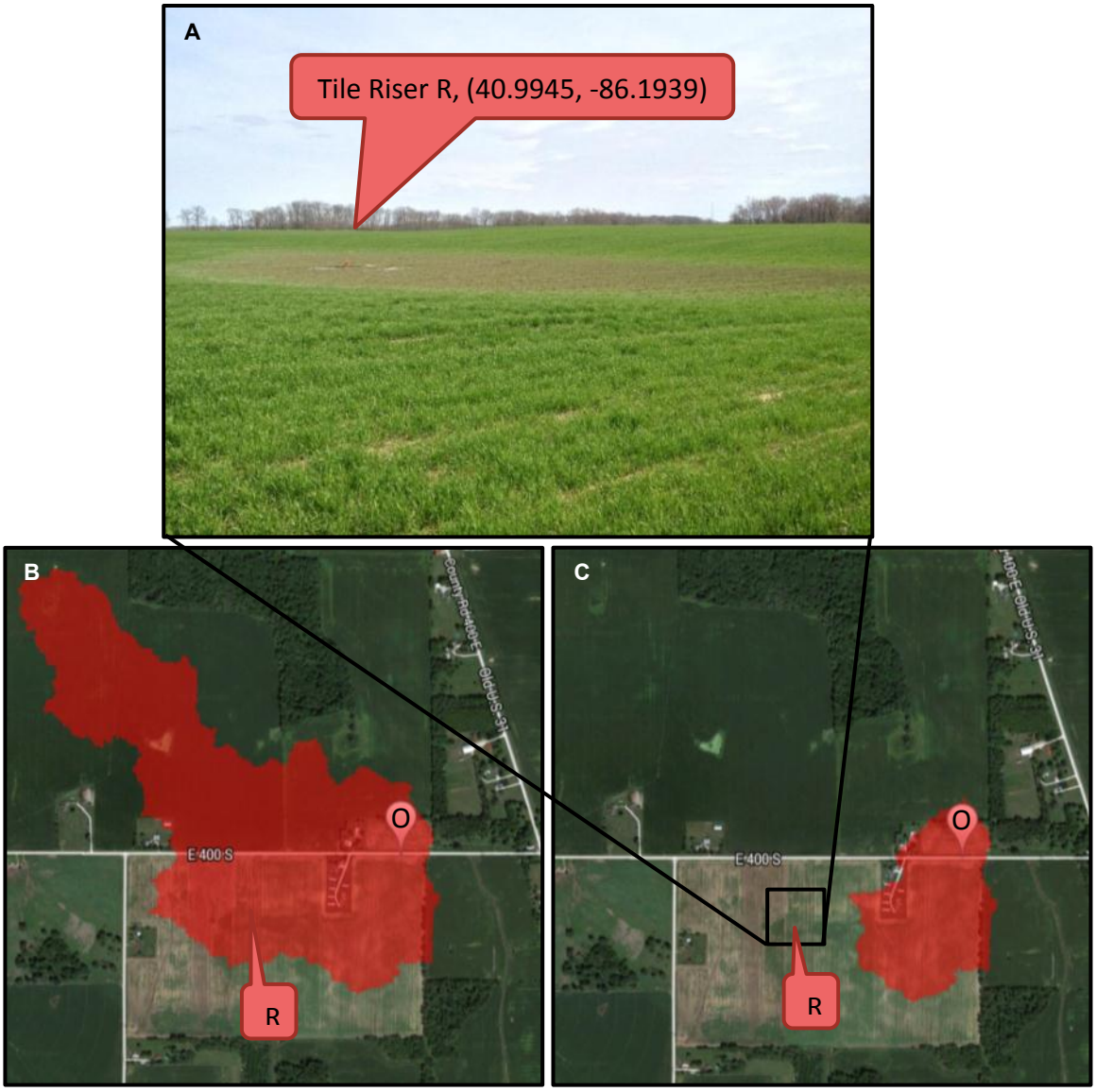


Figure 13. A) A natural depression with a tile riser at R. Notice the brown vegetation indicating standing water following a large rainfall event B) A watershed delineation for the outlet O without accounting for the tile riser R. C) A watershed delineation for the outlet O while accounting for the tile riser R.

4.5 Applicability Study

The relevance of sequential depression-filling hinges on the extent of hydrologic connectivity present across a given landscape as runoff begins. Landscapes with natural depressions and other features that inhibit surface flow connectivity (e.g. road embankments) will vary in connectivity more widely as a function of rainfall than landscapes lacking significant depression storage. It may be determined whether depressions are singular, localized features or spread relatively homogenously across a given region. As scales increase, the role of depressions may either diminish or they may dominate the hydrology of a landscape on the whole.

Additionally, scale is also an important consideration when determining an appropriate minimum DEM size acceptable for use on a mobile device. Because memory and computational resources are more limited on a mobile device than on a desktop platform, the DEM area footprint may be based on agricultural field boundaries rather than watershed boundaries to reduce file sizes. A recommendation of minimum DEM size can be issued after the scale at which depressions and their full contributing area has been determined.

Figure 24 shows how connectivity may vary across an entire DEM in response to several rainfall events. Each uniquely colored polygon represents an area that drains to a common location. Isolated polygons are depressions that have some storage capacity and are yet to overflow, while polygons connected to the edge of the DEM are no longer involved in the filling process. Figure 25 plots the corresponding area draining off of the DEM in response to

increasing rainfall for the DEM shown in Figure 24. Each step represents a threshold in depression storage that was breached, where a depression began to

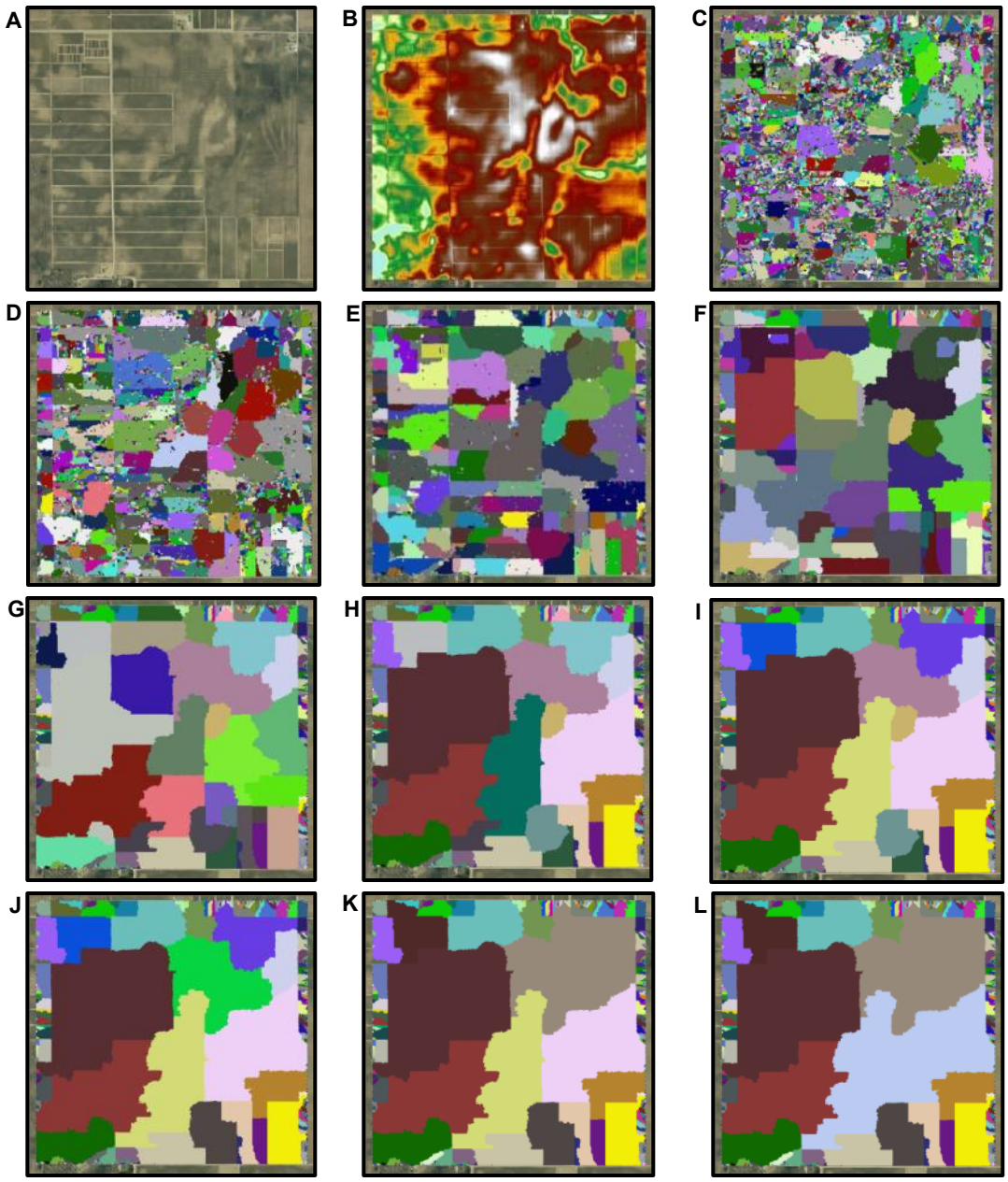


Figure 24.The variability of hydrologic connectivity as a function of rainfall excess for several plots at Throckmorton Purdue Agricultural Center in Tippecanoe County, IN. A) Orthophotography, B) DEM, and C-L) Catchment map showing the extent of hydrologic connectivity after 27, 34, 40, 57, 77, 96, 112, 129, 159, and 188 mm of rainfall excess, respectively. Each colored polygon represents a hydrologically connected "catchment."

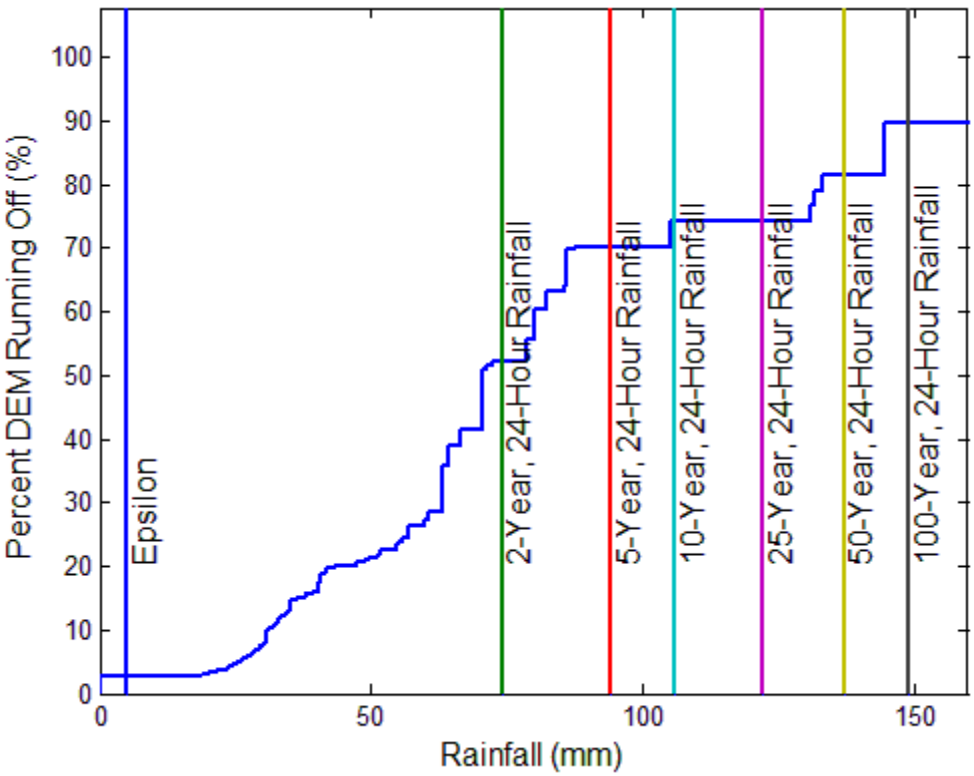


Figure 25. Percent of DEM area running off of the DEM versus rainfall corresponding to the DEM in Figure 24, and several SCS return-period storms for reference.

overflow and run off of the DEM. Rainfall excess has been converted to rainfall by back calculating losses using the SCS Curve Number Method assuming a curve number of 75.

Figure 26 shows the percent area running off as a function of DEM size for several rainfall events and three different counties in Indiana, USA. Each point on these curves represents the average percent area running off of the DEM after a given rainfall event. For example, for a DEM size of 100x100, 100 non-overlapping sub-DEMs (from a 1000x1000 DEM) were analyzed, and for these 100 sub-DEMs, the average percent of the DEM with a monotonically descending path off of the DEM edges after the epsilon rainfall event (0.1 mm of

rainfall) was 25.6 percent. Figure 26c demonstrates a landscape with relatively little change in hydrologic connectivity as a function of scale or rainfall excess. With very little precipitation, nearly 100 percent of the DEM runs off at all scales examined. In this type of landscape, filling all depressions prior to delineating watersheds is a safe assumption (though infiltration may shift these curves downward as will be pointed out later).

The landscapes plotted in Figures 26a and 26b exhibit a wider fluctuation in connectivity with rainfall and scale. At most scales, complete hydrologic connectivity is not achieved even after significant rainfall occurs. As scale increased (from 2 to 232 hectares) for a given rainfall amount, connectivity trended downward in general. This is due to the larger storage volume, per unit area, that exists at larger scales. This contradicts the common assumption made with coarser-resolution, large scale DEMs that sinks are erroneous. As DEM size increased, connectivity is more likely to be a function of rainfall. In these landscapes it is more important to be mindful of the implications of filling all depressions before delineating watersheds. Appropriate and accurate delineations performed in these landscapes are much more likely to vary with rainfall.

With landscapes that have significant depression storage, many of them exhibit a particular pattern as a function of DEM size where the percent running off begins very high, then decreases to a minimum around the 125-hectare scale, then they begin again to increase. This is because at small scales, depressions are not completely encompassed in the DEM. As a result, these areas may

immediately flow off of the edge of the DEM or else overflow quickly because the ridges are not entirely encompassed in the DEM. As the DEM size increases, the full depression storage area is encompassed in the DEMs and storage increases. Further increases in DEM size will include the full contributing area of

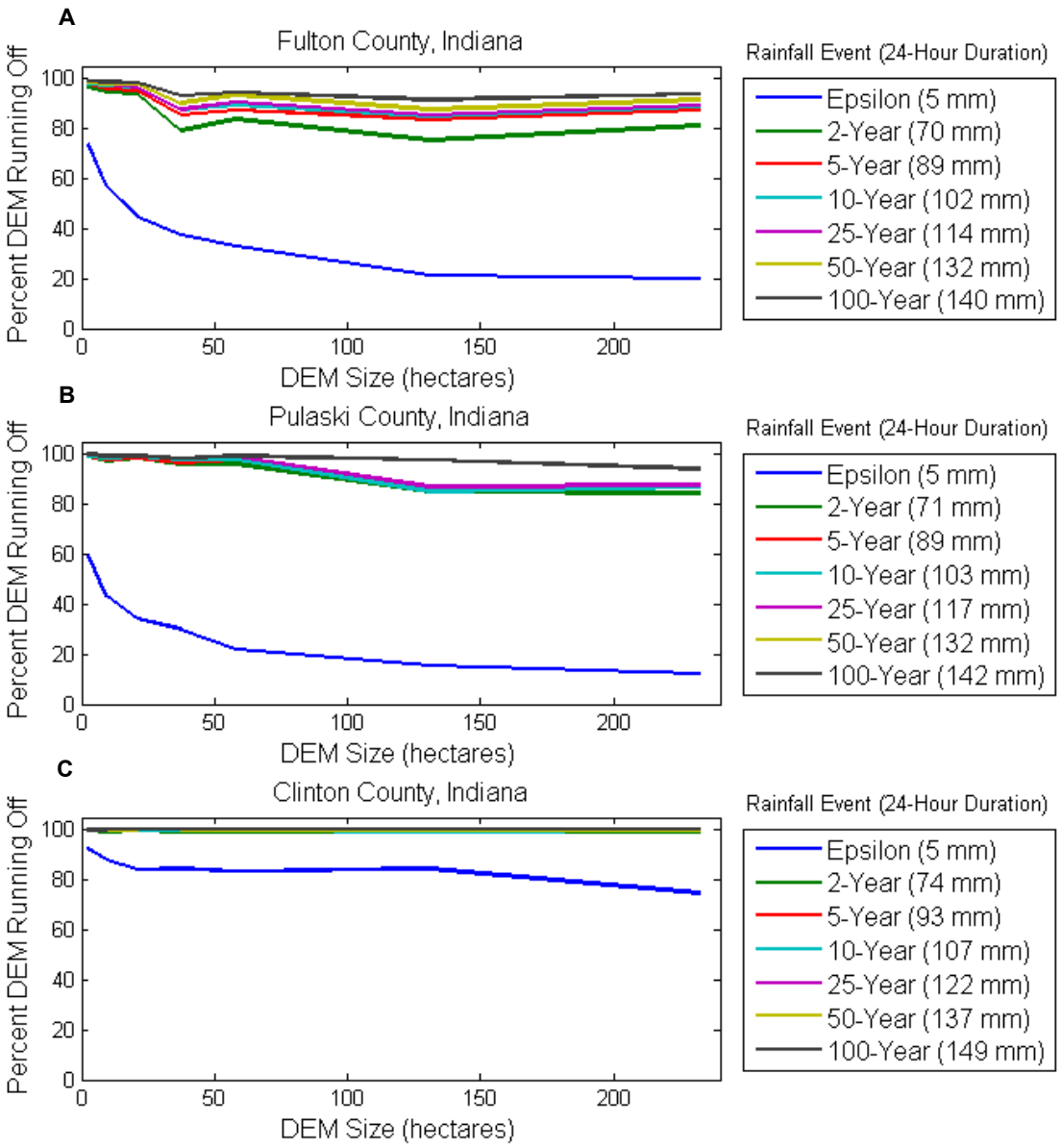


Figure 26. Percent DEM running Off vs. DEM size for several SCS return period rainfall events (assuming no infiltration) for DEMs located in A) Fulton County, IN, B) Pulaski County, IN, and C) Clinton County, IN

these depressions, working to reduce the storage per unit area; in other words, at this stage, the “funnel” of basins will increasingly be included in the DEM while the retention volume remains relatively constant.

It should also be noted that a certain portion of the DEM area will always run off because the DEM extent will likely never align with ridge features. For example, if a region were characterized by having many depressions one next to the other, then it would be expected that very little of the DEM runs off with a small amount of rainfall; however, unless the extent of all depressions were fit perfectly into the extent of the DEM, there will inevitably be depressions that will drain off the edge of the DEM immediately or otherwise overflow sooner than expected. Similarly, it is equally unlikely that the contributing area of depressions will align with the DEM bounding box so that some depressions may take longer to overflow than expected.

4.6 Effects of Infiltration

The plotted points in Figure 26 represent the percent of the DEM running off of the edges after the specified rainfall amount without considering infiltration or other losses. Figure 27 employs the SCS Curve Number Method to estimate the runoff-ready portion of rainfall before it is applied to the SDFA for analyses. Figures 27a, 27b, and 27c display the landscape from Figure 26a after considering losses resulting from curve numbers of 100, 75 and 50, respectively. Likewise, Figures 28a, 28b, and 28c corresponding to the landscape in Figure 26c after again considering losses resulting from curve numbers of 100, 75, and 50.

For the landscape in Figure 27, these considerations do not alter the outcome significantly. In a high infiltration scenario with a curve number of 50, 80% of the DEM runs off after the 100-year storm. This would indicate that, on average, 20% of the DEM will vary from the ArcGIS assumptions, meaning that delineations may be effected in this percentage of the DEM area on average.

For the landscape in Figure 28, the potential impacts of infiltration and other losses are substantial. For a curve number of 75, the plots have shifted

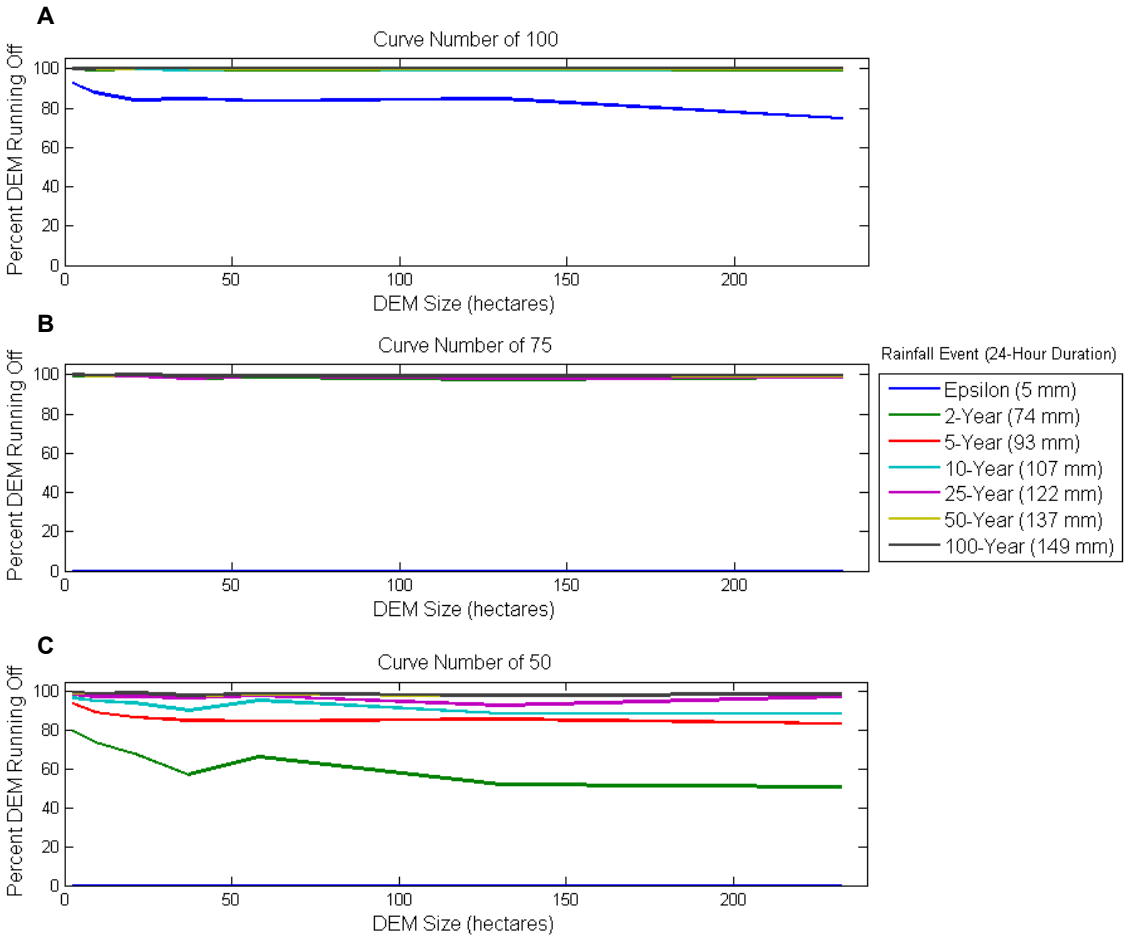


Figure 147. Percent DEM running off vs DEM size for several SCS return-period rainfall events for DEMS in Pulaski County, Indiana while taking into account losses using the SCS Curve Number Method. Shown are the same DEMs while accounting for losses using curve numbers of A) 100, B) 75, and C) 50.

notably to the point that the most extreme rainfall event plotted achieves only 85% connectivity. For a curve number of 50, the 100-year, 24-hour rainfall event results in only 60% of the DEM running off at the largest scale.

Subsurface drainage has a quantifiable and clear effect on field-scale hydrologic connectivity. Not all landscapes have complete hydrologic connectivity as runoff begins. Many models simulate overland flow in two simple, lumped steps: 1) conversion of rainfall into rainfall excess for each of many different HRUs (hydrologic response units: areas that all share a common land use + soil + slope classification) and 2) routing of streamflow (using, for example,

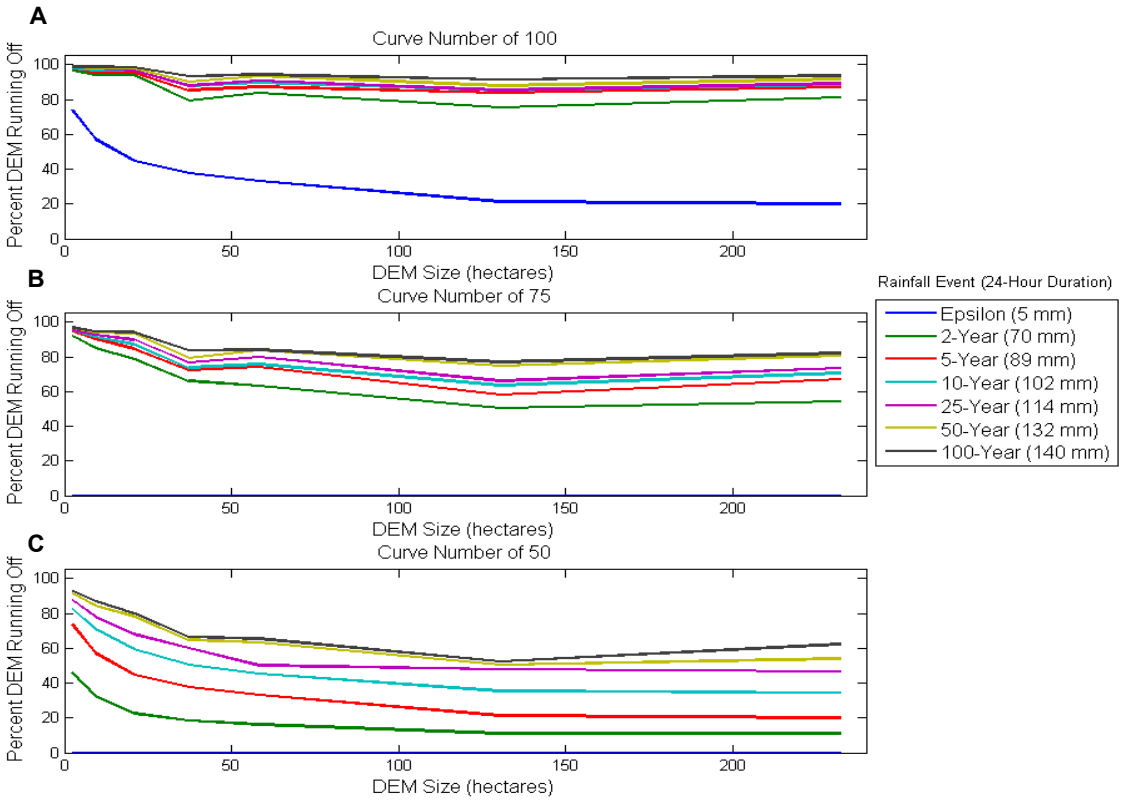


Figure 28. Percent DEM running off vs DEM size for several SCS return-period rainfall events for DEMS in Fulton County, Indiana while taking into account losses using the SCS Curve Number Method. Shown are the same DEMs while accounting for losses using curve numbers of A) 100 B) 75 and C) 50.

the Muskingum method). Hydrologic response is often modelled with the use of a lag time variable which is a function of the longest flow path length in the in the DEM after filling all depressions. Through the methods describing the SDFA, it should be clear that water may spend a considerable amount of time in surface depressions for highly disconnected landscapes, complicating the lag time calculation.

In landscapes lacking hydrologic connectivity, it often may be the case that drainage features have been implemented to enforce connectivity and keep agricultural lands productive. Plots and maps reflecting a landscape with low hydrologic connectivity demonstrate the extent to which human modification is necessary to achieve connectivity as we know it. However, the processes of infiltration and subsurface drainage are different than overland flow. As such, hydrologic connectivity may play a larger role in better understanding the interactions between surface and subsurface flow.

CHAPTER 5. WATERSHED DELINEATION MOBILE APPLICATION METHODS

As described in the introduction, it may be desirable to perform some tasks such as watershed delineation in the field in order to more rapidly resolve issues that are observed while in the field. A mobile application may be used to assist users' field observations, allowing them to make reliable estimations and suggest possible solutions to their clients without returning to the office, expediting the typical design and decision-making process.

5.1 Algorithm Development

The SDFA outlined in Chapter 3 was initially developed in Matlab to prove the concept and refine the algorithm. This allowed for rapid visualization of data, code development, robust testing, and generation of graphs and other outputs. By contrast, the Android development system is a more complex software ecosystem. Android makes use of Java, an object-oriented programming language. Specific restrictions are applied to code placed in the user interface (UI) thread such that resource-heavy computations likely to slow the user experience must be handled in the background. As far as the end-user experience, interfaces developed in Android have a higher ceiling than Matlab in terms of user-friendliness, simplicity, and visual appeal. Because mobile

application users are unlikely to be interested in modifying scripts, the number of variables will be minimized and presented in a convenient interface.

In order to begin developing Android applications, several components must be acquired in order to begin programming. One must typically make use of an integrated development environment (IDE) such as Eclipse that includes the workspace to write, build, compile, and debug code (Eclipse Foundation, 2014). An IDE is useful to organize the complex directory structure and project setup files of Android applications. The Android Developer Tools (ADT) contains all of the necessary software libraries and application programming interfaces (APIs) to specifically develop applications on the Android platform. The combination of the Eclipse IDE and the ADT are available online and packaged together in the Android Software Development Kit (SDK). Once this software is installed, applications can be developed and tested on Android devices (or a software emulation of an Android device on a computer).

Most mobile devices have several sensors and tools onboard that developers may utilize. The GPS, accelerometers, camera, network connection, and touchscreen enable the development of innovative applications and tools. Mapping services such as the Google Maps API may be used to display spatial information, and when used in tandem with a device's GPS, this can provide added functionality such as navigation and geographic data collection.

To publish signed apps online for the public to download at the Google Play Store (i.e. the marketplace for all Android apps, books, music, etc.), one must register as an Android Developer and pay a one-time 25 dollar fee. After

uploading the application file to the marketplace, developers may track their download statistics, receive bug reports, and publish updates to their application. The listing displayed on the Google Play Store may be edited to include a description and sample screenshots.

5.2 Android Libraries

The geospatial data abstraction library (GDAL) was used to incorporate additional GIS functionality such that most any raster data types and geospatial referencing systems may be supported by the Watershed Delineation App. GDAL is a library produced by the Open Source Geospatial Foundation (OSGeo) which allows for reading, writing, and even manipulating raster data. The GDAL library also contains OpenGIS Simple Features Reference Implementation (OGR), the vector data component of the GDAL library. The GDAL library is available in several languages including Perl, Python, Java, C#, C++, Ruby, and R. The Java version of GDAL is comprised of Simplified Wrapper Interface Generator (SWIG) Java bindings. The Java version of GDAL can be compiled for Android using the Android Native Development Kit (NDK). The Android NDK is a toolset that allows for usage of native code in Android apps. The app also makes use of the file-writing capabilities of the GDAL library to export raster watershed delineation datasets.

The Watershed Delineation App also makes use of OpenATKLib. This library was produced from the OpenATK project which focuses on farm management apps (OpenATK, 2014). It adds additional mapping user-interface functionalities on top of the Google Maps API such as clickable polygon features

(the Google Maps API does not include any listeners for click events on polygons, only markers). In addition to clickable polygons, the OpenATKLib offers an implementation of markers which includes the ability to give markers a “super draggable” quality. In the Google Maps API, markers are used to represent points, and in the Watershed Delineation App, a marker is used to indicate the outlet location of the watershed to be delineated. This “super draggable” feature allows users to simply touch and drag the marker without the standard “long-hold” touch required by the Google Maps API marker functionality. The long-hold marker movement requires the user to press in a specific location of the marker icon (in which the user’s fingertip is likely to entirely cover) and remain held in that position for around 3 seconds; if the user’s finger does not first make contact with the exact “hit box” location of the marker, the long-hold event will not be triggered. If the user moves their finger while long-holding the marker, it may instead pan the map. What is more, there is no inherent indication to the user that they must perform a long-hold to move the marker and so this procedure is not apparent unless previously demonstrated to the user. This is a clear example of how user experience can be improved.

The combination of GDAL and OpenATKLib libraries that provided GIS data and user-interface functionality to the Watershed Delineation App have been combined into a library which future watershed management mobile apps can utilize and on top of which they can build. This WMACLib will incorporate GDAL and OpenATKLib to handle raster DEM datasets as described in Section

6.2. WMACLib will also include hydrologic analysis GIS functions such as flow direction, flow accumulation, and pit-filling.

5.3 Implementation Verification

After implementation of the SDFA as an Android application, it was verified that the algorithm was correctly ported from Matlab to Java and the algorithm operates as demonstrated in the previous chapters. To make the comparison, catchment grids were generated from a DEM before rainfall and after applying 25, and 250 millimeters of rainfall using Matlab as well as the Android application.

5.4 DEM Size Performance Relationship Testing

While mobile device performance is rapidly improving, there are still resource constraints when compared to desktop computers and while attempting to process high-resolution DEMs. To get a better grasp of the limitations of the mobile device platform, the algorithm was run on several DEMs of differing sizes and resolutions until all depressions have been filled and the run times will be recorded. DEM sizes of 125 x 125, 250 x 250, 500 x 500, and 1000 x 1000 will be evaluated at 1-meter and 3-meter resolutions. It is expected that the algorithm's performance is directly related to the number of depressions in the DEM, and so the initial number of depressions will be reported for each run of the algorithm as well. Given a roughly uniform distribution of depressions across the landscape evaluated, the number of depressions is expected to vary proportionately with DEM size. To ensure roughly uniform distribution of depressions, all DEMs were be taken from the same area of Clinton County, Indiana, USA. Furthermore, it is expected that the initial amount of depressions

in the DEM are affected by the chosen DEM resolution due to elevated signal-noise effects as horizontal resolution increases while vertical accuracy remains constant. For each DEM size and resolution combination, run times and initial number of depressions were recorded.

CHAPTER 6. WATERSHED DELINEATION APPLICATION RESULTS

6.1 User Interface Design and Functionality

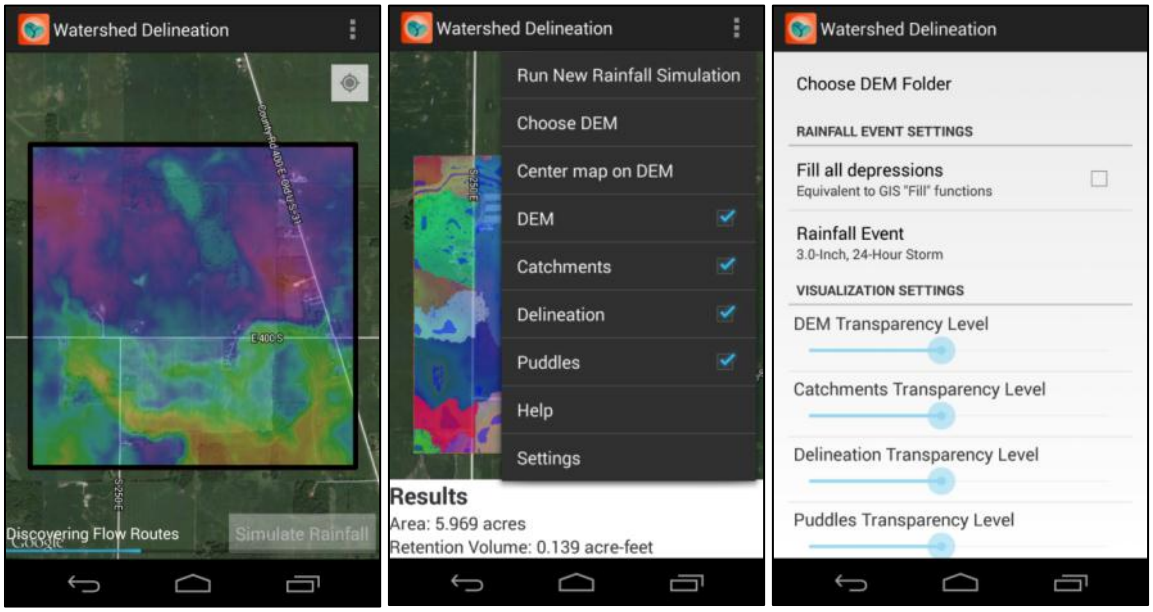
Upon installation of the application, a directory is created on the device called ‘dem’ and a sample DEM is placed in this directory so that the app may be demonstrated when used for the first time. A sample DEM was included to allow potential users to try out the app without downloading data.

All DEMs in the ‘dem’ directory are displayed on the map as a rectangular polygon of the DEM boundary. If the user taps inside of one of these boundaries, that DEM will be loaded as the current operating DEM. Alternatively, a DEM may be chosen from a list by navigating to Menu > Choose DEM (Figure 29b). Alternate DEM directories may be specified by going to Menu > Settings > Choose DEM Folder (Figure 29b). After choosing a DEM, it loaded and a colored overlay is created on the map.

When a DEM is selected, some preprocessing such as flow direction and initial pit identification occurs while the user is allowed to pan, zoom, and edit settings (Figure 29a). The Simulate Rainfall button, which fills depressions and prepares other datasets from which watershed delineations can be performed, remains grayed out until preprocessing is completed (Figure 29a). As described in the previous chapters, the extent of connectivity and resulting watershed

delineations may vary based on the amount of rainfall applied. The rainfall amount to be simulated may be edited from the settings screen. The option to fill all depressions may also be specified by tapping a checkbox (Figure 29c).

Several visual data layers are available after the SDFA has been run (Figure 29). The visibility of these layers may be toggled from the menu in the upper right while the transparency of each layer may be adjusted from the Settings screen (Figure 29c). In the *Catchments* overlay (Figure 30b), each uniquely colored polygon represents a different catchment (i.e. depression) after employing the SDFA. Some drain off of the map while others that have yet to be filled may appeared as isolated polygons. This is effectively a connectivity map where hydrologically connected cells which flow to a common destination share the same color. Notice that the polygon delineated in Figure30d is a subset of



A **B** **C**

Figure 29. Watershed Delineation App User Interfaces. A) preloading DEM parameters. B) Action Bar overflow menu. C) Settings screen.

the polygon marked with an X in Figure 30b. If a delineation were performed at the location where the watershed meets the edge of the map, the delineated watershed polygon would match the marked polygon in the catchment map. By comparing the DEM elevations before and after depressions have been filled, cells that have been raised may be visualized using the *Puddles* layer (Figure 30c) by coloring these altered cells blue.

The *Delineation* layer (Figure 30d) displays the watershed by displaying those cells draining to the red marker in red. Additional watersheds may be delineated by simply dragging the red marker to a new location within the DEM bounds. After the marker has been dropped at the desired location, the upslope contributing area is delineated by tracing the flow direction information to find all cells draining to that location. Initially, delineations were performed by hitting a "Delineate" button. However, Android allows for actions to be triggered upon the end of a marker drag event (i.e. upon letting go of and dropping the marker). This is an example of simplifying the user experience and de-cluttering the user interface.

Initially, watershed delineation attempts were performed by selecting a single pixel at the tip of the marker location. However, because of high data resolutions, a single pixel rarely has a high number of cells flowing through it (see flow accumulation discussion in Section 5.2). As a result, it took several attempts to find an interesting delineation with a considerable contributing area. To resolve this, a small buffer (a 7 x 7 window) was placed around the single pixel to increase the area of interest for the delineation. The resulting delineated

watershed represents the area draining to any of the cells in the buffered area of interest. Less precision and fewer attempts will be required to produce a non-trivial result. The delineated area value in acres is displayed in the results panel at the bottom of the screen. In the future, other parameters and information such as average slope and surface retention may be displayed in this panel.

This approach allows for watershed delineations to be performed in rapid

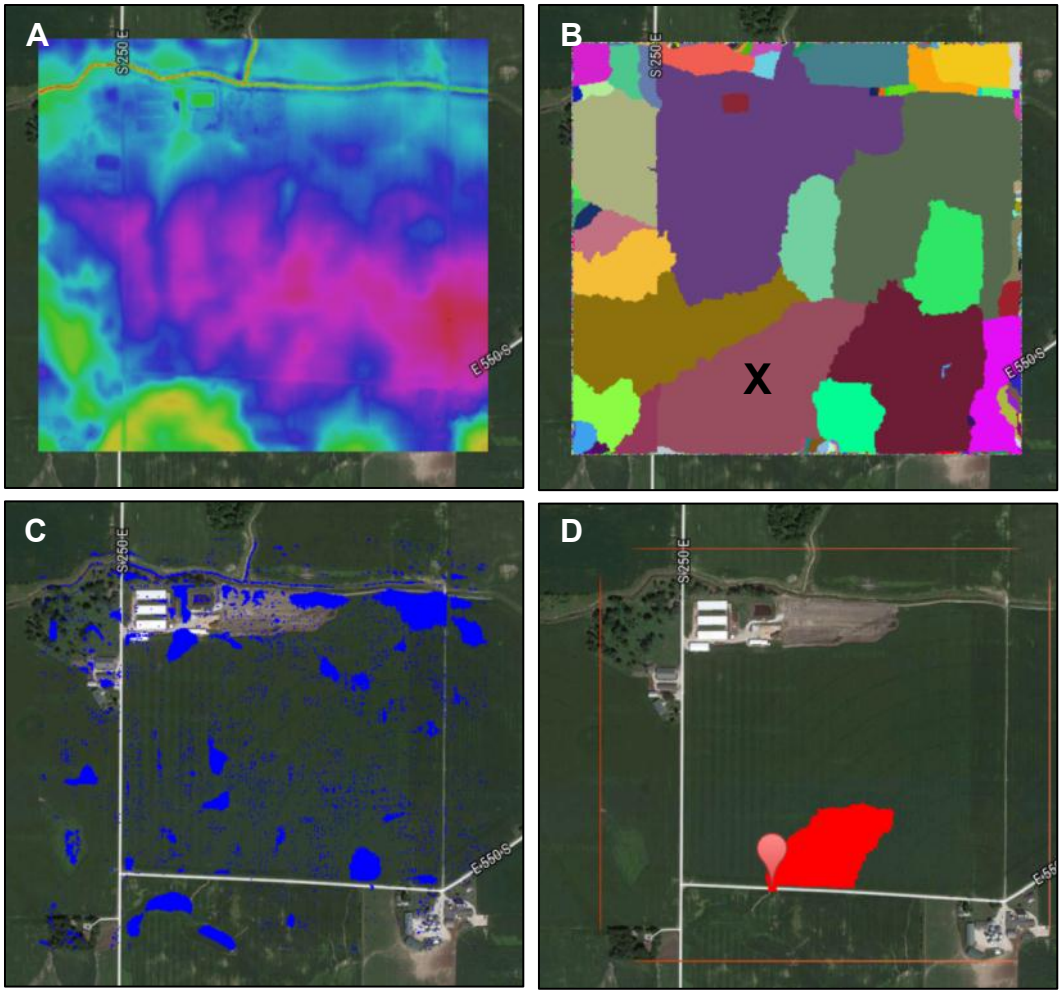


Figure 30. Visual Overlays. A) DEM Elevations. High elevations are pink and low elevations are yellow. B) Catchments. The polygon marked with an X indicates the catchment containing the watershed delineated in D> C) Puddles. D) Delineation. The marker indicates the outlet location while the area in red shows the watershed area draining to that outlet point.

succession across the DEM area under the selected state of connectivity (i.e. rainfall event). Alternatively, a second approach would allow for the selection of one or more watershed outlet points before the algorithm has been run, and then the SDFA could be run to find the watersheds for various rainfall events. The algorithm would only run a single time and watersheds could be delineated for several rainfall events of interest as depressions are filled with increasing rainfall. This approach may prove useful in the case of a known outlet point in an area that is known to vary widely as a function of rainfall. However, the first approach was selected because watershed delineations at the field scale are often a process that takes some trial-and-error. If a user were to choose a poor outlet point, it may become frustrating to have to repeatedly run the algorithm to find a suitable outlet point. While discussing these options, it is worth mentioning that the algorithm could be easily modified to allow it to be paused and continued if this were a desirable usage scenario. However, it is significantly more difficult to reverse the algorithm (see Section 7.3, Future Work). For example, imagine attempting to derive the individual elevations of several hundred adjacent cells that had been all raised to the same elevation in the filling process.

After a given depression has been filled, the filled area (i.e. the puddle) is treated as a collective feature where users cannot delineate the area draining only to portion of the puddled area. If a user clicks within a puddle feature, the entire puddle and all of those cells flowing into that puddle shall be included in the watershed delineation. If not handled in this matter, then it is likely that delineations performed inside a puddle will produce linear, artifactual watersheds

as a result of the patterns in flow direction that are produced when resolved during the filling process (see Section 4.2).

The Catchments and Delineation layers may be output as a raster geotiff file. For the Catchments layer, raster cells making up each catchment polygon are assigned their particular depression ID value. For the delineation layer, a value of 1 is assigned to all cells delineated within the watershed of interest while all remaining cells are assigned a value of 0. Each of these raster layers will be given the same output extent as the input DEM. In the future, these layers may be output as vector datasets (e.g. shapefile format).

The Watershed Delineation app can be found on the Google Play Store under the title “Watershed Delineation – WMAC” as shown in Figure 31 (https://play.google.com/store/apps/details?id=org.waterapps.watershed&hl=en).

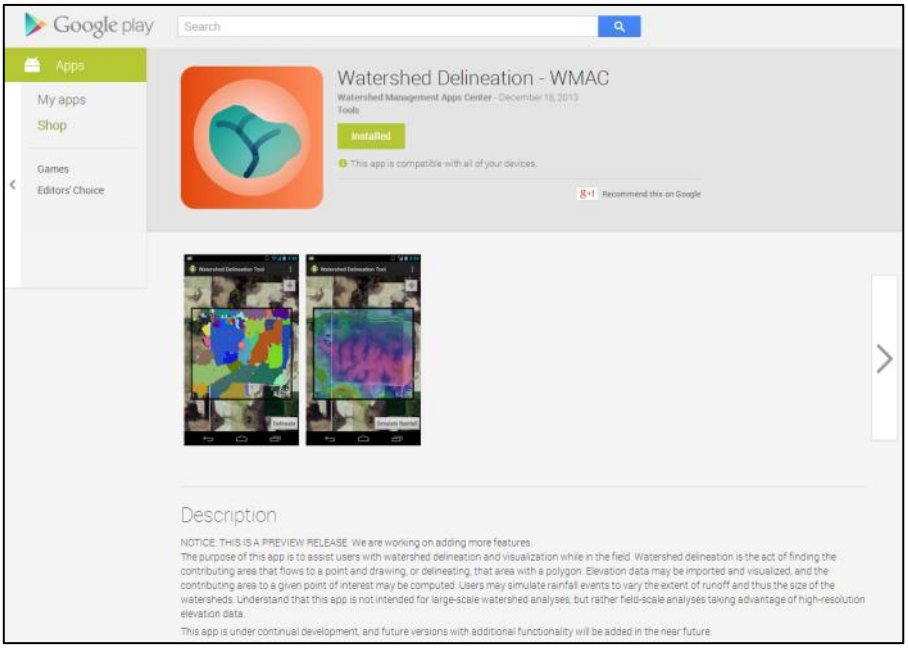


Figure 31. The Watershed Delineation app listing on the Google Play Store.

Similarly, the code for the entire Android application project can be downloaded as a GitHub repository (https://github.com/WaterApps/watershed-delineation-app). Critical Java classes implementing the SDFA are included in Appendix D

6.2 Implementation Verification

Figure 32 displays the catchments produced from a DEM for an agricultural field in Fulton County, Indiana before filling any depressions, and after filling depressions with 25 and 250 mm of rainfall. Before rainfall begins, the set of catchments produced by the two implementations match identically with 5427 depressions that must be filled and another 1352 polygons that run off of the DEM edges. After 10 mm of rainfall, it can be observed that the same polygons exist in each of the Matlab and Android catchment grids (Figures 32e and f). This indicates that the functionality of the rainfall-based SDFA has been implemented identically between the two. Furthermore, the final catchment grids produced after filling all depressions (250 mm) are identical.

Some differences are visible when comparing the catchment grids produced from the two implementations. These differences could have originated from any number of sources including the associated coding languages and the differences in implementation of the two platforms themselves. For example, in Matlab, the grid data is indexed with zero on the left and increases to right. However, in Java, an image read into memory is indexed in the opposite direction. One must reverse the order of all FOR loops with respect to only the x-axis to mimic the opposite platform. For example, when assigning flow directions, in the case of multiple eligible downhill neighbors of the same distance-weighted

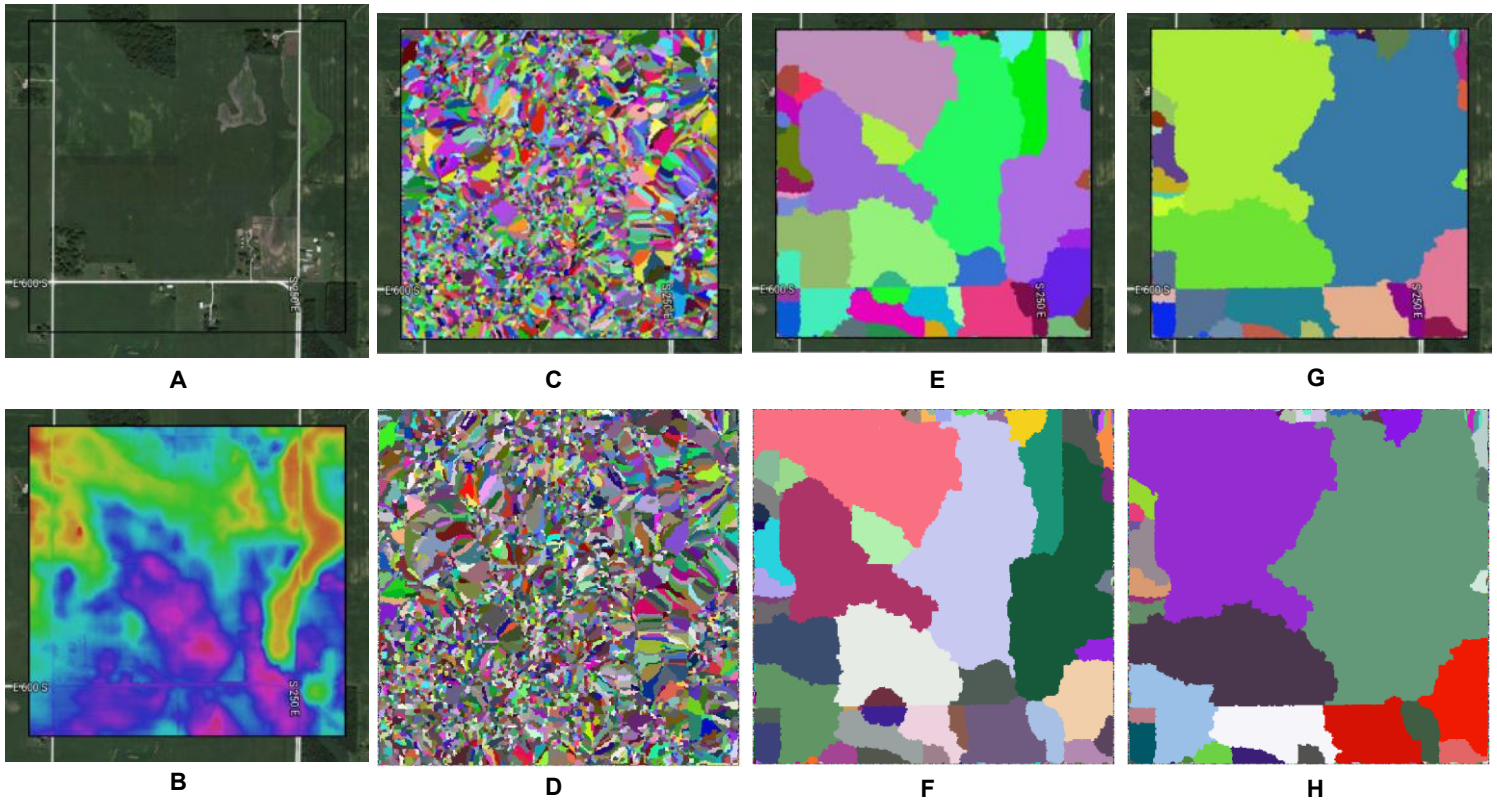


Figure 32. Verification of implementation between Matlab and Android. A) Imagery, B) DEM, and C-E) catchment grids produced from varying rainfall. C) Android 0 mm, D) Matlab 0 mm, E) Android 25 mm, F) Matlab 25 mm, G) Android 250 mm, H) Matlab 250 mm

drop, the first cell encountered is taken as the direction of flow; if traversing the data in the opposite direction with respect to the x-axis, different results will be obtained.

Differences may have also been encountered if attention was not given to all special cases. As explained in previous chapters, datasets with flat topography where differences in elevation between adjacent grid cells are at or below the vertical accuracy of the dataset results in several special cases which must be handled by the algorithm. Because the DEM analyzed in Figure 32 contains flat areas, these special cases are magnified, particularly when a comparison is made between implementations on two different platforms (i.e. Matlab on a PC versus Java on an Android device).

A number of special cases must be handled due to limited vertical precision such as the assignment of flow direction and the identification of spillover locations. Additionally, elevations were rounded to the nearest centimeter because the dataset had a vertical accuracy of about 15 cm. Any additional digits of precision could not be relied on, and, instead, any flat areas resulting from this loss of precision should be handled by the algorithm. With only centimeter precision, it becomes increasingly likely to have multiple depressions with the same volume and that require equivalent amounts of time to overflow if they also have the same area (which is quite common at the initial state of connectivity). As a result, the sequence that depressions are filled becomes random between depressions with the same spillover time, and differences in the sorting functions of Matlab and Java may lead to different results.

6.3 DEM Size Performance Relationship Testing

Table 4 presents the number of depressions and run times for various DEM sizes and resolutions. The number of depressions appears to be linearly related the DEM area, confirming that depressions are relatively uniformly distributed across the chosen area. For example, increasing DEM size from 125 x 125 to 250 x 250, a factor of four in area, similarly results in an increase in the number of depressions from 1715 to 6971, roughly a factor of four. However, with the increase in depressions, performance is impacted exponentially; the increase from 1715 to 6971 results in a sharp reduction in performance from two seconds to 94 seconds. Unfortunately, this means particular attention must be given to the size of the DEM to ensure that performance isn't affected usability.

Table 4. Performance Evaluation of SDFA on Android Device

	1-meter Resolution			3-meter Resolution		
DEM Size	DEM Area (hectares)	Number of Depressions	Run Time (s)	DEM Area (hectares)	Number of Depressions	Run Time (s)
125 x 125	1.56	1715	2.07	14.31	390	0.91
250 x 250	6.25	6971	94.0	56.25	1993	5.79
500 x 500	25	25682	1280 (21 min)	225	5032	67.5
1000 x 1000	100	---	---	900	27346	1340 (22 min)

6.4 Instructional

The Watershed Delineation app and the Water Plane app were utilized in a classroom setting as a lab assignment for students in Agricultural Systems Management 336: Environmental Systems Management (Appendix C, ASM 336 Lab Assignment). The Water Plane app is a relatively uncomplicated tool which

allows for the visualization of an area for which a DEM has been acquired. A virtual “water plane” of constant elevation across the entire DEM may be raised and lowered with the swipe of a finger along a slider. As the water plane moves, cells in the DEM are colored blue to indicate that they are below the water plane and left transparent (displaying imagery) when above the elevation of the water plane. To better conceptualize this, imagine it were the sea level raising or lowering.

The Water Plane app allows rapid identification the highest areas in the field, ridgelines, depressions, and channel or valley-shaped features. A marker locked to the users’ position can read out their current elevation and additional markers can be set so that relative elevations can be tracked between their elevation, the elevation of the water plane, and the elevations of these other markers throughout the DEM. This smooth, simple functionality utilizing high-resolution LiDAR-based DEMs makes for a wide range of potential uses such as planning a building site, familiarizing oneself with new terrain, etc. User interface designs are consistent between this app and the Watershed Delineation app in order to give users a sense of familiarity and intuitiveness when used together. For example, both applications display the available DEM bounding boxes on the map and allow users to click on them to load them as the operational DEM.

The assignment exposed students to mobile technologies to demonstrate how tools specially designed to be used in the mobile context (i.e. apps) can provide a new outlook when performing various tasks. The usage of the apps in this setting also demonstrated the functionality of the applications and served as

a means to receive some preliminary feedback regarding the usability of the applications.

In the assignment, students begin by using the Water Plane app to identify low spots in a particular field as potential wetland sites or depressions that may require tile drainage. Next, students begin the preliminary design of a grassed waterway by identifying a channel-like feature in the topography with the Water Plane app. The assignment then made use of the Watershed Delineation app to find the area draining to a culvert where the channel is intersected by a road. Students estimate runoff using the SCS Curve Number Method.

CHAPTER 7. CONCLUSIONS, RECOMMENDATIONS, AND FUTURE WORK

7.1 Conclusions

An algorithm was developed which fills depressional features sequentially as they would fill with water given an input rainfall event. This type of algorithm is advantageous in the presence of natural depressions which may alter surface flow patterns from those produced using existing watershed delineation methods. The algorithm enables field-scale analyses by accounting for field-scale topography and sub-surface drainage features in a dynamic way, providing a wide range of field-scale connectivity conditions to affirm users' on-site observations. Furthermore, modifications to the flow direction implementation have been made to allow connectivity between non-adjacent cells such as in the case of tile inlets draining a depression or a culvert draining water across a road.

The algorithm functions identically to the Jenson and Domingue (1988) method used in ArcMap with the exception of the sequential depression-filling component. In the cell-by-cell comparison of 37 watershed delineations across four different DEMs, an average percent difference of 5.8% was found between those watersheds produced from the SDFA and ArcGIS Hydrology Toolset with an overall percent difference of 3.11%. That is, in the scenario where all depressions are filled, the same depressions are identified within the DEM and

filled to equivalent elevations. Flow is routed similarly across the landscape and rerouted through the depression's minimum spillover location after they are filled. Minor differences are present, primarily in regarded to how a flow direction is decided in the case of flat areas, and it is not possible to say which is better.

An applicability study was performed to look for any major patterns in the response of hydrologic connectivity as a function of rainfall excess across various landscapes and DEM sizes. It was found that landscapes vary widely in the extent of hydrologic connectivity as a function of rainfall excess. While some landscapes exhibit no change in hydrologic connectivity based on rainfall, others with prominent natural depressions may require in excess of the 50-year return-period rainfall event in order to fill all depression and attain the complete hydrologic connectivity that is assumed by the current filling algorithms. Accounting for infiltration and losses further exaggerates the effects of depressions on hydrologic connectivity.

For those landscapes that fluctuate noticeably with natural or man-induced depression features, it was determined in Section 4.5 that DEM size is an important consideration. At small DEM sizes, depressions are not completely captured in the DEMs, causing full connectivity to be approached rapidly. In this case, edge effects comprise a majority of the DEM area. At medium scales, DEMs may be captured, but the areas contributing to them may not be, and, as a result, more rainfall excess may be required to achieve complete connectivity. Finally, at large scales, depressions and their contributing areas may be captured in the DEMs while edge effects occupy only a small proportion of the DEM. As a

result, the rainfall excess required to fill all depressions at the largest scales (232 hectares) was found to be less than medium scales (100 hectares) but greater than small scales (less than 50 hectares).

Finally, the algorithm has been implemented as a mobile application for the Android operating system. With the exception of drainage features, the full rainfall-based SDFA is carried out on mobile devices in the Watershed Delineation app. The Watershed Delineation app is free and publicly available on the Google Play Store. When using the application, DEM size should be considered carefully (based on resolution, e.g. limited to ~300 hectares at 3-meter resolution) to avoid experiencing a drop in performance.

7.2 Recommendations

Two conditions are recommended in order to acquire a good representation of field-scale surface flow: 1) the DEM should be large enough such that the entire contributing area of any desired features are likely to be contained in the DEM and 2) tile inlets and culvert connections should be known. The first condition may be challenging because this requires some knowledge of the solution before it has been found unless a very large DEM is acquired. This may encourage users to download large DEMs, but this is undesirable because of the greater processing and memory requirement associated with large DEMs. Given the results of Section 4.6 regarding suggested DEM size, it is recommended to acquire DEMs greater than 1.5 km^2 in size. However, those analyses utilized DEMs at a 1.5-meter resolution which may be unnecessarily high resolution for hydrologic modelling, even at the field scale. While some features that affect

drainage patterns such as small roadside ditches may not be captured even at 1.5-meter resolution, a DEM at 3-meter resolution will be a quarter of the file size while it will still capture features such as streams, ditches, and gently rolling field topography. As far as identifying tile inlets and other drainage features, a moderate rainfall event (e.g. 20 - 50 mm) may be simulated and any isolated catchments may be further investigated for the presence of such drainage features.

Properly accounting for drainage features makes the delineation algorithm into a time-based model. Although infiltration and losses were implemented in this research using the curve number method, rate-based soil infiltration could be implemented in a way similar to drainage features. This, however, starts down the path of a field scale, finite-element, completely distributed model that attempts to mimic and simulate the entire natural system. The current implementation of time-based depression filling (i.e. rainfall duration, intensity, and rates of drainage) completely ignores the fact that rainfall intensity is not uniform over a given storm and changes in soil moisture and infiltration over time should likely be accounted for to maintain model integrity at such a level of detail. Moreover, the case is rare that rainfall exceeds the drainage rates of a functioning subsurface drainage system to the point that a large surface depression overflows and connectivity is altered. With this in mind, drainage features as currently implemented should be used to enforce specific, known surface connectivity patterns rather than to calculate combatting effects of rainfall intensity to drainage rates.

7.3 Future Work

Currently, the algorithm is only capable of generating a single state of connectivity per run of the algorithm. In the future, an ideal interface allowing users to alter the rainfall amount and seamlessly view the corresponding hydrologic connectivity, and delineate watersheds would provide a truly powerful user experience. While this functionality is more challenging, it could be achieved in several ways. One option would require storing the history of each depression merger and the corresponding flow direction information. This option is immensely memory-intensive under the current data structure which requires gridded datasets as well as a database-like array of parameters for each existing depression; the most basic implementation of this would require duplicating these datasets at each state of connectivity. To improve upon this, a clever tree structure may be investigated that stores the hierarchy of depressions (e.g. Depressions 1 and 2 form Depression 3) along with the critical parameters to derive the corresponding gridded datasets at any desired state of connectivity.

Alternatively, the data may be precomputed and served to users over the internet. Post-SDFA flow direction grids may be delivered for various rainfall events or only critical design-storm rainfall events and, from this watershed delineations and the hydrologic connectivity grid can be derived. Caching and local storage would allow the data to be acquired before going to the job site in remote areas lacking WiFi or cellular internet connectivity. By removing the computational burden of the algorithm from the mobile device, a more complicated algorithm may be devised which applies the SDFA at larger scales

and accounts for infiltration, non-uniform rainfall, etc. This would enable watershed delineations at all scales, including delineations performed in streams and rivers. The algorithm would be run only once to generate the necessary datasets, after which it will be stored on a server. However, this will limit opportunities for the user to respond to on-site observations and alter inputs to the algorithm such as drainage features, topographic modifications, and infiltration. Again, this method would require a server to host the data with and a folder structure (if not a server with geospatial capabilities) that will allow for spatial queries to access tiled subsets of the data for a particular areas.

Another challenge that is faced concerns the ability to delineate larger streams and watersheds that may span outside of the area of interest when a field-scale DEM is analyzed. For example, Figure 14 (Section 4.14) shows a DEM of a large field with a ditch that runs across the northern edge. Because the DEM does not include the full upstream area draining to this ditch, the watershed area will be underestimated for a delineation performed in the ditch.

Alternative data structures such as TINs may also be investigated for possibilities to make gains in memory or computational performance over gridded DEM data. Unlike gridded DEM data which can be redundant in flat areas, a TIN-based data structure is more flexible in that it allows points to be spaced sparsely across invariable topography and more densely in areas with high topographic variability such as stream channels. Furthermore, ridge features, which are important in the determination of water flow, can be represented more precisely when elevation samples are associated with a specific point (as in a

TIN) rather than a small square plane (as in a gridded DEM). However, gridded DEMs have an inherent x-y spacing from which the geospatial location can be easily determined given the location of a corner point. To sustain the same file size, the TIN must use one third of the amount of points from the gridded DEM in order to store x and y positional information.

Additionally, there is room for improvement on the implementation of drainage features and non-adjacent flow connections in the algorithm. Currently, these drainage features are treated as infinite sinks for the purpose of enforcing known surface flow patterns (e.g. a particular depression will never overflow due to a tile inlet). In the future, the outlets of such drainage features should be accounted for so that the downstream hydrologic response more correctly reflects the linkages made by these features. In the future, a database of these known drainage feature connections may be of value for studies analyzing the impacts of increased hydrologic connectivity due to anthropogenic structures and drainage tile inlets.

As far as utilizing mobile applications to affect watershed management decisions in the field, watershed delineations are only one of many inputs into solving watershed management issues. After a watershed has been delineated, this information should be fed into a hydrologic model to analyze various watershed management options be they alternative practices or structures. Under a sister USDA-NIFA grant directed toward mobile apps for farm management, efforts have been made toward a note-taking mobile application called the Field Notebook. A map-view provides utilities that allow users to draw

points, lines, and polygons to which text and pictures may be associated. These notes may be searched and organized either spatially or in a list-view.

Such an app should prove useful to conservation agents whose work relies on variety of spatial notes. While the types of notes taken will differ, most of the design elements of the Field Notebook app can be shared between these two open-source projects. The primary difference will be that this app may also be utilized as the point of entry for specific, required hydrologic model inputs. It may be easiest to use this app to record observed land use information, record agricultural management decisions after speaking with the farmer, or look up rainfall and soil-type data for the area of interest.

As stated, this data would then be fed into a hydrologic model provide estimates of the risk associated with various management decisions (e.g. fertilizer and pesticide application) and the efficacy of various best management practice options. Similar methods and principles should be applied in the development of these apps; user inputs should be minimal and all designs should be based on realistic user scenarios recognizing the mobile context.

REFERENCES

REFERENCES

Acushla Antony, I. C. (2013, November 15). *Napra Web*. Retrieved from https://engineering.purdue.edu/napra

Administration, N. O. (2014, June). *NOAA Digital Coast*. Retrieved July 1, 2014, from United States Interagency Elevation Inventory: http://www.csc.noaa.gov/digitalcoast/tools/inventory

Antoine, M., Javaux, M., & Bielders, C. (2009). What indicators can capture runoff-relevant connectivity properties of the micro-topographyh at the plot scale? *Advances in Water Resources 32 (8)*, 1297-1310.

Antonic, O., Hatic, D., & Pernar, R. (2001). DEM-based depth in sink as an environmental estimator. *Ecological Modelling 138*, 247-254.

Antony, A., & Engel, B. A. (2009). Web-Based Decision Support Tool for Nutrient and Pesticide Analysis. *ASABE Annual International Meeting.* Reno, Nevada: ASABE.

Appels, W., Bogaar, P., & van der Zee, S. (2011). Influence of spatial variations of microtopography and infiltration on surface runoff and field scale hydrologic connectivity. *Advances in Water Resources 34 (2)*, 303-313.

Band, L. E. (1986). Topographic Partition of Watershed with Digital Elevation Models. *Water Resources Research*, 15-24.

Blanchoud, H., Moreau-Guigon, E., Farrugia, F., Chevreuil, M., & Mouchel, J. (2007). Contribution by urban and agricultural pesticide uses to water contamination at the scale of the marine watershed. *Science of the Total Environment 34 (2)*, 168-179.

Broscoe, A. J. (1959). *"Quantitative analysis of longitudinal stream profiles of small watersheds", Office of Naval Research, Project NR 389-042, Technical Report No. 18.* New York: Department of Geology, Columbia University.

Burrough, P., & McDonnell, R. (1998). Spatial Information Systems and geostatistics. In P. Burrough, & R. McDonnell, *Principles of Geographical Information Systems* (p. 333). New York: Oxford University Press.

Chu, X., Yang, J., Chi, Y., & Zhang, J. (2013). Dynamic puddle delineation and modeling of puddle-to-puddle filling-spilling-merging-splitting overland flow processes. *Water Resour. Res.*, 3825-3829. doi:10.1002/wrcr.20286

Chu, X., Zhang, J., Chi, Y., & Yang, J. (2010). An improved method for watershed delineation and computation of surface depression storage. *Watershed Management 2010: Innovations in Watershed Management under Land Use and Climate Change*, (pp. 1113-1122).

Collins, S. B. (1975). Terrain parameters directly from a digital terrain model. *The Canadian Surveyor 29*, 507-518.

Costa-Cabral, M. C., & Burges, S. J. (1994). Digital elevation model networks (DEMON): A model of flow over hillslopes for computation of contributing and dispersal areas. *Water Reesources Research*, 29.

Darboux, F., Davy, P., & Gascuel-Odoux, C. (2002). Effect of depression storage capacity on overland flow generation for rough horizontal surfaces: water transfer distance and scaling. *Earth Suface Processes and Landforms 27*, 177-191.

Darboux, F., Davy, P., Gascuel-Odoux, C., & Huang, C. (2001). Evolution of soil surface roughness and flowpath connectiviy in overland flow experiments. *Catena*, 125-139.

Dhun, K. (2011). *Application of LiDAR DEMs to the Modelling of Surface Drainage PAtterns in Human Modified Landscapes.* University of Guelph.

Eclipse Foundation. (2014). Retrieved June 19, 2014, from Eclipse: http://www.eclipse.org

Environmental Systems Research Institute (ESRI). (2013). ArcGIS Desktop: Release 10.2. Redlands, CA, United States.

Fairfield, J., & Leymarie, P. (1991). Drainage networks from grid digital elevation models. *Water Resources Research, 27(5)*, 709-717.

Foundation, O. S. (2014). *GDAL/OGR In Java.* Retrieved June 19, 2014, from GDAL: http://trac.osgeo.org/gdal/wiki/GdalOgrInJava

Freeman, T. G. (1991). Calculating Catchment Area with Divergent Flow Based on a Regular Grid. *Computers and Geosciences, 17(3)*, 413-422.

Garbrecht, J., & Martz, M. W. (1997). The assignment of drainage direction over flat surfaces in raster digital elevation models. *Journal of Hydrology*, 204-213.

Google. (2014). *Android SDK*. Retrieved June 19, 2014, from Android Developers: http://developer.android.com/sdk/index.html

Hayashi, A., & van der Kamp, G. (2000). Modelling and managing critical source areas of diffuse pollution from agricultural land using flow connectivity simulation. *Journal of Hydrology 237*, 74-85.

Heathwaite, A., Quinn, P., & Hewett, C. (2005). Modelling and managing critical source areas of diffuse pollution from agricultural land using flow connectivity simulation. *Journal of Hydrology 304(1-4)*, 446-461.

Heidemann, H. K. (2012, November). Lidar base specification. *U.S. Geological Survey Techniques and Methods, Book 11, chap. B4, 67 p. with appendices, ver 1.2*, http://dx.doi.org/10.3133/tm11B4. doi:http://dx.doi.org/10.3133/tm11B4

Hubbard, D., & Linder, R. (1986). Spring runoff retention in prairie pothole wetlands. *Journal of Soil and Water Conservation 41 (2)*, 122-125.

Indiana University. (2014). *Indiana Spatial Data Portal*. Retrieved August 13, 2013, from Single File Download Interface: http://gis.iu.edu/isdp_dl/map/m10000.html

IndianaMap. (2013). *IndianaMap Framework Data*. Retrieved from http://dx.doi.org/10.5069/G9959FHZ

Jenson, S. K., & Domingue, J. O. (1988). Extracting Topographic Structure from Digital Elevation Data for Geographic Information System Analysis. *Photogrammetric Engineering and Remote Sensing*, 1593-1600.

Lea, N. L. (1992). An aspect driven kinematic routing algorithm. In *Overland Flow: Hydraulics and Erosion Mechanics.* New York: Chapman & Hall.

Lindsay, J., & Creed, I. (2005). Removal of artefact depressions from DEMs: towards a minimum impact appraoch. *Hydrological Processes 19 (16)*, 3113-3126.

Lindsay, J., & Creed, I. (2006). Distinguishing actual and artefact depressions in digital elevation data. *Computers and Geosciences 32*, 1192-1204.

Louchart, X., Voltz, M., Andrieux, P., & Moussa, R. (2001). Herbicide transport to surface waters at field and watershed scales in a mediterranean vineyard area. *Journal of Environmental Quality 30(3)*, 982-991.

MacMillan, R. A., Furley, P. A., & Healey, R. G. (1993). Using hydrological models and geographic information systems to assist with the management of surface water in agricultural landscapes. In *Landscape Ecology and GIS* (pp. 181-209). London: Taylor & Francis.

MacMillan, R., Martin, T., Earle, T., & McNabb, D. (2003). Automated analysis and classification of landforms using high-resolution digital elevation data: applications and issues. *Canadian Journal of Remote Sensing 29 (5)*, 592-606.

Mark, D. (1984). Automated detection of drainage networks from digital elevation models. *Cartographica 21(2-3)*, 168-178.

Mark, D. (1988). Network models in geomorphology. In M. Anderson, *Modelling Geomorphological Systems* (pp. 73-97). New York, NY: Wiley.

Marks, D. M., Dozier, J., & Frew, J. (1984). Automated BAsin Delineation From Digital Elevation Data. *Geo-processing 2*, 299-311.

Martz, L. W., & Garbrecht, J. (1998). The treatment of flat areas and depression in automated drainage analysis of raster digital elevation models. *Hydrological Processes, 12*, 843-855.

Martz, L., & DeJong, E. (1988). CATCH: A FORTRAN program for measuring catchment area from digital elevation models. *Computers and Geosciences 14 (5)*, 627-640.

Martz, L., & Garbrecht, J. (1999). An outlet breaching algorithm for the treatment of closed depressions in a raster DEM. *Computeres & Geosciences 25 (7)*, 835-844.

McCormack, J., Hogg, J., & Hoyle, B. (1993). Feature-based derivation of drainage networks. *International Journal of Geographical Information Systems 7 (3)*, 263-279.

Metcalfe, R., & Buttle, J. (1999). Semi-distributed water balance dynamics in a small boreal forest basin. *Journal of Hydrology 226*, 66-87.

Moore. (n.d.).

Moore, I. D., Grayson, R. B., & Ladson, A. R. (1991). Digital Terrain Modelling: A Review of Hydrological, Geomorphological, and biological Applications. *Hydrological Processes, 5(1)*, 3-30.

Moore, I. D., O'Loughlin, E. M., & Burch, G. J. (1988). A countour-based topographic model for hydrological and ecological applications. *Earth Surfaces Processes Landforms, 13*, 306-320.

Muehrcke, P., & Muehrcke, J. (1998). *Map Use: Reading, Analysis, and Interpretation, Fourth ed.* Madison, WI: JP Publications.

National Oceanic and Atmospheric Administration (NOAA) Coastal Services Center. (2012). *Lidar 101: An Introduction to Lidar Technology, Data, and Applications. Revised.* Charleston, SC: NOAA Coastal Services Center.

O'Callaghan, J., & Mark, D. (1984). The extraction of drainage networks from digital elevation data. *Computer Vision, Graphics, and Image Processing 28*, 323-344.

OpenATK. (2013). Retrieved June 19, 2014, from Open Ag Toolkit: http://www.openagtoolkit.com

OpenATK. (2014). *OpenATKLib.* Retrieved June 19, 2014, from Github: https://github.com/OpenATK/OpenATKLib

OpenTopography. (2012). *OpenTopography*. Retrieved November 14, 2012, from 2011-2013 Indiana Statewide LiDAR: http://opentopography.org

Pan, F., Stieglitz, M., & McKane, R. B. (2011). An algorithm for treating flat areas and depressions in digital. *Water Resources Research, 48, W00L10, doi:10.1029/2011WR010735*.

Probst, J. (1985). Nitrogen and phosphorus exportation in the Garonne basin (France). *Journal of Hydrology 76 (3-4)*, 281-305.

Quinn, P. K., Beven, K., Chevallier, P., & Planchon, O. (1991). The Prediction of Hillslope Flow Paths for Distributed Hydrological Modeling Using Digital Terrain Models. *Hydrological Processes 5*, 59-80.

Rosenberry, D., & Winter, T. (1997). Dynamics of water-table fluctuations in an upland between two prarie-pothole wetlands in North Dakota. *Journal of Hydrology 191*, 266-189.

Rozemeijer, J. (2010). *Dynamics in groundwater and surface water quality: from field-scale processes to catchment-scale monitoring.* Utrecht: Faculty of Geosciences, Utrect University.

Simard, J. G., Beauchemin, S., & Haygarth, P. (2000). Potential for preferential pathways of phosphorus transport. *Journal of Environmental Quality*, 29(1): 97-105.

Speight, J. G. (1968). Parametric description of land form. *In Stewart, G.A., editor, Land Evaluation, Papers of a CSIRO Symposium in Cooperation with UNESCO*, 26-31.

Tarboton, D. G. (1997). A New Method for the Determination of Flow Directions and Upslope Areas in Grid Digital Elevation Models. *Water Resources Research, 33(2)*, 309-319.

Tarboton, D., Bras, R., & Rodriguez-Iturbe, I. (1991). On the extraction of channel networks from digital elevation data. *Hydrological Processes 5*, 81-100.

The MathWorks, Inc. (2013). MATLAB and Statistics Toolbox Release 2013b. Natick, Massachusetts, United States.

Tobler, W. R. (1966). Numerical map generalization and notes on the analysis of geographical distributions. *Michigan Inter-University Community of Mathematical Geographers*, Discussion Paper 8, University of Michigan.

Tribe, A. (1992). Automated recognition of valley lines and drainage networks from grid digital elevation models: a review and a new method. *Journal of Hydrology 139*, 263-293.

Turtola, E., & Jaakkola, A. (1995). Loss of phosphorus by surface runoff and leaching from a heavy clay soil under barley and grass ley in Finland. *Acta Agriculturae Scandinavica B-S P 45(3)*, 159-165.

United States Department of Agriculture. (1986). Urban hydrology for small watersheds SCS Technical Release 55.

Watershed Management Apps Center. (2014). *WMACLib.* Retrieved June 19, 2014, from Github: https://github.com/WaterApps/watershed-delineation-app

Yang, J., & Chu, X. (2012). Effects of DEM resolution on surface depression properties and hydrologic connectivity. *Journal of Hydrologic Engineering, 18(9)*, 1157-1169.

Zhou, Q., & Liu, X. (2004). Analysis of errors of derived slope and aspect related to DEM data properties. *Computers & Geosciences 30*, 369-378.

Zhou, Q., Pilesjo, P., & Chen, Y. (2011). Estimating surface flow paths on a digital elevation model using a triangular facet network. *Water Resources Research, 47, W07522, doi:10.1029/2010WR009961*.

APPENDICES

Appendix A Additional Figures for Section 4.2

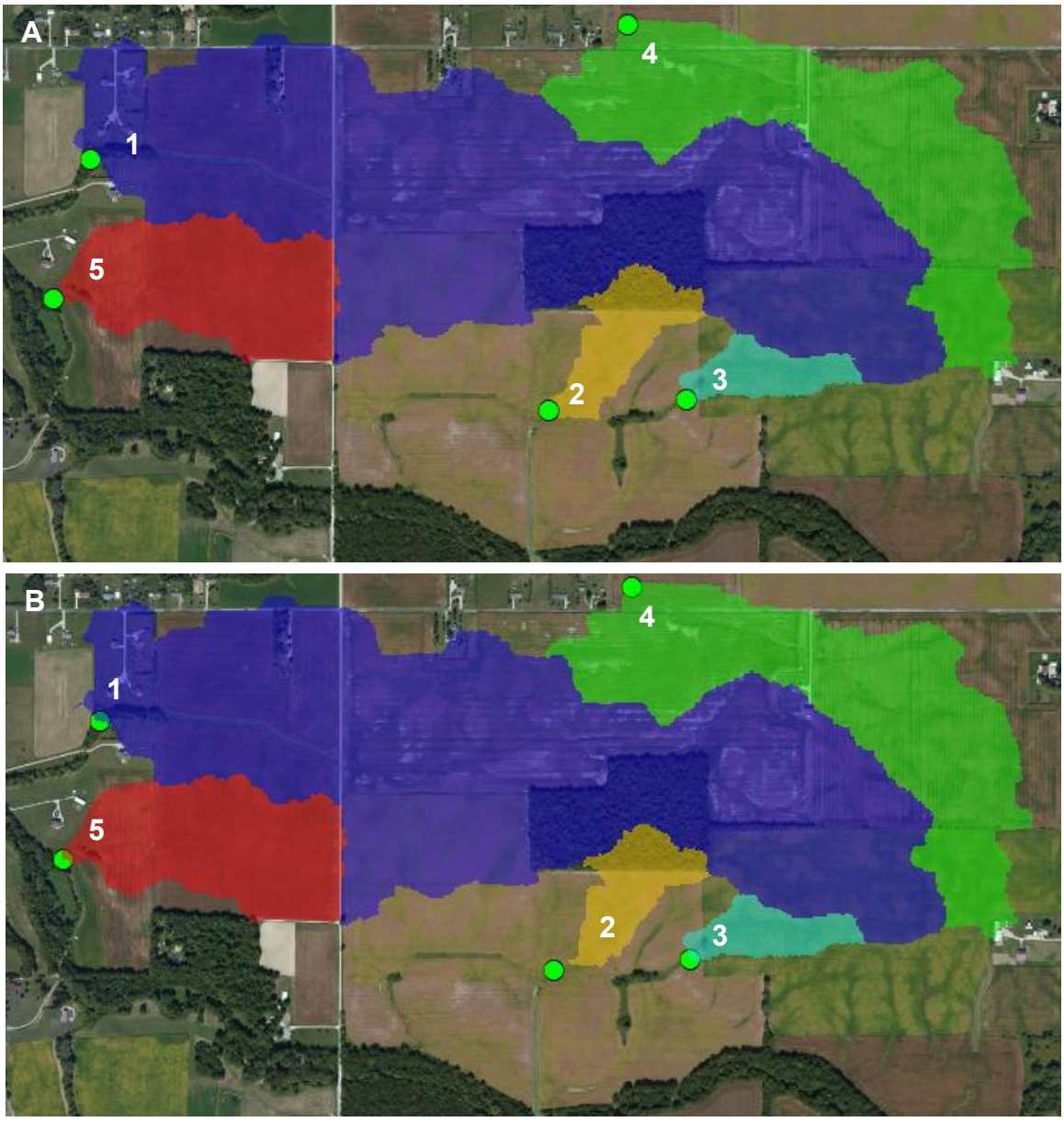


Figure B1. For an agricultural field in Clinton County, Indiana, USA: A) Watershed based on ArcGIS Hydrology toolset. B) Watersheds based on SDFA.

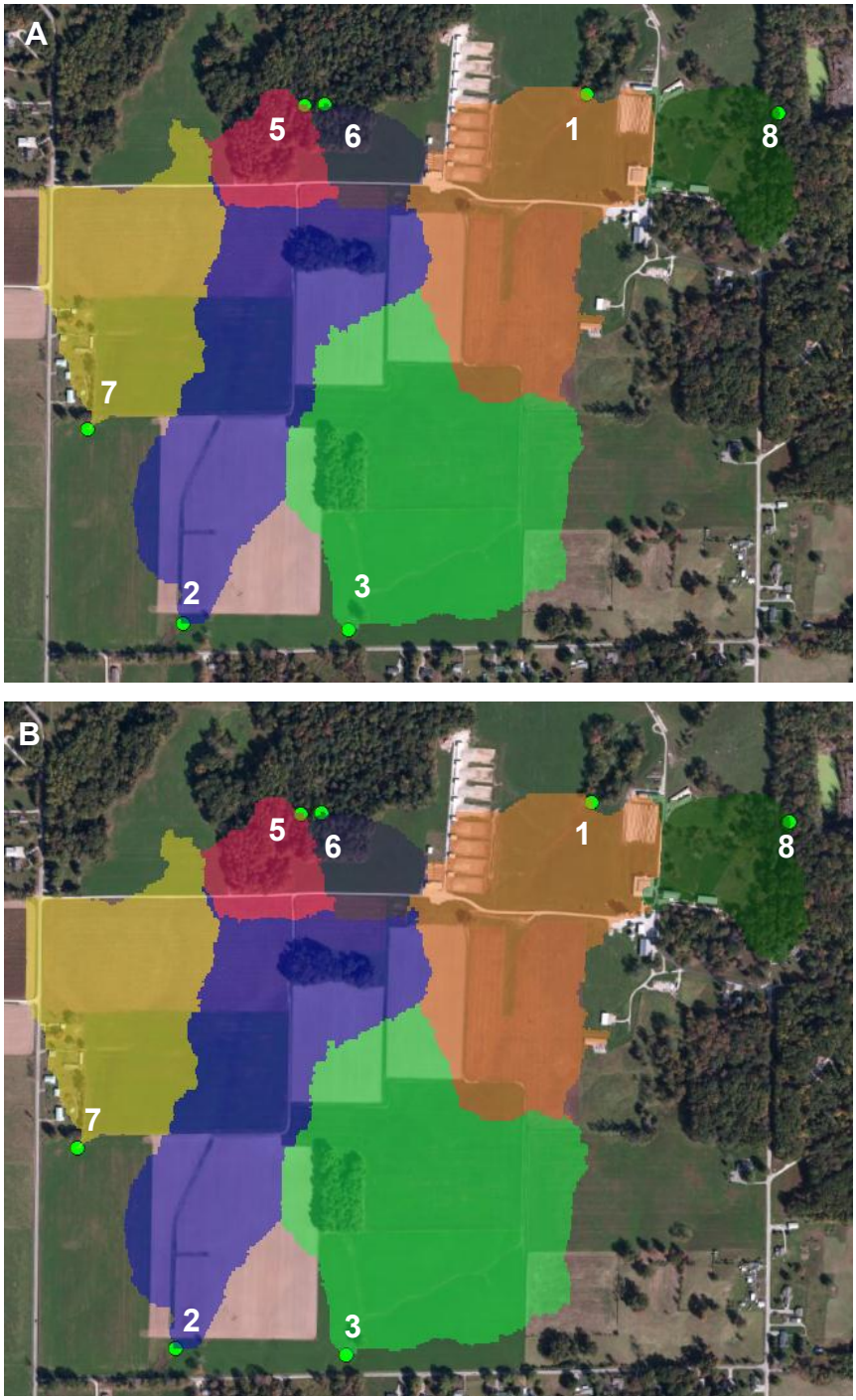


Figure B2. For an agricultural field in Lawrence County, Indiana, USA: A) Watershed based on ArcGIS Hydrology toolset. B) Watersheds based on SDFA.

Appendix B Additional Figures for Section 4.6

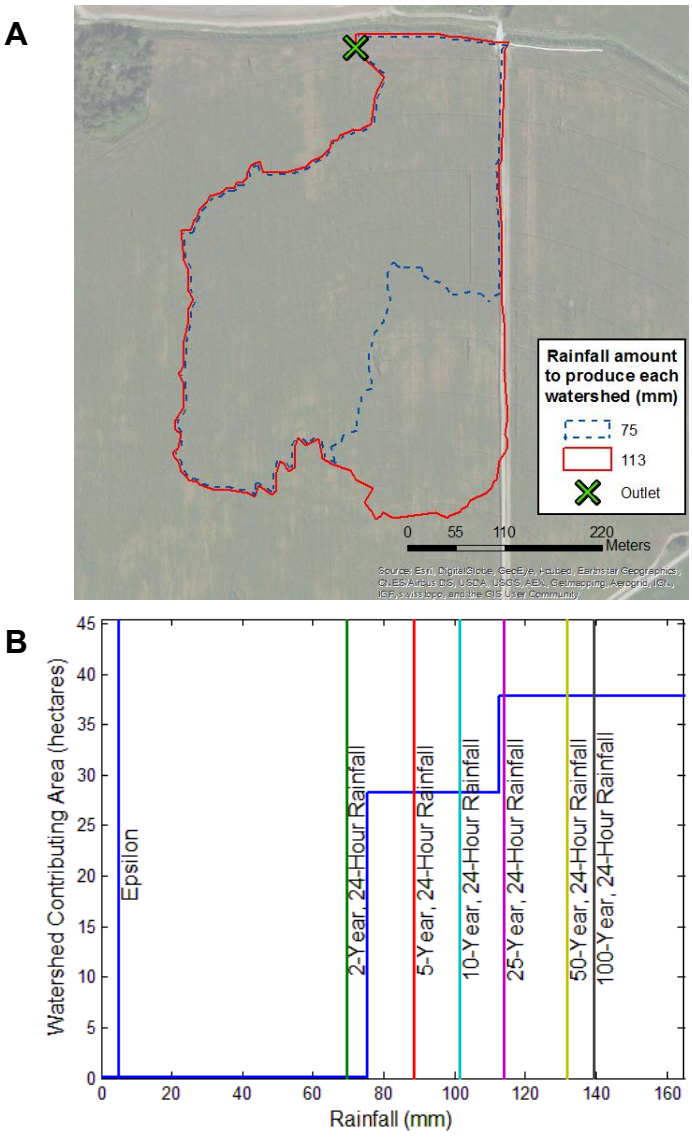


Figure A1. a) A watershed delineation performed in Fulton County, Indiana showing watershed area as a function of rainfall. Losses have been accounted for using the SCS Curve Number method with a curve number of 75. b) The associated full plot of contributing area versus the required rainfall amount to produce that watershed.

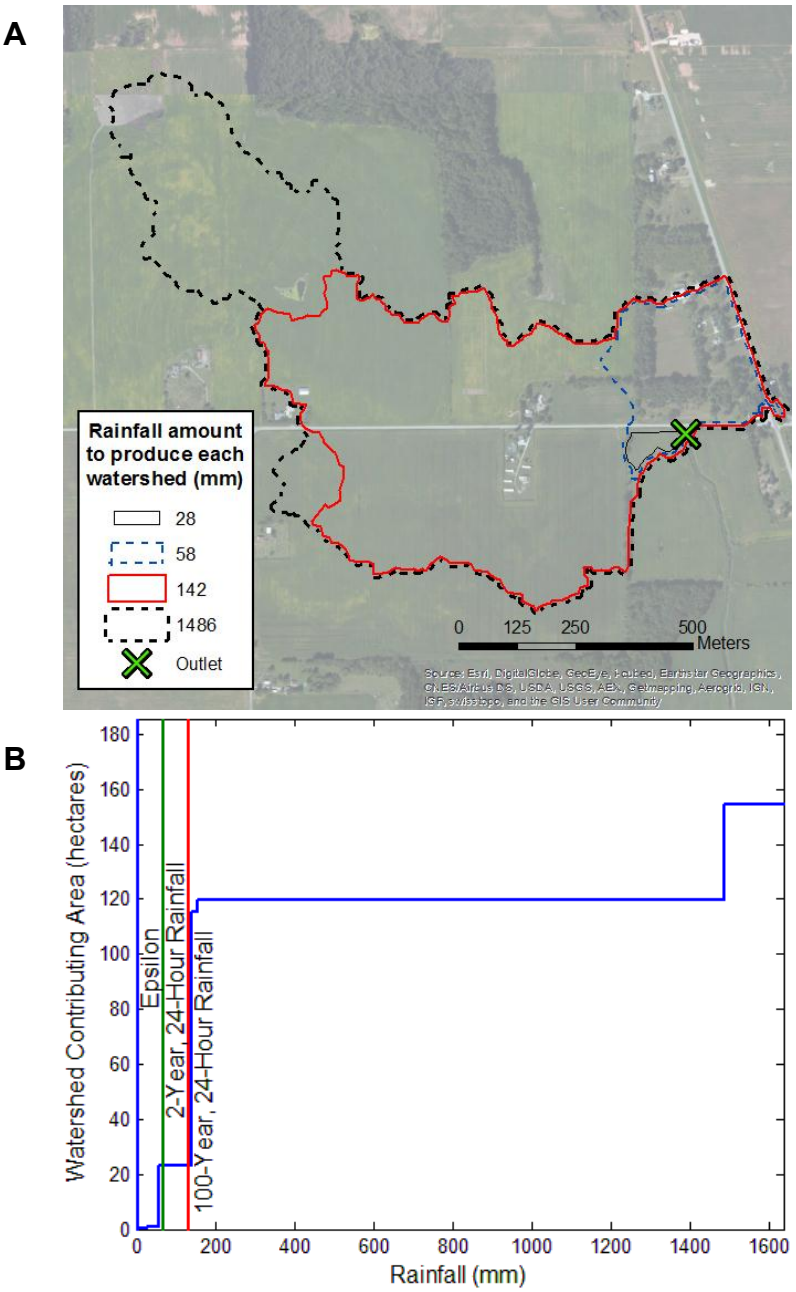


Figure A2. a) A watershed delineation performed in Fulton County, Indiana showing watershed area as a function of rainfall. Losses have been accounted for using the SCS Curve Number method with a curve number of 75. b) The associated full plot of contributing area versus the required rainfall amount to produce that watershed

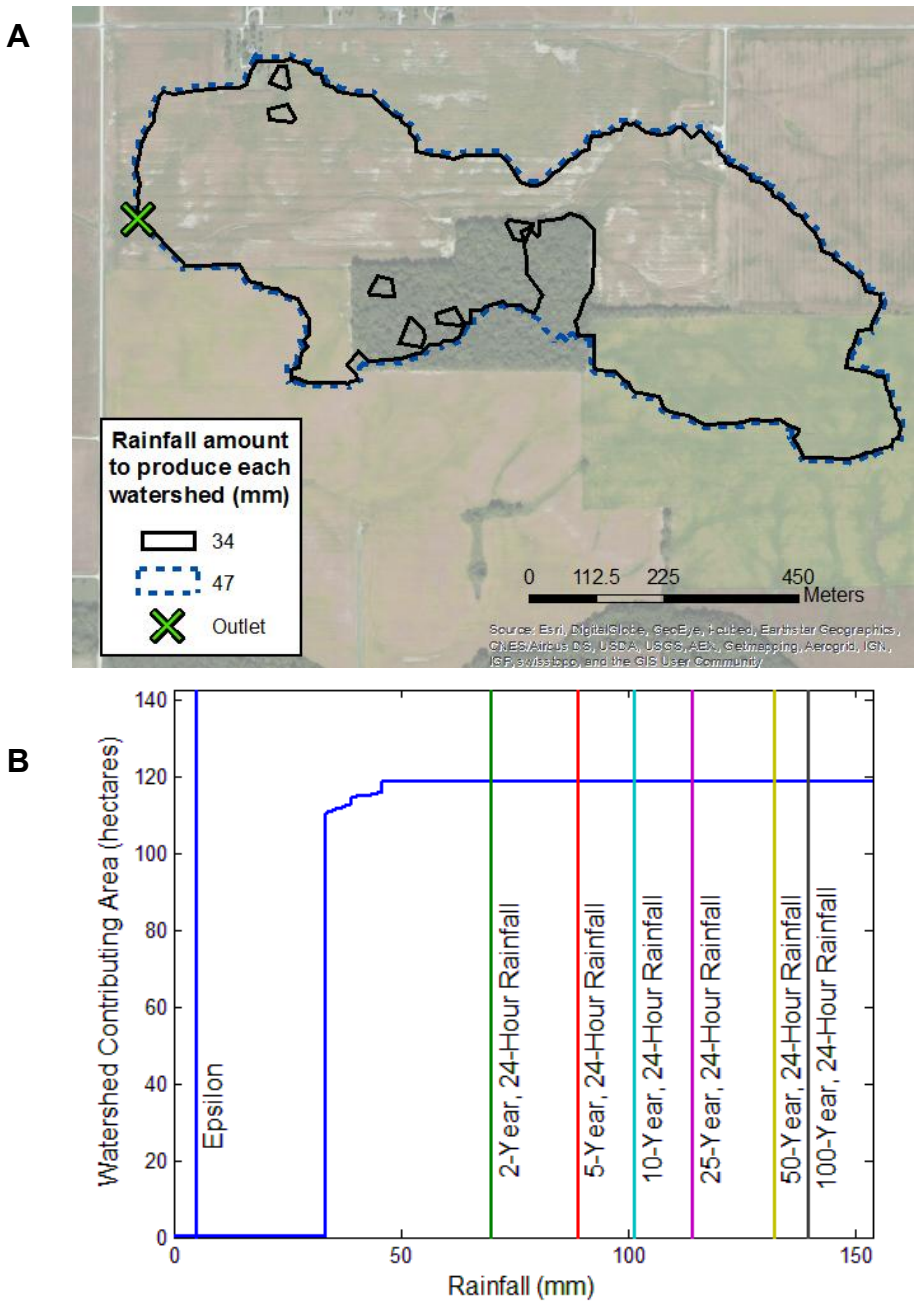


Figure A3. a) A watershed delineation performed in Clinton County, Indiana showing watershed area as a function of rainfall. Losses have been accounted for using the SCS Curve Number method with a curve number of 75. b) The associated full plot of contributing area versus the required rainfall amount to produce that watershed.

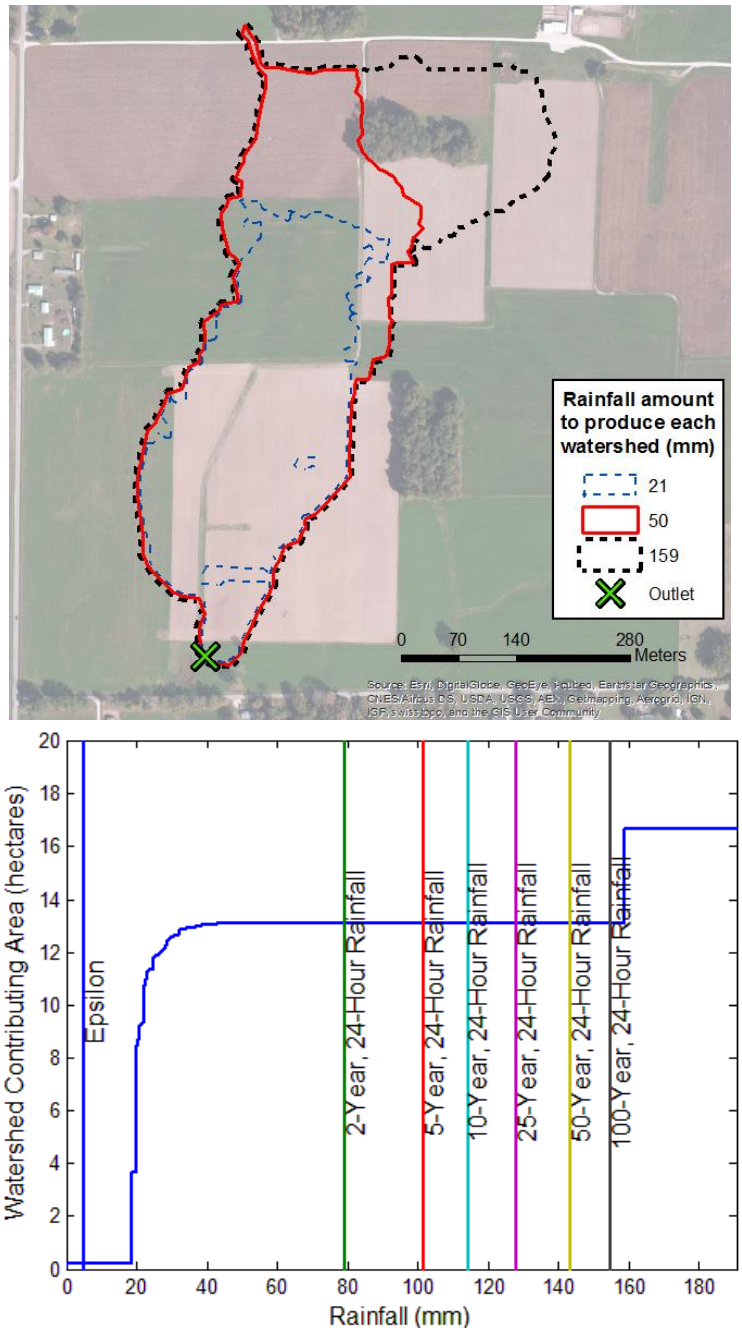


Figure A4. a) A watershed delineation performed in Lawrence County, Indiana showing watershed area as a function of rainfall. Losses have been accounted for using the SCS Curve Number method with a curve number of 75. b) The associated full plot of contributing area versus the required rainfall amount to produce that watershed.

Appendix C ASM 336 Lab Exercise

ASM336 Lab

Applications for Mobile Devices for Water

Learning Outcomes
At the end of this assignment, students should be able to:

1. Identify a potential location for a) a tile riser, b) a wetland, and c) a grassed waterway
2. Delineate a watershed
3. Establish a relationship between precipitation amount and the area inundated/contributing runoff
4. Implement SCS curve number method to estimate runoff volume

You will use two mobile device Applications (Apps), the Water Plane App and the Watershed Delineation App, to achieve the learning outcomes. These Apps are intended to be used as tools to assist your on-site observations, so keep this in mind as you proceed. Toshiba Thrive tablets will be supplied to use, but you are encouraged to use your personal Android devices.

Android Device Crash Course:
There are typically 3 navigation buttons on all Android devices – the back button, home button, and the Application drawer that allows you to switch between opened Applications. The home page typically has an icon for the *Google Play Store* which is the Android equivalent of *iTunes Store* where you may view and purchase Apps. There is also an Apps icon which allows you to view all installed Apps. Device settings such as WiFi/GPS access, screen brightness, etc. may be accessed by pressing in the lower right by the time and then pressing Settings.

Log onto PAL for Wifi Access
Go to the Settings screen and select Wireless and networks on the upper left side. From the right panel select Wi-Fi settings. Select PAL3.0 and select Forget. Find PAL3.0 again in the list of available networks and select it. Enter your login information – your Purdue Career Account ID and password.

Install the Apps
Go to the Watershed Management Apps Center website at www.waterapps.org. The two apps are shown in the upper right, and are linked to their *Google Play Store* listings. Download the apps on your device.

Open the Apps
Upon installation, icons of installed Apps are placed on the home screen. They may also be found using the Apps icon to list all installed Apps. Open them! We will start with the Water Plane App.

The Water Plane App

The Water Plane App is primarily a visualization and surveying tool that makes use of publicly available, state-wide high resolution elevation data. Using the slider at the bottom of the screen, the water level may be raised and lowered while the map shows how this water level intersects the landscape. By doing this, the highest and lowest regions may quickly be identified, and one may be able to get a feel for the landscape. If a puddle-like isolated area of water forms as you raise the water plane, then it is likely that this is a low spot surrounded by higher elevation, perhaps a suitable location for a tile drain or wetland. If you raise the water plane and visible “fingers” are present, these areas are likely channels where the surrounding area drains.

The Watershed Delineation App

A watershed is the area that drains to a given location. The Watershed Delineation App is a tool developed to assist in resolving water issues in the field. As you may expect, the area that drains to a given location depends on the amount of precipitation that falls upon the area; natural depressions are present in the terrain, capable of retaining water, and only after they puddle and overflow do they drain further downhill or downstream. Users may select a rainfall amount to simulate on the landscape using the Menu button in the upper right, then selecting Settings, then editing the Rainfall Event.

After selecting a rainfall amount, the event may be simulated using the Simulate Rainfall button. Following the rainfall simulation, several layers are displayed which may be toggled on or off from the Menu. The Puddles layer displays puddles in the field. The Catchments layer displays each area that drains together as a uniquely colored polygon. However, at times, you may want to know how an area drains within a subsection of these polygons. The marker placed on the map may be dragged (hold your finger on it for a moment and it becomes draggable), and the watershed draining to that location is then delineated.

Figure B-5 10-year, 24-hour rainfall

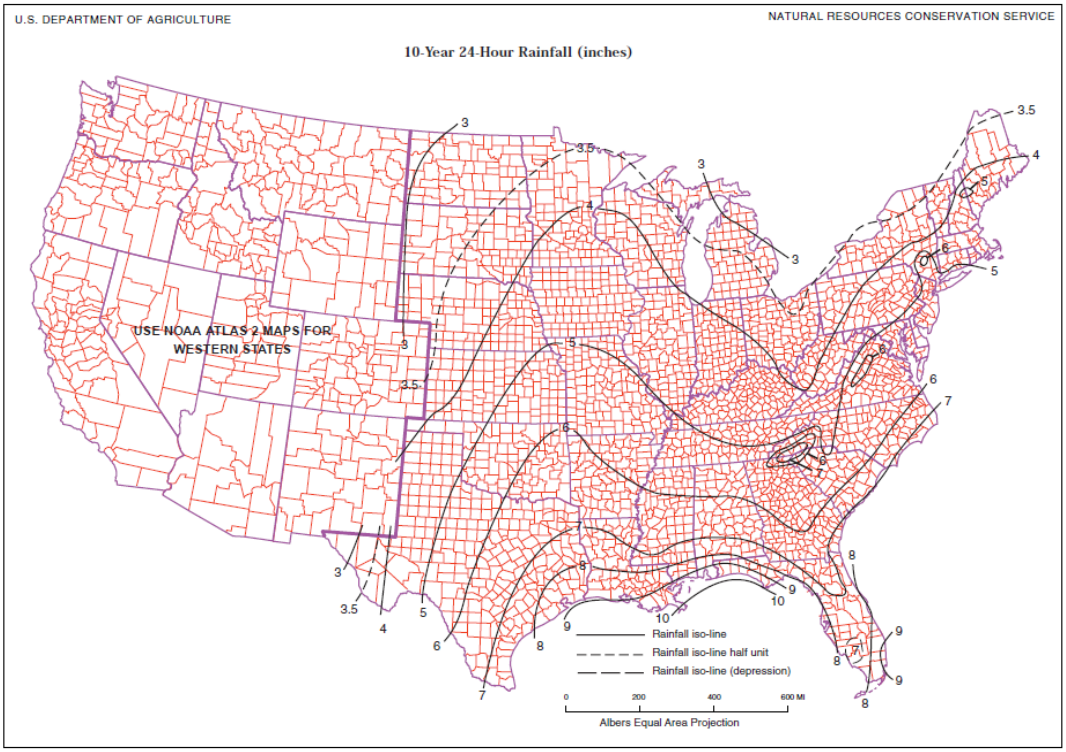


Figure 1. 10-Year 24-Hour Rainfall Map.
Reference: www.cpesc.org/reference/tr55.pdf (page 148, note: additional maps available).

Table 2-2b Runoff curve numbers for cultivated agricultural lands[1/]

Cover description			Curve numbers for hydrologic soil group			
Cover type	Treatment[2/]	Hydrologic condition[3/]	A	B	C	D
Fallow	Bare soil	—	77	86	91	94
	Crop residue cover (CR)	Poor	76	85	90	93
		Good	74	83	88	90
Row crops	Straight row (SR)	Poor	72	81	88	91
		Good	67	78	85	89
	SR + CR	Poor	71	80	87	90
		Good	64	75	82	85
	Contoured (C)	Poor	70	79	84	88
		Good	65	75	82	86
	C + CR	Poor	69	78	83	87
		Good	64	74	81	85
	Contoured & terraced (C&T)	Poor	66	74	80	82
		Good	62	71	78	81
	C&T+ CR	Poor	65	73	79	81
		Good	61	70	77	80
Small grain	SR	Poor	65	76	84	88
		Good	63	75	83	87
	SR + CR	Poor	64	75	83	86
		Good	60	72	80	84
	C	Poor	63	74	82	85
		Good	61	73	81	84
	C + CR	Poor	62	73	81	84
		Good	60	72	80	83
	C&T	Poor	61	72	79	82
		Good	59	70	78	81
	C&T+ CR	Poor	60	71	78	81
		Good	58	69	77	80
Close-seeded or broadcast legumes or rotation meadow	SR	Poor	66	77	85	89
		Good	58	72	81	85
	C	Poor	64	75	83	85
		Good	55	69	78	83
	C&T	Poor	63	73	80	83
		Good	51	67	76	80

[1] Average runoff condition, and I_a=0.2S

[2] Crop residue cover applies only if residue is on at least 5% of the surface throughout the year.

[3] Hydraulic condition is based on combination factors that affect infiltration and runoff, including (a) density and canopy of vegetative areas, (b) amount of year-round cover, (c) amount of grass or close-seeded legumes, (d) percent of residue cover on the land surface (good ≥ 20%), and (e) degree of surface roughness.

Poor: Factors impair infiltration and tend to increase runoff.

Good: Factors encourage average and better than average infiltration and tend to decrease runoff.

Figure 2 - SCS Curve Number Method – Curve Number Table.
Reference: www.cpesc.org/reference/tr55.pdf (page 18).

Chapter 2 Estimating Runoff

SCS runoff curve number method

The SCS Runoff Curve Number (CN) method is described in detail in NEH-4 (SCS 1985). The SCS runoff equation is

$$Q = \frac{(P - I_a)^2}{(P - I_a) + S} \qquad \text{[eq. 2-1]}$$

where

Q = runoff (in)
P = rainfall (in)
S = potential maximum retention after runoff begins (in) and
I_a = initial abstraction (in)

Initial abstraction (I_a) is all losses before runoff begins. It includes water retained in surface depressions, water intercepted by vegetation, evaporation, and infiltration. I_a is highly variable but generally is correlated with soil and cover parameters. Through studies of many small agricultural watersheds, I_a was found to be approximated by the following empirical equation:

$$I_a = 0.2S \qquad \text{[eq. 2-2]}$$

By removing I_a as an independent parameter, this approximation allows use of a combination of S and P to produce a unique runoff amount. Substituting equation 2-2 into equation 2-1 gives:

$$Q = \frac{(P - 0.2S)^2}{(P + 0.8S)} \qquad \text{[eq. 2-3]}$$

S is related to the soil and cover conditions of the watershed through the CN. CN has a range of 0 to 100, and S is related to CN by:

$$S = \frac{1000}{CN} - 10 \qquad \text{[eq. 2-4]}$$

Figure 2-1 and table 2-1 solve equations 2-3 and 2-4 for a range of CN's and rainfall.

Factors considered in determining runoff curve numbers

The major factors that determine CN are the hydrologic soil group (HSG), cover type, treatment, hydrologic condition, and antecedent runoff condition (ARC). Another factor considered is whether impervious areas outlet directly to the drainage system (connected) or whether the flow spreads over pervious areas before entering the drainage system (unconnected). Figure 2-2 is provided to aid in selecting the appropriate figure or table for determining curve numbers.

CN's in table 2-2 (*a* to *d*) represent average antecedent runoff condition for urban, cultivated agricultural, other agricultural, and arid and semiarid rangeland uses. Table 2-2 assumes impervious areas are directly connected. The following sections explain how to determine CN's and how to modify them for urban conditions.

Hydrologic soil groups

Infiltration rates of soils vary widely and are affected by subsurface permeability as well as surface intake rates. Soils are classified into four HSG's (A, B, C, and D) according to their minimum infiltration rate, which is obtained for bare soil after prolonged wetting. Appendix A defines the four groups and provides a list of most of the soils in the United States and their group classification. The soils in the area of interest may be identified from a soil survey report, which can be obtained from local SCS offices or soil and water conservation district offices.

Most urban areas are only partially covered by impervious surfaces: the soil remains an important factor in runoff estimates. Urbanization has a greater effect on runoff in watersheds with soils having high infiltration rates (sands and gravels) than in watersheds predominantly of silts and clays, which generally have low infiltration rates.

Any disturbance of a soil profile can significantly change its infiltration characteristics. With urbanization, native soil profiles may be mixed or removed or fill material from other areas may be introduced. Therefore, a method based on soil texture is given in appendix A for determining the HSG classification for disturbed soils.

Figure 3. SCS Curve Number Method – Equations.
Reference: www.cpesc.org/reference/tr55.pdf (page 13).

Name: _____

Computer Lab, #3

ASM 336

Assignment

1. Use the Water Plane App to identify potential locations for a) a tile riser, b) a wetland, and c) a grassed waterway using the instructions given. Describe what made this location a suitable location (based on the App, not indications from the Google Maps imagery). Label these points on the provided map view with the letters A, B, and C, respectively.

Figure 1. Location for delineation

2. Gather some preliminary information for a grassed waterway design. Grassed waterways are typically designed using the 10-year, 24-hour storm event. Identify this rainfall value for the Feldun Purdue Agricultural Center using Figure 1 (instructions).

3. The SCS curve number method is an approach to estimate runoff given a generalized set of land use and soil type characteristics. For example,

assume the grassed waterway will be placed in an area that is straight row cropped with corn on hydrologic soil group B soils in good condition. What would be the runoff produced (in inches) as a result of the 10-year, 24-hour storm found in Question 2? The curve number may be found in Figure 2, while the necessary equations are in Figure 3 (instructions).

4. Because the Watershed Delineation App doesn't take infiltration into account, the input rainfall event is actually "effective rainfall," or rainfall minus infiltration and initial abstractions. Use the runoff value found in Question 3 as the input rainfall amount to the Watershed Delineation App. Delineate the area that drains to the point specified in Figure 1 (above). (This could be the base of a grassed waterway.) Record the acreage of this watershed, and multiply this value by the runoff found in Question 3 to get a total volume of runoff (acre-inches of runoff).

5. **OPTIONAL** (bonus): Using the Watershed Delineation App, vary the rainfall amount and look for the changes in watershed area at a given location (Go to Menu > Run New Simulation, then Menu > Settings > Rainfall Amount to prepare a new rainfall simulation. Simulate 0.5", 1.0", and 2.0" of rainfall, and make a plot of the resulting watershed area vs. each of these rainfall amounts. Discuss this relationship.

Appendix D Algorithm Code

This appendix contains the critical Java classes used to implement the SDFA in the Watershed Delineation App on Android devices. The rest of the project code necessary to duplicate the app are available on Github (https://github.com/WaterApps/watershed-delineation-app).

Pit Class

```
package org.waterapps.watershed;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import android.graphics.Point;
import android.util.Log;

public class Pit implements Comparable{
   int pitId; // pit identification number.  Negatives flow off the DEM edge while positive value pits have storage
   Point pitPoint;
//   List<Point> allPointsList; // this is used to speed up several calculations when pits merge: Re-IDing the two merging pits, and finding new retention volume critical to causal sequential depression filling
   List<Point> pitBorderIndicesList; // list of indices along the border of the depression.  Used to find new spillover elevation when merger takes place
   float spilloverElevation = Float.NaN; // threshold elevation before the depression will overflow
   Point pitOutletPoint; // spillover elevation and
   Point outletSpilloverFlowDirection;  // cell to which this pit will overflow
   int area;
```

```
  float retentionVolume; // used to calculate spillover time.  Derived from elevation difference from spillover elevation to pit cells lower than spillover elevation
  float filledVolume; // critical to know when pits merge and calculate
  float spilloverTime; // used to order depressions
  float pitDrainageRate; // NOT USED CURRENTLY (left null) for accumulation rate calculation


  public Pit(int pitId, Point pitPoint) {
    this.pitPoint = pitPoint;
    this.pitId = pitId;
  }


  public void completePitConstruction(List<Point> indicesDrainingToPit, float[][] drainage, float cellSize, float[][] dem, int[][] pitIdMatrix) {
    area = indicesDrainingToPit.size();


    // Border-dependent variables and calculations
    pitBorderIndicesList = new ArrayList<Point>(indicesDrainingToPit);
    for (int i = 0; i < indicesDrainingToPit.size(); i++) {
      Point currentPoint = new Point(indicesDrainingToPit.get(i));
      int r = currentPoint.y;
      int c = currentPoint.x;
      boolean onBorder = false;
      for (int x = -1; x < 2; x++) {
        for (int y = -1; y < 2; y++){
          if (x == 0 && y == 0) {
            continue;
          }
          if (currentPoint.y+y > pitIdMatrix.length-1 || currentPoint.y+y < 0 || currentPoint.x+x > pitIdMatrix[0].length-1 || currentPoint.x+x < 0) {
            continue;
          }
```

```
            if (pitIdMatrix[r+y][c+x] != pitId || (r == pitIdMatrix.length-1 || r == 0 || c ==
pitIdMatrix[0].length-1 || c == 0)) {
              float currentElevation = dem[r][c];
              float neighborElevation = dem[r+y][c+x];
              onBorder = true;
              if ((Float.isNaN(spilloverElevation)) || (currentElevation <=
spilloverElevation && neighborElevation <= spilloverElevation)) {
                spilloverElevation = (float) Math.max(neighborElevation,
currentElevation);
                pitOutletPoint = currentPoint;
                outletSpilloverFlowDirection = new Point(c+x, r+y);
              }
            }
          }
        }
        if (onBorder == false) {
          pitBorderIndicesList.remove(currentPoint);
        }
      }

      // Volume/elevation-dependent variables and calculations
      retentionVolume = 0.0f;
      filledVolume = 0.0f;
      pitDrainageRate = 0.0f;
      float netAccumulationRate = (RainfallSimConfig.rainfallIntensity *
indicesDrainingToPit.size() * cellSize*cellSize);//cubic meters per hour          -
pitDrainageRate
      spilloverTime = retentionVolume / netAccumulationRate; //hours
    }


    public void completeNegativePitConstruction(List<Point> indicesDrainingToPit,
float[][] drainage, float cellSize, float[][] dem, int[][] pitIdMatrix) {
      area = indicesDrainingToPit.size();
```

```
    // Border-dependent variables and calculations
    pitOutletPoint = pitPoint;
    spilloverElevation = dem[pitOutletPoint.y][pitOutletPoint.x];

    pitBorderIndicesList = new ArrayList<Point>(indicesDrainingToPit);
    for (int i = 0; i < indicesDrainingToPit.size(); i++) {
      Point currentPoint = new Point(indicesDrainingToPit.get(i));
      int r = currentPoint.y;
      int c = currentPoint.x;
      boolean onBorder = false;
      for (int x = -1; x < 2; x++) {
        for (int y = -1; y < 2; y++){
          if (x == 0 && y == 0) {
            continue;
          }
          if (currentPoint.y+y > pitIdMatrix.length-1 || currentPoint.y+y < 0 ||
currentPoint.x+x > pitIdMatrix[0].length-1 || currentPoint.x+x < 0) {
            continue;
          }
          if (pitIdMatrix[r+y][c+x] != pitId || (r == pitIdMatrix.length-1 || r == 0 || c ==
pitIdMatrix[0].length-1 || c == 0)) {
            onBorder = true;
          }
        }
      }
      if (onBorder == false) {
        pitBorderIndicesList.remove(currentPoint);
      }
    }

    // Volume/elevation-dependent variables and calculations
```

```
    retentionVolume = 0.0f;
    filledVolume = 0.0f;
    pitDrainageRate = 0.0f;
    spilloverTime = Float.POSITIVE_INFINITY;
  }

  @Override
  public int compareTo(Object obj) {
    Pit f = (Pit) obj;
    if (spilloverTime > f.spilloverTime) {
      return 1;
    }
    else if (spilloverTime < f.spilloverTime) {
      return -1;
    }
    else {
      return 0;
    }
  }
}
```

PitRaster Class

```
package org.waterapps.watershed;

import java.util.ArrayList;
import java.util.List;
import org.waterapps.watershed.WatershedDataset.WatershedDatasetListener;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Color;
import android.graphics.Point;
```

```
public class PitRaster {
  // bitmap represents rasterized elevation data
  List<Pit> pitDataList;
  int[][] pitIdMatrix;
  int numrows;
  int numcols;
  private WatershedDatasetListener listener;
  static int status = 40;
  private float[][] dem;
  FlowDirectionCell[][] flowDirection;
  float cellSize;
  int maxPitId = 1; // positive pits need to be filled
  int minPitId = -1; // negative pits are connected to the edge of


  // constructor method
  public PitRaster(float[][] dem, float[][] drainage,FlowDirectionCell[][]
flowDirection, float inputCellSize, WatershedDatasetListener listener) {
    this.dem = dem;
    this.flowDirection = flowDirection;
    this.cellSize = inputCellSize;
    this.listener = listener;
  }


  public void constructPitRaster(int pitCellCount) {
    numrows = flowDirection.length;
    numcols = flowDirection[0].length;
    pitIdMatrix = new int[numrows][numcols];
    pitDataList = new ArrayList<Pit>(pitCellCount + (numrows*2) + (numcols*2) -
4);


    //identify catchments raster
```

```
    for (int c = 0; c < numcols; c++) {
      for (int r = 0; r < numrows; r++) {
        // Edge cells were marked with a null flow direction child.  Identify each
edge cell and
        // those cells that flow to it as a unique pit with a negative id.
        if (flowDirection[r][c].childPoint == null) {
          Point pitPoint = new Point(c, r);
          Pit currentPit = new Pit(minPitId, pitPoint);
          pitDataList.add(currentPit);
          minPitId--;
        } else if (flowDirection[r][c].childPoint.y < 0) {
          Point pitPoint = new Point(c, r);
          Pit currentPit = new Pit(maxPitId, pitPoint);
          pitDataList.add(currentPit);
          maxPitId++;
        }
      }
      status = (int) (40 + (25 * ((((c*numrows)))/((float) numrows*numcols))));
      listener.simulationOnProgress(status, "Locating Surface Depressions");
    }

    // After identifying the pits matrix, gather pit data for each pit
    for (int i = 0; i < pitDataList.size(); i++) {
      if (pitDataList.get(i).pitId > -1){
        List<Point> indicesDrainingToPit = findCellsDrainingToPoint(flowDirection,
pitDataList.get(i).pitPoint, pitDataList.get(i).pitId);
        pitDataList.get(i).completePitConstruction(indicesDrainingToPit, null,
cellSize, dem, pitIdMatrix);
      } else {
        List<Point> indicesDrainingToPit = findCellsDrainingToPoint(flowDirection,
pitDataList.get(i).pitPoint, pitDataList.get(i).pitId);
```

```
          pitDataList.get(i).completeNegativePitConstruction(indicesDrainingToPit,
null, cellSize, dem, pitIdMatrix);
        }
        status = (int) (65 + (25 * (i/(float)pitDataList.size())));
        listener.simulationOnProgress(status, "Computing Surface Depression
Dimensions");
      }
    }

    public List<Point> findCellsDrainingToPoint(FlowDirectionCell[][] flowDirection,
Point pitPoint, int pitId) {
      List<Point> indicesDrainingToPit = new ArrayList<Point>();
      indicesDrainingToPit.add(pitPoint);
      List<Point> indicesToCheck = new ArrayList<Point>();
      indicesToCheck.add(indicesDrainingToPit.get(0));
      while (!indicesToCheck.isEmpty()) {
        int r = indicesToCheck.get(0).y;
        int c = indicesToCheck.get(0).x;
        pitIdMatrix[r][c] = pitId;
        indicesToCheck.remove(0);

        if (flowDirection[r][c].parentList.isEmpty()) {
          continue;
        }
        for (int i = 0; i < flowDirection[r][c].parentList.size(); i++) {
          indicesDrainingToPit.add(flowDirection[r][c].parentList.get(i));
          indicesToCheck.add(flowDirection[r][c].parentList.get(i));
        }
      }
      return indicesDrainingToPit;
    }

    public int getIndexOf(int inputPitID) {
```

```
    for (int i = 0; i < pitDataList.size(); i++) {
      if (pitDataList.get(i).pitId == inputPitID) {
        return i;
      }
    }
    int pitListIndex = -1;
    return pitListIndex;
  }

  public Bitmap highlightSelectedPit(int selectedPitIndex) {
    BitmapFactory.Options options = new BitmapFactory.Options();
    options.inPurgeable = true;
    options.inInputShareable = true;
    Bitmap icon =
BitmapFactory.decodeResource(MainActivity.context.getResources(),
R.drawable.watershedelineation, options);
    Bitmap pitsBitmap = Bitmap.createScaledBitmap(icon,
this.pitIdMatrix[0].length, this.pitIdMatrix.length, false);
    Bitmap highlightedPitBitmap = Bitmap.createBitmap(pitsBitmap);;
    Pit selectedPit = this.pitDataList.get(selectedPitIndex);
    for (int i = 0; i < selectedPit.pitBorderIndicesList.size(); i++) {
      highlightedPitBitmap.setPixel(pitIdMatrix[0].length - 1 -
selectedPit.pitBorderIndicesList.get(i).x, selectedPit.pitBorderIndicesList.get(i).y,
Color.BLACK);
    }
    return highlightedPitBitmap;
  }
}
```

WatershedDataset Class

```
package org.waterapps.watershed;
```

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.Random;
import org.gdal.gdal.Dataset;
import org.gdal.gdal.gdal;
import org.gdal.gdalconst.gdalconst;
import org.gdal.gdalconst.gdalconstConstants;
import org.gdal.ogr.Driver;
import org.gdal.ogr.Feature;
import org.gdal.ogr.FeatureDefn;
import org.gdal.ogr.Geometry;
import org.gdal.ogr.Layer;
import org.gdal.ogr.ogr;
import org.gdal.osr.SpatialReference;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Color;
import android.graphics.Point;
import android.os.AsyncTask;
import android.util.Log;
import com.google.android.gms.maps.model.LatLng;
import com.openatk.openatklib.atkmap.models.ATKPolygon;

public class WatershedDataset {
   FlowDirectionCell[][] flowDirection;
   float[][] originalDem;
   float[][] dem;
   float[][] drainage;
   public static PitRaster pits;
```

```
  static float cellSize;
  static int status = 0;
  static String statusMessage = "Reading DEM";
  public static float noDataVal;
  public static boolean fillAllPits = false;
  public static int delineatedArea = 0;
  WatershedDatasetListener listener;
  DelineationListener delineationListener;
  static Layer layer;
  static org.gdal.ogr.DataSource dst;

  public interface WatershedDatasetListener {
    public void simulationOnProgress(int progress, String status);
    public void simulationDone();
  }

  public interface DelineationListener {
    public void delineationOnProgress(Bitmap bitmap);
    public void delineationDone();
  }

  // Constructor
  public WatershedDataset(float[][] inputDem, float inputCellSize, float
inputNoDataVal, AsyncTask task) {

    if(task instanceof WatershedDatasetListener) {
      listener = (WatershedDatasetListener) task;
    } else {
      throw new ClassCastException("WatershedDataset - Task must implement
WatershedDatasetListener");
    }
```

```
    noDataVal = inputNoDataVal;
    drainage = null;
    cellSize = inputCellSize;

    // Load the DEM
    listener.simulationOnProgress(status, "Reading DEM");
    originalDem = inputDem;
    dem = new float[originalDem.length][originalDem[0].length];
    for (int r = 0; r < originalDem.length; r++) {
      for (int c = 0; c < originalDem[0].length; c ++) {
        dem[r][c] = originalDem[r][c];
      }
    }

    listener.simulationOnProgress(status, "Discovering Flow Routes");
    // Compute Flow Direction
    int pitCellCount = computeFlowDirection();
    // Compute Pits
    listener.simulationOnProgress(status, "Identifying Surface Depressions");
    pits = new PitRaster(dem, drainage, flowDirection, cellSize, listener);
    pits.constructPitRaster(pitCellCount);
    listener.simulationOnProgress(status, "Done");

  }

  public void recalculatePitsForNewRainfall() {
    MainActivity.simulateButton.setEnabled(false);
    for (int i=0; i < WatershedDataset.pits.pitDataList.size(); i++) {
      if (WatershedDataset.pits.pitDataList.get(i).pitId < 0) {
        continue;
      }
    }
```

```
    MainActivity.simulateButton.setEnabled(true);
  }

  public void setTask(AsyncTask task){
    if(task instanceof WatershedDatasetListener) {
      listener = (WatershedDatasetListener) task;
    } else {
      throw new ClassCastException("WatershedDataset - Task must implement
WatershedDatasetListener");
    }
  }



  public int computeFlowDirection() {
    int pitCellCount = 0;

    flowDirection = new FlowDirectionCell[this.dem.length][this.dem[0].length];
    for (int c = 0; c < this.dem[0].length; c++) {
      for (int r = 0; r < this.dem.length; r++) {
        Point childPoint = null;
        // If the cell is along the border then it should remain a null
        if (r == this.dem.length-1 || r == 0 || c == this.dem[0].length-1 || c == 0) {
          flowDirection[r][c] = new FlowDirectionCell(childPoint);
          continue;
        }
        float minimumSlope = Float.NaN;
        for (int x = -1; x < 2; x++) {
          for (int y = -1; y < 2; y++){
            if (x == 0 && y == 0) {
              continue;
            }
            float distance = (float) Math.sqrt((Math.pow(x, 2) + Math.pow(y, 2)));
```

```
            float slope = (dem[r+y][c+x] - dem[r][c])/distance;
            //maintain current minimum slope, minimum slope being the steepest
downslope
            if (Float.isNaN(minimumSlope) || slope < minimumSlope) {
              minimumSlope = slope;
              childPoint = new Point(c+x, r+y);
              flowDirection[r][c] = new FlowDirectionCell(childPoint);
            }
          }
        }

        // Identification of pit cells (no downslope available) as (-1, -1) flow
direction childpoint
        if (minimumSlope >= 0) {
          pitCellCount++;
          childPoint = new Point(-1, -1);
          FlowDirectionCell flowDirectionCell = new FlowDirectionCell(childPoint);
          flowDirection[r][c] = flowDirectionCell;
        }
      }
      status = (int) (20 + (10 * (((c*this.dem.length))/((float)
this.dem.length*this.dem[0].length))));
      listener.simulationOnProgress(status, "Discovering Flow Routes");
    }

    // Now go back through and also build a list of parents so the tree structure
can be traversed either way.
    // Edge pixels may have parents, but lack the neighbors to determine a valid
flow direction (child).
    for (int c = 0; c < this.dem[0].length; c++) {
      for (int r = 0; r < this.dem.length; r++) {
        Point currentPoint = new Point(c, r);
        ArrayList<Point> parentList = new ArrayList<Point>(8);
```

```
          // Find all cells pointing to current cell.
          for (int x = -1; x < 2; x++) {
            for (int y = -1; y < 2; y++){
              if (x == 0 && y == 0) {
                continue;
              }
              if (r+y > this.dem.length-1 || r+y < 0 || c+x > this.dem[0].length-1 || c+x
< 0) {
                continue;
              }
              if (flowDirection[r+y][c+x].childPoint != null) {
                if (flowDirection[r+y][c+x].childPoint.x == currentPoint.x &&
flowDirection[r+y][c+x].childPoint.y == currentPoint.y) { //apparently, this is not
the same thing as "flowDirection[r+y][c+x].childPoint == currentPoint"; perhaps
checking == for two points doesn't work
                  parentList.add(new Point(c+x, r+y));
                }
              }
            }
          }
          flowDirection[r][c].setParentList(parentList);
        }
        status = (int) (30 + (10 *
(((c*this.dem.length))/((float)this.dem.length*this.dem[0].length))));
        listener.simulationOnProgress(status, "Discovering Flow Routes");
      }
      return pitCellCount;
    }


    public void findFlowDirectionParents(List<Point> cellsToFindParents) {
      for (int i = 0; i < cellsToFindParents.size(); i++) {
        int r = cellsToFindParents.get(i).y;
        int c = cellsToFindParents.get(i).x;
```

```
        ArrayList<Point> parentList = new ArrayList<Point>(8);
        for (int x = -1; x < 2; x++) {
          for (int y = -1; y < 2; y++){
            if (x == 0 && y == 0) {
              continue;
            }
            if (r+y > this.dem.length-1 || r+y < 0 || c+x > this.dem[0].length-1 || c+x <
0) {
              continue;
            }
            if (flowDirection[r+y][c+x].childPoint != null) {
              if (flowDirection[r+y][c+x].childPoint.y == r &&
flowDirection[r+y][c+x].childPoint.x == c) { //apparently, this is not the same thing
as "flowDirection[r+y][c+x].childPoint == currentPoint"; perhaps checking == for
two points doesn't work
                parentList.add(new Point(c+x, r+y));
              }
            }
          }
        }
        flowDirection[r][c].setParentList(parentList);
      }
    }

    public void resolveFlowDirectionParents() {
      for (int c = 0; c < this.dem[0].length; c++) {
        for (int r = 0; r < this.dem.length; r++) {
          if (r > this.dem.length-1 || r < 0 || c > this.dem[0].length-1 || c < 0) {
            continue;
          }
          Point currentPoint = new Point(c, r);
          ArrayList<Point> parentList = new ArrayList<Point>();
```

```
        // Find all cells pointing to current cell. This comes after assuring that the
current cell isn't on the border.
        for (int x = -1; x < 2; x++) {
          for (int y = -1; y < 2; y++){
            if (x == 0 && y == 0) {
              continue;}
            if (r+y >= this.dem.length-1 || r+y <= 0 || c+x >= this.dem[0].length-1 ||
c+x <= 0) {
              continue;
            }
            if (flowDirection[r+y][c+x].childPoint.x == currentPoint.x &&
flowDirection[r+y][c+x].childPoint.y == currentPoint.y) {
              Point parentPoint = new Point(c+x, r+y);
              parentList.add(parentPoint);
            }
          }
        }
        this.flowDirection[r][c].setParentList(parentList);
      }
    }
  }


  // Wrapper function that simulates the rainfall event to iteratively fill depressions
until the rainfall event ends or no more remain
  @SuppressWarnings("unchecked")
  public boolean fillPits() {
    long start = System.currentTimeMillis();
    statusMessage = "Filling and Merging Depressions";
    int fill_counter = 0;
    Collections.sort(WatershedDataset.pits.pitDataList);
    int numberOfPits = pits.pitDataList.size();
    long pre = System.currentTimeMillis();
    if (fillAllPits) {
```

```
        // Once a pit is connected to the edge of the map, it becomes negative.  All negative pits (and only negative pits) should have an
        // infinite spillover time, placing them at the end of the list.  If the first pit in the list has a negative ID,
        // then all remaining pits are negative and filling is complete.
        while (WatershedDataset.pits.pitDataList.get(0).pitId > 0) {
          altMergePits();
          Collections.sort(WatershedDataset.pits.pitDataList);

          fill_counter++;
          status = (int) (100 * (fill_counter/(float)numberOfPits));
          listener.simulationOnProgress(status, "Simulating Rainfall");
        }
      } else {
        // Handle rainfall/duration-based filling.
        while (WatershedDataset.pits.pitDataList.get(0).spilloverTime < RainfallSimConfig.rainfallDuration) {
          altMergePits();
          fill_counter++;
          status = (int) (100 * (fill_counter/(float)numberOfPits));
          listener.simulationOnProgress(status, "Simulating Rainfall");
        }
        Log.w("runtime", Long.toString((System.currentTimeMillis()-start)));
      }
      // time has expired for the storm event, filling is 100% complete for this simulation
      status = 100;
      listener.simulationOnProgress(status, "Finished");
      drawPuddles();
      long post = System.currentTimeMillis();
      System.out.println(Long.toString(post-pre) + ",");
      return true;
    }
```

```
  public boolean altMergePits() {
    Pit firstPit = WatershedDataset.pits.pitDataList.get(0);
    int secondPitId =
WatershedDataset.pits.pitIdMatrix[firstPit.outletSpilloverFlowDirection.y][firstPit.o
utletSpilloverFlowDirection.x];
    int secondPitListIndex = WatershedDataset.pits.getIndexOf(secondPitId);
    Pit secondPit = WatershedDataset.pits.pitDataList.get(secondPitListIndex);

    // Handle pits merging with other pits
    if (secondPitId > 0) {
      int mergedPitId = WatershedDataset.pits.maxPitId;
      WatershedDataset.pits.maxPitId++;

      secondPit.pitBorderIndicesList.addAll(firstPit.pitBorderIndicesList);
      secondPit.spilloverElevation = Float.NaN;
      // traverse in reverse order.  some of the border indices will be found to be
not on the border and removed from the list (necessitating the onBorder variable)
      for (int i = secondPit.pitBorderIndicesList.size()-1; i > -1; i--) {
        Point currentPoint = secondPit.pitBorderIndicesList.get(i);
        int r = currentPoint.y;
        int c = currentPoint.x;
        boolean onBorder = false;
        for (int x = -1; x < 2; x++) {
          for (int y = -1; y < 2; y++){
            if (x == 0 && y == 0) {
              continue;
            }

            if ((WatershedDataset.pits.pitIdMatrix[r+y][c+x] != firstPit.pitId) &&
(WatershedDataset.pits.pitIdMatrix[r+y][c+x] != secondPitId)) {
              float currentElevation = this.dem[r][c];
              float neighborElevation = this.dem[r+y][c+x];
```

```
                onBorder = true;
                if (Float.isNaN(secondPit.spilloverElevation) || (currentElevation <=
secondPit.spilloverElevation && neighborElevation <=
secondPit.spilloverElevation)) {
                  secondPit.spilloverElevation = (float) Math.max(neighborElevation,
currentElevation);
                  secondPit.pitOutletPoint = currentPoint;
                  secondPit.outletSpilloverFlowDirection = new Point(c+x, r+y);
                }
              }
            }
          }
          if (onBorder == false) {
            secondPit.pitBorderIndicesList.remove(currentPoint);
          }
        }
        secondPit.pitId = mergedPitId;
        secondPit.filledVolume = secondPit.filledVolume + firstPit.retentionVolume;
        secondPit.retentionVolume = secondPit.filledVolume;
        int raisedPointsCount = 0;
        List<Point> indicesToCheck = new ArrayList<Point>(firstPit.area);
        indicesToCheck.add(firstPit.pitPoint);
        for (int j = 0; j < firstPit.area; j++) {
          int r = indicesToCheck.get(j).y;
          int c = indicesToCheck.get(j).x;
          if (this.dem[r][c] <= firstPit.spilloverElevation) {
            raisedPointsCount++;
            this.dem[r][c] = firstPit.spilloverElevation;
          } else {
            WatershedDataset.pits.pitIdMatrix[r][c] = mergedPitId;
          }
          if (this.dem[r][c] < secondPit.spilloverElevation) {
```

```
            secondPit.retentionVolume += ((secondPit.spilloverElevation -
this.dem[r][c]) * cellSize * cellSize);
          }
          if (flowDirection[r][c].parentList.isEmpty()) {
            continue;
          }
          for (int i = 0; i < flowDirection[r][c].parentList.size(); i++) {
            indicesToCheck.add(flowDirection[r][c].parentList.get(i));
          }
        }

        indicesToCheck = new ArrayList<Point>(secondPit.area);
        indicesToCheck.add(secondPit.pitPoint);
        // re-ID second pit
        for (int j = 0; j < secondPit.area; j++) {
          int r = indicesToCheck.get(j).y;
          int c = indicesToCheck.get(j).x;
          WatershedDataset.pits.pitIdMatrix[r][c] = mergedPitId;
          if (this.dem[r][c] < secondPit.spilloverElevation) {
            secondPit.retentionVolume += ((secondPit.spilloverElevation -
this.dem[r][c]) * cellSize * cellSize);
          }
          if (flowDirection[r][c].parentList.isEmpty()) {
            continue;
          }
          for (int i = 0; i < flowDirection[r][c].parentList.size(); i++) {
            indicesToCheck.add(flowDirection[r][c].parentList.get(i));
          }
        }
        indicesToCheck = null;

        // Resolve flow direction to direct flow out of the pit
```

```
        List<Point> toCheckForNeighbors = new
ArrayList<Point>(raisedPointsCount);
        this.flowDirection[firstPit.pitOutletPoint.y][firstPit.pitOutletPoint.x].childPoint
= firstPit.outletSpilloverFlowDirection;

this.flowDirection[firstPit.outletSpilloverFlowDirection.y][firstPit.outletSpilloverFlo
wDirection.x].parentList.add(firstPit.pitOutletPoint);

WatershedDataset.pits.pitIdMatrix[firstPit.pitOutletPoint.y][firstPit.pitOutletPoint.x]
= mergedPitId;
        toCheckForNeighbors.add(firstPit.pitOutletPoint);
        for (int i = 0; i < raisedPointsCount; i++) {
          for (int x = -1; x < 2; x++) {
            for (int y = -1; y < 2; y++){
              if (x == 0 && y == 0) {
                continue;
              }
              Point neighborPoint = new Point(toCheckForNeighbors.get(i).x + x,
toCheckForNeighbors.get(i).y + y);
              // check if the point is part of the complete list to be resolved, but not
already on the "next up" list
              if
((WatershedDataset.pits.pitIdMatrix[neighborPoint.y][neighborPoint.x] ==
firstPit.pitId) &&
(WatershedDataset.pits.pitIdMatrix[neighborPoint.y][neighborPoint.x] !=
mergedPitId)) {
                this.flowDirection[neighborPoint.y][neighborPoint.x].childPoint =
toCheckForNeighbors.get(i);
                WatershedDataset.pits.pitIdMatrix[neighborPoint.y][neighborPoint.x]
= mergedPitId;
                toCheckForNeighbors.add(neighborPoint);
              }
            }
          }
        }
        findFlowDirectionParents(toCheckForNeighbors);
```

```
        toCheckForNeighbors = null;

        //Sum the drainage taking place in the pit
        secondPit.area = firstPit.area + secondPit.area;
        secondPit.pitDrainageRate = 0;
        float netAccumulationRate = (RainfallSimConfig.rainfallIntensity *
secondPit.area * cellSize * cellSize) - secondPit.pitDrainageRate;
        secondPit.spilloverTime = secondPit.retentionVolume/netAccumulationRate;

      // Handle pits that begin to run off the DEM
      } else if (secondPitId < 0) {
        int mergedPitId = WatershedDataset.pits.minPitId;
        WatershedDataset.pits.minPitId--;

        secondPit.pitBorderIndicesList.addAll(firstPit.pitBorderIndicesList);
        // traverse in reverse order.  some of the border indices will be found to be
not on the border and removed from the list (hence, the onBorder variable)
        for (int i = secondPit.pitBorderIndicesList.size()-1; i > -1; i--) {
          Point currentPoint = secondPit.pitBorderIndicesList.get(i);
          int r = currentPoint.y;
          int c = currentPoint.x;
          boolean onBorder = false;
          for (int x = -1; x < 2; x++) {
            for (int y = -1; y < 2; y++){
              if (x == 0 && y == 0) {
                continue;
              }
              if (r+y > WatershedDataset.pits.pitIdMatrix.length-1 || r+y < 0 || c+x >
WatershedDataset.pits.pitIdMatrix[0].length-1 || c+x < 0) {
                continue;
              }
              if ((WatershedDataset.pits.pitIdMatrix[r+y][c+x] != secondPit.pitId &&
WatershedDataset.pits.pitIdMatrix[r+y][c+x] != firstPit.pitId) || (r ==
```

```
WatershedDataset.pits.pitIdMatrix.length-1 || r == 0 || c ==
WatershedDataset.pits.pitIdMatrix[0].length-1 || c == 0)) {
                onBorder = true;
              }
            }
          }
          if (onBorder == false) {
            secondPit.pitBorderIndicesList.remove(currentPoint);
          }
        }
        secondPit.pitId = mergedPitId;

        // Fill the first pit and resolve flow direction. This must be completed before
the new pit entry is created or else retention volumes will be incorrectly
calculated (the first pit must be filled).
        int raisedPointsCount = 0;
        List<Point> indicesToCheck = new ArrayList<Point>(firstPit.area);
        indicesToCheck.add(firstPit.pitPoint);
        for (int j = 0; j < firstPit.area; j++) {
          int r = indicesToCheck.get(j).y;
          int c = indicesToCheck.get(j).x;
          if (this.dem[r][c] <= firstPit.spilloverElevation) {
            raisedPointsCount++;
            this.dem[r][c] = firstPit.spilloverElevation;
          } else {
            WatershedDataset.pits.pitIdMatrix[r][c] = mergedPitId;
          }

          if (flowDirection[r][c].parentList.isEmpty()) {
            continue;
          }
          for (int i = 0; i < flowDirection[r][c].parentList.size(); i++) {
            indicesToCheck.add(flowDirection[r][c].parentList.get(i));
```

```
        }
      }

      indicesToCheck = new ArrayList<Point>(secondPit.area);
      indicesToCheck.add(secondPit.pitPoint);
      // re-ID second pit
      for (int j = 0; j < secondPit.area; j++) {
        int r = indicesToCheck.get(j).y;
        int c = indicesToCheck.get(j).x;
        WatershedDataset.pits.pitIdMatrix[r][c] = mergedPitId;

        if (flowDirection[r][c].parentList.isEmpty()) {
          continue;
        }
        for (int i = 0; i < flowDirection[r][c].parentList.size(); i++) {
          indicesToCheck.add(flowDirection[r][c].parentList.get(i));
        }
      }
      indicesToCheck = null;

      // Resolve flow direction to direct flow out of the pit
      List<Point> toCheckForNeighbors = new
ArrayList<Point>(raisedPointsCount);
      this.flowDirection[firstPit.pitOutletPoint.y][firstPit.pitOutletPoint.x].childPoint
= firstPit.outletSpilloverFlowDirection;

this.flowDirection[firstPit.outletSpilloverFlowDirection.y][firstPit.outletSpilloverFlo
wDirection.x].parentList.add(firstPit.pitOutletPoint);

WatershedDataset.pits.pitIdMatrix[firstPit.pitOutletPoint.y][firstPit.pitOutletPoint.x]
= mergedPitId;
      toCheckForNeighbors.add(firstPit.pitOutletPoint);
      for (int i = 0; i < raisedPointsCount; i++) {
```

```
            for (int x = -1; x < 2; x++) {
              for (int y = -1; y < 2; y++){
                if (x == 0 && y == 0) {
                  continue;
                }
                Point neighborPoint = new Point(toCheckForNeighbors.get(i).x + x,
toCheckForNeighbors.get(i).y + y);
                // check if the point is part of the complete list to be resolved, but not
already on the "next up" list
                if
((WatershedDataset.pits.pitIdMatrix[neighborPoint.y][neighborPoint.x] ==
firstPit.pitId) &&
(WatershedDataset.pits.pitIdMatrix[neighborPoint.y][neighborPoint.x] !=
mergedPitId)) {
                  this.flowDirection[neighborPoint.y][neighborPoint.x].childPoint =
toCheckForNeighbors.get(i);
                  WatershedDataset.pits.pitIdMatrix[neighborPoint.y][neighborPoint.x]
= mergedPitId;
                  toCheckForNeighbors.add(neighborPoint);
                }
              }
            }
          }
          findFlowDirectionParents(toCheckForNeighbors);
          toCheckForNeighbors = null;

          secondPit.area = firstPit.area + secondPit.area;
          secondPit.filledVolume = secondPit.filledVolume + firstPit.retentionVolume;
          secondPit.retentionVolume = secondPit.filledVolume;
          secondPit.pitDrainageRate = 0.0f;
          secondPit.spilloverTime = Float.POSITIVE_INFINITY;
        }
        //Remove first pit
        WatershedDataset.pits.pitDataList.remove(firstPit);
```

```
    if (secondPit.spilloverTime == Float.POSITIVE_INFINITY) {
      WatershedDataset.pits.pitDataList.add(secondPit); // add to end of list;
shouldn't change order of list
      WatershedDataset.pits.pitDataList.remove(secondPit);
    }else {
      WatershedDataset.pits.pitDataList.remove(secondPit);
      for (int i = 0; i < WatershedDataset.pits.pitDataList.size(); i++) {
        if (WatershedDataset.pits.pitDataList.get(i).spilloverTime >
secondPit.spilloverTime) {
          WatershedDataset.pits.pitDataList.add(i, secondPit);
          break;
        }
      }
    }
    return true;
  }

  public Bitmap delineate(Point point, AsyncTask task) {
    if(task instanceof DelineationListener) {
      delineationListener = (DelineationListener) task;
    } else {
      throw new ClassCastException("WatershedDataset - Task must implement
DelineationListener");
    }

    BitmapFactory.Options options = new BitmapFactory.Options();
    options.inPurgeable = true;
    options.inInputShareable = true;

    Bitmap icon =
BitmapFactory.decodeResource(MainActivity.context.getResources(),
R.drawable.watershedelineation, options);
```

```
    Bitmap delinBitmap = Bitmap.createScaledBitmap(icon, this.dem[0].length,
this.dem.length, false);
    //skip outside cells
    for (int r = 1; r < this.dem.length-1; r++) {
      for (int c = 1; c < this.dem[0].length-1; c++) {
        delinBitmap.setPixel(this.dem[0].length - 1 - c, r, Color.TRANSPARENT);
      }
    }
    delineatedArea = 0; //number of cells in the delineation
    // discover adjacent points that may be part of a puddle
    List<Point> indicesToCheck = new ArrayList<Point>();
    List<Point> indicesToCheckPuddle = new ArrayList<Point>();
    float puddleElevation = this.dem[point.y][point.x];
    indicesToCheck.add(point);
    indicesToCheckPuddle.add(point);



    while (!indicesToCheckPuddle.isEmpty()) {
      int r = indicesToCheckPuddle.get(0).y;
      int c = indicesToCheckPuddle.get(0).x;
      indicesToCheckPuddle.remove(0);
      if (delinBitmap.getPixel(this.dem[0].length - 1 - c, r) == Color.RED){
        continue;
      }
      delinBitmap.setPixel(this.dem[0].length - 1 - c, r, Color.RED);
      delineatedArea++;
      for (int x = -1; x < 2; x++) {
        for (int y = -1; y < 2; y++){
          if (x == 0 && y == 0) {
            continue;
          }
```

```
            if (r+y >= this.dem.length-1 || r+y <= 0 || c+x >= this.dem[0].length-1 ||
c+x <= 0) {
              continue;
            }
            if (dem[r+y][c+x] == puddleElevation &&
delinBitmap.getPixel(this.dem[0].length - 1 - (c+x), (r+y)) != Color.RED) {
              indicesToCheckPuddle.add(new Point(c+x, r+y));
              indicesToCheck.add(new Point(c+x, r+y));
            }
          }
        }
      }

      // Add a buffer around the chosen pixel to provide a more likely meaningful
delineation
      for (int x = -3; x < 4; x++) {
        for (int y = -3; y < 4; y++) {
          if (point.y+y > this.dem.length-1 || point.y+y < 0 || point.x+x >
this.dem[0].length-1 || point.x+x < 0) {
            continue;
          }
          if (delinBitmap.getPixel(this.dem[0].length - 1 - (point.x+x), (point.y+y)) !=
Color.RED) {
            indicesToCheck.add(new Point(x+point.x, y +point.y));
            delinBitmap.setPixel(this.dem[0].length - 1 - (point.x+x), (point.y+y),
Color.RED);
            delineatedArea++;
          }
        }
      }

      // Now find all cells draining to either the puddle or the buffered delineation
point
      while (!indicesToCheck.isEmpty()) {
```

```
        int r = indicesToCheck.get(0).y;
        int c = indicesToCheck.get(0).x;
        indicesToCheck.remove(0);

        if (flowDirection[r][c].parentList.isEmpty()) {
          continue;
        }

        for (int i = 0; i < flowDirection[r][c].parentList.size(); i++) {
          if (delinBitmap.getPixel(this.dem[0].length - 1 -
flowDirection[r][c].parentList.get(i).x, flowDirection[r][c].parentList.get(i).y) !=
Color.RED) {
            indicesToCheck.add(flowDirection[r][c].parentList.get(i));
            delinBitmap.setPixel(this.dem[0].length - 1 -
flowDirection[r][c].parentList.get(i).x, flowDirection[r][c].parentList.get(i).y,
Color.RED);
            delineatedArea++;
          }
        }
      }
      return delinBitmap;
    }

    public Bitmap altDrawPits() {
      BitmapFactory.Options options = new BitmapFactory.Options();
      options.inPurgeable = true;
      options.inInputShareable = true;
      Bitmap icon =
BitmapFactory.decodeResource(MainActivity.context.getResources(),
R.drawable.watershedelineation, options);
      Bitmap pitsBitmap = Bitmap.createScaledBitmap(icon, dem[0].length,
dem.length, false);
      Random random = new Random();
      for (int i = 0; i < WatershedDataset.pits.pitDataList.size(); i++) {
```

```
        int red = random.nextInt(255);
        int green = random.nextInt(255);
        int blue = random.nextInt(255);
        int pitColor = Color.rgb(red,green,blue);
        List<Point> indicesToCheck = new
ArrayList<Point>(WatershedDataset.pits.pitDataList.get(i).area);
        indicesToCheck.add(WatershedDataset.pits.pitDataList.get(i).pitPoint);

        while (!indicesToCheck.isEmpty()) {
          int r = indicesToCheck.get(0).y;
          int c = indicesToCheck.get(0).x;
          pitsBitmap.setPixel(this.dem[0].length - 1 - c, r, pitColor);
          indicesToCheck.remove(0);
          if (flowDirection[r][c].parentList.isEmpty()) {
            continue;
          }

          for (int j = 0; j < flowDirection[r][c].parentList.size(); j++) {

            indicesToCheck.add(flowDirection[r][c].parentList.get(j));
          }
        }
      }
      return pitsBitmap;
    }

    public Bitmap drawPuddles() {
      int[] colorarray = new
int[WatershedDataset.pits.pitIdMatrix.length*WatershedDataset.pits.pitIdMatrix[0]
.length];
      Arrays.fill(colorarray, Color.TRANSPARENT);
      Bitmap.Config config = Bitmap.Config.ARGB_8888;
```

```
    Bitmap puddleBitmap = Bitmap.createBitmap(colorarray, this.dem[0].length,
this.dem.length, config);
    puddleBitmap = puddleBitmap.copy(config, true);

    for (int r = 1; r < this.dem.length-1; r++) {
      for (int c = 1; c < this.dem[0].length-1; c++) {
        if (r >= this.dem.length-1 || r <= 0 || c >= this.dem[0].length-1 || c <= 0) {
          continue;
        }
        if (originalDem[r][c] < dem[r][c]) {
          puddleBitmap.setPixel(this.dem[0].length - 1 - c, r, Color.BLUE);
        }
      }
    }
    return puddleBitmap;
  }

  public static void writeRaster(String rasterFilePath, int[][] rasterData, String
fileOutPath) {
    gdal.AllRegister();
    ogr.RegisterAll();
    Dataset demRaster = gdal.Open(rasterFilePath);

    // Transform from 2D array to 1D array
    int[] array = new int[rasterData.length * rasterData[0].length];
    int i = 0;
    for (int r = 0; r < rasterData.length; r++) {
      for (int c = 0; c < rasterData[0].length; c++) {
        array[i] = rasterData[r][rasterData[0].length - c - 1];
        i++;
      }
    }
```

```
    //mask the outer rows
    int[] mask = new int[rasterData.length * rasterData[0].length];
    i = 0;
    for (int r = 0; r < rasterData.length; r++) {
      for (int c = 0; c < rasterData[0].length; c++) {
        if ((r == 0) || (r == rasterData.length - 1) || (c == 0) || (c ==
rasterData[0].length - 1)) {
          mask[i] = 0;
          i++;
        } else {
          mask[i] = 1;
        }
      }
    }

    //Create a new file that is a copy of the DEM geotiff so that the
georeferencing data is identical and write the new band data to this file
    org.gdal.gdal.Driver rdriver = gdal.GetDriverByName("GTiff");
    Dataset catchmentRaster = rdriver.Create(fileOutPath, rasterData[0].length,
rasterData.length);
    catchmentRaster.AddBand(gdalconst.GDT_UInt16);
    catchmentRaster.WriteRaster(0, 0, catchmentRaster.getRasterXSize(),
catchmentRaster.getRasterYSize(), catchmentRaster.getRasterXSize(),
catchmentRaster.getRasterYSize(), gdalconstConstants.GDT_Int32, array, new
int[]{1}, 0, 0, 0);
    //TODO Test if this does anything useful
    catchmentRaster.FlushCache();

    demRaster.delete();
    catchmentRaster.delete();

    demRaster = null;
```

```
    catchmentRaster = null;
  }


  public static void polygonize(String rasterFilePath, int[][] rasterData, String
fileOutPath) {
    gdal.AllRegister();
    ogr.RegisterAll();
    Dataset demRaster = gdal.Open(rasterFilePath);

    // Transform from 2D array to 1D array
    int[] array = new int[rasterData.length * rasterData[0].length];
    int i = 0;
    for (int r = 0; r < rasterData.length; r++) {
      for (int c = 0; c < rasterData[0].length; c++) {
        array[i] = rasterData[r][rasterData[0].length - c - 1];
        i++;
      }
    }

    //mask the outer rows
    int[] mask = new int[rasterData.length * rasterData[0].length];
    i = 0;
    for (int r = 0; r < rasterData.length; r++) {
      for (int c = 0; c < rasterData[0].length; c++) {
        if ((r == 0) || (r == rasterData.length - 1) || (c == 0) || (c ==
rasterData[0].length - 1)) {
          mask[i] = 0;
          i++;
        } else {
          mask[i] = 1;
        }
```

```
        }
    }

    //Create a new file that is a copy of the DEM geotiff so that the
georeferencing data is identical and write the new band data to this file
    org.gdal.gdal.Driver rdriver = gdal.GetDriverByName("GTiff");
    Dataset catchmentRaster = rdriver.CreateCopy(fileOutPath, demRaster);
    catchmentRaster.WriteRaster(0, 0, catchmentRaster.getRasterXSize(),
catchmentRaster.getRasterYSize(), catchmentRaster.getRasterXSize(),
catchmentRaster.getRasterYSize(), gdalconstConstants.GDT_Int32, array, new
int[]{1}, 0, 0, 0);
    catchmentRaster.AddBand(gdalconst.GDT_UInt16);
    catchmentRaster.WriteRaster(0, 0, catchmentRaster.getRasterXSize(),
catchmentRaster.getRasterYSize(), catchmentRaster.getRasterXSize(),
catchmentRaster.getRasterYSize(), gdalconstConstants.GDT_Int32, mask, new
int[]{2}, 0, 0, 0);
    catchmentRaster.FlushCache();

    // When creating this new datasource, the file must not already exist.
    Driver shpDriver = ogr.GetDriverByName("ESRI Shapefile");
    org.gdal.ogr.DataSource catchmentVector =
shpDriver.CreateDataSource(fileOutPath+".shp");
    SpatialReference srs = new
SpatialReference(catchmentRaster.GetProjection());
    Layer catchmentLayer = catchmentVector.CreateLayer("NewLayer", srs);
    gdal.Polygonize(catchmentRaster.GetRasterBand(1),
catchmentRaster.GetRasterBand(2), catchmentLayer, 0);

    SpatialReference wgs = new SpatialReference("GEOGCS[\"WGS
84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS
84\",6378137,298.257223563,AUTHORITY[\"EPSG\",\"7030\"]],AUTHORITY[\"E
PSG\",\"6326\"]],PRIMEM[\"Greenwich\",0,AUTHORITY[\"EPSG\",\"8901\"]],UNIT
[\"degree\",0.01745329251994328,AUTHORITY[\"EPSG\",\"9122\"]],AUTHORITY
[\"EPSG\",\"4326\"]]");
    FeatureDefn outFeatureDef = catchmentLayer.GetLayerDefn();
    // for each polygon
```

```
    float epsilon = (float) Math.sqrt(2*Math.pow(cellSize, 2))/2;
    for (int i1 = catchmentLayer.GetFeatureCount()-1; i1 > -1; i1--) {
      Feature inFeature = catchmentLayer.GetNextFeature();
      Geometry geometry = inFeature.GetGeometryRef();
      Geometry geom = geometry.Simplify(epsilon);
      Log.w("geometry null", Boolean.toString(geom == null));
      inFeature.SetGeometryDirectly(geom);
    }
    for (int i1 = 0; i1 < catchmentLayer.GetFeatureCount(); i1++) {
      Feature inFeature = catchmentLayer.GetNextFeature();
      Geometry geometry = inFeature.GetGeometryRef();
      Log.w("error check", gdal.GetLastErrorMsg());
      geometry.TransformTo(wgs);
      List<LatLng> list = new ArrayList<LatLng>();
      for (int j = 0; j < geometry.GetGeometryRef(0).GetPointCount(); j++) {
        list.add(new LatLng(geometry.GetGeometryRef(0).GetPoint(j)[1],
geometry.GetGeometryRef(0).GetPoint(j)[0]));
      }
      ATKPolygon poly = new ATKPolygon("test", list);
      MainActivity.map.addPolygon(poly);
      poly.viewOptions.setFillColor(Color.TRANSPARENT);
      poly.viewOptions.setStrokeColor(Color.RED);
    }

    catchmentVector.SyncToDisk();
    demRaster.delete();
    catchmentRaster.delete();
    catchmentVector.delete();
    catchmentLayer.delete();

    demRaster = null;
    catchmentRaster = null;
```

```
    catchmentVector = null;
    catchmentLayer = null;
  }
}
```

FlowDirectionCell Class

```
package org.waterapps.watershed;

import java.util.ArrayList;
import android.graphics.Point;

public class FlowDirectionCell {
  Point childPoint;
  ArrayList<Point> parentList = null;

//   // Constructor method
  public FlowDirectionCell(Point inputChildPoint) {
    childPoint = inputChildPoint;
  }

  public void setParentList(ArrayList<Point> inputParentList) {
    if (parentList != null) {
      parentList.clear();
    }
    parentList = inputParentList;
  }
}
```