Fast Summarizing Algorithm for Polygonal Statistics over a Regular Grid

Scott Haag[1,2], David Tarboton[3], Martyn Smith[4], and Ali Shokoufandeh[5]

Abstract

We describe a data structure and associated algorithm called Fast Zonal Statistics (FZS) for the retrieval of the summary characteristics of an arbitrary polygon derived from a regular grid. The FZS algorithm can return numerical (e.g., mean, sum, and count) attributes for a polygonal object over a regular grid (e.g., raster data model). The computational complexity of the FZS algorithm is constant in relation to the length of the polygon perimeter. This contrasts with existing approaches which scale in relation to the polygon area, therefore we expect and measure geometric decreases in execution time using the proposed approach for simple polygon surfaces. We demonstrate applications of the algorithm and data structure on example datasets extracting the sum of impervious surface for watershed boundaries in the Chesapeake Bay watershed, a common use case.

Keywords: Spatial Summary Statistics, Green's Theorem, Geospatial area average, Zonal Statistics

[1]Drexel University, United States of America

[2]Corresponding author at: Children's Hospital of Philadelphia, 2716 South Street, Philadelphia, PA 19146, United States of America. E-mail address: haags@email.chop.edu (S. Haag).

[3]Utah State University, United States of America

[4]United States Geological Survey (USGS), United States of America

[5]Drexel University, United States of America

This is the accepted version of the following published article. Please refer to the published article for the final version of the manuscript

Haag, S., D. Tarboton, M. Smith and A. Shokoufandeh, (2020), "Fast summarizing algorithm for polygonal statistics over a regular grid," Computers & Geosciences, 142: 104524, https://doi.org/10.1016/j.cageo.2020.104524.

1. Introduction

1.1. Problem

A common problem in the geo-spatial sciences is the extraction of numerical attributes from a regular grid for a specific overlay area (i.e., a polygon stored as a piece-wise list of vertexes). Algorithms to produce these results are found in a variety of tools including zonal statistics from Environmental Systems Research Institutes (ESRI), the System for Automated Geoscientific Analyses (SAGA) zonal statistics module (Conrad et al., 2015), zonal attribute from Erdas Imagine, class statistics in Ecognition, and a variety of custom Python libraries (e.g., rasterstats (Perry, 2013)). These types of analysis are routine and have applications to a variety of fields of study including hydrology and watershed modelling, urban planning and medical imaging. This manuscript introduces an algorithm and an associated data structure called Fast Zonal Statistics (FZS). The goal of FZS is to decrease the complexity of retrieving certain polygon zonal statistics from a regular grid data structure.

Geospatial processing techniques are increasingly becoming web enabled to allow real time interactive query processing. These techniques rely on optimized retrieval and processing techniques. Existing zonal statistics algorithms, to the best of our knowledge must visit all internal grid cells to return zonal summary statistics. The need to increase the speed of these types of queries has led to a number of projects focused on clustering frameworks and parallelization approaches, enabling many computational devices to coordinate resources. For example, Geotrellis, a raster processing Application Program Interface (API) developed by Azavea, utilizes Apache spark to achieve geospatial processing in real time to support web services (Kini

and Emanuele, 2014). In a similar approach Google Earth Engine provides a tiled raster processing service allowing independent rendering of raster imagery (Gorelick et al., 2017). These approaches are focused on enlarging the data processing pipeline not reducing the computational complexity of the calculations. This manuscript provides an alternative approach that can reduce the number of operations necessary to return numerical attributes from a regular grid for a specific overlay area (zonal statistics). Utilizing this approach, coupled with enlarging the data processing pipeline might provide a best case scenario allowing the retrieval of large polygonal summary statistics over dense grids in real time applications.

The methods discussed in this manuscript are to our knowledge novel to the field of Geo-Spatial modelling and the sub-fields of Geographic Information Systems. We focus on the problem of returning statistics for a region (polygon over a regular grid (i.e., the raster data model)) which is a fundamental question in this field. As an example, a number of existing libraries have been created (Perry, 2013) and (Geo, 2018) to answer this specific question.

Similar techniques were used by (Viola and Jones, 2001) and (Pham et al., 2010) by creating integral images to speed up the process of object detection and localization. These methods provided constant, $O(1)$ time complexity for localization of rectangular regions of interest in intensity images. (Viola and Jones, 2001) argue that accurate object detection in 2D iamges require a large number of summary statistics, the cost of preprocessing (creating the integral image) is paid off during the object localization step. We are making a similar argument, in that for raster datasets which are subject to a large number of polygonal object queries, the cost

of pre-processing can be amortized by the reduced cost of individual calculations. The significance of the method described by (Viola and Jones, 2001) is apparent from its impact in computer vision community citations ($> 20,000$ citation). We believe that similar benefits can be expected in the field of Geo-Spatial modelling using the techniques described in this manuscript.

1.2. Approach

The FZS algorithm uses Green's theorem (Green, 1828) which relates the line integral around a perimeter of a two dimensional field to the area integral of the derivative of the field. We reduce the complexity of the evaluation of area integrals, which require summing values from each grid cell within the area to a traverse of the perimeter evaluating the line integral and summing only grid cells along the perimeter. This is enabled by first summing down the columns of the grid representation of the field to be evaluated and then performing the perimeter line integration on the integral field result. Perimeter integration is much faster for large areas where there is a smaller ratio of perimeter grid cells to area grid cells. However, there is a requirement to first calculate the integral field. This is done as a pre-processing step, and is most beneficial for static fields where repeated extraction of zonal statistics for many regions are required.

As an example, we provide a Jupyter notebook with code comparing FZS to an existing well known Python implementation for calculating zonal statistics called rasterstats. Empirical timing results are shown using watersheds of varying sizes within the Chesapeake Bay drainage area. We show that the benefits of our approach are the most pronounced with larger

geometric structures (i.e, larger vs smaller polygons in area) and we expect similar increases with higher resolution raster datasets.

Lastly, we assume that polygons submitted to the fast zonal algorithm are simple or non-intersecting. Self-intersections or bowties can be identified using several existing libraries, such as the repair self intersection tool from ArcGIS or is.valid from the shapely GIS library, and simplified prior to processing.

2. Material and Methods

2.1. Notation

In this manuscript we devise an efficient algorithm for computation of zonal statistics of aggregated summary statistics for the intersection between a query of a closed polygonal chain, P, with no intersections and a given grid, G. The polygonal chain, P, will be represented by its ordered set of vertices $\langle v_1, v_2, ..., v_\ell \rangle$, each uniquely identified by its (x, y) coordinates in $\mathbb{R}^2$. The underlying structure used throughout the paper is a regular grid, G, with $n \times m$ cells, each of which is a subset of $\mathbb{R}^2$. Without loss of generality, we will use the notation $G[i, j]$ to denote the cell located at column $i \in \{1, ...n\}$ and row $j \in \{1, ..., m\}$ of grid G, see Figure 1. When it is clear from context, we will use the same notation, $G[i, j]$, for referring to the cell at location $(i, j) \in \{1..., n\} \times \{1, ..., m\}$ or its centroid.

We will assume each cell, $G[i, j]$ has a set, $M[i, j]$ of associated attributes capturing numerical and/or categorical attributes; this includes physical attributes (e.g., temperature or precipitation) or categorical (e.g., land use land cover). As an example, consider the scenario where P corresponds to

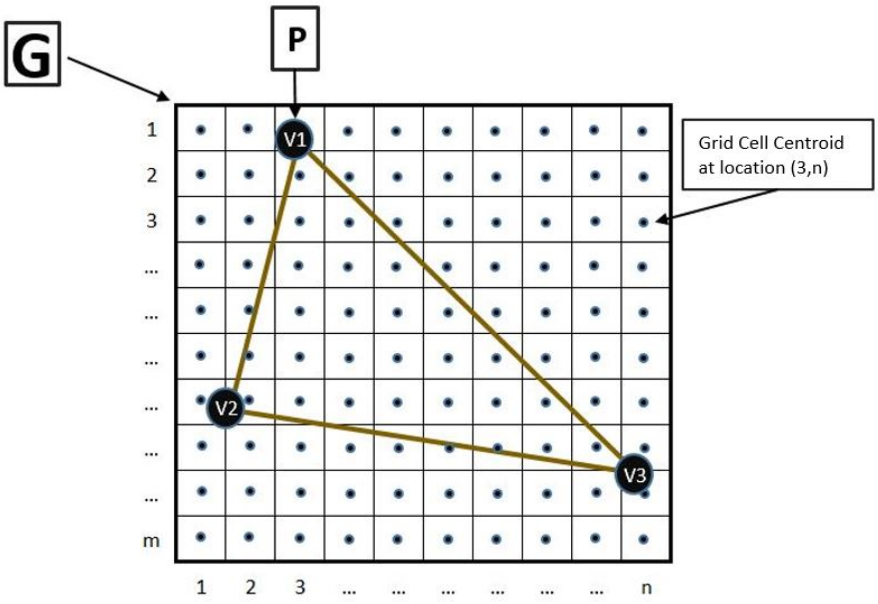

Figure 1: Overview of underlying data grid structure, G, and region of interest, P.

a watershed boundary and the matrix, M, represents estimates of precipitation at each grid cell of G. To use the fast zonal statistics, the method described in this manuscript will modify the value for each cell $G[i, j]$ from a measure of the local amount of precipitation to a sum of precipitation values for all grid cells above and including the cell $G[i, j]$. Specifically, assuming a NORTH to SOUTH cardinal direction, we define $\mathcal{F} : G \rightarrow R$ as

$$\mathcal{F}[i, j] = \sum_{k=1}^{j} M[i, k], \forall (i, j) \in \{1, ..., n\} \times \{1, ...m\}.$$

Figure 2 and Algorithm 1 provide a schematic overview of this computation.

Given a pair of consecutive vertices, u and v, representing a boundary edge, $e = (u, v)$, of P, we will define the *support chain* of edge e, as $\mathcal{C}(u, v) \subset$

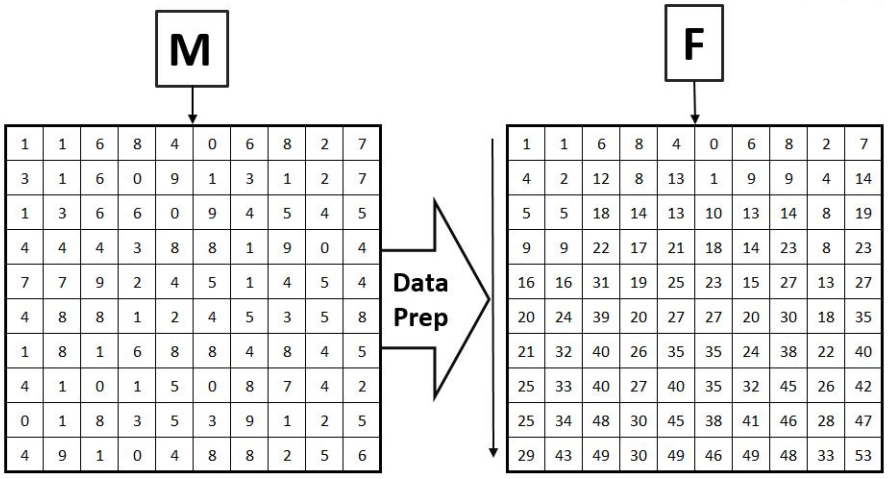

Figure 2: Creating NORTH to SOUTH cardinal direction, aggregate statistics $\mathcal{F}$ from matrix M.

G consisting of cells uniquely identified by their (x, y) grid coordinates whose centroids are both closest and strictly above the edge the e_i.

Because $\mathcal{C}(u, v)$ is created by connecting consecutive vertices that define a region using a counter clockwise walk, this directly leads to a counter clockwise ordering among cells $\langle c_1, ..., c_k \rangle$ that form $\mathcal{C}(u, v)$. We note that due to its properties, the numbers of cells in any chain $\mathcal{C}(u, v)$ for $1 \leq i \leq \ell$ can not exceed n. Observe that there are degenerate cases for which $|\mathcal{C}(u, v)| = 0$, i.e., there are no cells in chain $\mathcal{C}(u, v)$. Given the proximity requirement in the definition of cells in supporting curves, any cell c in G can belong to, at most, one edge e in P. Since otherwise c has to be equidistant from two edges, e and e', in P while being above both of them, which is impossible for simple non-intersecting polygons. It should be noted that if the centroid of any of the cells in supporting chain $\mathcal{C}(u, v)$ corresponding to an edge $e = (u, v)$ is in the interior of P then the centroid of every cell

Algorithm 1 – **FZS_DT**(G): A procedure for computing grid value $\mathcal{F}$ along NORTH to SOUTH cardinal direction.

Input: Regular grid G with numerical value stored in grid M,
Output: Regular grid G with summed value grid $\mathcal{F}$.
For $i = 1$ to n with increments of 1
    $\mathcal{F}[i, 1] = M[i, 1]$
    For $j = 2$, to m with increments of 1
        $\mathcal{F}[i, j] = \mathcal{F}[i, j - 1] + M[i, j]$
    end
end

in $\mathcal{C}(u, v)$ will be in the interior of P. We will refer to a chain with all its cells in the interior of P as a *positive supporting chain*. Conversely, if the centroid of any of the cells in supporting chain $\mathcal{C}(u, v)$ corresponding to an edge $e = (u, v)$ is outside P then the centroid of every cell in $\mathcal{C}(u, v)$ will be in the exterior of P. We will refer to such a chain as a *negative supporting chain*. We note that if c belongs to the positive supporting chain $\mathcal{C}(u, v)$ then the triplet $\langle u, c, v\rangle$ will form a clockwise turn, while for a negative supporting chain the triplet $\langle u, c, v\rangle$ form a counterclockwise turn. This observation provides a straightforward computational problem for verifying whether a supporting curve is positive or negative. Let $c = (c_x, c_y)$ denote the centroid of a cell belonging to support chain $\mathcal{C}(u, v)$ for vertex pair $u = (u_x, u_y)$ and $v = (v_x, v_y)$ in P. To classify $\mathcal{C}(u, v)$, we can use the sign of the following determinant for determining the orientation of the turn

$\langle u, c, v\rangle$ (Berg et al., 2008):

$$\Delta(u, c, v) = \begin{vmatrix} u_x & u_y & 1 \\ c_x & c_y & 1 \\ v_x & v_y & 1 \end{vmatrix}.$$

Specifically, if $\Delta(u, c, v) < 0$ then $\langle u, c, v\rangle$ represents a clockwise turn, while $\Delta(u, c, v) > 0$ will correspond to a counterclockwise turn, and $\Delta(u, c, v) = 0$ implies c lies on the edge $e = (u, v)$. Figure 3 is an illustration of sample support chains associated with a closed polygonal chain P in G.

In our current study, all integrations in the integration field are pulled in the same cardinal direction, e.g., NORTH to SOUTH. This assumption is reflected in the computation of summary statistic $\mathcal{F}$ and will allow for a more efficient probe mechanism for classification of positive and negative support chains and their associated cells. Specifically, given the chain $\mathcal{C}(u, v)$ corresponding to a counterclockwise oriented edge $e = (u, v)$ in P, if $u_x < v_x$ then $\mathcal{C}(u, v)$ can be characterized as a positive supporting chain. Conversely, if $u_x > v_x$ then $\mathcal{C}(u, v)$ can be characterized as a negative supporting chain. Finally, having $u_x = v_x$ results in a degenerate case of an empty support chain, i.e., $\mathcal{C}(u, v) = \emptyset$.

In the FZS algorithm we stores the vertices of the polygon in a counter clock-wise march returning to the first vertex to denote the closed nature of the polygon. Note that the methods discussed in this paper will work equally well for a polygon that undergoes a clock-wise march, except that the sum and count results will end up being negative based on the probe conditions described above, a technique to solve this problem involves an absolute value calculation at the last step before returning the final results. This step will make this algorithm equal for both clockwise and counter

clockwise polygons.

Figure 3 shows in red grid cells values that must be subtracted and in blue values that must be added from grid M to calculate simple summary statistics. This is true because we want to get the summary value of the grey cells, and it is obvious that if the red is subtracted from the blue, that we are left with the sum value of grey region. This technique, is highly parallelizable, as the order of the vertex pairs does not matter due to the commutative nature of addition.

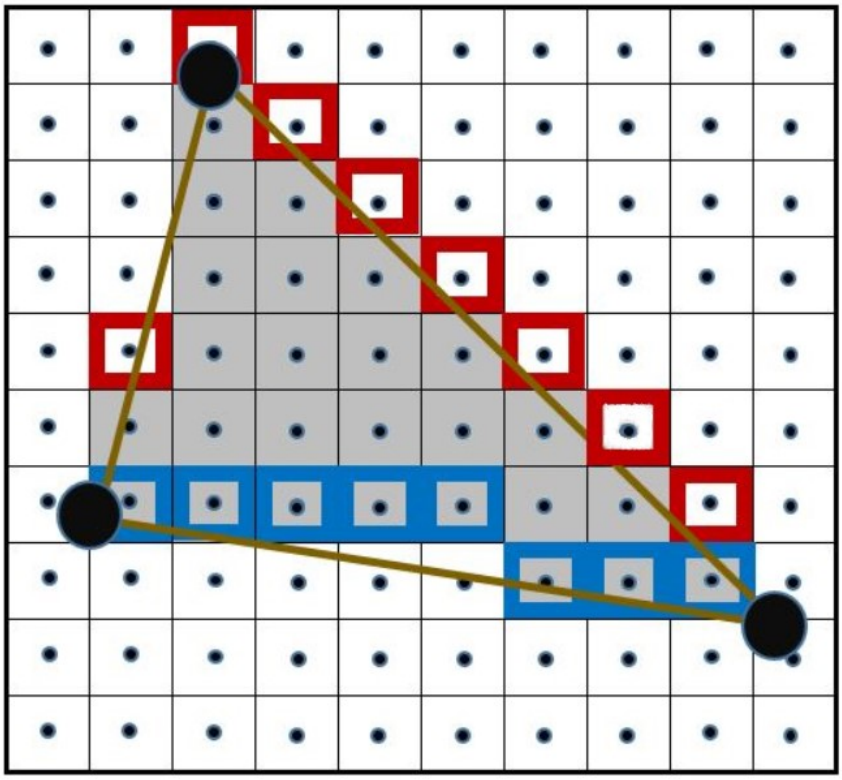

Figure 3: An illustration of positive (blue) and negative (red) supporting chains associated with edges of a closed chain P. The area in grey shows grid cells to be summarized by traditional algorithms

2.2. Greens theorem

Green's theorem, Green (1828) as commonly stated: Let C be a positively oriented, piecewise smooth, simple closed curve in a plane, and let D be the region bounded by C. If L and M are functions of (x, y) defined on an open region containing D and have continuous partial derivatives then

$$\oint_c (Ldx + Pdy) = \iint_D (\frac{\partial P}{\partial x} - \frac{\partial L}{\partial y})dxdy$$

Here we set the quantity we are interested in integrating, M, as $\partial L/\partial y$ and integrate this along columns to obtain, $F = L$, and $\partial P/\partial x$ are set to 0. In other words

$$M = \frac{\partial L}{\partial y},$$

$$\mathcal{F} = L = \int M dy,$$

and using Greens theorem

$$\iint_d M dxdy = -\oint_c \mathcal{F} dx.$$

2.3. Source Data

Any regular grid that stores numeric or categorical data can take advantage of the FZS approach. Categorical data pose a challenge in that each category requires it's own F data structure for the FZS algorithm to operate over. For this paper we demonstrate the utility of the proposed algorithm for a numerical dataset by using the United States Geologic Surveys' Impervious surface dataset, that has been subset for the Chesapeake River Basin and stored in an Albers equal area projection, NAD 83 (EPSG 5070) with dimensions 17,038 columns and 25,699 rows, bottom left coordinate x = 1295369 and y = 1669437).

Impervious surface in this case is stored as a percent value with values ranging between 0 and 100 for each individual grid cell. Impervious surface is used to model the percent of precipitation that will percolate into the ground vs the percent of horizontal flow. As an example, built surfaces such as roads and buildings typically have high impervious surface while natural surfaces such as woods and fields have lower values. This can be dependent on many factors including soil type and compaction. Impervious surface is used in many models of nutrient and flow modelling to estimate the relationship between rain and stream flow and associated sediment and nutrient loads. For Example the mapsheds model (noa, 2019) uses impervious surface to model the amount of waters running off of terrestrial areas, and to differentiate fluvial and non-fluvial flow rates.

We used a selection of standard USGS watershed boundary dataset cataloging units as the arbitrary polygons, they ranged in size from 12 digit hydrologic unit code, ($\approx 110\ km^2$) to the 2 digit HUC ($\approx 180 * 10^3\ km^2$) to test retrieval speeds of the fast zonal statistics algorithm. HUC codes are a form of nested hierarchy, where smaller codes contain larger codes. So for example the 2 digit code for the Middle Atlantic region of the United States ('02') contains the Chesapeake bay watershed ('0211') which in turn contains a number of smaller watersheds down to the HUC-14 code. For a more complete description refer to the Natural Resource Conversations Services website (Natural Resource Conservation Services, 2019).

2.4. Overview of Algorithm

The fast zonal statistics algorithm relies on a method to produce the integration field required by Green's theorem. In our example, we describe

a simple sum down algorithm that takes grid cell values and converts them into vectors fields in one dimension. The dimension is arbitrary, but in this example (and in the associated Jupyter notebook) we sum down the y dimension. Noting the restriction that the polygon P is a simple closed polygon with no intersections.

The Fast Zonal Data Preparation procedure, illustrated in Algorithm 1, simply pushes down the sum of the numeric grid (see Figure 2). This converts the scalar grid values into an integration field integrated over the y dimension. This pre-processing step can be accomplished in $O(n \times m)$ time, for a grid G with n columns and m columns, and can be implemented in a massively parallel way since each column can be independently processed.

Algorithm 2, *SupportChain*(G, u, v), is an auxiliary method for computing the subset of cells in G that form the support chain $\mathcal{C}(u, v)$ of a pair of consecutive vertices u and v in P. The algorithm is designed to mimic the default setting of rasterstats zonal summary algorithm. In this setting, grid cells whose centroid are inside the boundary of a polygon P are considered for computing the zonal statistics. Given a polygonal edge $(u, v) \in P$, our goal is to determine whether $\mathcal{C}(u, v)$ is in or out of the polygon and needs to be added or subtracted. The algorithm begins by identifying the first column, G_{u_x}, and the last column, G_{v_x}, of grid G that are stabbed by the polygonal edge (u, v). This computation is depicted in lines 5 and 6 of Algorithm 2. The algorithm for computing $\mathcal{C}(u, v)$ traverses the columns of G residing in the range $[G_{u_x}, G_{v_x}]$, and in each iteration will identify the unique cell in that column that belongs to supporting chain of (u, v), if one exists. The algorithm iterative constructs elements in $\mathcal{C}(u, v)$ be identifying cells of minimum distance from the edge $u, v)$). These latter steps are

depicted in lines 9-12 of Algorithm 2. These complete detail of $\mathcal{C}(u, v)$ are presented in Algorithm 2.

As an example, consider the execution of *SupportChain*(G, u, v) on vertices $u = (2.25, 10)$ and $v = (7.25, 100)$ with $\alpha = 1$. We note that $u_x < v_x$ so the pair u and v are not swapped. Next, G_{u_x} is set to $\lfloor 2.25 + \frac{1}{2} \rfloor$, i.e. 2. Similarly, the The last column G_{v_x} is calculated as $\lfloor 7.25 - \frac{\alpha}{2} \rfloor$, i.e., 6. Next, the algorithm iterates over the list of columns $[1, 2, ..., 6]$ identifying the row index of the cell belonging to support chain in each column using the slope of the line connecting u and v. For, this example the edge slope $m = 18$, and for column index $i = 2$, the row index will have values $j = \lfloor (2 - 2.25 + \frac{1}{2}) * 18 + 10 - \frac{1}{2} \rfloor = 14$. Using similar computation, we can identify all the cells for $i \in [2, 6]$ as $\mathcal{C}(2, 6) = \{(2, 14), (3, 32), (4, 50), (5, 68), (6, 86)\}$.

Algorithm 3, FZS(G, P), computes the sum of grid M for polygon P over the grid G as a real number. We will assume that the summed value grid $\mathcal{F}$ has been pre-calculated by FZS_DT(.) (Algorithm 1). Next we utilize the procedure SupportChain(.), (Algorithm 2), to project edges in polygon P onto the grid cells in G. The algorithm iterates over all the consecutive vertex pairs (u, v) that form an edge in P. Every edge is used to identify the support curve $\mathcal{C}(u, v)$ by a call of the form SupportChain(G, u, v). After identifying the support chain the algorithm decides whether the contributions of cells in $\mathcal{C}(u, v)$ are positive or negative by determining if their center are inside or outside P. As described in Section 2.1, this can be easily verified by comparing u_x and v_x. More precisely, if $u_x > v_x$ for each cell $G[i, j] \in \mathcal{C}(u, v)$ the sum flow $\mathcal{F}[i, j]$ should be added to total sum and

Algorithm 2 – **SupportChain**(G, u, v): Auxiliary method for computing support chains (refer to Figure 3).

Input: Regular grid G and two consecutive vertices $u, v \in P$.

Output: the *support chain* $\mathcal{C}(u, v)$ consisting of grid cells in G.

$\mathcal{C}(u, v) = \emptyset$

If $u_x > v_x$ then Swap u and v

$G_{u_x} = \left\lfloor u_x + \frac{\alpha}{2} \right\rfloor$

$G_{v_x} = \left\lfloor v_x - \frac{\alpha}{2} \right\rfloor$

If $G_{u_x} < G_{v_x}$

    $m = (u_y - v_y)/(u_x - v_x)$

    For $i \in [G_{u_x}, G_{v_x}]$ with increments of 1

        $j = \left\lfloor \left(i - u_x + \frac{\alpha}{2}\right) * m + u_y - \frac{\alpha}{2} \right\rfloor$.

        $\mathcal{C}(u, v) = \mathcal{C}(u, v) \ \cup \{G[i, j]\}$

    end

end

Return $\mathcal{C}(u, v)$

subtracted otherwise, this corresponds to cases when centroid of cell $G[i, j]$ is outside and inside P. Lastly, in the case where no centroids are covered, the algorithm returns a zero (e.g, if $P = ((1.24, 10), (1.45, 20), (1.24, 10))$). Algorithm 3 provides an overview of these processes.

Algorithm 3 – **FZS**(G, P): Algorithm to return summary statistics for polygon P over grid G (refer to Figure 3).

Input: Regular grid G with summed value grid $\mathcal{F}$ and a Polygon P made up of vertices $\langle v_1, v_2, ..., v_\ell \rangle$.
Output: Sum of grid cells grid F for cells centroids above P in the y dimension.
$\Sigma = 0$
For each edge $(u, v) \in P$
    For $(i, j) \in$ **SupportChain**(G, u, v)
        Σ += **Sign**$(v_x - u_x)\mathcal{F}[i, j]$
    end
end
Return Σ

2.5. Implementation

We demonstrate a working example of the FZS algorithm using a Github repository containing Jupyter notebook[6]. The note book is called FastZonalStatistics.ipynb and it relies on a number of accessory algorithms not described in this manuscript to move between coordinate systems in the real

[6]This repository is located at https://github.com/ScottHaag/fast_zonal_statistics.

world and the grid data structure G. We implemented the FZS algorithm using the default centroid in polygon method applied by raster stats.

3. Time and Space Complexities

The Time and space complexity of the overall system can be separated in terms of single pre-processing step necessary to create grid structure $\mathcal{F}$ vs the frequent retrieval query complexity of the FZS zonal statistic for arbitrary polygons. The pre-processing step is executed once for any input grid, it requires each grid cell to be visited exactly one time, which, for a $n \times m$ grid G will result in an $O(nm)$ computational complexity. Conversely, the complexity of the retrieval queries is dependant only on the shape and complexity of polygon P. As we will discuss below, the approach described here requires a trade-off between pre-processing time to reduce retrieval time, noting that the traditional approaches do not require the pre-processing step.

Given a polygonal region $P = \langle v_1, v_2, ..., v_\ell \rangle$, we will use $\tilde{\mathcal{C}}$ to denote the union of all support chains for the edges in P, i.e., $\tilde{\mathcal{C}} = \bigcup_{(u,v)\in P} \mathcal{C}(u,v)$. Let $N = |\tilde{\mathcal{C}}|$, it is easy to see that the FZS algorithm retrieves the polygon summary statistic from the grid data structure in $O(N)$. This is a direct consequence of the constant access time to grid data $\mathcal{F}[i,j]$ for a cell $G[i,j]$ within a grid of a finite size. The size of $\tilde{\mathcal{C}}$ is dependent on the structure of polygon P. We can see that by projecting vertex pairs corresponding to edges of P onto G, at worst $|\mathcal{C}(u,v)| = \lceil |u_x - v_x| \rceil$, and this implies $N = \sum_{(u,v)\in P} |u_x - v_x|$, which is the total distance traveled by coordinate pairs (edges) of P across G in one dimension. In the degenerative case where P spirals (i.e., P has a large surface to volume ratio) across G our

proposed method will perform similar to traditional techniques. We assume here that all coordinates of P are inside of G. Figure 4 depicts one such degenerative configuration for polygon P.

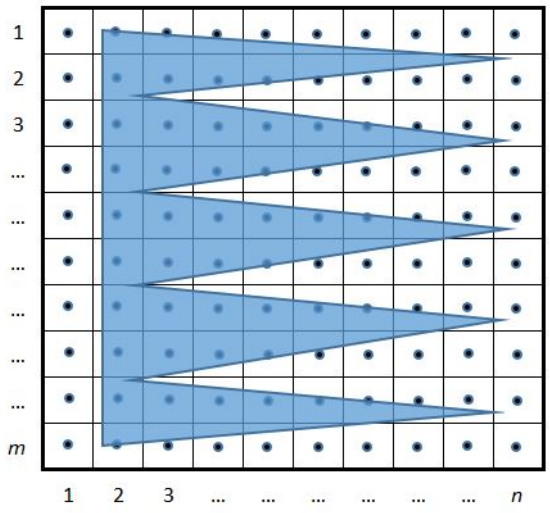

Figure 4: Degenerative case for the FZS algorithm where a polygon has a large surface area and a small interior.

The storage complexity for the FZS algorithm is primarily a function of the auxiliary data structure utilized for storing grid $\mathcal{F}$. This complexity is function of the size of the numbers that must be stored. For example given an integer unsigned 8-bit representation of grid $\mathcal{M}$ with values $[0, 1, ..., 255]$ and a grid G with dimensions 1000×1000, the max value possible would be $(2^8 - 1) * 1000$. Therefore an upper-bound on the storage complexity for a grid $\mathcal{F}$ for an $n \times m$ grid computed by FZS_DT() is $O(nm \log_2 \mathcal{S})$, where $\mathcal{S} = m \times (\max_{i,j} M[i, j])$ represents an upper-bound on the largest value stored in $\mathcal{F}$.

As we mentioned earlier, the computational complexity of creating $\mathcal{F}$

from M is linear with respect to the number of cells in G. Note that grid $\mathcal{F}$ only needs to be created once, as long as there are no changes to the grid G. After its creation, an unlimited number of polygon queries can be submitted to the FZS algorithm. Because the complexity and cost of creating $\mathcal{F}$ can be much higher then any one call to the FZS algorithm the most efficient use case for this algorithm is when many summary statistics are required for many polygon objects against a stable version of G. As a rule of thumb if every grid cell is involved in exactly one summary statistic calculation then the amortized pre-processing cost of the proposed algorithm will largely be negated. This can be violated if the arbitrary polygons are degenerative as shown in Figure 4, and it is not exactly true because the perimeter must still searched. But for large polygons over dense grids (e.g., the Alaska Region in table 1), it is very close.

4. Generalization of FZS

In this section we will provide an overview for generalizing the FZS algorithm in several ways. First, we provide the necessary modifications for extending the algorithm for handling categorical data. Next, we will consider the effect of missing data in computation of fast zonal statistics. Finally, we will discuss typical modifications to FZS algorithm for handling statistical functions such as standard deviation for a given region of interest.

4.1. Categorical data

Categorical data provides a unique challenge in comparison to continuous or numeric data types. In this case the most common output is the count per category which can then be easily converted to a percentage based

on the sum of all the categorical values that are returned. To use the Fast Zonal algorithm for categorical values requires a separate summed down grid for each categorical value. This has the effect of increasing the storage size by the number of categorical values, therefore storage size increases in a linear fashion based on the number of categorical values. This can be a high price to pay if the dataset has a large number of categorical values. We argue that this price might be worth it if the dataset is highly useful. As an example, several of the authors of this manuscript are currently working on the creation of an Application Program Interface (API) to share the 1 meter land use land cover dataset for the Chesapeake Watershed. This dataset is used as the basis for the CAST modelling tool (Chesapeake-Bay-Program, 2019), and underlies the assessment of nutrient loading to the Chesapeake Bay estuary system. Fast queries using the fast zonal statistics algorithm, will allow real time queries for arbitrary polygon objects, supporting the creation of decision support tools with web-query times (seconds). The objective of this work is to move the analysis from the desktop GIS environment to web based decision support tools.

4.2. Missing data

The impact of missing data does not effect the sum for numerical or count for categorical. But when the results need to be normalized over an area (e.g., percents or mean), then it is necessary to store missing data. In a similar way to the categorical example above, it is possible to store missing data as a categorical value (1 if not missing and 0 of missing). The count of missing values is then equal to the number of observations, and can be used in any mean or percentage calculation as the denominator. The cost

of this is the storage of another categorical grid, or a constant increase in storage costs for every grid dataset.

4.3. Standard Deviation

The calculation of standard deviation can be enabled by storing the summed down sum of squares, that is for every cell storing the squared value plus the sum of all the squared values above. To calculate the standard deviation for a polygon the sum is calculated in a normal way and divided by the sum of the missing value grid - 1. Storing the summed down grid, will require using a larger data type vs the categorical values, as over-flow issues might occur (Shafait et al., 2008). An efficient test to measure whether an overflow has occurred involves traversing each column from the top to the bottom, if a preceding grid cell value is found to be larger then a lower value then an overflow has occurred. This is true because each column is monotonic, that is values for every grid cell from the top to the bottom will all ways be equal or greater.

5. Results

5.1. Theoretical results for the complexity of retrieving statistics for watershed boundaries

To better understand the computational complexity for the FZS algorithm and traditional techniques for a number of existing geometric objects, we used the Watershed Boundary Data set (WBD) released on September 2, 2018 (NRCS, 2018). We compared the number of 30 meter grid cells on the boundary of the 22 watersheds with the number of 30 meter grid cells that make up the area of each watershed. These are not exact estimates as

a number of grid cells would overlap (i.e., would be on the border). But these errors should be consistent between the two techniques, and therefore should not impact the overall results. To calculate the perimeter length and area we reprojected the data to Albers equal area. A summary tabulation comparing the complexity of traditional techniques versus FZS is shown in Table 1 and Figure 5. According to the table, traditional approaches to return zonal statistics for the Alaska Region would require visiting approximately 2 trillion grid cells using traditional methods versus 720 thousand grid cells using the FZS.

5.2. Empirical results for selected watersheds of the Chesapeake River Basin

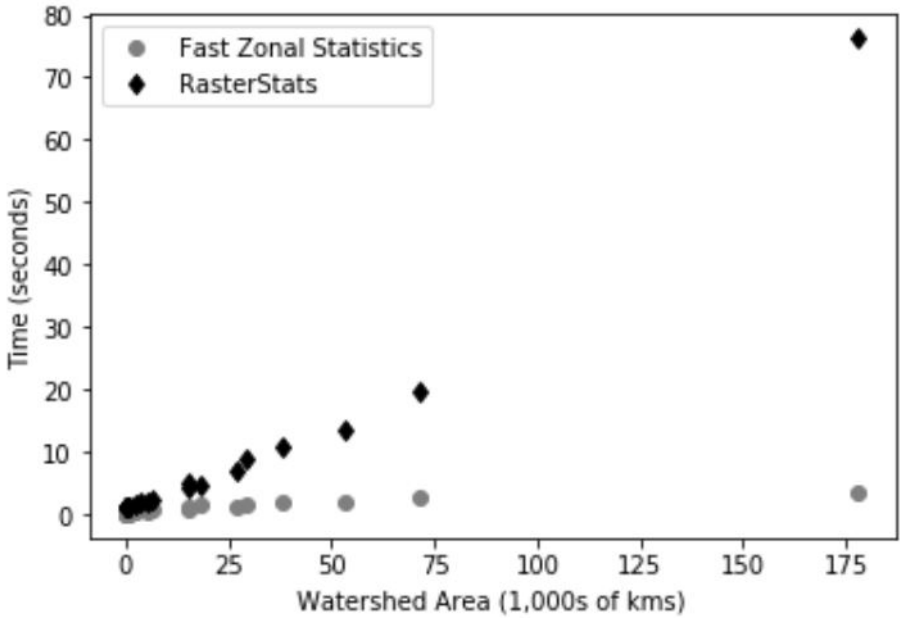

Figure 5: Comparison of the timing results between the fast zonal statistics algorithm and the existing raster stats Python library(from Jupyter Notebook)

WBD	**Perimeter** (km)	**Area** (km^2)	FZS (complexity)	**Rasterstats** (complexity)
Upper Mississippi Region	7,446	492,033	248,207	546,703,278
Souris-Red-Rainy Region	7,437	259,213	247,910	288,014,189
California Region	6,048	436,624	201,592	485,138,189
Upper Colorado Region	4,346	293,569	144,855	326,187,800
Hawaii Region	1,628	25,558	54,259	28,398,278
Pacific Northwest Region	8,531	836,517	284,371	929,463,322
Ohio Region	5,949	421,966	198,300	468,851,311
New England Region	4,488	198,838	149,602	220,930,633
Texas-Gulf Region	4,403	474,544	146,759	527,271,222
Lower Colorado Region	5,963	424,357	198,776	471,507,700
Mid Atlantic Region	5,051	276,482	168,369	307,202,222
Lower Mississippi Region	5,029	276,036	167,620	306,706,244
Caribbean Region	1,100	15,567	36,656	17,297,056
Arkansas-White-Red Region	7,075	642,212	235,842	713,569,411
South Atlantic-Gulf Region	7,194	739,947	239,801	822,163,833
Tennessee Region	3,443	105,949	114,752	117,721,189
South Pacific Region	524	3,376	17,477	3,750,556
Rio Grande Region	9,537	597,883	317,910	664,314,378
Great Basin Region	6,664	367,049	222,148	407,832,200
Missouri Region	9,998	1,349,418	333,275	1,499,353,611
Great Lakes Region	15,260	617,043	508,657	685,603,511
Alaska Region	21,531	1,866,798	717,688	2,074,220,422

Table 1: Estimated retrieval complexity (number of operations) for the 22 2-digit Hydrologic Units from the Watershed Boundary Dataset using the proposed algorithm (FZS) vs. traditional techniques over a 30 meter grid

We compared the actual retrieval time for a list of watersheds in the Chesapeake River Basin including examples from 2, 4 and 8 digit Hydrologic Units from the WBD (Table 2). Watersheds sizes ranged from 0.11 to $178 * 10^3$ km^2. The results showed that the proposed algorithm performed much faster than the standard raster stats library with the largest watershed returning in $\approx$ 76 seconds for rasterstats vs. $\approx$ 3.5 seconds using the FZS algorithm for a watersheds of $\approx 178 * 10^3$ km^2.

6. Discussion

The proposed FZS algorithm and associated data model has been shown to retrieve summary statistics faster than existing techniques and libraries. This increase of speed is a trade off with an increase in memory storage. For standard numerical grids (e.g. a grid storing impervious surface or precipitation), the increase in memory storage costs are related to the size of the maximum number stored in grid data M vs $\mathcal{F}$; for categorical datasets the problem is exasperated by the need to store a grid data $\mathcal{F}$ for every category.

A shortcoming of the proposed strategy is the inability to return the full array of statistics for a given polygon. For example rasterstats allows users to describe polygons based on their min, max, mean, count, sum, standard deviation, median, majority, minority, unique values, and range. The FZS algorithm could return the mean, count, sum, majority (for categorical), minority (for categorical), and standard deviation (by creating a summed down sum of squares), but it will not be able to return the other values in it's present form. We argue that this, while an important shortcoming

HUC Level	FZS Σ	**Raster Stats** Σ	FZS time (Secs.)	**Raster Stats** time (Secs.)	**Polygon Area** 10^3 km^2
2L	464,936,066	464,936,066	3.468	76.262	178.02
4L	75,097,455	75,097,455	1.328	4.836	15.21
4L	138,308,609	138,308,609	1.875	10.656	38.02
4L	137,562,599	137,562,599	2.587	19.705	71.22
4L	113,967,403	113,967,403	1.878	13.421	53.57
6L	75,097,455	75,097,455	0.739	4.327	15.21
6L	15,682,206	15,682,206	1.451	4.738	18.07
6L	43,807,532	43,807,532	1.638	8.753	29.28
6L	75,733,292	75,733,292	1.289	6.948	26.79
8L	30,722,824	30,722,824	0.605	2.226	6.44
8L	2,307,558	2,307,558	0.618	1.984	3.59
8L	7,606,454	7,606,454	0.428	2.120	5.24
8L	3,527,751	3,527,751	0.312	1.572	2.14
10L	463,238	463,238	0.203	1.270	0.64
10L	355,028	355,028	0.281	1.275	0.83
10L	6,180,231	6,180,231	0.317	1.305	0.89
10L	457,054	457,054	0.061	1.215	0.63
12L	57,067	57,067	0.084	1.149	0.11
12L	86,581	86,581	0.079	1.134	0.12
12L	38,039	38,039	0.097	1.140	0.11
12L	163,525	163,525	0.047	1.141	0.12

Table 2: Results from Jupyter Notebook comparing the proposed implementation with the rasters stats Python library (Perry, 2013). The columns indicate the USGS Hydrologic Unit Code (HUC) for the input polygon boundary, the sum results for the Fast Zonal Statistics, the sum result for the Raster Stats library, the running time for the Fast Zonal Statistics algorithm and the running time for the Raster Stats Python library and lastly the area of the input polygon.

of the proposed method, it is justified by the considerable reduction in the retrieval complexity.

Many common problems only require the sum and mean statistics. For example, the authors of this manuscript have worked on a number of projects that only require count (for categorical) or sum and mean statistics for continuous data. As an example consider computational models that are used to calculate the impact of landscape features to the loading of nutrients into streams and watersheds. These include the wiki-watershed (Stroud Water Research Center, 2019), the Stream Reach Assessment Tool (Academy of Natural Sciences of Drexel University, 2019), and the USGS Stream Stats web application (United States Geological Survey, 2019). These projects only require the count and associated area of land use and land cover categorical datasets, which we have shown can be accomplished using the methods described in this manuscript. In addition USGS Streamstats online web applications is one of the mostly highly utilized analytical websites in USGS. The utility of FZS could greatly increase our delivery speed of characteristic computation and easily allow for a standard suite of basin characteristics nationally.

This is of particular importance for the work in the Chesapeake Bay watershed because several local GIS based organization have focused on the development of high resolution (1 meter) land use and land cover datatsets. These data layers require significant processing time to resolve the count and sum measures required for nutrient modelling, inhibiting online decision support tools with standard zonal summary methods. With support from the United States Environmental Protection Agency, we have begun implementing working versions of the algorithm described in this manuscript

to solve these problems.

Lastly both the testing and discussion above, have not considered issues with disk to memory or Input/Output. A standard way of converting algorithms from in memory to out of memory involves replacing arrays with memory maps (for an example refer to Numpy's memmap package (Oliphant, 2006)). The Jupyter notebook associated with this manuscript loads into memory the entire raster data-set for the FZS algorithm. For very large polygons that can not be stored in memory raster stats throws an out of memory error. We argue that the advantages we describe here will be much larger if the calculations are done using out of memory data structures (such as memory maps), as the size of the I/O operation will be much smaller than traditional techniques. This is due to the fact that the proposed method scales in relation to the polygon boundary length, vs existing techniques that scale in relation to the polygon size.

7. Conclusions and Future Work

7.1. Conclusions

We describe an application of Green's theorem to extracting zonal statistics for a polygon from a regular grid data structure. This is a common operation with multiple existing libraries (e.g., ESRI Zonal Stats, SAGA, Rasterstats). To accomplish this we describe a data structure and an associated algorithm called Fast Zonal Statistics that can return zonal statics (mean, sum, count, standard deviation) geometrically faster than existing techniques in the field of geo-spatial sciences. The application of these integral image and summed down table to the field of image object detection has been profound with the Viola paper receiving over 20,000 citations.

Empirical results for several (n=21) watersheds in the the Chesapeake bay drainage show a decrease of between 0 and 72 seconds (Table 2), with the largest decrease occurring in the largest watershed going from approximately 3.5 seconds using the fast zonal statistics algorithm approximately 76 seconds using the existing rasterstats package (Table 1). We estimate that returning the average impervious surface for the Alaskan region of the watershed boundary dataset would require approximately 720 thousand operations using the FZS vs 1 billion questions using traditional techniques a 99.83% reduction in computational complexity.

The proposed methodology has two potential drawbacks, the first is the size of the auxiliary data structure which can be quite large especially if the source dataset is categorical. We estimate that the grid structure necessary to support the fast zonal algorithm for the 2011 USGS National Land Cover categorical Dataset (NLCD) would be about 77 times larger than the original grid structure and take up around 715 GB of disk space. The second disadvantage is the inability of the FZS to retrieve all the summary statistics provided by existing packages such as rasterstats. For example, the FZS algorithm cannot return the min or max values for a polygon.

7.2. Future Work

Because of the above listed constraints and the need to create an auxiliary grid structure the proposed technique is an excellent candidate for an online cloud resources that could be be used by multiple researchers for independent projects. For example, an API that provided summary statistics for any arbitrary polygon on a national precipitation map or the USGS National Land Cover Data could be of use to 1,000's of research projects

saving large amounts of computational time system wide.

Additionally, there is the issue of complex zones used with the FZS Algorithm such as with overlapping edges or internal holes. Interior holes can be dealt with by treating them as their own regions and subtracting them from the outside hull although we do not do that in our test code.

Lastly, integrating this proposed algorithm with the Fast Watershed Marching Algorithm (Haag et al., 2018) and (Haag and Shokoufandeh, 2017) will allow the creation of watershed boundaries with raster attribute information in web query time. Development of these technologies will support the creation of online decision support systems.

8. Computer Code Availability

The computer code library called fast zonal statistics version1, developed as part of this research is available at https://github.com/ScottHaag/fast_zonal_statistics. This code repository contains a jupyter notebook that created the output tables and statistics published in this manuscript. The code is distributed with a GNU General Public License v3.0.

CRediT authorship contribution statement

Dr. Haag in conversation with Dr. Tarboton came up with the Conceptualization for the algorithm, wrote a substantial portion of the manuscript, developed the code repository with Martyn Smith and worked on the Formal analysis to understand the probe for inside and outside of the polygon region with Dr. Shokoufandeh. Dr. Tarboton in conversation with Dr. Haag came up with the Conceptualization for the algorithm and helped to write and edit the manuscript. Martyn Smith helped develop the code library,

provided test cases and reviewed the manuscript. Dr. Shokoufandeh helped to write the manuscript and developed the method used to determine if a grid cell was on the left or right-hand side of a polygon region. In addition, Dr. Shokoufandeh worked with Dr. Haag to determine the computational complexity of the two proposed algorithms.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by William Penn Foundation Grants number 103-14 and 12-17 to Stroud Water Research Center for the development of Model My Watershed functionality.

References

Academy of Natural Sciences of Drexel University, 2019. Stream reach assessment tool. URL https://www.streamreachtools.org/

Azavea, 2018. GeoTrellis Documentation Release 1.0.0 Azavea. Tech. rep., Azavea. URL https://media.readthedocs.org/pdf/geotrellis/latest/geotrellis.pdf

Berg, M. d., Cheong, O., Kreveld, M. v., Overmars, M., 2008. Computational geometry: algorithms and applications. Springer-Verlag TELOS.

Chesapeake-Bay-Program, 2019. Chesapeake assessment scenario tool (cast). URL https://cast.chesapeakebay.net/

Conrad, O., Bechtel, B., Bock, M., Dietrich, H., Fischer, E., Gerlitz, L., Wehberg, J., Wichmann, V., Böhner, J., jul 2015. System for Automated Geoscientific Analyses (SAGA) v. 2.1.4. Geoscientific Model Development 8 (7), 1991–2007.
URL https://www.geosci-model-dev.net/8/1991/2015/

Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., Moore, R., dec 2017. Google Earth Engine: Planetary-scale geospatial analysis for everyone. Remote Sensing of Environment 202, 18–27.
URL https://www.sciencedirect.com/science/article/pii/S0034425717302900

Green, G., 1828. An Essay on the Application of mathematical Analysis to the theories of Electricity and Magnetism. Nottingham.
URL https://arxiv.org/pdf/0807.0088.pdf

Haag, S., Shakibajahromi, B., Shokoufandeh, A., nov 2018. A new rapid watershed delineation algorithm for 2D flow direction grids. Environmental Modelling & Software 109, 420–428.
URL https://www.sciencedirect.com/science/article/pii/S1364815218303530

Haag, S., Shokoufandeh, A., Jul. 2017. Development of a data model to facilitate rapid watershed delineation. Environmental Modelling & Software.
URL http://www.sciencedirect.com/science/article/pii/S1364815216308623

Kini, A., Emanuele, R., 2014. Geotrellis: Adding geospatial capabilities to spark. Spark Summit.

Natural Resource Conservation Services, 2019. Huc-codes.
URL https://www.nrcs.usda.gov/Internet/FSE_DOCUMENTS/stelprdb1042207.pdf

NRCS, 2018. Natural resource conservation services, watershed boundary dataset.
URL https://nrcs.app.box.com/v/gateway/folder/39640323180

Oliphant, T., 2006. NumPy: A guide to NumPy. USA: Trelgol Publishing, [Online; accessed 4/2019].
URL http://www.numpy.org/

Penn State University, 2019, Mapshed download page,
URL: http://www.mapshed.psu.edu/download.htm

Perry, M., 2013. Rasterstats python library.
URL https://github.com/perrygeo/python-rasterstats

Pham, M.-T., Gao, Y., Hoang, V.-D., Cham, T.-J., 06 2010. Fast polygonal integration and its application in extending haar-like features to improve object detection. pp.

942–949.

Shafait, F., Keysers, D., Breuel, T. M., 2008. Efficient implementation of local adaptive thresholding techniques using integral images. In: Yanikoglu, B. A., Berkner, K. (Eds.), Document Recognition and Retrieval XV. Vol. 6815. International Society for Optics and Photonics, SPIE, pp. 317 – 322.
URL https://doi.org/10.1117/12.767755

Stroud Water Research Center, 2019. Wiki-watershed.
URL https://wikiwatershed.org/

United States Geological Survey, 2019. Stream statistics.
URL https://streamstats.usgs.gov/ss/

Viola, P., Jones, M., 2001. Rapid object detection using a boosted cascade of simple features. pp. 511–518.