MONITORING EASTERN OKLAHOMA LAKE WATER QUALITY USING LANDSAT

By

CLAY BARRETT

Bachelor of Science in Environmental Science

Oklahoma State University

Stillwater, Oklahoma

2004

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2015

MONITORING EASTERN OKLAHOMA LAKE

WATER QUALITY USING LANDSAT

Thesis Approved:

Dr. Amy E. Frazier

Thesis Adviser

Dr. Jonathan C. Comer

Dr. Chris Zou

ACKNOWLEDGEMENTS

The author wishes to thank Marga and Xavier Barrett for their patience, understanding, and support during the completion of this thesis.

The encouragement of family and friends made the burdens of graduate school less difficult to bear. Those I would like to thank especially are Nadeen Barrett, Jill Holmes, and Lori Edgmand.

The perseverence and guidance provided by my committee members were invaluable in completing this research. Thank you all.

Name: CLAY BARRETT

Date of Degree: MAY, 2015

Title of Study: MONITORING EASTERN OKLAHOMA LAKE WATER QUALITY USING LANDSAT

Major Field: GEOGRAPHY

Abstract: The monitoring of public waters for recreational, industrial, agricultural, and drinking purposes is a difficult task assigned to many state water agencies. The Oklahoma Water Resources Board (OWRB) is only physically monitoring a quarter of the lakes it is charged with monitoring in any given year. The minimal sample scheme adopted by the OWRB is utilized to determine long-term trends and basic impairment but is insufficient to monitor the water quality shifts that occur following influx from rains or to detect algal blooms, which may be highly localized and temporally brief. Recent work in remote sensing calibrates reflectance coefficients between extant water quality data and Landsat imagery reflectance to estimate water quality parameters on a regional basis. Remotely-sensed water quality monitoring benefits include reduced cost, more frequent sampling, inclusion of all lakes visible each satellite pass, and better spatial resolution results. The study area for this research is the Ozark foothills region in eastern Oklahoma including the many lakes impacted by phosphorus flowing in from the Arkansas border region. The result of this research was a moderate r^2 regression value for turbidity during winter (0.52) and summer (0.65), which indicates that there is a seasonal bias to turbidity estimation using this methodology and the potential to further develop an estimation equation for this water quality parameter. Refinements that improve this methodology could provide state-wide estimations of turbidity allowing more frequent observation of water quality and allow better response times by the OWRB to developing water impairments.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The Clean Water Act of 1972 (EPA 2014b) was enacted to make the waters of the United States fishable, drinkable, and swimmable in recognition of their vital role. This recognition extends beyond them being necessary for life and includes their use for industry, recreation, and food production. The Oklahoma Legislature tasked the Oklahoma Water Resources Board (OWRB) with monitoring the state's waters to ensure they were complaint with the Clean Water Act regulations, with those not in compliance being listed on the 303d or impaired waters list. Oklahoma's water issues vary from a lack in the west, high sediment loads in the middle due to heavy clay soils, and anthropogenic phosphorus causing undesirable algal growth in the east.

The monitoring of public waters for recreational and drinking purposes is a difficult task assigned to many state water agencies. Currently, the OWRB's Beneficial Use Monitoring Program (BUMP) is only able to physically monitor a quarter of the lakes it is charged with monitoring in any given year (30 out of 130), with each lake being sampled a minimum of once every five years. Field sample campaigns are labor and time intensive outings requiring a full day per lake (or lake arm for a large lake) for sample and data collection, with some samples requiring additional processing in the lab after the field campaign.

Four sample points are average for all but the largest lakes in Oklahoma and serve to represent the mean condition of the entire lake. This average sample is sufficient for compliance purposes, but lakes are not homogenous and are comprised of distinct zones. Where each tributary to a lake or reservoir enters the water body is the riverine zone, which is typified by the highest nutrient and sediment loads. As these waters mingle with the lake water, solids settle out while nutrient loads dilute and they enter the transitional zone. These first two zones spatially expand and contract in response to increased and decreased flow from rivers. The lacustrine zone is the clearest part of the lake, nearest to the dam, and will typically contain the least amount of sediment and other suspended or dissolved matter.

The minimal sample scheme adopted by the OWRB is utilized to determine long-term trends and basic impairment but is insufficient to monitor the water quality shifts that occur following influx from rains or to detect algal blooms, which may be highly localized and temporally brief. The spatial heterogeneity of many parameters like sediment is supposed to be represented by including the lacustrine, riverine, and transitional zones in the sample regime (OWRB 2014b), but these variations are lost when samples are averaged together to represent an entire lake.

If the OWRB were able to monitor every lake potentially twice a month, the increased sample rate would provide more timely information and allow faster response time to water impairment events. The mandate of the OWRB is to implement the Clean Water Act not only by defining the Beneficial Uses of Oklahoma’s lakes and streams, but also to set Total Maximum Daily Loads (TMDL) which support these Beneficial Uses, monitor to ensure they do not exceed the TMDL for each water quality parameter, and grant or revoke permits in order to control point source pollution. In addition to more frequent samples, a remotely-sensed image could allow the discernment of which tributary or tributaries contributed an impairing level of a pollutant and produce faster, more accurate diagnosis of the source leading to reduced non-compliance with TMDL limits.

The potential benefits of using satellite monitoring to estimate Oklahoma lake water quality parameters include reduced sampling cost, increased sampling frequency, inclusion of all lakes visible (not hidden by clouds) at each satellite pass, better understanding of where in a lake water quality is impaired, and faster result processing. The output of this process would be lake-wide estimations of specific water quality parameters, which would be unsuitable for regulatory or punitive uses but would be well-suited to observe selected water quality issues more closely and allow better response times by the OWRB to developing water impairments.

The parts necessary to implement this type of remote observation technique for water quality monitoring are already present. The Landsat program collects imagery of the entire Earth every sixteen days, BUMP collects the water quality data necessary to calibrate the relationship with the satellite images, and remote sensing analysts have devised the techniques necessary to integrate the two. This research puts these pieces together with the objective of testing these methods in Oklahoma and providing these tools to improve monitoring of water quality by the OWRB and other water quality analysts.

Remote Sensing of Water Quality

The acquisition of aerial imagery while concurrently sampling water quality began in the early 1970's (Goldman et al. 1974) and remains the backbone of remote sensing-based water quality studies (Harrington, Schiebe, and Nix 1992; Chavez 1996; Jiazhu 2006). Traditionally, collection of *in situ* water quality samples was timed to coincide with flights contracted to gather aerial photographs. One simply waited until the plane flew past to get out in a boat, collected samples, and regressed the collected data against manually assigned color values from the photograph. Goldman et al. (1974) studied sediment flowing into Lake Tahoe, Nevada, from a tributary stream with recent urban development using this method. Soil matter suspended in the stream water influx contrasted highly with the unsedimented lake water due to the difference in spectral properties between water and sediment. Original studies such as this one utilized analog photographs and required manual assignment of subjective density values for sediment loads, which therefore limited comparisons to within a single image. While there have been continual improvements to the tools utilized in water research, the core methodology has not varied: collect water reference data as close to image acquisition as possible and then analyze the reflectance data to determine a relationship between the two.

There have been two technological advances enabling better water quality research. The first was the rise of the desktop computer with its image processing and Geographic Information System (GIS) software. This technological advance has allowed the sheer volume of image analysis to expand and the pace of image processing to quicken display and analysis of increasingly large and complex datasets. The second, and perhaps greatest, technological change has occurred in development and continued improvement of remote sensing satellites. Technically, satellite sensors were capable of capturing a greater range of spectra than film, and digital media gained prevalence with the spread of desktop computers.

The Landsat Earth observing platforms were the first satellites launched for the specific purpose of documenting the change of the Earth's surface (USGS 2013c). This series of polar-orbiting satellites is the longest running continuous repository of earth imagery. While initially developed for remote sensing of land cover, upgrades over the decades have expanded their range of operation to include water and thermal applications (USGS 2013b), and costs for procuring Landsat images were eventually eliminated. The Landsat archive is now a free repository of remotely sensed images of the earth's surface dating back to 1972. The availability of such a large pool of data has enticed water quality researchers to adapt Landsat data to their purposes. This is despite the availability of other instruments capturing hundreds of spectral bands (Han, Jin, and Yun 2006), specialized satellites capable of discriminating between sediment at the surface and at depth (Kunte 2008), and satellites designed solely for monitoring water (Lepistö et al. 2010). The archive of Landsat Thematic Mapper (TM) data are still continually applied for remote sensing of water quality due to the depth of the catalog, accessibility, and global coverage.

The pioneering efforts by water quality analysts to use satellite imagery were conducted using the earlier Landsat Multi Spectral Scanner (MSS) instrument, with only four wide spectral bands at a spatial resolution of 80 x 80m, on water bodies of various conditions. While the MSS sensor has been phased out, the techniques developed to correlate remote sensing information with water quality data developed using MSS imagery were applied to the next generation sensor, the Thematic Mapper (TM). While the Landsat TM sensor generates seven narrower spectral bands and a reduced spatial resolution of 30 x 30m, the MSS image analysis techniques were transferred with no significant difference in the correlation in some cases (Ritchie, Cooper, and Schiebe 1990).

One of the most important contributions was the more than decade-long collection of water quality data collected temporally near Landsat overpasses of Lake Chicot, Arkansas, by Harrington, Schiebe and Nix (1992). This type of long-term analysis was possible in part due to the digital format of the Landsat images, the authors' choice of optical water quality parameters instead of physical (i.e., turbidity instead of sediment), and their sheer persistence. Their analysis of 79 different sample dates that included four sample sites on the lake allowed the relationship between remote reflectance and sediment to finally become predictive. Ritchie, Schiebe and Cooper (1989) and Schiebe, Harrington and Ritchie (1992) investigated alternate models to linear regression for comparison of water reference data and image values, finding that exponential transformation techniques controlled for photosynthetic reflectance, thereby improving the estimation of sediment for clear lakes. However, the authors' choice of Landsat bands for high sediment water bodies was the Near Infrared (NIR) Band, as high concentrations of sediment at the surface caused these wavelengths to be reflected rather than absorbed, improving satellite returns.

Radiometric Correction

When satellites replaced planes for image acquisition, the increase in the distance between the Earth and the sensor increased the effect of atmospheric scattering (e.g., water molecules) on light captured by the sensor, with estimates ranging as high as 90 percent (Kutser et al. 2005) or in some cases even 100 percent (Hadjimitsis 1999). Atmospheric scatter is undesirable because the light reaching the sensor does not contain any reflectance information about the actual target, in this case water quality. Light can be reflected towards the sensor from the atmosphere, from the water surface, from the substrate at the bottom of the water body, or from constituents suspended within the water column (Bukata 1995). Only the light reflected from constituents suspended within the water column contains true information about water quality, and the minimization of all other reflectance paths is necessary to improve estimation results. Early efforts to correct for atmospheric scattering required collection of reference radiometric data during water quality sample events, which unfortunately are not available for archival images, as well as specialized software. In addition, some satellite sensors develop flaws and the imagery they collect needs to be corrected using ancillary data provided by the satellite management agency. This combined correction of error due to sensor flaws and atmospheric effects on the sensed image is called radiometric correction.

Radiometric correction was the first addition to the basic template for water quality research using remotely-sensed images. Correction for atmospheric diffusion and sun angle by Verdin (1985) was an early success in image processing for this purpose. Curran and Novo (1988) quickly declared diffusion and sun angle correction a necessary step in any water quality estimation research in order to correct error introduced during atmospheric transmission. Unfortunately, the technique developed by Verdin (1985) required field measurements of radiometric conditions or imperfect estimations based upon the highest lake brightness values and only accounted for one type of atmospheric error. The choice between estimating corrective factors or additional field data collection made this technique unattractive for both reasons.

Dark Object Subtraction (DOS) was developed to reduce both types of atmospheric error by accounting for the angle of the sun at different times of the year. Chavez (1996) accomplished this feat by expanding upon DOS radiometric correction by using the cosine of solar zenith (COST) to accurately estimate error for all Landsat MSS bands without any field radiometric measurements. The development of the COST-DOS method allowed researchers to correct images only using data from within the image and known properties of the sun, rather than inflicting additional data collection burdens. Some additional methods for image-based correction have been proposed utilizing more complicated filtering (Hadjimitsis and Clayton 2009) or by locating large Lambertian surfaces (Hadjimitsis, Clayton, and Retalis 2009), but COST-DOS remains the most frequently utilized method. This refinement of the radiometric correction technique unlocked archival data for use by researchers, a development that has been important for remote sensing of water bodies along with improved technology.

With correction techniques developed to account for the error introduced by moving the remote sensing apparatus beyond the Earth's atmosphere, the basic outline for remote sensing of water quality was complete. However, some researchers (Curran and Novo 1988) noted that research on different lakes still came up with differing relationships between reflectance and suspended sediment, regardless of the improvements in technique. They suggested that a better understanding of the multitude of environmental factors influencing the color and composition of suspended sediment could improve water quality estimation abilities using satellites.

Components of Water Reflectance

The difficulties faced when trying to decode water quality remotely led to research on individual water quality parameters. Dekker, Vos and Peters (2002) modeled Total Suspended Matter (TSM) on a regional basis, but TSM is an aggregate measure of water quality and does not inform about specific individual parameters such as Suspended Sediment Concentration (SSC or often just sediment) or chlorophyll. Onderka and Rodný (2010) derived a relationship between NIR saturation and SSC when there was adequate variation in reflectance within an image. The use of software-assisted classification was successful for detecting sediment plumes without *in situ* sampling (Guneroglu, Karsli, and Dihkan 2013), supporting the concept of completely remote analysis of some water quality parameters. However, this still did not resolve the difficulties of adapting a region-specific reflectance equation to new regions.

Preliminary research efforts were focused on investigating which Landsat spectral bands optimally represented specific water quality parameters. The primary factors influencing the spectral reflectance, or color, of water are the relative concentrations of chlorophyll, dissolved organic matter, and suspended minerals (Bukata 1995; Jensen 2007). While these three constituents are known, their interactions at varied concentrations in solution have been difficult to decipher. Testing Landsat application to lakes, Carpenter and Carpenter (1983) selected the MSS Green band as the optimal band for detecting sediments in all but the most turbid water bodies. Ritchie, Cooper and Yongqing (1987) concluded that both the MSS Green and Red bands were optimal for sediment detection in clear lakes, but Ritchie, Schiebe and Cooper (1989) eventually noted that reflectance between lakes varied even for the same sediment level.

Specific water quality parameters of interest to monitoring water bodies *in situ* are often limited to nitrogen concentration, phosphorus concentration, and the amount of sediment suspended in the water column. Unfortunately, nitrogen and phosphorus have not yet been directly detected optically. Instead, chlorophyll has been used as a proxy for remote sensing of phosphorus and nitrogen (Chen and Quan 2012; Song et al. 2012). This is an imperfect arrangement, as each of those substances can be the limiting factor for algae growth, and sediment can carry these elements bound to their surface by ionic bonds. Despite these challenges, chlorophyll has been the most widely used method for remote phosphorus prediction to date.

The estimation of chlorophyll using remote sensing is also nuanced. Some researchers suggest that the change in plant communities between seasons means the expected chlorophyll color varies with season (Schalles, Rundquist, and Schiebe 2002; Zhou et al. 2006) and additionally also varies between lakes (Schalles et al. 1998). Schalles, Rundquist and Schiebe (2002) attempted to model the interaction between sediment and chlorophyll by taking spectral reflectance values by spectrometer for tanks of water with known chlorophyll concentrations while adding clays of various colors. The results showed a consistent increase in albedo with clay addition but an inconsistent shift of the location of the maximum reflectance within the green visible light range depending on the color of clay added. While Landsat data are averaged together to make a broad green band and peak shift effects are moderated, these intricacies in the relationship between chlorophyll and sediment reflectance led some researchers to advocate against a unified sediment reflectance theory (Dekker, Vos, and Peters 2002; Jiazhu 2006; Nas et al. 2010).

Landscape Scale Research

Rather than continuation of research on a global relationship between water quality parameters and reflectance, researchers focused on establishing the optimal water reflectance relationship for specific regions, often bounded by a single satellite image scene and for a single water quality parameter. This shift to the landscape scale often led to a larger sample pool, and was often based upon extant water quality data collected over longer time scales than earlier studies.

Kloiber, Brezonik and Bauer (2002) analyzed 25 years of Secchi Disk Depth (SDD) data with Landsat images for Minnesota. The Secchi Disk is a disk with alternating black and white color attached to a chain and lowered into a water body to measure optical clarity, which is recorded as the lowest depth at which the disk can still be seen. There were SDD data for some lakes in the region, but the authors extrapolated using the relationship calibrated by the few lakes with sample data to include all lakes in each Landsat scene. They found that the temporal window for statistically significant correlations for SDD extended to images captured up to seven days before or after the sample event. SDD is an inorganic measurement and is very stable during the summer dry period, which enables the wide temporal range of useable imagery. Additionally, Kloiber et al. (2002) discovered a significant relationship between SDD and the combination of the Blue band/Red band.

The SDD band combination and landscape level application techniques that were established by Kloiber, Brezonik and Bauer (2002) and Kloiber et al. (2002) have been verified in other geographic areas including Michigan (Olmanson, Bauer, and Brezonik 2008), Turkey (Nas et al. 2010), and western Maine (McCullough, Loftin, and Sader 2012). In Maine, it was found that clear lakes were less suited for remote estimation of water quality due to the risk of light hitting the bottom of the lake and reflecting the bottom substrate rather than reflecting just the components of water column (McCullough, Loftin, and Sader 2012), requiring masking to exclude these portions of the lake from analysis (Allan et al. 2011). Although these researchers used the same spectral bands, Kallio et al. (2008) cautioned during a review of other SDD works that reflectance equations require recalibration when applied to new areas due to regional differences in water composition.

While these studies successfully modeled a single water quality parameter for a subset of lakes with *in situ* reference values, the ability to model multiple parameters at a landscape scale was the next evolution in water quality research. Whereas studies had been utilizing a few or single lakes to calibrate a scene for a single parameter (Kloiber et al. 2002) or even multiple parameters within a few lakes (Allan et al. 2011), the ability to effectively model multiple parameters at a regional scale was not demonstrated until recently. An example of overcoming this shortcoming is the recent work by Hicks et al. (2013) in New Zealand. Using extant Turbidity, Total Suspended Sediment, and SDD data for a few lakes, they expanded use of the regressed relationship between these water quality parameters and Landsat imagery to all lakes in a Landsat scene, even lakes that were lacking *in situ* monitoring data. The study period of ten years comprised 53 Landsat scenes, each containing a variable number of water bodies free of cloud and shadow. Therefore, the authors developed automated techniques for processing a vast amount of data into corrected images displaying only the open water used to extract reflectance values.

The result of this study was an exceptionally high r^2 value for Turbidity (0.924) utilizing the NIR band and a good r^2 value (0.670) for SDD utilizing the ratio between the Red and Blue bands. While the Hicks et al. (2013) study applied the resultant formula to all water bodies in a Landsat image, this research only estimates water quality parameters on water bodies that have contributed water quality sample data to developing the reflectance coefficients.

Objectives

The Hicks et al. (2013) findings were specific to New Zealand. In order to validate their methodology for other regions, the method needs to be tested in different regions and on other water bodies. Additionally, many researchers including Hicks et al. (2013) use Landsat image boundaries to define the study region. While use of the Landsat image coverage is convenient, it does not ensure similarity of environmental factors pertaining to lake classification. Oklahoma is an ecologically diverse state but contains a region in the Ozark foothills with the greatest similarity to the clear, shallow lakes of New Zealand. Therefore, Hicks et al.'s (2013) methodology will be validated in Oklahoma's region most similar to New Zealand to see if their methodology can be successfully applied in this new region with a different definition of the study area.

The primary objective of this thesis is to test the methodology developed by Hicks et al. (2013) for remote water quality analysis for a region in Oklahoma. Specifically, this research aims to calibrate the water reflectance equation developed by Hicks et al. (2013) for the parameters of turbidity and chlorophyll, for the lakes located in the eastern region of Oklahoma. Successful calibration of the water reflectance equation for each water quality parameter will allow lake-wide estimation of turbidity and chlorophyll each time that lake is imaged by a Landsat sensor, thereby decreasing the sampling interval and improving lake monitoring and management. In addition to developing remotely-sensed water reflectance equations for eastern Oklahoma lakes, this study also aims to develop image processing scripts, which will automate the repetitive data processing tasks associated with image calibration and analysis, thereby making the methodology accessible and useful to individuals involved with water quality management in Oklahoma. If successful, these scripts will be released for any parties interested in using Landsat to monitor lake water quality remotely. It is hoped that the methodology of remotely-sensing water quality can ultimately be adopted by the OWRB and expanded to the rest of Oklahoma.

CHAPTER II

STUDY AREA

The study area includes all major lakes in the Ozark foothills region of eastern Oklahoma. Some of the waters in this region have experienced serious impairments due to eutrophication. Common impairments are high total phosphorus value and subsequent low Dissolved Oxygen due to consumption by algae (EPA 2010). These lakes are also of particular political interest and have been the focus of scientific studies due to the nature of their impairment being caused by activities upstream. While not the primary motivation for selection of a study area, the political and economic importance of this area was taken into consideration while selecting an area in Oklahoma to test remote-sensing of water quality. The EPA Ecoregion Level II boundary for the Ozark, Ouachita-Appalachian Forest Ecoregion approximates the area comprising the clear waters of Oklahoma and form the starting boundary for the study area. An ecoregion is not a homogeneous area, but it is defined by landscapes with common characteristics (McCullough, Loftin, and Sader 2012). For example, the soils in this ecoregion are Permian, unlike the oxidized Pennsylvanian units under the western half of the state (OGS 2008).

The Ozark, Ouachita-Appalachian Forest Ecoregion includes many rivers leaving the Ozark Plateau and draining through Oklahoma on their way back into Arkansas as the Arkansas River. The longest water system is the Neosho River and associated impoundments, with sources in

Kansas, Missouri, and Arkansas. Important uses of these watersheds include Eucha-Spavinaw as Tulsa's freshwater supply, Lake Eufaula as the largest recreational lake in Oklahoma, and tourism on Oklahoma Scenic Rivers like the Illinois River. However, the EPA ecoregion boundary is not perfect for this study as it often crosses back and forth across streams and water bodies as seen in Figure 1.

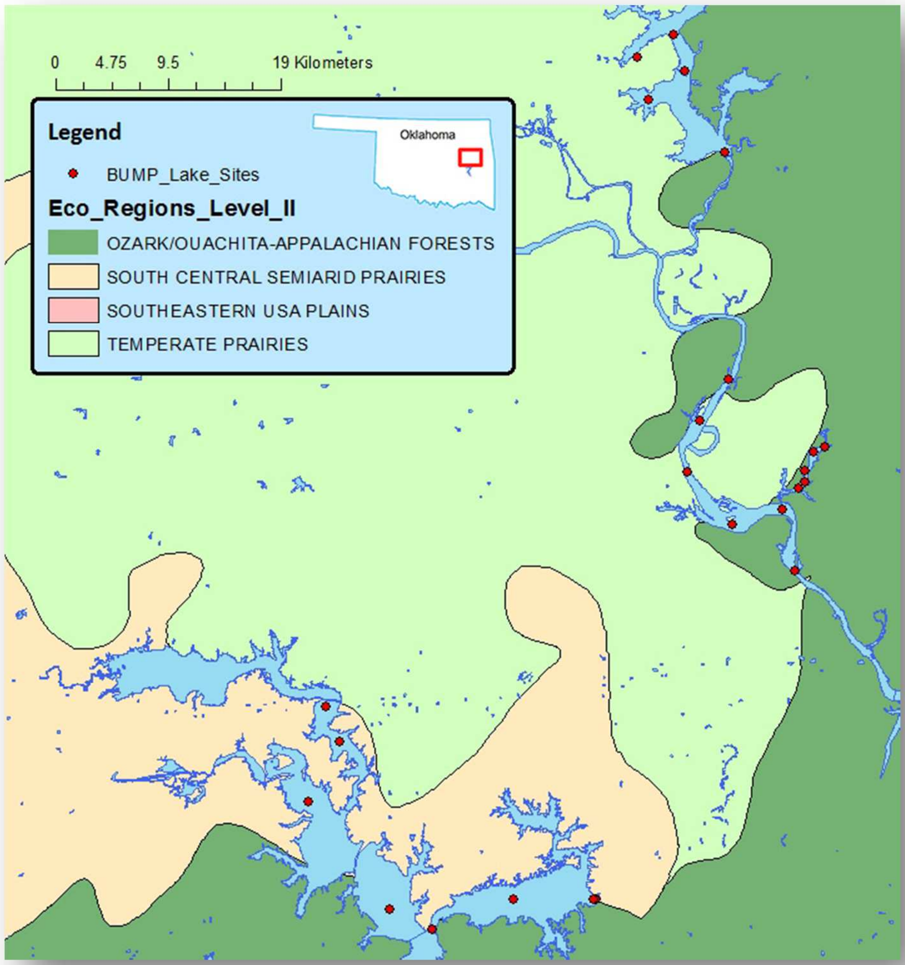

Figure 1: Example of the location of the EPA Level II Ecoregions in relationship to water bodies in eastern Oklahoma.

Therefore, the boundary of the study area has been manually extended to include the entirety of water bodies belonging to an Ozark, Ouachita-Appalachian Forest Ecoregion stream system.

Subsequently, all of the lakes along the Neosho River are included, all arms of Lake Eufaula have been contained within the study area, and the entirety of Pine Creek Lake has also been included. These modifications can be seen in Figure 2, where it is also clear that the Ozark, Ouachita-Appalachian Forest Ecoregion extends beyond Oklahoma's borders. The study area includes Oklahoma exclusively because the reconciliation of data from different state agencies is outside the scope of this thesis. Inclusion of water quality data from Missouri and Arkansas and estimating water quality for their lakes is a logical extension for future work from the base study provided here.

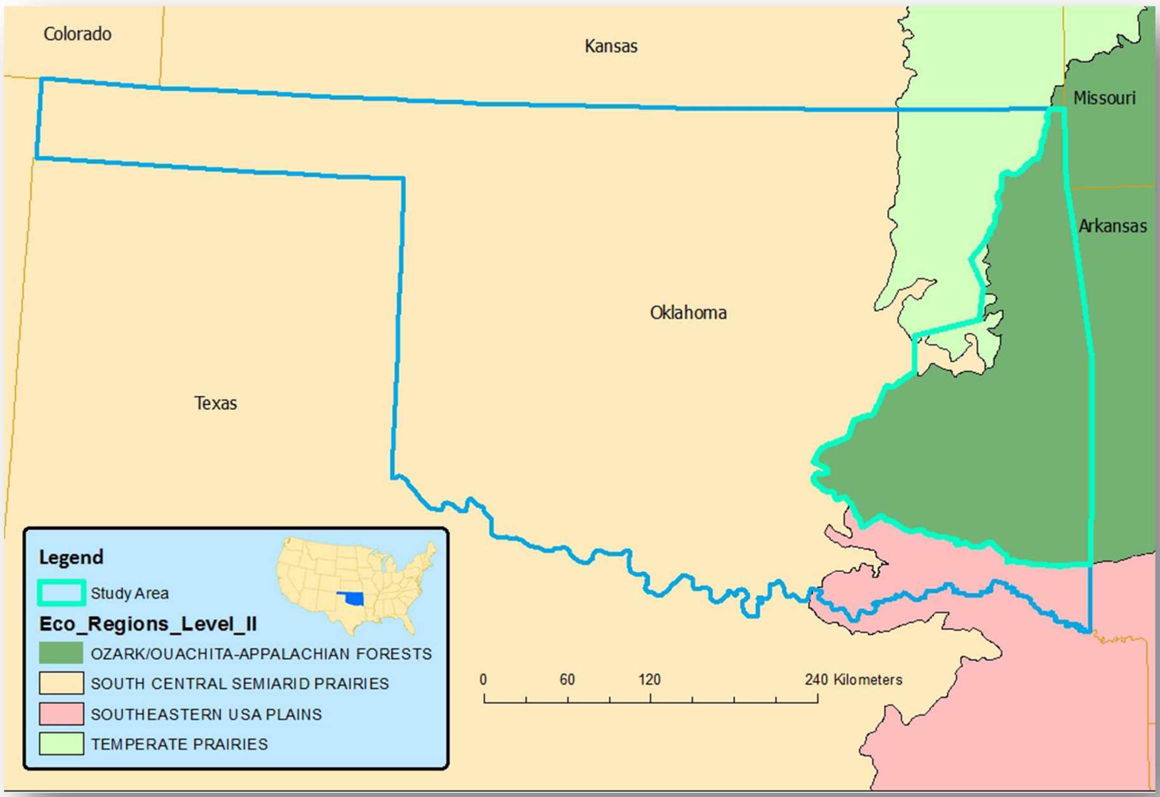

Figure 2: Oklahoma's EPA Level II Ecoregions showing the modified study area in eastern Oklahoma.

The primary water quality issue for waters in the study area is the presence of phosphorus in streams exiting Arkansas causing high algal production in lakes on the Oklahoma side of the border, thereby degrading water quality and leading to these lakes being listed as impaired by high levels phosphorus, chlorophyll, and low dissolved oxygen (EPA 2014a). Because of these deleterious effects on watersheds due to excessive phosphorus, reduction of influx and methods to mitigate phosphorus's effect on Oklahoma's waters are the subject of ongoing research (Haggard, Moore, and DeLaune 2005; Haggard and Soerens 2006; Busteed et al. 2009). The state of Oklahoma has sued both the EPA and municipalities within Arkansas in defense of these waters, resulting in the establishment of a separate TMDL on phosphorus loads for Oklahoma's designated Scenic Waterways enforceable at the Oklahoma border (EPA 2010). The ability to remotely monitor these waters is essential for researchers focused on this area.

CHAPTER III

DATA

The data for this thesis are all extant and are available freely from the BUMP database and the Landsat Archive. The methodology used to process these two datasets into the final reflectance equation is detailed in the next section.

Beneficial Use Monitoring Program

The OWRB's BUMP database provides the primary data component for this study. The database contains all the water quality data needed as ground reference, which negates the need for field work. The BUMP monitoring program began in 1998 in order to monitor 130 of Oklahoma's largest lakes to

> "document Beneficial Use impairments, identify impairment sources (if possible), detect water quality trends, provide needed information for the Water Quality Standards, and facilitate the prioritization of pollution control activities" (OWRB 2012: 3)

OWRB designates different levels of water quality standard depending upon the Beneficial Use(s) designated for each lake. The five Beneficial Uses are (1) Fish & Wildlife Propagation, (2) Aesthetics, (3) Agriculture, (4) Primary Body Contact Recreation, and (5) Public & Private Water Supply. Each lake is sampled for parameters that have tolerance ranges for each activity and these samples are used to determine whether the lake is compliant for each designated Beneficial Use or Uses. The water quality parameters sampled by OWRB include nutrients such as nitrogen and phosphorus, optical clarity, dissolved metals, salt concentrations, biological indicators such as chlorophyll, and physical parameters including depth.

These data are collected using a combination of digital mensuration tools, field measurements, and lab analysis to evaluate the multitude of parameters sampled (OWRB 2012). For this project, the focus is on the levels of suspended sediment and phosphorus in the lakes. Chlorophyll-α is utilized as a proxy for remote detection of phosphorus. To measure chlorophyll-α, water samples are collected in a flask at each sample point and returned to the lab for extraction and measurement by spectrometer. Chlorophyll-α is ratio data, precise to four significant figures, and reported in units of milligrams per liter (µg/L).

Suspended sediments and turbidity have both been used to successfully estimate water quality (Hicks et al. 2013), but field measurement of suspended sediment was discontinued for lakes other than Lake Thunderbird after August of 2000. Turbidity has been collected in its stead, which is the composite of suspended sediments, chlorophyll, dissolved organic matter, and even air bubbles (particularly in moving streams). Turbidity is interval data, precise to four significant figures, and reported in Nephelometric Turbidity Units (NTU). The turbidity parameter is detected with a digital turbidity meter that calculates the amount of light passing through the water column. Perfectly clear water has no turbidity and a reading of 0 NTU, whereas heavily sedimented water has a turbidity value as high as 1600 NTU.

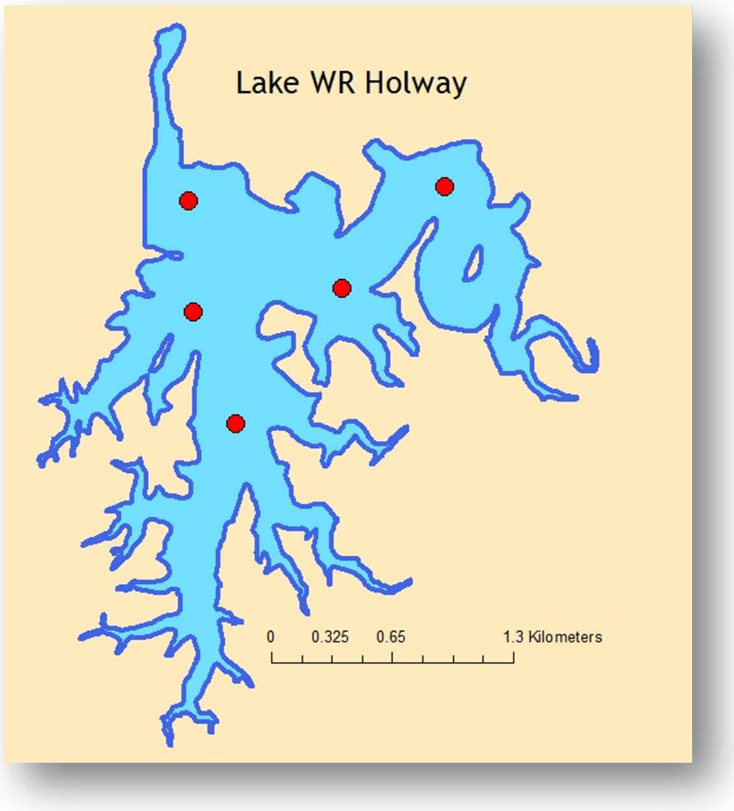

Figure 3: Lake WR Holway Beneficial Use Monitoring Program (BUMP) sample points (red dots).

Each lake has multiple sample points situated to capture the conditions in the riverine, transitional, and lacustrine zones (Figure 3). Larger lakes tend to have more sample sites and even separate sample events for different arms when necessary. OWRB sampling campaigns occur quarterly and all lakes are sampled at least once every five years. However, sampling intervals are not systematic, and the repeat rate for the larger lakes is more frequent than for the smaller water bodies. This sporadic sampling regime makes for spotty availability of data events for each particular lake.

The greatest complication when using BUMP data is the inconsistency of parameter collection, even among just chlorophyll and turbidity. For example, there are no sampling events in the database between 1998 and 2002 when these two parameters are captured during the same sample event. Despite these issues, the database contains many thousands of entries and is an invaluable asset that enables this and many other water quality research projects in Oklahoma.

Landsat Imagery

There have been six functioning Landsat satellite missions, but this thesis is limited to the data produced by Landsat 5 (LS5) and Landsat 7 (LS7) for reasons explained below. First, some general information about the Landsat satellites will be described. Each satellite was placed in an offset polar orbit with a temporal resolution of sixteen days. First generation Landsat (LS1 – LS3) carried a Multi Spectral Scanner instrument (MSS), but the second generation (LS4 – LS7) phased the MSS out and contained the Thematic Mapper (TM) and eventually Enhanced Thematic Mapper (ETM+) instruments, which generated the data for this project. The most recent satellite in the Landsat family (LS8) contains a new instrument, the Operational Land Imager (OLI), which is not compatible with the TM data due to band width contraction. As is shown in Table 1, only one boundary for one band was kept in place between the TM and OLI instruments. Effectively, every band is new since the sampled wavelengths narrowed by at least one micrometer, a 14 percent reduction in the Blue band. NIR was reduced from thirteen micrometers to three, a reduction of 69 percent. As the table shows, all the critical bands for water quality research have been altered in the latest Landsat sensor (red highlight), and cannot be compared with prior satellite data.

Table 1: Landsat (LS) band comparison showing wavelength ranges

	Aerosol	Blue		Green		Red		NIR	
LS 4-7	-	0.45	0.52	0.52	0.60	0.63	0.69	0.77	0.90
LS 8	0.43–0.45	0.45	0.51	0.53	0.59	0.64	0.67	0.85	0.88
	Wavelengths are in micrometers								

Landsat Spectral Bands

There are multiple nomenclature methods for identifying the Landsat bands. Unfortunately the simplest, numerological, was offset with the addition of a new band at the spectral origin point (as is seen in Figure 4). Therefore each band will be described by the nominal system using the visible light range the band corresponds with or its common usage (e.g. Red, Blue, Aerosol, and NIR).

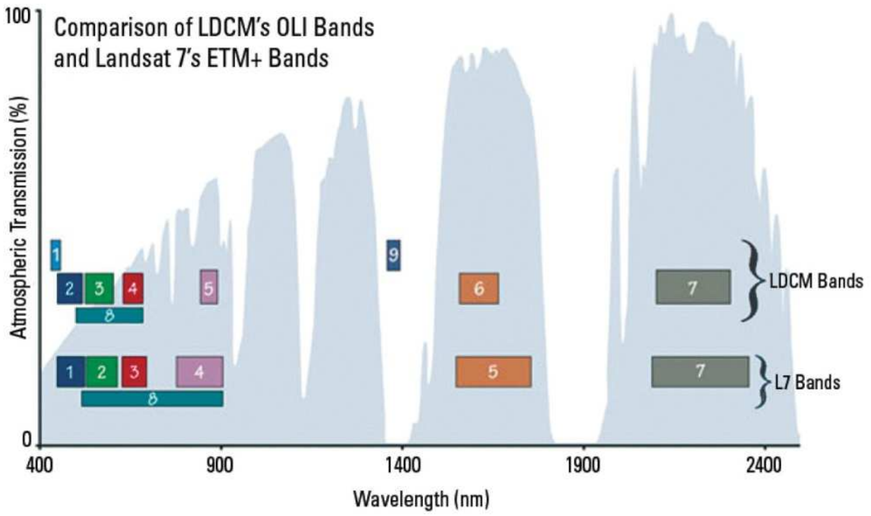

Figure 4: Landsat 8 & Landsat 7 spectral band comparison (image acquired from http://pubs.usgs.gov/fs/2013/3060/pdf/fs2013-3060.pdf).

The Coastal Band is only available on LS8 and was added to detect coastal aerosol and shallow water. There have only been a few studies testing this band, so its utility for water quality studies is not yet known. It has been suggested that it can be used not only to correct the Blue Band for diffusion of light but also to monitor airborne pollutants. The Blue Band is the band that best penetrates water. The flip side is that water vapor interferes with this band the most, making it inconsistent and unsuitable for many research projects.

The Green Band corresponds with the visible reflectance of vegetation. This band works in conjunction with other bands to detect boundaries between vegetated areas and soil. The Red Band includes the wavelengths absorbed by chlorophyll. The variation in absorption by this band is one indicator of plant vigor. The NIR band is highly absorbed by water, making finding water bodies with this band simple due to their high contrast against surrounding soil and vegetation. However, a lake reflecting this band indicates there is suspended sediment reflecting light back at the sensor.

The remaining Landsat bands are not typically used for detecting water quality directly, but some are used during image processing and therefore all will be described here for completeness. The Short Wave Infrared 1 (SWIR1) is used to detect soil moisture and is capable of discerning between cloud, snow, and ice. Therefore it will be used to build the not-cloud mask. Short Wave Infrared 2 (SWIR2) is used for geological surveys as it can tell between geologic formations. The Thermal Band is used to detect temperature, useful in this study only for cloud detection. Finally the Panchromatic band is a higher spatial resolution composite of the lower visible bands. This is used for cosmetic improvements to images and has no scientific utility.

Timing of Landsat Collection Events

Prior research has indicated that there is a narrow window of time to collect appropriate imagery to utilize preceding or following an *in situ* data collection event in a water body: the day before, the day of, or the day after (Kloiber et al. 2002; Hicks et al. 2013). Therefore, the primary limitation for the application of Landsat images to this project is the availability of cloud-free images of the lakes taken within this narrow time window around a data collection event from the OWRB containing both turbidity and chlorophyll. Narrowing the field further is the discovery that incidence of both parameters only falls between the years of 2002 and 2012 within the BUMP data collected for this project. These factors have reduced the pertinent datasets to only include LS5 and LS7 as seen in Figure 5.

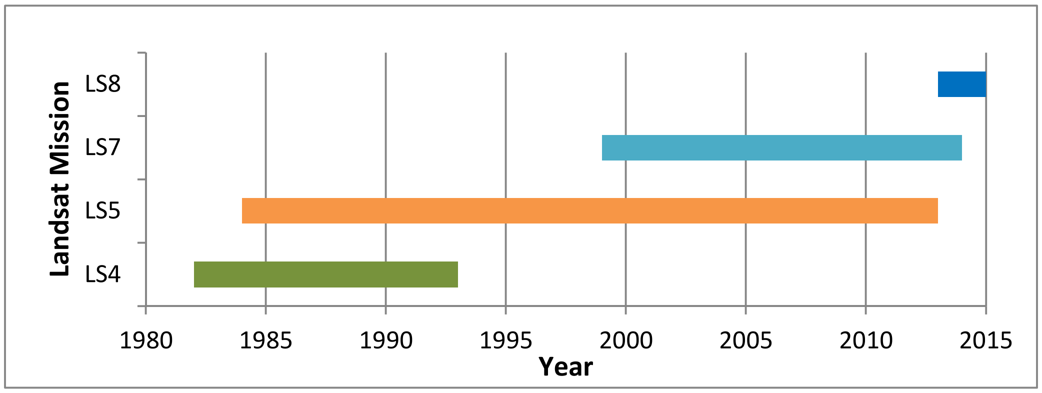

Figure 5: Landsat (LS) mission timelines.

The LS7 collection was disrupted in 2003 when a component on the satellite malfunctioned and began producing images with stripes devoid of data on every image. The post-2003 LS7 datasets are now designated with ‘SLC-off’ to denote these problematic images. This thesis utilizes this part of the archive in spite of the missing data since data selection only includes small pixel sampling windows around BUMP sample points. The sporadic selection of data from each image allows these SLC-off images to be used for analysis, but will not be able to fill in the missing data when estimating water quality parameters.

CHAPTER IV

METHODS

The methodology for this thesis is a scripted way of clipping, correcting, analyzing, and selecting pixels associated with each BUMP sample point. Mean pixel values associated with each sample point are then assessed by regression to determine the presence or absence of a statistical relationship between reflectance and sampled values for turbidity and chlorophyll (Figure 6). The primary tools for image pre-processing and processing are the Python scripting language (PSF 2014) and the ArcMap GIS package. Image manipulation is performed using ArcMap (ESRI 2014) via the 'arcpy' module, a version of Python built by ESRI with external access to ArcMap's functions.

This means Python is utilized to control ArcMap from outside the program allowing ArcMap functions to be called the same way any Python function is called. The development of these script tools in Python means that they can be distributed and easily utilized for other research projects. However, the usage of arcpy for this methodology requires the presence of both an ArcMap and Spatial Analyst license.

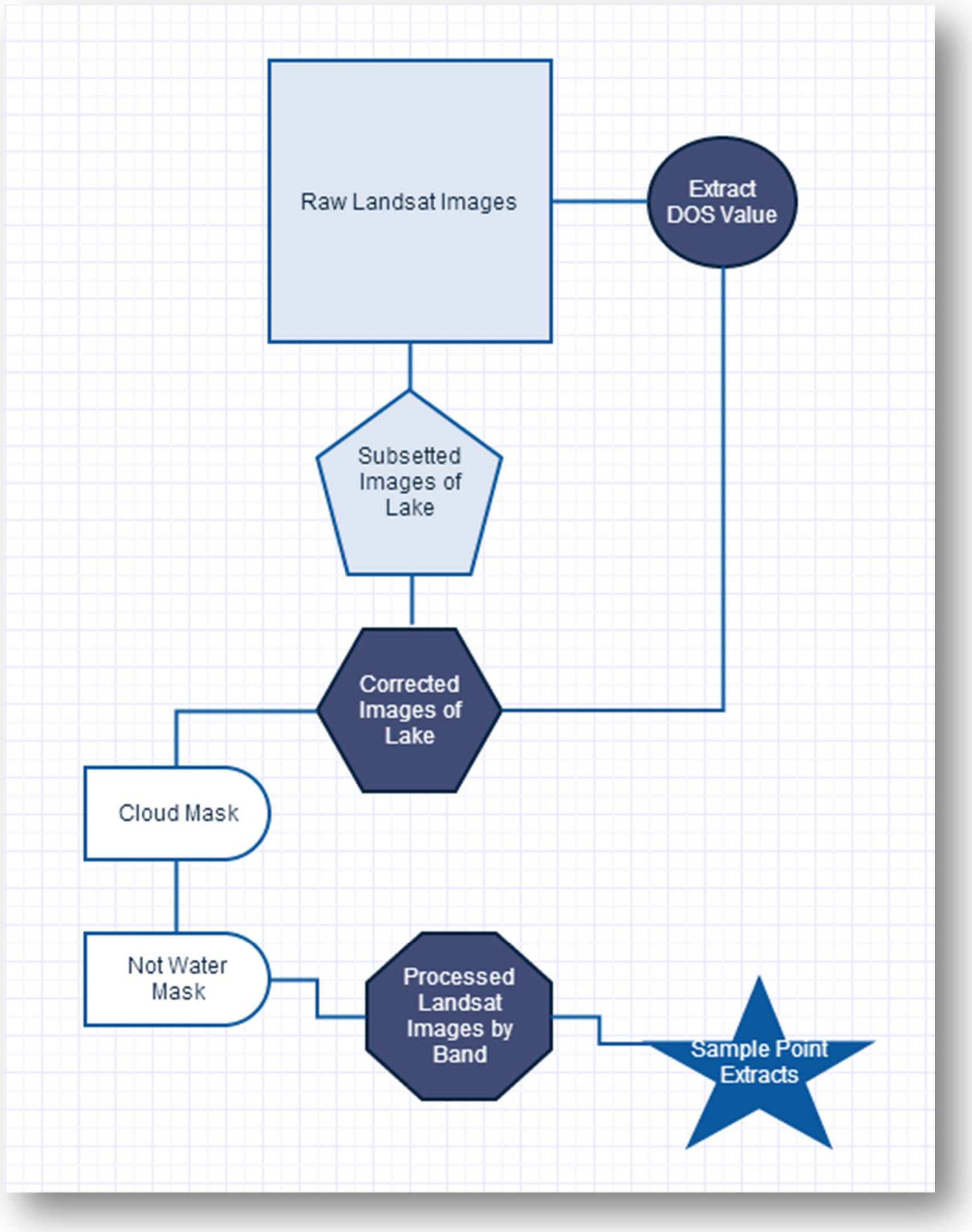

Figure 6: Methodology flowchart.

Image Analysis: Subsetting Images

The majority of the methodology comprises the selection and reduction of Landsat images into a form comparable with the BUMP water quality data. The first step is determining the Landsat images within a one day temporal window of a BUMP sample events. Images that were collected within one day of a BUMP sample event and containing less than 100 percent cloud cover are downloaded from the USGS Global Visualization Viewer. The size of each uncompressed Landsat scene is approximately 3 gb for the entire 31,110 km^2 scene in all bands. Each lake is only a fraction of that image, so to save space and processing time, images are reduced (i.e., spatially subsetted) as soon as possible. There is one measurement required in the radiometric correction portion of the methodology (discussed below) from image before subsetting; the DN_{min}. DN_{min} is the lowest pixel value (or Digital Number) which has a count of at least 100 from the entire image, representing the actual value of 'black' in the scene for each band. This value is assessed for each scene once and saved, meaning that if any lakes are sourced from the same Landsat scene they would all have the same DN_{min} value.

Once DN_{min} is determined for the scene, each band's image is subset to just the lake. Data files for the lake outline are available from the OWRB (OWRB 2014a), but lake levels fluctuate and complex vectors are computationally expensive inputs for clipping. Instead the lake outlines are reprocessed to rectangles bounding the outer extent of the lake's shoreline (Figure 7) to expedite the image subsetting process.

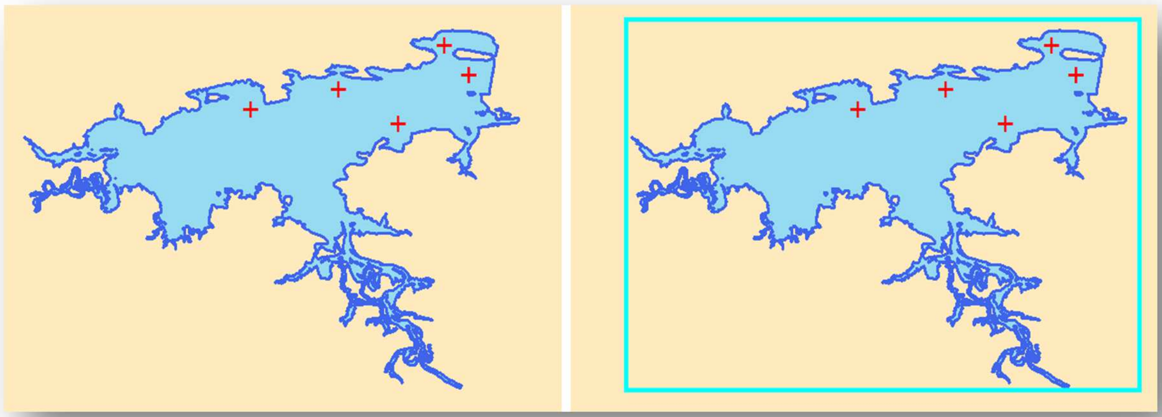

Figure 7: Subsetting illustration for Sardis Lake.

Image Analysis: Correction and Masking

Clipped lake images are smaller and faster to process than entire scenes, but they are still uncorrected radiometrically and contain data reflected off vegetation, lakeshore, cloud, man-made structures like docks, and the water surface. First, the images require atmospheric correction and analysis to mask out the non-water features. Once processed, they are ready for extraction of the corrected pixel values.

Band Corrections

All bands except the Thermal band are corrected using Chavez's (1996) COST-DOS method which requires the DN_{min} value collected before image subsetting. The Thermal band, which is used only to detect clouds, is instead converted to temperature for use in cloud mask building (see Band Corrections below). Both of these processes require Top of Atmosphere reflectance (TOA) as input instead of the DN that Landsat images have been reprocessed and delivered containing.

COST-DOS is a sophisticated method for mitigating the effects of atmosphere recorded by Landsat instruments including scattering, absorbing, and refracting light from the particles in the air such as water vapor and air-borne particulate matter. The inputs for this method include the DN_{min} collected from the whole scene, solar zenith angle, distance between the earth and sun, and the scene's pixel values recalculated to the TOA values originally captured by the sensor. The COST-DOS process includes correction for both components of error with the COST portion addressing multiplicative error by accounting for the variation in sun angle through the year. The DOS portion mitigates additive error through a histogram shift, making the DN_{min} value the reference zero value in the image.

The conversion of the image from DN to the TOA reflectance (Equation 1) is the first step in the correction process for all bands (USGS 2011). The inputs for this equation are the DNs in the image and calibration readings taken at-satellite for each scene and contained in the metadata included with Landsat scene downloads.

$$L_\lambda = M\rho * Qcal + A\rho \tag{1}$$

The pixel values, *Qcal* in DN, are multiplied by the multiplicative scaling factor, *Mρ*, and then the additive rescaling factor, *Aρ*, is summed to give the TOA reflectance L_λ. The result of this first equation converting image data from DN to TOA reflectance is uncorrected in any way, it only returns it to the original units of capture at the sensor. Next, all bands other than thermal undergo COST-DOS, which calculates rescaling factors from the minimum and maximum calibration values and includes DOS correction and sun angle correction in this process. The generic and expanded form of this equation are below (Equations 2-4).

$$L_{haze} = HL_{min} - L_{1\%} \tag{2}$$

Where:

$$HL_{min} = L_{min} + DN_{min} \frac{(L_{max} - L_{min})}{(QCAL_{max} - QCAL_{min})} \tag{3}$$

$$L_{1\%} = (0.01 + ESUN_\lambda * \cos(\emptyset_s \pi / 180)^2) / (d^2\pi) \tag{4}$$

The majority of the necessary data are present in the Landsat metadata, which accompanies each image downloaded from the USGS site. L_{max} and L_{min} are maximum and minimum spectral radiances values scaled to $QCAL_{max}$ and $QCAL_{min}$ by band. The difference between $QCAL_{max}$ and $QCAL_{min}$ has been 254 (255 – 1) since a recalibration in 2004, and this term is often substituted with that value when published. DN_{min} is the lowest pixel value (originally in DN but then converted to TOA reflectance in HL_{min}) with at least a count of 100 in the raw scene. $ESUN_\lambda$ is solar spectral irradiance at TOA which are known values published for each satellite (Chander and Markham 2003; USGS 2013a). The solar zenith angle in degrees, $\emptyset_s$, is calculated from the sun elevation included in the metadata minus 90^0. Finally, *d* is the distance between the Earth and sun, which are known values referenced using the Julian date within the Landsat scene name. The outcome, L_{haze} is the rescaled and corrected DN_{min} value, which is used to correct the rescaled image values during their correction process (Equation 5).

$$P_\lambda = \frac{\pi(L_\lambda - L_{haze})d^2}{ESUN_\lambda \cos(\pi / 180\, \theta_s)} \tag{5}$$

The atmospherically corrected reflectance (P_λ) is the final outcome using the TOA reflectance (L_λ) minus the rescaled and corrected DN_{min} value, corrected the variations in the angle, distance, and brightness of the sun throughout the year. The output of this step in the methodology is an atmospherically corrected version of lake clips for each desired band.

The Thermal band is used to calculate cloud cover mask through a temperature detection algorithm. These data were converted to spectral radiance (Equation 1), and need to be converted to temperature (Equation 6) for the masking process. This conversion also requires calibration constants (USGS 2013a); in this case K_1 and K_2 are thermal conversion constants and the output is in kelvins.

$$T = \frac{K_2}{\ln(\frac{K_1}{L\lambda} + 1)} \tag{6}$$

Not-Cloud Mask

The Landsat data production process includes a percentage counter for the amount of each image covered by cloud. The source pixel-by-pixel map of cloud cover is not made available with the scene. However, the same process used in the Landsat image processing can be used to generate our own cloud cover map to exclude these pixels from data extraction and analysis utilizing the now converted Green, Red, NIR, SWIR1, and Thermal bands.

The official Landsat process was detailed clearly by Irish (2000) and consists of a two pass process using thresholds and indices to check for dense clouds and filter out sand and snow which both exhibit similarities to cloud reflectance. Cirrus clouds are not detected by this method due to their lack of a strong thermal response, and are pervasive enough to be undesirable for exclusion according to Irish (2000).

The first pass is comprised of eight filters, which progressively select areas that are definable as cloud or not cloud. During this process the amount of pixels in the scene that are snow and desert are counted for later use. The remainder of pixels unclassified by this first pass are classified as ambiguous and handed to the second pass for analysis.

The second pass reevaluates any ambiguous pixels, if there were any, on basic statistics of the clouds detected during pass one. If there is an excess of brightly illuminated desert pixels detected in pass one, pass two is not executed. If there is snow present in the scene, the clouds are divided into a warm and cold class with the statistics of each used to evaluate ambiguous pixels, otherwise the entire cloud class is utilized. These cloud statistics set minimum and maximum classification thresholds for the algorithm reevaluating ambiguous pixels to determine if they are cloud or not cloud.

The final result of this Not-Cloud Mask is a value of 0 if Not-Cloud is false, and 1 if Not-Cloud is true. Simply stated, values of 1 are pixels where cloud is not detected and these values are extracted for analysis.

Water Mask

Due to the seasonal fluctuations in lake levels, it is necessary to also exclude any pixels that are unidentifiable as water. The Normalized Difference Water Index (NDWI) was designed to mask out exactly these features for MSS data, and has been redesigned to work with current Landsat bands as the Modified NDWI (Xu 2006) (Equation 7).

$$MNDWI = \frac{Green - SWIR1}{Green + SWIR1} \tag{7}$$

The result of MNDWI is an image that is reclassified into a mask to filter out pixels where water was not detected. Reclassification leaves 0 where Water was not detected and 1 where Water was detected. The mask value of 1 allows those pixel values to be extracted for analysis.

Image analysis is completed with lake specific clips of the raw Landsat image corrected for atmospheric interference, and masks computed to exclude non-water and cloud pixels from automated extraction. The images are now completely processed and ready for pixel value extraction.

Image Analysis: Pixel Extraction

Each processed lake image contains a variable number of BUMP sample sites. The purpose of this step is to extract values from each sample site and present them as a summary containing mask results and individual pixel values for each band. If data are present for all nine pixels within the sample window and the masks permit, the nine pixel values are averaged for each band. This aggregated output of a mean by band for each site is combined with the BUMP sample data for turbidity and chlorophyll for the final step: regression analysis.

The OWRB provides a digital file containing the GPS points for all its BUMP sample sites (OWRB 2014a). Using the sample point GPS point as the center cell, a sample window 3 pixels by 3 pixels is generated for each sample site. These nine pixels are sampled for all corrected bands and both masks at each BUMP sample site. This method averages each BUMP water sample over a 9 pixel, or 900 m^2, area of reflectance from the water surface.

There are a number of reasons for this averaging of a window around the sample point pixel, the first being that samples are taken by boats floating on the lake surface, which makes staying in place difficult, thereby contributing to spatial variation in actual sample location. A second reason is a given sample point could lie a meter away from the boundary of a pixel and be as representative of that neighboring pixel as the one the GPS point lies within. Third, water is heterogeneous and often in flux due to seasonal, solar, and meteorological factors. Finally, the water quality samples are a few cups of water taken to represent reflectance data averaged into a 900 m^2 surface with a single digital reflectance value. By averaging nine pixel reflectance averages as regression values rather than using a single pixel's reflectance, these influences are reduced by increasing the area of lake each sample represents. Kloiber et al. (2002) investigated the result of increasing sizes of region of interests (up to 25 x 25 pixels) finding that more than 3 x 3 pixels was not statistically advantageous.

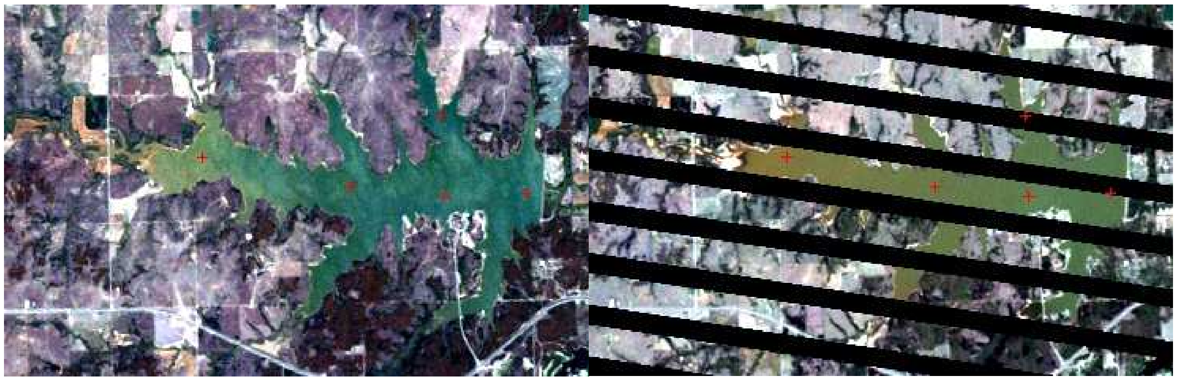
Figure 8: Lake Carl Blackwell without (left) and with Landsat 7 ETM+ scan line correction void zones (right).

It is possible that aside from the masked-out cloud and land features, the LS7 data will contain SLC-off voids (Figure 8) or even be outside the scene but within the file extent. The image on the left is a SLC-on picture of Lake Carl Blackwell, with the red crosses indicating the five BUMP sample points. On the right, these horizontal stripes in the image reduce the number of pixels available for image selection. It is clear that three of the sample points are clear of the void zones, one is within the void, but the 9-pixel window nearest the dam could contain both void and data. A closer look (Figure 9) shows how SLC-off voids interfere completely (a), not at all (b), or occasionally only partially (c) with the 9-pixel extraction window.

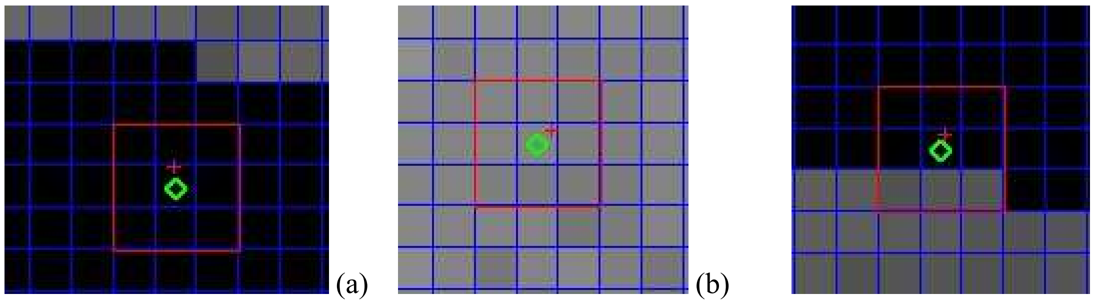
(a) (b) (c)

Figure 9: Landsat 7 ETM+ scan line correction void & extraction window interactions. Nine-pixel sample window shown in red.

The cases where the SLC-off voids partially occlude the sample area (Figure 8c) are not explicitly dealt with in the literature (Allan et al. 2011; Hicks et al. 2013). The simplest way to deal with the partial extractions is to drop them from the analysis, and with sufficient data available this was the path taken. A worthy refinement of these methods for future consideration would be to investigate the proper thresholds for accepting or rejecting partial captures from sample windows. Lakes clipped completely out of the no data portion of Landsat scenes are detected as such and not processed further.

Statistical Analysis

Statistical analysis of extracted pixel values is accomplished using SPSS (IBM 2014) and includes basic statistics, correlations, and regressions. The mean of the extracted pixel values for each band is correlated with the turbidity and chlorophyll water quality parameters. The basic minimum and maximum values will also be useful to indicate any anomalous results, as reflectance values should range between one and zero. The values taken at each sample point within each lake are treated independently, even though the locations sampled are static. The dynamic nature of lakes means that the same geographic point does not consist of the same water from sample event to sample event, especially when the rate of return for each lake is measured in years. The samples within each lake are treated independently since this reflectance relationship is not dependent on location, only the light reflected by each pixel of water.

Basic data explorations of standard histograms, summary statistics, and normality tests are performed. At this point, candidates for transformation are identified since these are environmental data and often exhibit high skew which usually present more normal distributions when subjected to transformation. Transformed and raw pixel means are then tested for correlations, by band, using Pearson's *r* against the water quality parameter values. Band/parameter pairs with significant and meaningful Pearson's values are then tested with linear bivariate regression. All reported correlations and regression results reported are significant at the 0.01 level ($p = 0.99$) with correlations being two-tailed.

CHAPTER V

RESULTS

Over the ten years of BUMP data that include turbidity and chlorophyll available for Oklahoma, there were 130 Landsat scenes taken within the study area. The study area boundary for the Ozark, Ouachita-Appalachian Forest Ecoregion within Oklahoma included twenty lakes sampled by the OWRB. After verification that each sample window contained lake surface reflectance and was complete for all bands by BUMP sample site, the number of paired pixel extracts and BUMP samples was 432 ($n = 432$).

The chlorophyll and turbidity variables both exhibited a logarithmic curve, and transformation by natural log improved the normality of each. Even when transformed, the chlorophyll did not correlate often or well with any Landsat bands or band combinations. The only significant results at the $p = 0.99$ level for Chlorophyll and lnChlorophyll was the band combination of SWIR2/NIR (Pearson's r values of -0.154 and -0.155 respectively). LnChlorophyll also had a weak, positive correlation (0.150) with lnTurbidity.

The correlations between lnTurbidity and single bands was highest for SWIR1 and SWIR2 transformed by natural log (-0.142 and -0.145 respectively). These correlations are improved by band combinations that include Red/SWIR1 (0.421), NIR/SWIR1 (0.409), and Red/SWIR2

(0.384). Linear regression of lnTurbidity proceeded utilizing the significant although moderate correlations. The outcome was a slightly less moderate r^2 value (0.431) for the factors Red/SWIR1, Green/Red, lnSWIR2, and lnChloropyll.

The corrected water reflectance data for all bands was all within the expected range ($0 < x < 1$). While it appeared that they would respond well to a logarithmic transformation, they instead presented a bimodal distribution after natural log transformation (Figure 10).

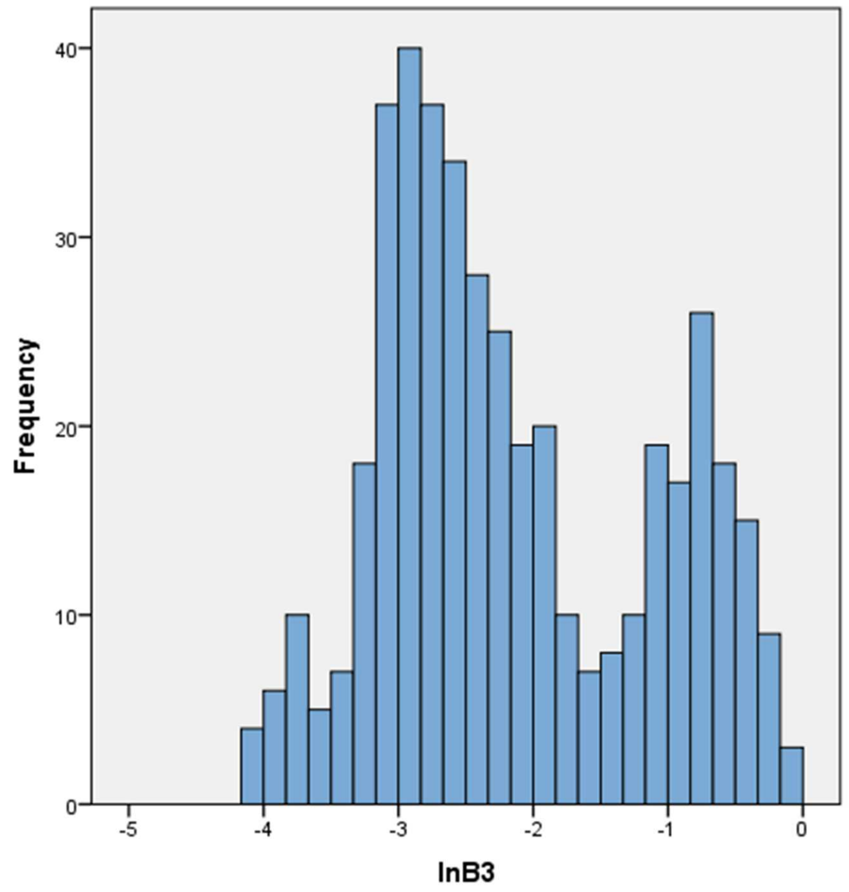

Figure 10: Frequency histogram for the natural log of band 3 (lnB3) for all turbidity

During this study period there were infrequently significant correlations with the transformed band data, and they were always weak. The band ratio data were significant more frequently and were usually stronger as well. It is possible that a more complex transformation schedule, such as log-log, square root, multiplicative inverse, and combinations of half-logarithmic band ratios may produce distributions more similar to the water quality parameters, allowing a more direct relationship between reflectance and water quality to emerge than by using the band ratios alone.

Given the rather weak relationship between water quality samples and Landsat reflectance for the entire study period and the much greater sample size compared to those of Hicks et al. (2013), these results are further analyzed by approximate seasons. The four seasons are rounded to the nearest month rather than calculating based on annual equinoxes and solstices, shifting the start of each season back approximately one week (e.g., January through March designated as winter, April through June as spring, July through September as summer, and October through December as fall).

Winter

The winter season data subset consisted of 102 events (*n* = 102), which was reduced to 74 when correlated pairwise with chlorophyll. The two water quality variables were significantly but weakly positively correlated (0.289) with one another once both were transformed. The number of significant correlations for Chlorophyll exceeded lnChlorophyll in their strength and quantity. LnChlorophyll was only significantly correlated to SWIR1/SWIR2 and lnTurbidity (0.379 and -0.324 respectively). The significant correlations associated with Chlorophyll all included the SWIR2 band: SWIR1/SWIR2 (0.498), Blue/SWIR2 (0.273), and SWIR2/Green (0.238). The moderate strength of the SWIR1/SWIR2 correlation remained significant during regression analysis, contributing the majority of the r^2 value of 0.318 along with Blue/SWIR2. These variables' relationship with lnChlorophyll was unsatisfactory based upon the skew of the residual histogram and clustering of the residual scatterplot.

Strong correlations were more common during winter for turbidity than for chlorophyll. The strongest significant correlations for Turbidity included the mineral detection bands: Green/SWIR2 (0.523) and Green/SWIR1 (0.455). These relationships were stronger with lnTurbidity, which became more normally distributed during transformation. There were many significant correlations with band combinations, the strongest including the mineral bands again: Red/SWIR1 (0.510), NIR/SWIR1 (0.503), and Red/SWIR2 (0.470). In a linear regression, the r^2 value of 0.648 comprised of Red/SWIR1 and Blue/SWIR1. There was some abnormality in the positive end of the residual histogram and a pronounced downward pull on the scatterplot. Since both band combinations utilize SWIR1 as the denominator the Variance Inflation Factor was in the double digits, therefore the least contributing input (Blue/SWIR1) was removed and the regression rerun. This modification lowered the r^2 value to 0.521 at the same time it normalized the residual histogram and scatterplot by reducing the redundant influence of SWIR1. The band combinations for this regression relationship with lnTurbidity were Red/SWIR1, NIR/Blue, and Green/Red.

Spring

The data within the spring classification included 118 sample events ($n = 118$) reduced to a minimum of 94 pairwise. There was no significant correlation between chlorophyll and turbidity for this season. Chlorophyll had no significant correlations in its untransformed state. LnChlorophyll was slightly and negatively correlated with a selection of mineral bands: NIR/SWIR1 (-0.356), Green/SWIR1 (-0.346), Red/SWIR1 (-0.319), Green/ SWIR2 (-0.306), and NIR/SWIR2 (-0.303). In a linear stepwise regression, only NIR/SWIR was selected and residuals were unsatisfactory.

Turbidity correlated with more bands and combinations in spring, of varying significance. LnTurbidity results superseded those in terms of significance and strength except for in the case of single bands and natural log transformations of single bands where a drop of 0.5 was common. The strongest lnTurbidity correlations were not dominated by the mineral bands in spring, though they were present: Green/Red (-0.464), Blue/Red (-0.422), SWIR2/NIR (0.329), and SWIR2/Blue (0.273). Linear regression of these band combinations resulted in a weak r^2 value of 0.303 which included Green/Red, Blue/NIR, and SWIR2/Green.

Summer

Filtering the processed data to only analyze the summer season selected 120 data (*n* = 120) with a minimum of 99 when correlated pairwise. LnChlorophyll and lnTurbidity were moderately positively correlated (0.495) for this season. This was also the only significant correlation for chlorophyll. Turbidity has a selection of significant correlations, but all were improved by transformation.

The band correlations for lnTurbidity included either the Red band or a mineral band (SWIR1 & SWIR2). These ten significant correlations were moderate in strength at best (ranging from 0.277 to 0.496), but together accomplished a greater r^2 value of 0.654 during stepwise regression. The selected variables of influence were lnChlorophyll, and Red/Blue, and SWIR2/NIR which had a normally distributed residual histogram and scatterplot.

Fall

Fall contained a total of 92 data entries (*n* = 92), which was reduced to a minimum of 80 during pairwise analysis. Chlorophyll nearly achieved significance for many minor correlations with single band and band transformations, but fell short at the selected significance level (*p* = 0.99). LnChlorophyll failed to make a significant correlation with any data. The Turbidity correlations were again weaker and less often significant than lnTurbidity relationships.

LnTurbidity during the fall season achieved a significant correlation with all six of the bands, their natural log transformations, and ten independent band ratios varying in strength from -0.292 to 0.461. All of the band ratios included an infrared band: NIR, SWIR1, or SWIR2. In a stepwise regression of the significantly correlated variables, there was overlap with the Green band showing up three times. When controlled for excessive influence of that one band, the r^2 = 0.555 and included the NIR, lnRed, and lnBlue bands. However, the Variance Inflation Factor shot to triple digits due to multicollinearity between the lnRed and lnBlue bands. Scaling back either natural log transformation band further reduced the r^2 below 0.200 and still failed to normalize residuals.

CHAPTER VI

DISCUSSION

The BUMP water quality data were not found to be highly correlated with Landsat reflectance data as was found by Hicks et al. (2013). The strong predictive relationship between the NIR Band and Turbidity as well as the relationship between the ratio of the Blue band and the Red band that Hicks et al. (2013) found for lakes in New Zealand do not appear to hold for the lakes in the Ozark, Ouachita-Appalachian Forest Ecoregion of Oklahoma. The detection of chlorophyll remains elusive. However, this study did find a relationship between Landsat reflectance and turbidity, albeit with a seasonal bias.

There are a number of factors that may have contributed to the Hicks et al. (2013) findings not being directly applicable to Oklahoma. Of primary importance is likely the difference between the two regions in terms of lake morphology and water composition. Additional research on the factors which best define similarity between lakes in order to create a study area could refine the relationships discovered. Both studies utilize Landsat data from the last decade, but Hicks et al. (2013) utilized just six scenes over nine years with thirty-five total sample events from ten lakes, while this research analyzed hundreds of sample events spanning one-hundred Landsat images and twenty lakes. The larger study area used in this study may have introduced additional variability, and the higher sample size rate may have compounded this issue.

That water quality parameter relationships with bands varied seasonally means that further analysis upon the boundaries when strong relationships are found between turbidity and reflectance are necessary. Uneven portions of the year may uncover periods of water stability with stronger relationships than four even seasons did. There is also the risk of decreased temporal range reducing the sample size drastically, and so research on recovery of partial sample window results is also important for continuation of remote-sensing of water quality. Alternately, the acceptable temporal window between a BUMP sample event and a Landsat scene could be extended for turbidity, since physical parameters stabilize during low influx times of the year (Olmanson, Bauer, and Brezonik 2008) and the reduced correlation strength for additional observations is offset by the increase in the number of observations (Kloiber et al. 2002).

Although chlorophyll was not definitively revealed via reflectance, there were some clues that may lead to better estimating this parameter remotely. The frequent correlations with mineral bands (SWIR1 and SWIR2) in winter and spring suggest that there may be a relationship between those spectra during those seasons. A regression relationship between mineral band reflection with algae and plant production may still be possible through further refinement of methodology. It is quite surprising that chlorophyll is not significantly correlated with any tested bands or band combinations during the growing season. However, algae blooms can be temporally brief phenomena which are difficult to detect with a temporal resolution of sixteen days.

Spring turbidity data exhibited a weaker correlation with single band data when transformed, unlike the other seasons, and resulted in a very weak regression result. Investigation into the causes and corrections for this phenomenon could improve the ability to estimate turbidity during the spring season. It could also simply be due to the lake turnover effect, when the equalization of the thermal gradient in the lake induces mixing of surface and bottom waters. The fall turbidity correlations included every single band and natural log transformation of single bands, but no reversal of transformed correlation strength as seen for spring. Investigation into the possible reasons for fall's special relationship with band rather than band ratio data or finding a way to reduce the overlap between the significant bands may yet lead to a relationship for this season as well.

The most significant result of this research was the establishment of moderate regression relationships for turbidity during winter and summer (Table 2), when lake water quality is most stable. The winter regression yielded an r^2 value of only 0.521 and was comprised of band combinations including the Red, SWIR1, and NIR bands. Previous research had indicated possible relationships with Red (Kloiber, Brezonik and Bauer 2002) and NIR (Hicks et al. 2013), but not SWIR1. More significant was the summer regression result of 0.654, which again included Red, SWIR1, and NIR in band combinations as well as lnChlorophyll. The significant correlation between chlorophyll and turbidity found for summer may mean that this relationship is less stable than the winter regression result, as biologic entities are more dynamic than other particles suspended in the water. While this is the strongest regression result, it falls a bit short of being a robust estimator for lake turbidity values. Additional refinements in methodology which build on this study may increase this relationship to enable researchers with the OWRB to check turbidity levels in all lakes in eastern Oklahoma on a sixteen-day cycle without mandatory field work.

Table 2: Linear regression results for the natural log of turbidity (lnTurbidity)

Season	**n**	**Equation**	r^2	***p***
Summer	96	lnTurbidity = -2.59 + .457 x lnChlorophyll + 6.34 x Red/Green – 4.46 x SWIR2/NIR	0.654	<0.001
Winter	70	lnTurbidity = 6.7 + 0.62 x Red/NIR - .882 x NIR/Blue – 3.22 x Green/Blue	0.521	<0.001

Finally, the development of scripted image processing tools using Python and ESRI's arcpy was successful in processing a large volume of image data quickly. The design of these tools around the OWRB's water quality database and lake shapefiles allows rapid reapplication of this methodology across Oklahoma. With minor changes, they could also be applied to other water quality databases and even integrate LS8 once it has been confirmed compatible with current Landsat data. In the development of these scripts, code was reviewed and portions reused which had been published by prior researchers **(**Gómez-Dans 2013; Kochaver 2014)**.** The insight these other scripts provided expedited the development of these tools, and hopefully their public release helps future researchers.

The preference going into this project was that proprietary programs would be avoided, allowing the application of the scripts by anyone with internet access, not just those with an ArcMAP license. Due to the time constraints of a thesis and familiarity with the arcpy module, the project was completed using ArcMAP. Future work on these script tools will be focused on replacing arcpy functions with free tools.

Conclusions

In conclusion, this thesis set out to apply the methodology of Hicks et al. (2013) that was developed for lakes in New Zealand to the Ozark, Ouachita-Appalachian Forest Ecoregion of Oklahoma. The first objective was to test that methodology for two critical water quality parameters, turbidity and chlorophyll, and if successful translate the regression relationship into a predictive equation for each parameter. The ability to estimate the state of Oklahoma's waters with remote-sensing data would greatly increase the OWRB's response time to emerging water quality issues, cost very little compared to field campaigns, and hopefully lead to improved water quality for all Beneficial Uses and users.

There was no relationship discovered for chlorophyll, but moderate to fair regression relationships were discovered for turbidity during winter and summer. This finding is important not only as it can be used to crudely estimate turbidity in eastern Oklahoma during half of the year, but serves as the foundation for increased understanding of the relationship between Landsat reflectance and turbidity for Ozark, Ouachita-Appalachian Forest Ecoregion and potentially all of Oklahoma.

The second objective of this study was to develop an automated method for the image processing workflow. These Landsat image processing scripts are another important by-product of this research. The scripts will be released to the public along with guidelines for their use. The availability of these tools accelerates research on remotely-sensed water quality estimation, and enables agencies tasked with monitoring water quality, such as the OWRB, to investigate remotely-sensed water quality observation.

Future work stemming from this research includes better transformations of the corrected LS band data, parameters which define similar water quality bodies, and whether specific time periods are more effective at estimating turbidity. The frequent correlations between the mineral bands and chlorophyll during winter and spring also merit further analysis, as there may be an undiscovered relationship. Finally, there are minor modifications to the script parameters which may increase the ability to estimate water quality from Landsat reflectance which include increasing the acceptable time between image and water sample collection and recovery of partial sample window results. It is hoped that this research and the release of the scripts themselves encourages interest in the application of Landsat imagery to monitoring water quality.

REFERENCES

Allan, M. G., D. P. Hamilton, B. J. Hicks, & L. Brabyn. 2011. Landsat remote sensing of chlorophyll a concentrations in central North Island lakes of New Zealand. *International Journal of Remote Sensing* 32:2037-2055.

Bukata, R. P. 1995. *Optical properties and remote sensing of inland and coastal waters*. Boca Raton, FL: CRC Press.

Busteed, P. R., D. E. Storm, M. J. White, & S. H. Stoodley. 2009. Using SWAT to Target Critical Source Sediment and Phosphorus Areas in the Wister Lake Basin, USA. *American Journal of Environmental Sciences* 5:156-163.

Carpenter, S. M., & D. J. Carpenter. 1983. Modeling inland water quality using Landsat data. *Remote Sensing of Environment* 13:345-352.

Chander, G., & B. Markham. 2003. Revised Landsat-5 TM radiometric calibration procedures and postcalibration dynamic ranges. *Geoscience and Remote Sensing, IEEE Transactions on* 41:2674-2677.

Chavez, P. S., Jr. 1996. Image-based atmospheric corrections--revisited and improved. *Photogrammetric Engineering & Remote Sensing* 62:1025.

Chen, J., & W. Quan. 2012. Using Landsat/TM imagery to estimate nitrogen and phosphorus concentration in Taihu Lake, China. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of* 5:273-280.

Curran, P. J., & E. M. M. Novo. 1988. The Relationship Between Suspended Sediment Concentration and Remotely Sensed Spectral Radiance: A Review. *Journal of Coastal Research* 4:351-368.

Dekker, A. G., R. J. Vos, & S. W. M. Peters. 2002. Analytical algorithms for lake water TSM estimation for retrospective analyses of TM and SPOT sensor data. *International Journal of Remote Sensing* 23:15-35.

Environmental Protection Agency (EPA). 2010. "TMDL Development for Lakes Eucha and Spavinaw in Oklahoma". www.epa.gov/waters/tmdldocs/2Eucha_Spavinaw_TMDL_TP_Apr_10 (last accessed 20 February 2015).

———. 2014a. "2014 Oklahoma 303(d) List of Impaired Waters (DRAFT)". http://www.deq.state.ok.us/wqdnew/305b_303d/2014DraftReport/2014 Appendix C - 303d Draft.pdf (last accessed 22 February 2015).

———. 2014b. "Summary of the Clean Water Act". http://www2.epa.gov/laws-regulations/summary-clean-water-act (last accessed 1 November 2014).

Environmental Systems Research Institute (ESRI). ArcGIS Desktop, Version 10.2.2. Environmental Systems Research Institute, Redlands, CA.

Goldman, C. R., R. C. Richards, H. W. Paerl, R. C. Wrigley, V. R. Oberbeck, & W. L. Quaide. 1974. Limnological studies and remote sensing of the Upper Truckee River sediment plume in Lake Tahoe, California-Nevada. *Remote Sensing of Environment* 3:49-67.

Guneroglu, A., F. Karsli, & M. Dihkan. 2013. Automatic detection of coastal plumes using Landsat TM/ETM+ images. *International Journal of Remote Sensing* 34:4702-4714.

Hadjimitsis, D. G. 1999. The application of atmospheric correction algorithms in the satellite remote sensing of reservoirs. University of Surrey, United Kingdom.

Hadjimitsis, D. G., & C. Clayton. 2009. Assessment of temporal variations of water quality in inland water bodies using atmospheric corrected satellite remotely sensed image data. *Environmental Monitoring and Assessment* 159:281-292.

Hadjimitsis, D. G., C. R. I. Clayton, & A. Retalis. 2009. The use of selected pseudo-invariant targets for the application of atmospheric correction in multi-temporal studies using satellite remotely sensed imagery. *International Journal of Applied Earth Observation and Geoinformation* 11:192-200.

Haggard, B., P. Moore, & P. DeLaune. 2005. Phosphorus flux from bottom sediments in Lake Eucha, Oklahoma. *Journal of environmental quality* 34:724-728.

Haggard, B. E., & T. S. Soerens. 2006. Sediment phosphorus release at a small impoundment on the Illinois River, Arkansas and Oklahoma, USA. *Ecological Engineering* 28:280-287.

Han, Z., Y. Q. Jin, & C. X. Yun. 2006. Suspended sediment concentrations in the Yangtze River estuary retrieved from the CMODIS data. *International Journal of Remote Sensing* 27:4329-4336.

Harrington, J. A., F. R. Schiebe, & J. F. Nix. 1992. Remote sensing of Lake Chicot, Arkansas: Monitoring suspended sediments, turbidity, and Secchi depth with Landsat MSS data. *Remote Sensing of Environment* 39:15-27.

Hicks, B., G. Stichbury, L. Brabyn, M. Allan, & S. Ashraf. 2013. Hindcasting water clarity from Landsat satellite images of unmonitored shallow lakes in the Waikato region, New Zealand. *Environmental Monitoring and Assessment* 185:7245-7261.

International Business Machines Corporation (IBM). IBM SPSS Statistics, Version 22. International Business Machines Corporation, Armonk, NY.

Irish, R. R. 2000. Landsat 7 automatic cloud cover assessment. In *AeroSense 2000*, 348-355. International Society for Optics and Photonics.

Jensen, J. R. 2007. *Remote sensing of the environment: an earth resource perspective*. Upper Saddle River, NJ: Pearson Prentice Hall.

Jiazhu, G. J. W. Y. H. 2006. A model for the retrieval of suspended sediment concentrations in Taihu Lake from TM images. *Journal of Geographical Sciences* 16:458-464.

Kallio, K., J. Attila, P. Harma, S. Koponen, J. Pulliainen, U. M. Hyytiainen, & T. Pyhalahti. 2008. Landsat ETM+ images in the estimation of seasonal lake water quality in boreal river basins. *Environmental Management* 42:511-522.

Kloiber, S. M., P. L. Brezonik, & M. E. Bauer. 2002. Application of Landsat imagery to regional-scale assessments of lake clarity. *Water Research* 36:4330-4340.

Kloiber, S. M., P. L. Brezonik, L. G. Olmanson, & M. E. Bauer. 2002. A procedure for regional lake water clarity assessment using Landsat multispectral data. *Remote Sensing of Environment* 82:38-47.

Kunte, P. 2008. Sediment concentration and bed form structures of Gulf of Cambay from remote sensing. *International Journal of Remote Sensing* 29:2169-2182.

Kutser, T., D. C. Pierson, K. Y. Kallio, A. Reinart, & S. Sobek. 2005. Mapping lake CDOM by satellite remote sensing. *Remote Sensing of Environment* 94:535-540.

Lepistö, A., T. Huttula, S. Koponen, K. Kallio, A. Lindfors, M. Tarvainen, & J. Sarvala. 2010. Monitoring of spatial water quality in lakes by remote sensing and transect measurements. *Aquatic Ecosystem Health & Management* 13:176-184.

McCullough, I. M., C. S. Loftin, & S. A. Sader. 2012. Combining lake and watershed characteristics with Landsat TM data for remote estimation of regional lake clarity. *Remote Sensing of Environment* 123:109-115.

Nas, B., S. Ekercin, H. Karabork, A. Berktay, & D. J. Mulla. 2010. An Application of Landsat-5TM Image Data for Water Quality Mapping in Lake Beysehir, Turkey. *Water Air and Soil Pollution* 212:183-197.

Oklahoma Geologic Survey (OGS). 2008. "Geologic History of Oklahoma". http://www.ogs.ou.edu/pubsscanned/EP9_2-8geol.pdf (last accessed 2 October 2014).

Olmanson, L. G., M. E. Bauer, & P. L. Brezonik. 2008. A 20-year Landsat water clarity census of Minnesota's 10,000 lakes. *Remote Sensing of Environment* 112:4086-4097.

Onderka, M., & M. Rodný. 2010. Can suspended sediment concentrations be estimated from multispectral imagery using only image-derived information? *Journal of the Indian Society of Remote Sensing* 38:85-97.

Oklahoma Water Resources Board (OWRB). 2012. "2012 Oklahoma Lakes Report". http://www.owrb.ok.gov/quality/monitoring/bump/pdf_bump/archives/2012_LakesBUMPReport.pdf (last accessed 4 October 2014).

———. 2014a. "Data & Maps: Surface Water". http://www.owrb.ok.gov/maps/pmg/owrbdata_SW.html (last accessed 22 April 2014).

———. 2014b. "Status of Water Quality Monitoring in Oklahoma: Surface Water Monitoring Strategy Document". http://www.owrb.ok.gov/quality/monitoring/monitoring_pdf/2014WaterQualityMonitoringStrategy.pdf (last accessed 14 February 2015).

Python Software Foundation (PSF). Python, Version 2.7. Python Language Reference, Delaware, USA.

Ritchie, J. C., C. M. Cooper, & F. R. Schiebe. 1990. The relationship of MSS and TM digital data with suspended sediments, chlorophyll, and temperature in Moon Lake, Mississippi. *Remote Sensing of Environment* 33:137-148.

Ritchie, J. C., C. M. Cooper, & J. Yongqing. 1987. Using landsat multispectral scanner data to estimate suspended sediments in Moon Lake, Mississippi. *Remote Sensing of Environment* 23:65-81.

Ritchie, J. C., F. R. Schiebe, & C. M. Cooper. 1989. Landsat digital data for estimating suspended sediment in inland water. *International Association of Hydrological Sciences Publ*:151-158.

Schalles, J. F., A. A. Gitelson, Y. Z. Yacobi, & A. E. Kroenke. 1998. Estimation of chlorophyll a from time series measurements of high spectral resolution reflectance in an eutrophic lake. *Journal of Phycology* 34:383-390.

Schalles, J. F., D. Rundquist, & F. Schiebe. 2002. The influence of suspended clays on phytoplankton reflectance signatures and the remote estimation of chlorophyll. In *Verh. Int. Ver. Theor. Angew. Limnol./Proc. Int. Assoc. Theor. Appl. Limnol./Trav. Assoc. Int. Limnol. Theor. Appl.*, 3619-3625.

Schiebe, F., J. Harrington, & J. Ritchie. 1992. Remote sensing of suspended sediments: the Lake Chicot, Arkansas project. *International Journal of Remote Sensing* 13:1487-1509.

Song, K., L. Li, S. Li, L. Tedesco, B. Hall, & L. Li. 2012. Hyperspectral remote sensing of total phosphorus (TP) in three central Indiana water supply reservoirs. *Water, Air, & Soil Pollution* 223:1481-1502.

United States Geologic Survey (USGS). 2011. "Landsat 7 Science Data Users Handbook". http://landsathandbook.gsfc.nasa.gov/data_prod/prog_sect11_3.html (last accessed May 15 2015).

United States Geologic Survey (USGS). 2013a. "ACCA Parameters". http://landsathandbook.gsfc.nasa.gov/cpf/prog_sect9_2.html#section9.2.4 (last accessed 15 March 2015).

———. 2013b. "Landsat 4 History". http://landsat.usgs.gov/about_landsat4.php (last accessed 29 September 2014).

———. 2013c. "Landsat Project Description". http://landsat.usgs.gov/about_project_descriptions.php (last accessed 29 September 2014).

Verdin, J. P. 1985. Monitoring water quality conditions in a large western reservoir with Landsat imagery. *Photogrammetric Engineering and Remote Sensing* 51:343-353.

Xu, H. 2006. Modification of normalised difference water index (NDWI) to enhance open water features in remotely sensed imagery. *International Journal of Remote Sensing* 27:3025-3033.

Zhou, W., S. Wang, Y. Zhou, & A. Troy. 2006. Mapping the concentrations of total suspended matter in Lake Tailm, China, using Landsat-5 TM data. *International Journal of Remote Sensing* 27:1177-1191.

APPENDICES

Below are the source code for the Python scripts developed and utilized during this research project. These include code developed by other researchers but are implemented specifically for the OWRB's database format. The settings variables in the header of each script are customizable by the user to facilitate application to other data. There are three main scripts: Read Images and Clip, Image Correction, and BUMP and Processed Data Assembly. The first, Read Images and Clip reads a master list of lake and scene names, selects scenes from the master list for processing, reads the metadata for that image, and then clips each Landsat scene down to just the lake. Next these subset images are processed to convert them back to TOA reflectance using the metadata parameters, corrected using COST-DOS, masked for water and cloud detection, and then sampled for the nine pixels surrounding and including the BUMP sample point. Finally, the results of the Landsat image correction and sampling script is matched back up with the BUMP water quality data file for statistical analysis.

Read Images and Clip

```
try:

    import arcpy

except ImportError:

    print "You need arcpy installed"

    sys.exit ( -1 )

from arcpy import env

env.overwriteOutput = True


import fileinput

import os

from os import listdir
from os.path import isfile, join
import csv
try:
    import numpy as np
except ImportError:
    print "You need numpy installed"
    sys.exit ( -1 )
import inspect
import logging
import pickle

# Set environment settings
env.workspace = "F:/py"
working_dir = os.getcwd()

settings = {
'Image Dump': '',
'Image List': 'Master List.csv',
'Output Dir': 'Processed',
'Clip Dir': 'Clips',
'Scene Dir': 'image dump',
# reading master list
'Code Header': 'Code',
'Scene Header': 'Scene Name',
'Pick Bands': ['1', '2', '3', '4', '5', '6', '7'],
'Parameter List': ['dnmin', 'gain', 'bias', 'lmax', 'lmin', 'qc_lmax', 'qc_lmin'],
# metadata store
'MD file':'metadata.txt'
}

# code adapted from J Gomez-Dans
def process_metadata ( fname , scene):
    """A function to extract the relelvant metadata from the
    USGS control file. Returns dicionaries with LMAX, LMIN,
    QCAL_LMIN and QCAL_LMAX for each of the bands of interest."""
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))

    # band vars
    lmax = {} # Dicts to store constants
    lmin = {}
    qc_lmax = {}
    qc_lmin = {}
    gain = {}
    bias = {}
    p_list = ['lmax', 'lmin', 'qc_lmax', 'qc_lmin', 'gain', 'bias']
    parameters = [lmax, lmin, qc_lmax, qc_lmin, gain, bias]

    # scene vars
    sun_elevation = 0
    julian_date = 0
    sat_id = ''

    # lets just go ahead and assemble these as a dict for return
    result = {}

    with open(fname, 'r') as fp: # Open metadata file
        for line in fp:
          if ( line.find ("RADIANCE_MULT_BAND") >= 0 ):
            s = line.split("=") # Split by equal sign
            the_band = s[0].strip()[-1] # Band number as string
            # use this as dict name
            band_name = 'B' + s[0].split("_")[3].strip() + '.TIF'
            logger.debug('test {} in {}: {}'.format(the_band, settings['Pick Bands'], the_band in
settings['Pick Bands']))
            if the_band in settings['Pick Bands']: # Is this one of the bands we want?
                gain[band_name] = float ( s[-1] ) # Get constant as float
                logger.debug('gain[{}]: {}'.format(band_name, gain[band_name]))
          elif ( line.find ("RADIANCE_ADD_BAND") >= 0 ):
            s = line.split("=") # Split by equal sign
            # test this one
            the_band = s[0].strip()[-1] # Band number as string
            # use this as dict name
            band_name = 'B' + s[0].split("_")[3].strip()
            if the_band in settings['Pick Bands']: # Is this one of the bands we want?
                bias[band_name] = float ( s[-1] ) # Get constant as float
          elif ( line.find ("QUANTIZE_CAL_MAX_BAND") >= 0 ):
            s = line.split("=") # Split by equal sign
            # test this one
            the_band = s[0].strip()[-1] # Band number as string
            # use this as dict name
            band_name = 'B' + s[0].split("_")[4].strip()
            if the_band in settings['Pick Bands']: # Is this one of the bands we want?
                qc_lmax[band_name] = float ( s[-1] ) # Get constant as float
          elif ( line.find ("QUANTIZE_CAL_MIN_BAND") >= 0 ):
            s = line.split("=") # Split by equal sign
            # test this one
            the_band = s[0].strip()[-1] # Band number as string
            # use this as dict name
            band_name = 'B' + s[0].split("_")[4].strip()
            if the_band in settings['Pick Bands']: # Is this one of the bands we want?
                qc_lmin[band_name] = float ( s[-1] ) # Get constant as float
          elif ( line.find ("RADIANCE_MAXIMUM_BAND") >= 0 ):
```

```
            s = line.split("=") # Split by equal sign
            # test this one
            the_band = s[0].strip()[-1] # Band number as string
            # use this as dict name
            band_name = 'B' + s[0].split("_")[3].strip()
            if the_band in settings['Pick Bands']: # Is this one of the bands we want?
                lmax[band_name] = float ( s[-1] ) # Get constant as float
        elif ( line.find ("RADIANCE_MINIMUM_BAND") >= 0 ):
            s = line.split("=") # Split by equal sign
            # test this one
            the_band = s[0].strip()[-1] # Band number as string
            # use this as dict name
            band_name = 'B' + s[0].split("_")[3].strip()
            if the_band in settings['Pick Bands']: # Is this one of the bands we want?
                lmin[band_name] = float ( s[-1] ) # Get constant as float
        elif ( line.find('SUN_ELEVATION') >= 0):
            s = line.split('=')
            sun_elevation = s[1].strip()
            log.debug('sun_elevation: {}'.format(sun_elevation))

    # other is scene wide metadata
    sat_id = scene[2]
    julian_date = scene[13:16]
    log.debug('sat id: {}, julian_date: {}'.format(sat_id, julian_date))
    other = [sat_id, julian_date, sun_elevation]

    # make named dict for result
    for (x, i) in zip(p_list, parameters):
            result[x] = i
    log.debug('result: {}'.format(result))
    return [result, other]

# code adapted from J Gomez-Dans
def get_metadata ( fname ):
    """This function takes `fname`, a filename (opionally with a path), and
    and works out the associated metadata file"""
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
    original_fname = os.path.basename ( fname )
    metadata_fname = original_fname.split("_")[0] + "_MTL.txt"
    metadata_fname = os.path.join ( os.path.dirname ( fname ), metadata_fname )

    return metadata_fname

def discover_files(loc):
    """
```

```
Search curdir for a csv or txt in the Raw Input dir.
This is the list of locations and dates which are used to select images from
the other directory full of LS metadata "Target List"
Lets use a working dir, and then a file name "Test list.xls". load the xlreader.
Read check for clip, read for dnmin, then clip and save to new dir.
"""
log = logger.getChild(inspect.currentframe().f_code.co_name)
log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
result = []
# Load all Input_Types files in curdir to a list
file_list = [ f for f in listdir(loc) if isfile(join(loc,f)) \
    and os.path.splitext(f)[1] in settings['Input_Types']]

log.debug('found anything?: {}'.format(file_list))

# See if it's a CSV or TXT file
for file in file_list:
    extention =  os.path.splitext(file)[1]
    log.debug('checking extention: {}'.format(extention))
    if extention.lower() == '.csv':
        log.debug('found a csv file: {}'.format(file))
        result.append(join(loc,file))
    elif extention.lower() == '.txt':
            log.debug('found a txt file: {}'.format(file))

            result.append(file)

        else:

            log.debug('no csv or txt file was found')

    if verbose: print ('discover result: ', result)

    if result:

        log.debug('found an input')

    return result


def list_find(list, search_term):

    """

    Find if something is found in a list of lists, and return that column's data


    Returns ['R, C', 'List of stuff -header']

    """

    log = logger.getChild(inspect.currentframe().f_code.co_name)

    log.debug('Initializing {}'.format(inspect.currentframe().f_code.co_name))

    logger.debug('for {} in {}'.format(search_term, list))

    result = []


    # see if term is in the list

    term_row = [idx for idx in list if search_term in idx]

    log.debug('term row: {}'.format(term_row))
    # possible outcomes are 0 = not found, 1 = expected to find one time, >1 = problem...
    # this is a list comp, so result is a list. check length!
    if len(term_row) == 1:
        # get the index of the row term was found in
        term_row_i = [i for i, x in enumerate(list) if search_term in x][0]
        log.debug('term row_i: {}'.format(term_row_i))
        # get the column index it was found in
        term_col_i = [x for x, i in enumerate(term_row[0]) if i == search_term][0]  # returns i as list,
so [0]
        log.debug('term col_i: {}'.format(term_col_i))
        # with start row and col found, find last col and return a selection
        # only get from header row +1 to last row
        # can deal with an extra, blank line result on return
        selection = [x[term_col_i] for idx, x in enumerate(list) if idx in range(term_row_i +1,
len(list))] # but this leaves blank on the end of names
        log.debug('list_find selection dump: {}'.format(selection)) # should gaurd against blanks
above header row
        result = [[term_row_i, term_col_i], selection]
    else:
        log.warning(u'Failed to find {} exactly once in list'.format(search_term))
        result = [0, 0]
    log.debug('list_find result: {}'.format(result))
    return result
```

```
def main():
    pass


if __name__ == '__main__':
    main()


    # Setup Logger
    logger = logging.getLogger(os.path.basename(__file__))
    logger.setLevel(logging.DEBUG)


    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
    format2 = logging.Formatter('%(levelname)s - %(message)s')


    # Logging to conlsole/standard output
    ch = logging.StreamHandler()
    ch.setLevel(logging.DEBUG)
    ch.setFormatter(formatter)
    logger.addHandler(ch)


    # read master list
    # first is it present?
    input_file = settings['Image List']
    input_list = []
```

```
clip_list = []
if os.path.isfile(input_file):
    with open(input_file, 'r') as csvfile:
        logger.debug('opened: {}'.format(input_file))
        reader = csv.reader(csvfile)
        for row in reader:
            input_list.append(row)
            logger.debug('read: {}'.format(row))

# check for all scenes in image dump
    # get list of clip codes
    clip_list = list_find(input_list, settings['Code Header'])[1]
    logger.debug('code len & list: {} {}'.format(len(clip_list), clip_list))
    # reduce codes to just first half (scene code)
    clip_list = [i[:2] for i in clip_list]
    clip_list = set(clip_list)
    logger.debug('uniq code & scene list: {} {}'.format(len(clip_list), clip_list))

    # check each is present in clips
    clip_dir = os.path.join(os.getcwd(), settings['Clip Dir'])
    logger.debug('clip_dir: {}'.format(clip_dir))

    # get just the prefix, only for SHP files
```

```
clips_in_dir = [i[:2] for i in listdir(clip_dir) if i.partition('.')[2] == 'shp']
logger.debug('clips_in_dir 1 len & list: {} {}'.format(len(clips_in_dir), clips_in_dir))

# see if all of clip_list can be found in clips_in_dir list
clips_missed = []
for i in clip_list:
    if i in clips_in_dir:
        logger.debug('clip {} found already clipped'.format(i))
    else:
        logger.warning('clip {} NOT found already clipped'.format(i))
        clips_missed.append(i)

if clips_missed:
    logger.warning('clips NOT all found already clipped: {}'.format(clips_missed))
else:
    logger.debug('clips all found already clipped')

# get list of scenes
scene_list = list_find(input_list, settings['Scene Header'])[1]
logger.debug('scene len & list: {} {}'.format(len(scene_list), scene_list))
scene_list = set(scene_list)
logger.debug('uniq len & scene list: {} {}'.format(len(scene_list), scene_list))
```

```
# Start with each scene in the input_list

scene_dir = os.path.join(os.getcwd(), settings['Scene Dir'])

logger.debug('scene_dir: {}'.format(scene_dir))

for x in scene_list:

   logger.debug('processing scene: {}'.format(x))

   parameter_dict = {}

# read metadata to store in dict

   metadata_file = get_metadata(os.path.join(scene_dir, x))

   logger.debug('metadata name: {}'.format(metadata_file))

   if os.path.isfile(metadata_file):

      parameter_dict , scene_vars = process_metadata(metadata_file, x)

      logger.debug('metadata sneak peak: {}'.format(parameter_dict))

      logger.debug('scene metadata: {}'.format(scene_vars))


      # Get all the scenes

      scenes = [i for i in listdir(scene_dir) if x in i and i.partition('.')[2] == 'TIF' ]

      logger.debug('scenes selected from scene dir 1: {}'.format(scenes))


      # drop bands not on in the Pick List

      scenes = [i for i in scenes if i.partition('_')[2][1] in settings['Pick Bands']]

      logger.debug('reduced scenes selected from scene dir : {}'.format(scenes))


      # collect each scene's DNmin
```

```
dnmin = {}
for i in scenes:
    scene = os.path.join(settings['Scene Dir'], i)
    logger.debug('scene path: {}'.format(scene))
    img = arcpy.Raster(scene)
    # take of extention to match what the metadata keys look like
    this_band = i.partition('_')[2].split('.')[0]

    # check if RAT present or needs built
    test = img.hasRAT
    if test:
        logger.debug('{} has RAT'.format(i))
    else:
        logger.debug('{} has no RAT'.format(i))
        t = arcpy.BuildRasterAttributeTable_management(scene, "NONE")
        if t:
            logger.debug('{} had RAT built'.format(i))
        else:
            logger.debug('{} failed to build RAT'.format(i))

    if test or t:
        # these should be acending from lowest value
        with arcpy.da.SearchCursor(scene, ["VALUE", "COUNT"]) as rows:
                for row in rows:
                    d = 0
                    val = row[0]
                    count = row[1]
                    # find the lowest value with a count of at least 100
                    if int(val) > 0 and int(count) > 100:
                        d = int(val)
                        dnmin[this_band] = d
                        logger.debug('{} is DNmin'.format(d))
                        break
                    else:
                        logger.debug('{} is not DNmin'.format(val))
            logger.debug('DNmin: {}'.format(dnmin))
            # add dnmin to parameter dict
            parameter_dict['dnmin'] = dnmin
            logger.debug('parameters as dict: {}'.format(parameter_dict))

            # store it by scene name
            Correction_Parameters[x] = parameter_dict
            Correction_Parameters[x]['sat_id'] = scene_vars[0]
            Correction_Parameters[x]['julian_date'] = scene_vars[1]
            Correction_Parameters[x]['sun_elevation'] = scene_vars[2]
            logger.debug('assembed correction parameters for {}: {}'.format(x,
Correction_Parameters))
```

```
# Get all the lakes that need clipped out of this image
process_list = [i for i in input_list if x in i]
# Make sure you have something:
if process_list:
    logger.debug('lakes to be clipped: {}'.format(process_list))
    # Process this list
    for i in process_list:
        # Set clip variables
        output_dir = os.path.join(settings['Output Dir'], i[0])
        rectangle = '#'
        clip_feature = clip_dir + '/' + i[0][:2] + '_box.shp'
        nodata_value = '0'
        clipping_geometry = 'ClippingGeometry'
        maintain_clipping_extent = 'NO_MAINTAIN_EXTENT'

        # make sure the output dir exists, if not, make
        out = os.path.join(working_dir, settings['Output Dir'], i[0])
        if not os.path.exists(out):
            os.makedirs(out)

        # Clip and output all bands of the scene
        for j in scenes:
```

```
                    in_raster = os.path.join(scene_dir, j)

                    out_raster = os.path.join(working_dir, output_dir, 'c_'+j)

                    # Execute Clip

                    #Clip_management in_raster rectangle out_raster {in_template_dataset}
{nodata_value}  {NONE | ClippingGeometry}

                    clip = arcpy.Clip_management(in_raster, rectangle, out_raster, \

                        clip_feature, nodata_value, clipping_geometry, maintain_clipping_extent)

                    # reclass 0 to be NoData

                    if clip:

                        logger.debug('clipped scene {} to {}'.format(j, output_dir))

                    else:

                        logger.warning('failed to clip: {}'.format(j))

                else:

                    logger.warning('process_list couldnt find: {}'.format(x))


            else:

                logger.critical('metadata not found!: {}'.format(metadata_file))


        # write Correction Parameters dictionary to Output's dir

        logger.debug('writing corr parameters to {}'.format(settings['MD file']))

        with open(os.path.join(settings['Output Dir'], settings['MD file']), 'wb') as handle:

            pickle.dump(Correction_Parameters, handle)


    else:
        logger.warning(u'Missing the input file: {}'.format(settings['Image List']))


    logger.debug('finished!')
```

Image Correction

```
try:

    import arcpy

except ImportError:

    print "You need arcpy installed"

    sys.exit ( -1 )

from arcpy import env

env.overwriteOutput = True

from arcpy.sa import *


#Check out the Spatial Analyst extension

try:

    arcpy.CheckOutExtension("spatial")

except ImportError:

    print "You need access to Spatial Analyst"

    sys.exit ( -1 )


import os

from os import listdir

from os.path import isfile, join, isdir

import csv
import numpy as np

import math

import inspect

import logging

import pickle

from itertools import islice


working_dir = os.getcwd()


settings = {

'Image Dump': '',

'Image List': 'Master List.csv',

'Output Dir': 'Processed',

'Clip Dir': 'Clips',

'Scene Dir': 'image dump',

'Cloud Dir': 'cloud mask',

'Extr Dir': 'Extracts', #Extract dump

'Agg File': 'agg.dbf',   # Extract file name

'Agg Output': 'Aggregated.csv',

'SD File': 'd.csv',     # Solar distance file from NASA

# reading master list

'Code Header': 'Code',

'Scene Header': 'Scene Name',
'Pick Bands': ['1', '2', '3', '4', '5', '6', '7'],
# metadata store
'MD file':'metadata.txt',
'Clip Prefix': 'c_',
'Rad Prefix': 'r',  # Converted to TOA radiance
'Corr Prefix': 'c', # COST DOS corrected
'Temp Prefix': 't', # For B6 once it's temperature
'NDWI Prefix': 'water_', # prefix for water mask
'Blue Band': 'B1',
'Green Band': 'B2',
'Red Band': 'B3',
'NIR Band': 'B4',
'SWIR Band': 'B5',
'Thermal Band': 'B6',
}


sun_d = {}


Do_COSTDOS = 1
# switch for Testing or Production
Testing = 0
if Testing:
    log_location = 'log.txt'
    output_level = logging.DEBUG
else:
    log_location = 'log.txt'
    output_level = logging.INFO



#////////////////////////// Subfunctions //////////////////////////#


# code adapted from Steve Kochaver
def calc_radiance (LMAX, LMIN, QCALMAX, QCALMIN, QCAL, outfolder):
    """
    Calculate the TOA radiance from metadata on each band.
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))


    log.debug('with {}/{} {}/{} {}/{}'.format(LMAX, LMIN, QCALMAX, QCALMIN, QCAL,
outfolder))
    LMAX = float(LMAX)
    LMIN = float(LMIN)
    QCALMAX = float(QCALMAX)
    QCALMIN = float(QCALMIN)
    offset = (LMAX - LMIN)/(QCALMAX-QCALMIN)
    input_ras = Raster(QCAL)
    outname = os.path.join(outfolder, (settings['Rad Prefix'] + QCAL.split('\\')[-1]))
    log.debug('output name: {}'.format(outname))


    out_raster = (offset * (input_ras - QCALMIN)) + LMIN
    log.debug('saving output as: {}'.format(outname))
    out_raster.save(outname)


    return outname


# code adapted from Steve Kochaver
def calc_reflectance(solarDist, ESUN, solarElevation, radianceRaster, Lhaze, outfolder):
    """
    COSTDOS correction in the Lhaze parameter. Toggle Global var Do_Costdos to use 0.0.
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
    log.debug('with: {} {} {} {} {} {}'.format(solarDist, ESUN, solarElevation, radianceRaster,
Lhaze, outfolder))


    outname = os.path.join(outfolder, (settings['Corr Prefix'] + radianceRaster.split('\\')[-1]))
    #Value for solar zenith is 90 degrees minus solar elevation (angle from horizon to the center of
the sun)
    #http://landsathandbook.gsfc.nasa.gov/data_properties/prog_sect6_3.html
    solarZenith = (90.0 - float(solarElevation)) * (math.pi / 180)    #Converted from degrees to
radians
```

```
    #solarZenith = math.pow(((90.0 - float(solarElevation))* (math.pi / 180)), 2)

    solarDist = float(solarDist)

    ESUN = float(ESUN)

    radiance = Raster(radianceRaster)


    #outraster = (math.pi * radiance * math.pow(solarDist, 2)) / (ESUN * math.cos(solarZenith)) *
scaleFactor

    outraster = (math.pi * (radiance - Lhaze) * math.pow(solarDist, 2)) / (ESUN *
math.cos(solarZenith))

    outraster.save(outname)

    log.debug('saved: {}'.format(outname))


    return outname


# code adapted from Steve Kochaver

def get_ESUN(bandNum, SIType):

    """

    """

    log = logger.getChild(inspect.currentframe().f_code.co_name)

    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))


    SIType = SIType

    ESUN = {}

    #from NASA's Landsat7 User Handbook Table 11.3
http://landsathandbook.gsfc.nasa.gov/pdfs/Landsat7_Handbook.pdf
```

```
    #ETM+ Solar Spectral Irradiances(generated using the Thuillier solar spectrum)
    #if SIType == 'ETM+ Thuillier':
    if SIType == '7':
        ESUN = {'B1':1997,'B2':1812,'B3':1533,'B4':1039,'B5':230.8,'B7':84.90,'B8':1362}

    #from NASA's Landsat7 User Handbook Table 11.3 http://landsathandbook.gsfc.nasa.gov/data_prod/prog_sect11_3.html
    #ETM+ Solar Spectral Irradiances (generated using the combined Chance-Kurucz Solar Spectrum within MODTRAN 5)
    if SIType == 'ETM+ ChKur':
        ESUN = {'b1':1970,'b2':1842,'b3':1547,'b4':1044,'b5':225.7,'b7':82.06,'b8':1369}

    #from NASA's Landsat7 User Handbook Table 9.1 http://landsathandbook.gsfc.nasa.gov/pdfs/Landsat7_Handbook.pdf
    #from the LPS ACCA algorith to correct for cloud cover
    if SIType == 'LPS ACAA Algorithm':
        ESUN = {'b1':1969,'b2':1840,'b3':1551,'b4':1044,'b5':225.7,'b7':82.06,'b8':1368}

    #from Revised Landsat-5 TM Radiometric Calibration Procedures and Postcalibration
    #Dynamic Ranges Gyanesh Chander and Brian Markham. Nov 2003. Table II. http://landsathandbook.gsfc.nasa.gov/pdfs/L5TMLUTIEEE2003.pdf
    #Landsat 4 ChKur
    #if SIType == 'Landsat 5 ChKur':
    if SIType == '5':
        ESUN = {'B1':1957,'B2':1825,'B3':1557,'B4':1033,'B5':214.9,'B7':80.72}
    #from Revised Landsat-5 TM Radiometric Calibration Procedures and Postcalibration
    #Dynamic Ranges Gyanesh Chander and Brian Markham. Nov 2003. Table II.
http://landsathandbook.gsfc.nasa.gov/pdfs/L5TMLUTIEEE2003.pdf
    #Landsat 4 ChKur
    if SIType == 'Landsat 4 ChKur':
        ESUN = {'b1':1957,'b2':1826,'b3':1554,'b4':1036,'b5':215,'b7':80.67}

    bandNum = str(bandNum)

    return ESUN[bandNum]

def b6_to_temp(sat_id, radiance_ras):
    """
    Converts B6 as radiance to temperature in Kelvins
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
    log.debug('with {}'.format(radiance_ras))
    result = []

    name = radiance_ras.rsplit('\\', 1)
    desc=arcpy.Describe(radiance_ras)
    print 'no data =', desc.noDataValue, type(desc.noDataValue)
```

```
input_ras = Raster(radiance_ras, noDataValue=desc.noDataValue)
outname = os.path.join(name[0], (settings['Temp Prefix'] + name[1]))
log.debug('output name: {}'.format(outname))
#from http://landsathandbook.gsfc.nasa.gov/data_prod/prog_sect11_3.html 11.5
calibration_constants = {
'7': [666.09, 1282.71],
'5': [607.76, 1260.56]
}

# from http://landsathandbook.gsfc.nasa.gov/data_prod/prog_sect11_3.html
#T = (k2)/(ln(k1/L +1))
#Where:
#T    =  Effective at-satellite temperature in Kelvin
#K2   =  Calibration constant 2 from Table 11.5
#K1   =  Calibration constant 1 from Table 11.5
#L    =  Spectral radiance in watts/(meter squared * ster * ????m)

d1 = (calibration_constants[sat_id][0] / input_ras) +1
out_raster = calibration_constants[sat_id][1] / Ln(d1)

log.debug('saving output as: {}'.format(outname))
out_raster.save(outname)
```

```
    return outname


def utm_window(args):
    """
    Accepts pair of UTM coords (list).
    Return the surrounding 8 coords in a 3x3 pixel window  plus the given "center"
    As list of list objects
    """
    center = args
    result = []

    for (x, y) in center:
        result.append([x -30, y -30])
        result.append([x -30, y -0])
        result.append([x -30, y +30])
        result.append([x -0, y -30])
        result.append([x -0, y -0])
        result.append([x -0, y +30])
        result.append([x +30, y -30])
        result.append([x +30, y -0])
        result.append([x +30, y +30])
    return result


def make_pointfile(points, out_name):
    """
    Accept a list of points and a name.
    Make a shapefile containing those points named name, return the name.
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
    log.debug('with {}'.format(points))
    result =[]

    pt = arcpy.Point()
    ptGeoms = []
    for p in points:
        pt.X = p[0]
        pt.Y = p[1]
        pg = arcpy.PointGeometry(pt)
        ptGeoms.append(pg)
    final_name = out_name + ".shp"
    print final_name
    arcpy.CopyFeatures_management(ptGeoms, final_name)
    del ptGeoms
    return final_name



def ndwi_mask(folder_path, B2, B5):
    """
    Accepts the path and two required bands as inputs (2 & 5)
    Creates ndwi_Imagename.TIF in same dir
    Returns status code
    Water = 1 (True), otherwise false (0)
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
    log.debug('in {} with {} and {}'.format(folder_path, B2, B5))
    result = 0
    output_name = os.path.join(folder_path, settings['NDWI Prefix'] + B2.rsplit('_', 1)[0] + '.TIF')

    Green = os.path.join(folder_path, B2)
    NIR = os.path.join(folder_path, B5)

    #Create Numerator and Denominator rasters as variables and NDVI output (note that
arcpy.sa.Float returns a floating point raster)
    numerator = arcpy.sa.Float(Raster(Green) - Raster(NIR))
    denominator = arcpy.sa.Float(Raster(Green) + Raster(NIR))
    log.debug("Dividing")
    NDWI_eq = arcpy.sa.Divide(numerator, denominator)
    # save intermidiate before reclass
    NDWI_eq.save(os.path.join(folder_path, 'raw_ndwi.TIF'))

    # Reclassify to get a mask
    result = Reclassify(NDWI_eq, "Value", RemapRange([[-2, 0, 0], [0.01, 2, 1]]))

    #Saving output to result output you specified above
    try:
        result.save(output_name)
        log.debug("NDWI Successful: {}".format(output_name))
        result = output_name
    except:
        log.debug("NDWI Unsuccessful: {}".format(result))
        result = 0

    return result

def get_raster_count(args):
    """
    Subfunction to deal with ensuring there's a RAT and then getting the cell
    count out of the table using a cursor.
```

```
Accepts raster.

Returns integer value of total cell count or -1.

"""

log = logger.getChild(inspect.currentframe().f_code.co_name)

result = -1


x = args

# see if it's all null before the bother

null_result = arcpy.GetRasterProperties_management(x, 'ALLNODATA')

# make result useful

null = bool(int(null_result.getOutput(0)))

if null:

   result = 0

else:

# bother

   if x.hasRAT:

      logger.debug('{} has RAT'.format(x))

   else:

      try:

         arcpy.BuildRasterAttributeTable_management(x, "OVERWRITE")

      except:

         logger.debug('{} failed to build RAT'.format(x))
```

```
        # now RAT is built
        total_count = 0
        with arcpy.da.SearchCursor(os.path.join(x.path, x.name), ["VALUE", "COUNT"]) as rows:
            for row in rows:
                print row
                # nodata isn't in the RAT, so no need to filter it out
                # but we do skip the 0's
                if row[0] > 0:
                    total_count += row[1]
        result = float(total_count)

    log.debug('{} result = {}'.format(x.name, result))
    return result


def not_cloud_mask(*args):
    """
    0 = cloud, 1 = not cloud, 2 = ambig
    Return path of mask?
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
    log.debug('with {}'.format(args))
    result = -1
```

```
folder_path = args[0]
images = [os.path.join(folder_path, i) for i in args[1]]
#print images
B2 = Raster(images[0])
B3 = Raster(images[1])
B4 = Raster(images[2])
B5 = Raster(images[3])
B6 = Raster(images[4])

# save final mask in folder_path dir
# all the other stuff can be in the cloud_mask dir
working_path = os.path.join(folder_path, settings['Cloud Dir'])
# make sure the output dir exists, if not, make
out = working_path
if not os.path.exists(out):
    os.makedirs(out)

#total_count = 0
nodata_value = arcpy.Describe(B2).noDataValue
output_name = os.path.join(folder_path, 'not_cloud.TIF')
agg_name = os.path.join(working_path, 'agg_mask.TIF')
name_0 = os.path.join(working_path, 'cloud.TIF')
```

```
name_1 = os.path.join(working_path, 'not_cloud.TIF')
name_2 = os.path.join(working_path, 'ambig.TIF')

sql_exp = """{} > {}""".format(arcpy.AddFieldDelimiters(B2, 'VALUE'), nodata_value)
log.debug('test sql statement: {}'.format(sql_exp))

# test_raster just to get in count of data pixels
test_raster = Test(B2, sql_exp)
test_raster.save(os.path.join(working_path, 'test.TIF'))
total_count = get_raster_count(test_raster)

# set test_image to 0 to seed masks
test_raster = Con(test_raster > 0, 0)

# number of raster cells
snow_count = 0
hot_count = 0
cold_count = 0
desert_count = 0

# filter 1: B3 < .08 = 1/not cloud
f1 = Con(B3 < .08, 1, 0)
f1.save(os.path.join(working_path, 'f1.TIF'))
```

```
# add f1 to an aggregate mask
# add to aggregate (binary mask, assigned a category or not)
agg = BooleanOr(f1, test_raster)
agg.save(os.path.join(working_path, 'agg1.TIF'))

# running sums
sum_0 = 0
sum_1 = get_raster_count(agg)
sum_2 = 0
log.debug('f1- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count, \
    sum_1 / total_count, sum_2 / total_count))

# filter 2: NSDI (B2 - B5)/(B2 + B5). >.7 = 1/not cloud (get count of snow?)
NSDI = ((B2 - B5)/(B2 + B5))
sf2 = Con(NSDI > 0.7, 1, 0)
sf2.save(os.path.join(working_path, 'sf2.TIF'))

# just f2 additions
f2 = Diff(sf2, agg)
f2.save(os.path.join(working_path, 'f2.TIF'))

# get snow count
get_raster_count(f2)
```

```
# use aggregate + selection
agg = BooleanOr(agg, sf2)
agg.save(os.path.join(working_path, 'agg2.TIF'))
# running sums
sum_0 = 0
sum_1 = get_raster_count(agg)
sum_2 = 0
log.debug('f2- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count, \
   sum_1 / total_count, sum_2 / total_count))

# filter 3: B6 > 300K = 1
sf3 = Con(B6 > 300, 1, 0)
sf3.save(os.path.join(working_path, 'sf3.TIF'))
f3 = Diff(sf3, agg)
f3.save(os.path.join(working_path, 'f3.TIF'))
agg = BooleanOr(agg, sf3)
agg.save(os.path.join(working_path, 'agg3.TIF'))

# copy not-cloud aggregate to (name_1) since switching to detect Ambig
mask_1 = agg
# running sums
sum_0 = 0
```

```
sum_1 = get_raster_count(mask_1)
sum_2 = 0
log.debug('f3- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count, \
    sum_1 / total_count, sum_2 / total_count))

# filter 4: B5/6 Composite (1 - B5) * B6 > 225 = 2
f4 = ((1 - B5) * B6)
f4.save(os.path.join(working_path, 'rf4.TIF'))
sf4 = Con(((1 - B5) * B6) > 225, 1, 0)
sf4.save(os.path.join(working_path, 'sf4.TIF'))

f4 = Diff(sf4, agg)
f4.save(os.path.join(working_path, 'f4.TIF'))

# add to aggregate
agg = BooleanOr(agg, sf4)
agg.save(os.path.join(working_path, 'agg4.TIF'))
# start to cloud name_1 (ambig)
mask_2 = f4
mask_2.save(os.path.join(working_path, 'f4_a.TIF'))
# running sums
sum_0 = 0
sum_1 = get_raster_count(mask_1)
```

```
    sum_2 = get_raster_count(mask_2)

    print sum_1, sum_2, total_count, sum_2/total_count

    log.debug('f4- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count,  \

        sum_1 / total_count, sum_2 / total_count))


    # filter 5: B3/4 Ratio (B4/B3) > 2.0 = 2

    sf5 = Con((B4/B3) > 2.0, 1, 0)

    sf5.save(os.path.join(working_path, 'sf5.TIF'))

    f5 = Diff(sf5, agg)

    f5.save(os.path.join(working_path, 'f5.TIF'))

    # add to aggregate (binary mask, assigned a category or not)

    agg = BooleanOr(agg, f5)

    agg.save(os.path.join(working_path, 'agg5.TIF'))

    # add to cloud name_1 (ambig)

    mask_2 = BooleanOr(mask_2, f5)

    mask_2.save(os.path.join(working_path, 'f5_a.TIF'))

    # running sums

    print 'mask checks:', type(mask_1), mask_1.path, mask_1.name, type(mask_2), mask_2.path,
mask_2.name

    sum_0 = 0

    sum_1 = get_raster_count(mask_1)

    sum_2 = get_raster_count(mask_2)

    log.debug('f5- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count,  \

        sum_1 / total_count, sum_2 / total_count))
```

```
# filter 6: B4/2 Ratio (B4/B2) > 2.0 = 2

sf6 = Con((B4/B2) > 2.0, 1, 0)

sf6.save(os.path.join(working_path, 'sf6.TIF'))

f6 = Diff(sf6, agg)

f6.save(os.path.join(working_path, 'f6.TIF'))


# add to aggregate

agg = BooleanOr(agg, f6)

# add to cloud name_1

mask_2 = BooleanOr(f6, mask_2)

mask_2.save(os.path.join(working_path, 'f6_a.TIF'))

# running sums

sum_0 = 0

sum_1 = get_raster_count(mask_1)

sum_2 = get_raster_count(mask_2)

log.debug('f6- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count,  \

   sum_1 / total_count, sum_2 / total_count))


# filter 7: B4/5 Ratio (B4/B5) > 1.0 = 2 (get count of desert pixels)

sf7 = Con((B4/B5) > 1.0, 1, 0)

sf7.save(os.path.join(working_path, 'sf7.TIF'))

f7 = Diff(sf6, agg)
```

```
f7.save(os.path.join(working_path, 'f7.TIF'))
desert_count = get_raster_count(f7)
log.debug('desert_count: {}'.format(desert_count))

# add to aggregate (binary mask, assigned a category or not)
agg = BooleanOr(f7, agg)
# add to cloud name_1
mask_2 = BooleanOr(f7, mask_2)
mask_2.save(os.path.join(working_path, 'f7_a.TIF'))
# running sums
sum_0 = 0
sum_1 = get_raster_count(mask_1)
sum_2 = get_raster_count(mask_2)
log.debug('f7- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count,  \
   sum_1 / total_count, sum_2 / total_count))

# filter 8: Everything remaining = 0 (hot and cold clouds)
sf8 = BooleanNot(agg)
sf8.save(os.path.join(working_path, 'sf8.TIF'))
f8 = Diff(sf8, agg)
f8.save(os.path.join(working_path, 'f8.TIF'))

mask_0 = sf8
```

```
# running sums
sum_0 = get_raster_count(mask_0)
sum_1 = get_raster_count(mask_1)
sum_2 = get_raster_count(mask_2)
log.debug('f8- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count,  \
    sum_1 / total_count, sum_2 / total_count))

#finalize masks
log.debug('saving masks')
agg.save(agg_name)
mask_0.save(name_0)
mask_1.save(name_1)
mask_2.save(name_2)

# pass 2: only fires if there's clouds and all the 3 checks fail
# if mask_1 is all 0: cloudfree, else try checks
c_mask_0 = SetNull(mask_0, 1, "VALUE = 0")
null_result = arcpy.GetRasterProperties_management(mask_0, 'ALLNODATA')
null = bool(int(null_result.getOutput(0)))
log.debug('{} all no data check: {}'.format(f8.name, null))
if null:
# no clouds = no pass 2
    log.debug('no clouds, no pass 2')
    result = 1
else:
    check = 0   # if check == 3, do pass 2

    # if desert_index > 0.5: increment check
    if desert_count == 0:
        desert_index = 0
    else:
        #print desert_count, type(desert_count), total_count, type(total_count)
        desert_index = desert_count / total_count
    if desert_count > 0.5:
        check += 1
        log.debug('di did trip'.format(desert_index))
    else:
        log.debug('di did not trip'.format(desert_index))

    # if cold clouds are < .4%, skip pass 2
    if cold_count == 0:
        cold_index = 0
    else:
        # use B6 since it's the truest count of final image/filter
        cold_index = cold_count / get_raster_count(B6)
    if cold_index  < .004:
            log.debug('ci count did not trip'.format(cold_index))

        else:

            check += 1

            log.debug('ci did trip'.format(cold_index))


        # if mean temp of cold class < 295K

        #cold_mean = GetRasterProperties_management (in_raster, {property_type}, {band_index})
/ cold_count

        cold_mean_result = arcpy.GetRasterProperties_management(ExtractByMask(B6, f8),
'MEAN')

    # maybe extract B6 using Mask_0, and then get average from this.

        cold_mean = float(cold_mean_result.getOutput(0))

    # but just to test, really have to decide which.

        if cold_mean < 295:

            check += 1

            log.debug('cm count did trip'.format(cold_mean))

        else:

            log.debug('cm did not trip'.format(cold_mean))


        if check == 3:

            log.debug('need pass 2')

            # setup some vars and things


            # split cloud into warm and cold for p2
# B5/6 Composite (1 - B5) * B6 > 210 = warm, rest are cold
warm_cloud = Con((f8 * ((1 - B5) * B6)) > 210, 1, 0)
warm_cloud.save(os.path.join(working_path, 'warm_cloud.TIF'))
warm_count = get_raster_count(hot_cloud)
cold_cloud = Con((f8 * ((1 - B5) * B6)) <= 210, 1, 0)
cold_cloud.save(os.path.join(working_path, 'cold_cloud.TIF'))
cold_count = get_raster_count(cold_cloud)

#selection of which cloud groups to process
if snow_count:
    snow_pct = snow_count / total_count
    log.debug('snow_pct: {}'.format(snow_pct))
    # just assign the selection to a var
    if snow_pct < .01:
        # snow-free, use hot_cloud and cold_cloud AKA mask_0
        log.debug('snow free, use mask_0')
        test_clouds = mask_0
    else:
        # use cold cloud only only
        log.debug('snow found, use cold and move warm to ambig')
        test_clouds = cold_cloud
        # reassign warm_cloud as ambig/mask_2
        mask_2 = BooleanOr(mask_2, warm_cloud)
        mask_2.save(os.path.join(working_path, 'ambigP2.TIF'))
else:
    log.warning('no snow_count: {}'.format(snow_count))

# with selection, run through threshold evaluation
raster_stats = CalculateStatistics_management(test_clouds, "MINIMUM")
print 'cloud min', raster_stats.getOutput(0)

test_arr = np.array(test_clouds)
min = np.amax(test_arr)
max = np.amin(test_arr)
std_dev = np.std(test_arr)
mean = np.mean(test_arr)
n = 0
for i in test_arr:
    n += (i - mean)^3
skew = n / std_dev
skew = sp.stats.skew(test_arr)
log.debug('min {}. max {}. mean {}. std dev {}. skew {}'.format( \
    min, max, std_dev, skew))

max_threshold = (max - min) * .9875 + min
high_threshold = (max - min) * .975 + min
```

```
low_threshold = (max - min) * .825+ min

if skew > 0:
    # no adjustment to thresholds
    log.debug('no shift')
else:
    # adjust thresholds up
    shift_factor = skew * std_dev
    log.debug('shift factor: {}'.format(shift_factor))
    low_threshold = low_threshold * shift_factor
    high_threshold = high_threshold * shift_factor

    # back high down if too high
    if high_threshold > max_threshold:
        high_threshold = max_threshold
        log.debug('high_threshold reassigned: {}'.format(high_threshold))
    else:
        log.debug('high_threshold passed: {}'.format(high_threshold))

# Evaluate "termal effects"
g1 = Con(test_clouds < high_threshold, 1, 0)
g1 = Con(test_clouds < low_threshold, 0, g1)
g2 = Con(g1 < low_threshold, 1, 0)
```

```
# compute stats for upper

g1_count = get_raster_count(g1)

g12 = np.count_nonzero(~np.isnan(np.array(g1)))

print 'count test: {} vs {}'.format(g1_count, g12)

g1_pct = g1_count / total_count

g1_mean = np.mean(g1)


if g1_pct > .40 or g1_mean > 295:

    # then g1 are classified non-cloud

    mask_1 = BooleanOr(mask_1, g1)

    log.debug('rejecting g1')

    # continue to evaluate g2

    # compute stats for lower

    g2_count = get_raster_count(g2)

    g2_pct = g2_count / total_count

    g2_mean = np.mean(g2)


    if g2_pct > .40 or g2_mean > 295:

        # then all ambig are scrapped

        mask_1 = BooleanOr(mask_1, mask_2)

        log.debug('rejecting all ambig')

    else:
```

```
                    # accept the lower group into cloud
                    mask_0 = BooleanOr(mask_0, g2)
                    log.debug('uniting the lower with cloud')
            else:
                # unite all 3 cloud classes
                log.debug('uniting the 3 cloud classes')
                mask_0 = BooleanOr(mask_0, mask_2)

            # CHEQUES
            mask_0.save(os.path.join(working_path, 'cloudP2.TIF'))
        else:
            log.debug('at least one check was breeched, no Pass 2.')
            mask_0 = BooleanOr(test_raster, B6)
        # save final masks
        mask_0.save(os.path.join(working_path, 'cloudP2.TIF'))
        mask_1.save(os.path.join(working_path, 'not_cloudP2.TIF'))
        result = 2
    # and then cloud holes filled in?
        mask_1.save(output_name)
    log.debug('not cloud mask named {}'.format(output_name))
    return output_name

def select_values(*args):
```

```
"""
Get values from points around sample sites, if masks say its ok.
Return as a table in Proccessed dir?
"""
log = logger.getChild(inspect.currentframe().f_code.co_name)
log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
folder_path = args[0]
#need joined to path corrected_scenes = args[1]
corrected_scenes = [os.path.join(folder_path, i) for i in sorted(args[1])]
# now that htey are pathed, append not_cloud mask in
corrected_scenes.append(args[2])
log.debug('with {} ~ {}'.format(folder_path, corrected_scenes))
result = []

extract_dir = os.path.join(working_dir, settings['Extr Dir'], os.path.basename(folder_path))
log.debug('extract dir {}'.format(extract_dir))

# Get the sample point shapefile out of the dir corresponding with first 2 of this folder
clip_dir = os.path.basename(folder_path)[:2]
point_file = os.path.join(working_dir, settings['Clip Dir'], (clip_dir +'_pnts.shp'))
log.debug('selecting sample points from dir: {}'.format(point_file))
# have SR ready for features made
spatial_reference = arcpy.Describe(point_file).spatialReference
```

```
fields = ['SITE_ID', 'X', 'Y']
# make sure there's things
count = int(arcpy.GetCount_management(point_file).getOutput(0))
log.debug('len of feature {}: {}'.format(point_file, count))
if count:
    # list of files to aggregate after cursor
    extract_list = []

    # make output dir
    out = extract_dir
    if not os.path.exists(out):
        os.makedirs(out)

    # use sample on the XX_pnt file to get cell address
    xy_table = (os.path.join(extract_dir, 'xy.dbf'))
    Sample(corrected_scenes[0], point_file, xy_table)
    # join the SITE_ID from original file to xy_table
    arcpy.JoinField_management (xy_table, clip_dir +'_pnts', point_file, 'FID', 'SITE_ID')

    # iterate by site_id
    with arcpy.da.SearchCursor(xy_table, fields) as cursor:
        for row in cursor:
```

```
print row
log.debug('starting {}, {}/{}'.format(row[0], row[1], row[2]))
# validate shortens, we just need dash out of name
clean_siteid = row[0].replace('-', '_').strip()
# get points, make into window coords
points = []
points.append([row[1], row[2]])
window_coords = utm_window(points)
log.debug('window coords: {}'.format(window_coords))

# final extract table name
output_name = os.path.join(extract_dir, clean_siteid.partition('_')[2] +'.dbf')
extract_list.append(output_name)
name = os.path.join(extract_dir, clean_siteid)
log.debug('extract name: {}, pointfile name: {}'.format(output_name, name))
# pointfile of window points
pf_name = make_pointfile(window_coords, name)
# apply SR from source file
arcpy.DefineProjection_management(pf_name, spatial_reference)
# set the id field to the site_id

# Set local variables
inFeatures = pf_name
```

```
        fieldName1 = "SID"
        fieldType = 'TEXT'
        fieldLength = 20

        # Sample
        Sample(corrected_scenes, pf_name, output_name)
        logger.debug('sampled {}'.format(output_name))
        # read sampled values
        arcpy.AddField_management(output_name, fieldName1, fieldType, \
            '#', '#', fieldLength)
        logger.debug('just added field {}, goign to populate it with {}' \
            .format(fieldName1, clean_siteid))
        with arcpy.da.UpdateCursor(output_name, '*') as cur:
            for row in cur:
                # should be last one since most recently appended
                row[-1] = clean_siteid
                #print row
                cur.updateRow(row)

# aggregated sample file
agg_ext = os.path.join(extract_dir, 'agg.dbf')
# merge data into single file
arcpy.Merge_management(extract_list, agg_ext)
```

```
        logger.debug('merged all of {}'.format(clean_siteid.partition('_')[0]))

    else:
        log.debug('{} was empty'.format(point_file))

    return result

def aggregator(arg):
    """
    Accepts output dir and aggrated file name.
    Reads them all and puts them into a single CSV in the Extracts dir
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
    log.debug('with {}'.format(arg))
    result = []

    extract_dir = os.path.join(os.getcwd(), settings['Extr Dir'])

    # get folders in the Extract dir
    folders_in_dir = [i for i in listdir(extract_dir) if isdir(os.path.join(extract_dir, i))]
    log.debug('folders_in_dir len & (short) list: {} {}'.format(len(folders_in_dir),
folders_in_dir[:3]))
```

```
# initialize output
output = []

# walk dirs and open the file
for f in folders_in_dir:
    logger.debug('iterating with {}'.format(f))
   # if settings['Agg File'] in listdir(os.path.join(extract_dir, f)):
    f_path = os.path.join(extract_dir, f, settings['Agg File'])

# need to pick out some field headings here, and just read those in cursor
    # if first folder, initialize field names
    if output == []:
        field_names = [i.name for i in arcpy.ListFields(f_path)]
        field_names.insert(0, 'folder name')
        log.debug('field names {}'.format( field_names))
        output.append(field_names)
# first read, append header selection
    try:
        with arcpy.da.SearchCursor(f_path, "*") as cursor:
            for row in cursor:
                x = [i for i in row]
                x.insert(0, f)
                output.append(x)
        except:
            log.debug('{} did not have a {}'.format(f, settings['Agg File']))

    # write output into source folder
    with open(os.path.join(extract_dir, settings['Agg Output']), 'wb') as file:
        writer = csv.writer(file)
        writer.writerows(output)
        logging.info('output saved as {}\{}'.format(extract_dir, settings['Agg Output']))

    return result



#/////////////////////////   Correction Funcion  /////////////////////////#

def main():
    pass

# Setup Logger
logger = logging.getLogger(os.path.basename(__file__))
logger.setLevel(logging.DEBUG)

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
format2 = logging.Formatter('%(levelname)s - %(message)s')
```

```
# Logging to file
fh = logging.FileHandler(log_location)
fh.setLevel(logging.DEBUG)
fh.setFormatter(formatter)
logger.addHandler(fh)

# Logging to conlsole/standard output
ch = logging.StreamHandler()
ch.setLevel(output_level)
ch.setFormatter(formatter)
logger.addHandler(ch)

# open the metadata dict
with open(os.path.join(settings['Output Dir'], settings['MD file']), 'rb') as handle:
   Correction_Parameters = pickle.loads(handle.read())

# open the solar distance file, read into a dict
with open(os.path.join(working_dir, settings['SD File']), 'rb') as handle:
   reader = csv.reader(handle)
   for row in islice(reader, 1, None): #skip header row
      sun_d[row[0]] = float(row[1])
```

```
# read the folders to process in the Processed folder
image_dir = os.path.join(os.getcwd(), settings['Output Dir'])
logger.debug('processed_dir is: {}'.format(image_dir))

folders_in_dir = [i for i in listdir(image_dir) if isdir(os.path.join(image_dir, i))]
logger.debug('folders_in_dir len & list: {} {}'.format(len(folders_in_dir), folders_in_dir))

if __name__ == '__main__':
    main()

# cycle thorugh contents of Process folder
for folder in folders_in_dir:
    folder_path = os.path.join(working_dir, settings['Output Dir'], folder)
    logger.debug('folder path: {}'.format(folder_path))
    # get list of scenes with clip prefix and tif ext
    scenes_in_dir = [i for i in listdir(folder_path) if i[:2] == settings['Clip Prefix'] \
        and i.partition('.')[2] == 'TIF']
    logger.debug('scenes_in_dir {} len & list: {} {}'.format(folder, len(scenes_in_dir),
scenes_in_dir))

    scene_key = set([i.split('_')[1] for i in scenes_in_dir])
    logger.debug('scene_key {}'.format(scene_key))

    # load a subset of Corr Para for this scene
    scene_d = dict((i, Correction_Parameters[i]) for i in Correction_Parameters if i in scene_key)
    # having to use scene name sucks, since it's arelady pulled just "what we want"
    scene_d = scene_d.values()[0]
    logger.debug('scene parameters {}'.format(scene_d))

# code modified from ...
    for scene in scenes_in_dir:
        scene_path = os.path.join(folder_path, scene)
        #scene_path = os.path.join(working_dir, settings['Output Dir'], folder, scene)
        band = scene.split('_')[2].split('.')[0]
        logger.debug('reading {}, band = {}'.format(scene_path, band))
        # check if the image contains any data
        radiance_ras = Raster(scene_path)
        # see if the image is all NoData
        #   this returns a result object... part of which is string of result that needs converted
        #   to binary to be useful
        nd_check = arcpy.GetRasterProperties_management (radiance_ras, 'ALLNODATA')
        logger.debug('nd_check = {}'.format(nd_check))
        t = bool(int(nd_check.getOutput(0)))
        print 'int of t', t, type(t), bool(t)

        if t:
            logger.warning('image {} is all NoData, no processing'.format(scene))
        else:
            #radianceRaster = calcRadiance(LMAX, LMIN, QCALMAX, QCALMIN, BANDFILE,
outfolder)
           # (LMAX, LMIN, QCALMAX, QCALMIN, QCAL, outfolder)
            radiance_ras = calc_radiance(scene_d['lmax'][band], scene_d['lmin'][band], \
                scene_d['qc_lmax'][band], scene_d['qc_lmin'][band], scene_path , folder_path)


            # B6 > Temp, everything else goes for Correction
            if band == 'B6':
                test = b6_to_temp(scene_d['sat_id'], radiance_ras)
                if test:
                    logger.debug('{} successfully converted to kelvins'.format(scene_path))
                else:
                    logger.debug('{} failed to convert to kelvins'.format(scene_path))
            else:
                # DNMin is in DN. Has to be converted to Radiance seperatley form scene.
                # COST DOS correction (includes scaling DNMin)
                dnmin_as_radiance = 0.0
                if Do_COSTDOS:
                    offset = (scene_d['lmax'][band] - scene_d['lmin'][band])/(scene_d['qc_lmax'][band] -
scene_d['qc_lmin'][band])
                    HLmin = (scene_d['qc_lmin'][band] + scene_d['dnmin'][band] * offset)
                    logger.debug('HLmin: {}'.format(HLmin))
                    # now correct it

                    solarZenith = (90.0 - float(solarElevation))* (math.pi / 180)    #Converted from
degrees to radians

                    logger.debug('solar zenith in rad: {}'.format(solarZenith))

                    logger.debug('n = {}'.format((0.01 + get_ESUN(band, scene_d['sat_id']) *
math.cos(solarZenith))))

                    logger.debug('d = {}'.format((math.pow(sun_d[scene_d['julian_date'].lstrip('0')], 2) *
math.pi)))

                    L1percent = ((0.01 + get_ESUN(band, scene_d['sat_id']) * math.cos(solarZenith)) / \

                        (math.pow(sun_d[scene_d['julian_date'].lstrip('0')], 2) * math.pi))

                    logger.debug('L1percent: {}'.format(L1percent))

                    dnmin_as_radiance = HLmin - L1percent

                    logger.debug('DNMin of {} converted to radiance
{}'.format(scene_d['dnmin'][band], dnmin_as_radiance))

                else:

                    logger.debug('DNMin is 0 (preset)')

                reflectanceRaster = calc_reflectance(sun_d[scene_d['julian_date'].lstrip('0')], \

                    get_ESUN(band, scene_d['sat_id']), scene_d['sun_elevation'], radiance_ras,
dnmin_as_radiance, folder_path)


    # now from the corrected scenes grab band 2 & 5 (if any)

    corrected_prefix = settings['Corr Prefix'] + settings['Rad Prefix'] + settings['Clip Prefix']

    corrected_scenes = [i for i in listdir(folder_path) if corrected_prefix in i \

        and i.partition('.')[2] == 'TIF']

    logger.debug('corrected scene list 2: {}'.format(corrected_scenes))
```

```
# need to get TRC in there for cloud_mask

if len(corrected_scenes) > 2:
    B2 = [i for i in corrected_scenes if settings['Green Band'] in i][0]
    B5 = [i for i in corrected_scenes if settings['SWIR Band'] in i][0]

    # send to ndwi
    if B2 and B5:
        ndwi = ndwi_mask(folder_path, B2, B5)
    else:
        logger.warning('ndwi failed to process: {} {}'.format(B2, B5))
    B3 = [i for i in corrected_scenes if settings['Red Band'] in i][0]
    B4 = [i for i in corrected_scenes if settings['NIR Band'] in i][0]
    unwanted_thermal = 'VCID_2'
    thermal_prefix = settings['Temp Prefix'] + settings['Rad Prefix'] + settings['Clip Prefix']
    B6 = [i for i in listdir(folder_path) if thermal_prefix in i \
        and i.partition('.')[2] == 'TIF' and unwanted_thermal not in i][0]
    logger.debug('found thermal band: {}'.format(B6))
    corrected_scenes.append(B6)

    # do cloud mask
    cloud = not_cloud_mask(folder_path, [B2, B3, B4, B5, B6])
```

```
        # ndwi gets caught in corrected scenes!
        select_values(folder_path, corrected_scenes, cloud)
        # which searches out the masks from before to exlude from extraction
    else:
        logger.debug('No corrected scenes to process: {}'.format(corrected_scenes))
aggregator(1)

logger.debug(u'finished')
```

BUMP and Processed Data Assembly

```
import os
from os import listdir
from os.path import isfile, join, isdir
import csv
from datetime import date, datetime, timedelta
import logging
import inspect

settings = {
'Image List': 'Master List.csv',
'Processed Dir': 'Output',
'Processed List': 'Aggregated.csv',
'Bump Data': 'Bump Data.csv',
'Exclude Site': 'B',
'date allowance': timedelta(days=1),
'Final Output': 'final.csv'
}

working_dir = os.getcwd()

def read_csv(args):
    """

    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
    log.debug(u'with {}'.format(args))
    result = []

    with open(args, 'rb') as csvfile:
        log.debug(u'Reading {} as csvfile.'.format(args))
        reader = csv.reader(csvfile)
        for row in reader:
            result.append(row)
    return result
```

```
def main():
    pass


if __name__ == '__main__':
    main()


# Setup Logger
logger = logging.getLogger(os.path.basename(__file__))
logger.setLevel(logging.DEBUG)


formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
format2 = logging.Formatter('%(levelname)s - %(message)s')


# Logging to conlsole/standard output
ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG)
ch.setFormatter(formatter)
logger.addHandler(ch)


# Read Master list
master_list = read_csv(os.path.join(working_dir, settings['Image List']))


# Read Processed image data
```

```
aggregated_list = read_csv(os.path.join(working_dir, settings['Processed Dir'], settings['Processed
List']))

result = []

# initialize final output headers

merge_result = [['folder name', 'lake name', 'scene', 'scene date', 'site id', 'B1', 'B2', 'B3', \

    'B4', 'B5', 'B7', 'other result','Corrected Chlorophyl A', 'Turbidity, Field', 'season']]

# initialize mask check values, 0 = do not use

test_value = ['0.0', '0.0']


# Use folder name to select data for each scene out of processed data

folder_list = [i[0] for i in master_list[1:]]

for f in folder_list:

    logger.debug('iterating folder {}'.format(f))

    data_selection = [i for i in aggregated_list if f in i]

    logger.debug('folder data selection: {}'.format(data_selection))

    if data_selection:

        # iterate by scene and evaluate final values (- Bottom readings)

        site_list = set([i[-1] for i in data_selection if settings['Exclude Site'] not in i[-1]])

        logger.debug('site_list: {}'.format(site_list))

        line_seed =[i for i in master_list[1:] if f in i][0]

        for s in site_list:

            logger.debug('processing site {}'.format(s))

            # seed is Folder Name, Lake Name, Scene Name, and Scene Date

            line = []
```

```
line.extend(line_seed)
logger.debug('line initialized: {}'.format(line))
# append site_id converting '_' back to '-'
line.append(s.replace('_', '-'))
site_selection = [i for i in data_selection if s in i]
logger.debug('site_selection: {}/{}'.format(len(site_selection), site_selection))
test_masks = [i[11:13] for i in s]
if test_value in test_masks:
    logger.debug('all masks not ok')
    line.append('X')
    line.append('X')
    line.append('X')
    line.append('X')
    line.append('X')
    line.append('X')
    line.append('reason code')
else:
    # take a mean of the 4 desired bands
    index = [5, 6, 7, 8, 9, 10]
    for j in index:
        num = [i[j] for i in site_selection]
        # magically turn strings into numbers for division
        L = [float(n) for n in num if n]
```

```
                avg = sum(L)/len(L) if L else '-'
                logger.debug('avg: {}'.format(avg))
                line.append(avg)
            line.append('np')
        # Check if we should ship writing due to all bands being NoData
        if all(line[5:11]):
            merge_result.append(line)
        else:
            logger.debug('nd for all bands, not writing to output')
        logger.debug('line: {}'.format(line))
    else:
        logger.debug('{} had no output'.format(f))
logger.debug('master list and aggregated merged result: {}'.format(merge_result))

# open bump metadata
bump_list = read_csv(os.path.join(working_dir, settings['Processed Dir'], settings['Bump Data']))
# for each line in result compare to site & date in bump list, appending sample data
for (x, i) in enumerate(merge_result):
    # skip header
    if x > 0:
        r = i
        logger.debug('seeded r {}'.format(r))
        site = i[4]
```

```
date = i[3]
# dates are in '02/25/2001' format
do = datetime.strptime(date, "%m/%d/%Y").date()
date_check = []
date_check.append(do)
date_check.extend([do - settings['date allowance']])
date_check.extend([do + settings['date allowance']])
logger.debug('date objects for check {}'.format(date_check))
# bump list has date as string, so put these back to string
acceptable_dates = [datetime.strftime(d, "%m/%d/%Y") for d in date_check]
logger.debug('acceptable string dates: {}'.format(acceptable_dates))
logger.debug('check for {} {} in bump'.format(site, date))
selection = [i for i in bump_list if site in i and i[3] in acceptable_dates]
logger.debug('selection of bump: {}'.format(selection)) # try for id + date range
if selection:
    # each data seems to have its own row... assume there are no dupes here
    chlor = ''
    turb = ''
    for j in selection:
        logger.debug('checking {}'.format(j[5:7]))
        if j[5]:
            chlor = j[5]
        if j[6]:
                turb = j[6]
        r.extend([chlor, turb])


        logger.debug('month: {}'.format(do.month))
        if do.month > 9:
            season = 'fall'
        elif do.month > 6:
            season = 'summer'
        elif do.month > 3:
            season = 'spring'
        elif do.month > 0:
            season = 'winter'
        else:
            season = 'error'
        r.append(season)
        logger.debug('keeping line {}'.format(r))
        result.append(r)
    else:
        logger.warning('delete due to failing to match processed scene to a bump sample date')
else:
    logger.debug('seed result with header: {}'.format(i))
    result.append(i)
```

```
# write output into output folder

output_path = working_dir, settings['Processed Dir'], settings['Final Output']

logger.debug('final output: {}'.format(output_path))

type(output_path)

#with open(os.path.join(output_path), 'wb') as file:

with open(os.path.join(working_dir, settings['Processed Dir'], settings['Final Output']), 'wb') as
file:

    writer = csv.writer(file)

    writer.writerows(result)

    logger.info(u'result saved as {}'.format(output_path))


logger.debug('result {}'.format(result))


logger.info(u'finished')
```

VITA

Clay Barrett

Candidate for the Degree of

Master of Science

Thesis: MONITORING EASTERN OKLAHOMA LAKE WATER QUALITY USING LANDSAT

Major Field: Geography

Biographical:

Education:

Completed the requirements for the Master of Science in Geography at Oklahoma State University, Stillwater, Oklahoma in May, 2015

Completed the requirements for the Bachelor of Science in Environmental Science at Oklahoma State University, Stillwater, Oklahoma in July, 2004

Experience:

Currently employed as the GIS Specalist for Cartography Services in the Department of Geography at Oklahoma State University

Professional Memberships:

Association of American Geographers
Gamma Theta Upsilon
South Central Arc User Group
American Society for Photogrammetry and Remote Sensing