GIS MODELING OF RIPARIAN ZONES UTILIZING DIGITAL ELEVATION MODELS AND FLOOD HEIGHT DATA

By

Lacey A. Mason

A THESIS

Submitted in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE IN FORESTRY

MICHIGAN TECHNOLOGICAL UNIVERSITY

2007

TABLE OF CONTENTS

TABLE OF CONTENTS iii
ACKNOWLEDGEMENTS iv
LIST OF TABLES v
LIST OF FIGURES vi
ABSTRACT viii
INTRODUCTION 1
METHODS 6
RESULTS AND DISCUSSION 24
CONCLUSIONS 31
LITERATURE CITED 34
APPENDIX A: Watershed Area Raw Data……..……..……..A-1
APPENDIX B: R-Script for Statistical Analysis of Watershed Area Data……..…B-1
APPENDIX C: Python Code for Variable-Width Delineation Riparian Model…….C-1

ACKNOWLEDGEMENTS

This study was funded by a grant from the National Science Foundation, Materials Use: Science, Engineeering and Society (MUSES) program (grant award number 0524872) and the School of Forest Resources and Environmental Science, Michigan Technological University, Houghton, Michigan. I also wish to thank Timothy Aunan of the Minnesota Department of Natural Resources, Forestry Resource Assessment Program, for providing valuable input into the model development.

I am extremely grateful to my advisor, Dr. Ann L. Maclean, whose patience, kindness, knowledge and encouragement has been invaluable to me. Thanks to my committee, Dr. Christopher Webster, Dr. Linda Nagel and Dr. John Sutherland for all of their guidance and interest in my project. Thanks to James Rivard for help with coding the riparian model, Dr. Robert Froese for advice on statistical analysis and Michael Hyslop for his experience and guidance when troubleshooting problems. Additionally, I would like to thank my office mates, Clara Antón Fernández, Bharat Pohkarel, Chris Miller and Jessie Vital, for all the support, laughter and open-mindedness they continue to foster.

To my parents, Lee and Corine Charles, thank you for continuing to love, support and encourage me. The knot I tied 7 years ago has held strong and now it is time to move on. To my husband, Jim Mason, my utmost gratitude and love, you have had unwaivering patience through the whole process. Thank you for being my rock, without you none of this would have been possible.

LIST OF TABLES

Table 1: Forest best management practices (BMPs) summary for the states of Michigan, Minnesota and Wisconsin. 4

Table 2: Initial data inputs and download sources for the riparian variable-width delineation model. 7

Table 3: Area summaries for the 3 study sites using the variable-width buffer model and fixed-width buffers of 30 and 60 meters. 25

Table 4: Results from sequential F-test of fixed effects, linear-mixed effects model of buffer delineation type and landform, for all study sites. 30

Table 5: Statistical significance test results of riparian delineation methods across 3 landforms at a level of 0.10. The delineation methods included: 1) 10 m DEM; 2) 30m DEM; 3) 30m fixed-width; and 4) 60m fixed-width, NS = not significant. 30

LIST OF FIGURES

Figure 1: Cross-sectional view of a riparian area that links the aquatic habitat to the terrestrial habitat. From Wisconsin DNR (*http://www.dnr.state.wi.us/ORG/LAND/Forestry/Publications/Guidelines/PDF/chapter5.pdf).* 2

Figure 2: A) Northeastern Minnesota study areas (MN Plot). This area includes 3 noncontiguous areas with groups of 5, 4 and 27 adjacent watersheds. B) Upper Peninsula of Michigan study area (UP Plot). This area is composed of 16 adjacent watersheds. C) Lower Peninsula of Michigan study area (Plot LP). The area is composed of 9 watersheds in and near Oscoda County. 10

Figure 3: Regression graphs for one example site showing the hydrologic estimators for determining an approximate flood height. Calculations shown are A) flow rate vs. recurrence interval fitted lograthmically; B) cross-sectional area vs. flow rate; and C) width vs. cross-sectional area, stepping through these calculations results in an approximate flood height. 13

Figure 4: Flowchart of the riparian zone model. The model is composed of 5 modules to facilitate editing and customization. The heavy black line indicates the transition into the next module 15

Figure 5: An example of how the Dissolve tool works in ArcGIS on a section of streams. In the upper box, there are 18 individual stream segments that are dissolved by reach code to create 8 stream segments in the lower box. 16

Figure 6: Example showing: A) streams and lakes layer input into Module 1; and B) the streams feature class after the erase function. 16

Figure 7: Sample points, shown in magenta, generated by Module 2 along all stream segements. 17

Figure 8: Black dots show the location of transect points generated in Module 3. 18

Figure 9: Boundary of the riparian area, shown as yellow points calculated in module 4 of the model. The magenta dots are the original sample points from which the transects originate and the red dots are transect points outside of the study area. 20

Figure 10: Rasterization of points within the riparian zone. The spatial resolution is equal to that of the DEM elevations used as input into the model. The raster is cleaned using the Boundary Clean Tool with a one-way sort. 21

Figure 11: A close-up look at part of the Northern Minnesota study site showing elevation across the landscape, notice the elevation change inside the white oval. . 26

Figure 12: The same site with the 10 m and 30 m variable-width buffers overlaid. Notice the 30 m variable-width buffer did not recognize the elevation change inside the white oval. 26

Figure 13: A close-up look at part of the Northern Minnesota study site, comparing the riparian variable-width model 10 m DEM resolution and 30 m DEM resolution results with a fixed-width buffer distance of 60 m. Note the 30 m fixed-width buffer (not shown) would be half as wide as the 60 m fixed-width buffer. 27

Figure 14: A digital infrared orthophoto close-up look at part of the Lower Peninsula of Michigan study site. The 10 m DEM variable-width riparian boundary is shown in magenta and the 30 m fixed-width buffer is shown in yellow. 28

Figure 15: A digital infrared orthophoto close-up look at part of the Upper Peninsula of Michigan study site. The 10 m DEM variable-width riparian boundary is shown in magenta and the 30 m fixed-width buffer is shown in yellow. 29

ABSTRACT

Riparian ecotones are unique, diverse networks of vegetation and soils in close proximity to freshwater streams, rivers and lakes. These ecotones are linked to the watercourse network via flooding and intercepting upland runoff. Vegetation communities along stream banks often delineate riparian boundaries. However, geology, soil chemistry, and hydrologic changes need to be considered as well. Previous approaches to riparian boundary delineation have utilized fixed width buffers, but this methodology has proven to be inadequate as there are two factors that all riparian ecotones are dependent on: the watercourse and its associated floodplain. Using a fixed width riparian buffer only takes the watercourse into consideration. Past research has determined the 50-year floodplain is the optimal hydrologic descriptor of a riparian ecotone. By hydrologically defining a riparian ecotone to occur at the 50-year flood height and incorporating digital elevation data, the spatial modeling capabilities of ArcMap GIS were utilized to map riparian zones accurately and efficiently. The main objective of the study was to determine if there was a significant delineation difference between using 10 m versus 30 m Digital Elevation Models (DEM). The second objective was to determine if delineation of variable-width riparian areas was more accurate then current best management practice (BMP) suggestions. The study areas for model development included watersheds in northeastern Minnesota, the central Upper Peninsula of Michigan and the central Lower Peninsula of Michigan. The result of this study is a GIS-based model that delineates a variable-width riparian boundary. The model offers advantages over previously used methods of riparian ecotone mapping, such as fixed width buffers, by better characterizing the watercourse and its associated floodplain. Riparian zones delineated using 10 and 30 m

DEMs, along with stream course information from the National Hydrological Data, were statistically significant. This indicates that the coarser scale of 30 m DEMs may not sufficient to adequately map elevation changes for riparian area delineation in the upper Midwest states.

INTRODUCTION

Riparian ecotones are unique, diverse networks of vegetation and soils in close proximity to freshwater streams, rivers and lakes. For this study, a riparian ecotone is defined as "…a three-dimensional space of interaction that includes terrestrial and aquatic ecosystems that extend down into the groundwater, up above the canopy, outward across the floodplain, up the near-slopes that drain to the water, laterally into the terrestrial ecosystem, and along the water course at a variable width" (Verry and others 2004) and is illustrated in Figure 1. The ecotone is linked to the watercourse network via flooding and intercepting upland runoff (Mitsch and Gosselink 2000). It is important to note that riparian ecotones are typically defined by local conditions but respond to climatic and geological processes on continental scales via interconnecting watersheds. Hence any riparian zone delineation model must be scale independent. Vegetation communities along stream banks often delineate riparian boundaries. However, geology, soil chemistry, hydrologic changes and animal habitats also need to be considered (Naiman and McClain 2005).

Riparian areas contributes many important functions to the landscape, in addition to linking aquatic and terrestrial habitats. Riparian areas have heavily vegetated banks creating stability that reduces erosion and flooding. The vegetation and rich soils also have high rates of nutrient absorption and filter pollutants from the entire watershed, and all of these factors increase the habitat for animal diversity. With continuous corridors over the landscape, it is an ideal situation for species dispersal of animals and plants.

Most importantly riparian areas create improved environmental conditions and an aesthetically pleasing area for recreationl activities.

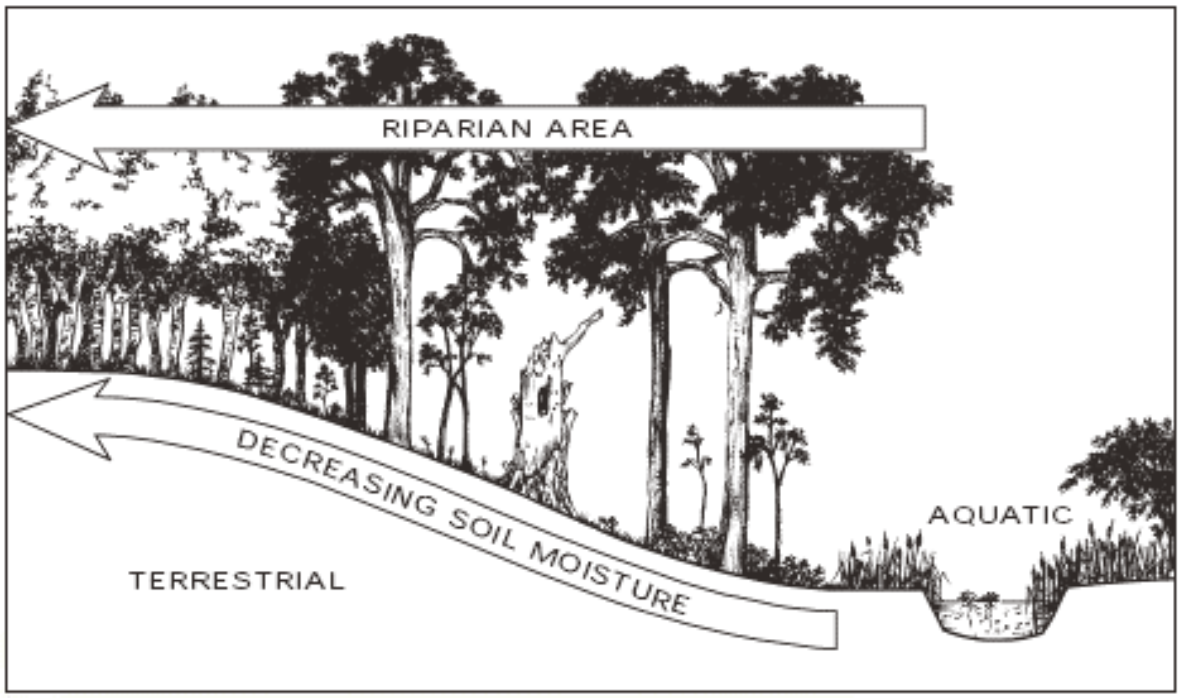

Figure 1: Cross-sectional view of a riparian area that links the aquatic habitat to the terrestrial habitat. From Wisconsin DNR (*http://www.dnr.state.wi.us/ORG/LAND/Forestry/Publications/Guidelines/PDF/chapter5.pdf*).

Previous approaches to riparian area delineation have utilized fixed width buffers, but this methodology has proven to be inadequate. Palik and others (2000) determined that fixed-width buffers do not emulate natural riparian corridors since they have no functional relationship to the naturally varying watercourse. Suggested buffer width guidelines from the Minnesota Forest Resources Council were evaluated by Skally and Sagor (2001) in a single-case pilot study. Their report described the difficulty in using the designated guidelines of fixed-width buffers because of many watercourse variables,

such as site condition, water body type and forest management history which need to be incorporated into the delineation process. Their research also concluded that the riparian ecotone boundary was on average 2.5 times farther from the stream than the suggested buffer width within the selected study area. Hanowski and others (2000) studied the effects of riparian buffers on breeding bird populations and used fixed-width buffers of 28.5 m and 57 m based on the spatial resolution of the Landsat TM sensor. The method was easy to use for the study, but did little to explain the relationship of buffer widths to the water bodies, and also concluded, to protect these areas would likely require wider buffers in addition to other management strategies. Hanowski and others (2007) later determined that, "management of riparian forests should focus on the protection of fine scale habitat features that may be important wildlife habitat features in riparian forests." This is also in agreement with the research of Macdonald and others (2003), who determined that management of riparian forests should meet specific conservation and management objectives through 1) variable-width buffers; 2) fine-scale management of specific habitat features; and 3) landscape-level planning for fire-suppresions and uncut forested areas.

Currently, fixed-width riparian buffer guidelines are specified by best management practices (BMPs) for forests and are implemented in Michigan, Minnesota and Wisconsin on a voluntary basis. The guidelines vary considerably, ash shown in Table 1. Michigan's BMPs are based on slope of the adjacent land to a stream. Minnesota's riparian management zones (RMZs) are based on whether the stream is habitat for trout and how the adjacent forest stands are managed (even vs. uneven aged). Wisconsin's

BMPs are determined by the stream width and whether or not the stream is naviagable. The definition of naviagable is whether or not a canoe can float in the stream. Justification as to how these fixed-width buffers were defined is difficult, due to the lack of references and literature cited in the BMP manuals. The disparity in BMP guidelines is interesting in that the 3 states, for the most, share the same ecological provinces as defined by Albert (1995).

Table 1: Forest best management practices (BMPs) summary for the states of Michigan, Minnesota and Wisconsin.

Michigan	
Slope (%)	***Minimum Width (ft)***
0-10	100
10-20	115
20-30	135
30-40	155
40-50	175
50+	No activity acceptable
Minnesota	
Basis	***Minimum Width (ft)***
Even-aged stands	150
Uneven-aged stands	200
Non-trout surface waters	50-100
Trout surface waters	50-200
Wisconsin	
Basis	***Minimum Width (ft)***
Lakes and navigable perennial streams	100
Naviagable intermittent streams	35
Non-navigable streams	35

Developing an all-encompassing definition for riparian ecotones, because of their high variability, is challenging. However, there are two factors that all riparian ecotones are

dependent on: the watercourse and its associated floodplain. Using a fixed width riparian buffer only takes the watercourse into consideration and ignores the critical surrounding geomorphology.

Research by Ilhardt and others (2000) determined the 50-year floodplain was the optimal hydrologic descriptor of a riparian ecotone. This flood recurrence interval was selected because the 50-year flood elevation, in most cases, intersects the first terrace or other up sloping surface and supports the same microclimate and geomorphology as the stream channel. The 50-year flood plain also coincides with measurements that quantify a valley to its streams via two measurements: 1) the entrenchment ratio (valley width at the first terrace or up slope to the stream width at full bank), and 2) the belt width ratio visible on aerial photos or maps (Ilhardt and others 2000).

Lakes are not as impacted by floodwaters compared to moving watercourses, but typically have a defined high water mark. This presents an issue of how to define a riparian ecotone boundary around standing, open water bodies. Within 30m (100 ft) of lakes, forest cover contributes 60-80% of its influencing habitat function, such as shade, woody debris recruitment, bank stability and litter fall as noted by Ilhardt and others (2000). Given more information about a lake's total area, watershed basin size and geomorphology a variable-width buffer could also be applied, but was beyond the scope of this study.

By hydrologically defining a riparian ecotone boundary as occurring at the 50-year flood height and incorporating digital elevation data, the spatial modeling capabilities of ArcMapGIS can be utilized to map variable-width riparian zones adequately and efficiently. Aunan and others (2005) initiated work on developing a variable-width riparian model utilizing ArcGIS, DEMs and flood heigh information. Their research proved the feasibility of developing such a model. However, the model had a flaw in that it coverted the raster DEM to a vector based format, which consistently contributed to an overestimate of riparian area. The underlying science of their model is sound. Given improvements in ArcGIS and computing capabilities over the past several years we felt that initial model could be improved. As part of upgrading the model, two objectives were formulated. The first was to determine the impact of the 10 m versus 30 m DEMs. The second was to compare the variable-width delineated riparian areas to the fixed-width buffers recommended in the BMPs for mapping accuracy.

METHODS

Data Inputs and Study Areas

The riparian delineation model utilized ArcGIS Desktop 9.1 produced by ESRI, Inc. (ESRI 1999-2005) for all data manipulation, management and spatial analyses. Inputs into the model were set up as geodatabases. The riparian zone delineation model used the coding language Python. The model creates variable-width riparian ecotone boundaries based on stream and lake locations, digital elevation data and the 50-year flood height

variable associated with stream segment order. Specific data inputs and their sources are listed in Table 2 and discussed below.

Table 2: Initial data inputs and download sources for the riparian variable-width delineation model.

Input Data	Sources
Streams	USGS National Hydrography Dataset (NHD) http://nhd.usgs.gov/
	Michigan Center for Geograhic Information http://www.michigan.gov/cgi
	Minnesota DNR Data Deli http://deli.dnr.state.mn.us/
Lakes	Michigan Center for Geographic Information http://www.michigan.gov/cgi
	Minnesota DNR Data Deli http://deli.dnr.state.mn.us/
10m Digital Elevation Model	GIS Data Depot http://data.geocomm.com/
30m Digital Elevation Model	USGS, The National Map http://nationalmap.gov/

The National Hydrography Dataset (NHD) is a feature-based dataset organized into ArcMap geodatabases. The data provide continuous, national coverage of stream reaches and water drainage systems and is overseen by the United States Geological Survey (USGS). The NHD is comprised of water-related entities such as natural river courses, lakes, ditches, industrial discharges and drinking water supplies. Each entity has an assigned address that establishes its location and connections to other entities in the drainage network (USGS, 2005). Currently there is nationwide coverage at 1:100,000 with larger scale coverage being developed at 1:24,000 and 1:12,000. For this study the

1:24,000 data was used where available. Data gaps were supplemented with information from state supported GIS systems (Table 2).

The USGS Digital Elevation Models (DEMs) are raster based elevations sampled at regularly spaced ground locations and registered to the UTM projected coordinate system. DEMs with spatial resolutions of 10 m and 30 m were used for this study. The 10 m DEM data, which has a per pixel area of 100 m^2 (0.025 acres), were downloaded in 7.5' quadrangle format from the GIS Data Depot (GeoCommunity 2007) and mosaiced to create a continuous coverage. The 30 m DEMs, covering 900 m^2 per pixel (0.22 acres), were downloaded from The National Map seamless dataset (USGS Geography 2007).

Flood height data was downloaded in a tabular format from the USGS Real-Time Water data site (USGS 2007). The USGS Real-Time water data collection system is composed of monitoring sites that record data at 15-60 minute intervals. The information is either stored onsite or transmitted to a USGS office in 1 to 4 hour increments. The data are transmitted via satellite, telephone or radio, and are available for viewing within minutes of arrival. During critical events, recording and transmission times are more frequent.

The study sites were comprised of multiple watersheds in 3 locations: northeast Minnesota with 36 watersheds, the central Upper Peninsula (UP) of Michigan with 16 and the eastern Lower Peninsula (LP) of Michigan with 9 watersheds (Figure 2). These locations were selected based on 10 m DEM data availability, which is more widely

available in areas of federally owned lands. The landforms present in the study areas are complex and diverse.

The northeastern Minnesota study sites consist of two landforms - border lakes and Lake Superior highlands - both with numerous lakes. The border lakes are composed of scoured bedrock uplands or shallow soils on bedrock interspersed with outwash plains. Ground moraine and end moraine of the Superior Lobe label this area part of the Lake Superior Highlands. A clay lake plain forms a broad band along the Lake Superior shoreline that is flat to rolling with steep narrow ravines creating numerous short, 15 to 25 km (10-15 miles), streams (Albert 1995).

The Michigan UP study site is also made up of two major landforms - the Grand Marais sandy end moraine and outwash, and the Seney sand lake plain - both of lacustrine origin. The Grand Marais landform is composed of sandy ridges of end moraine. The moraine contains droughty sand dunes and beach ridge deposits as well as poorly and very poorly drained glacial lacustrine deposits (Albert 1995). The Seney sand lake plain contains broad, poorly drained embayments with beach ridges and swales, sand spits, transverse sand dunes and sand bars. Along the northern margins of the embayments deltaic deposits occur where glacial streams carried massive amounts of sand into shallow waters (Albet 1995).

The Michigan LP study site is located on a high plateau. This landform is mostly outwash plain with large sandy ground moraines and end moraines and ice-contact

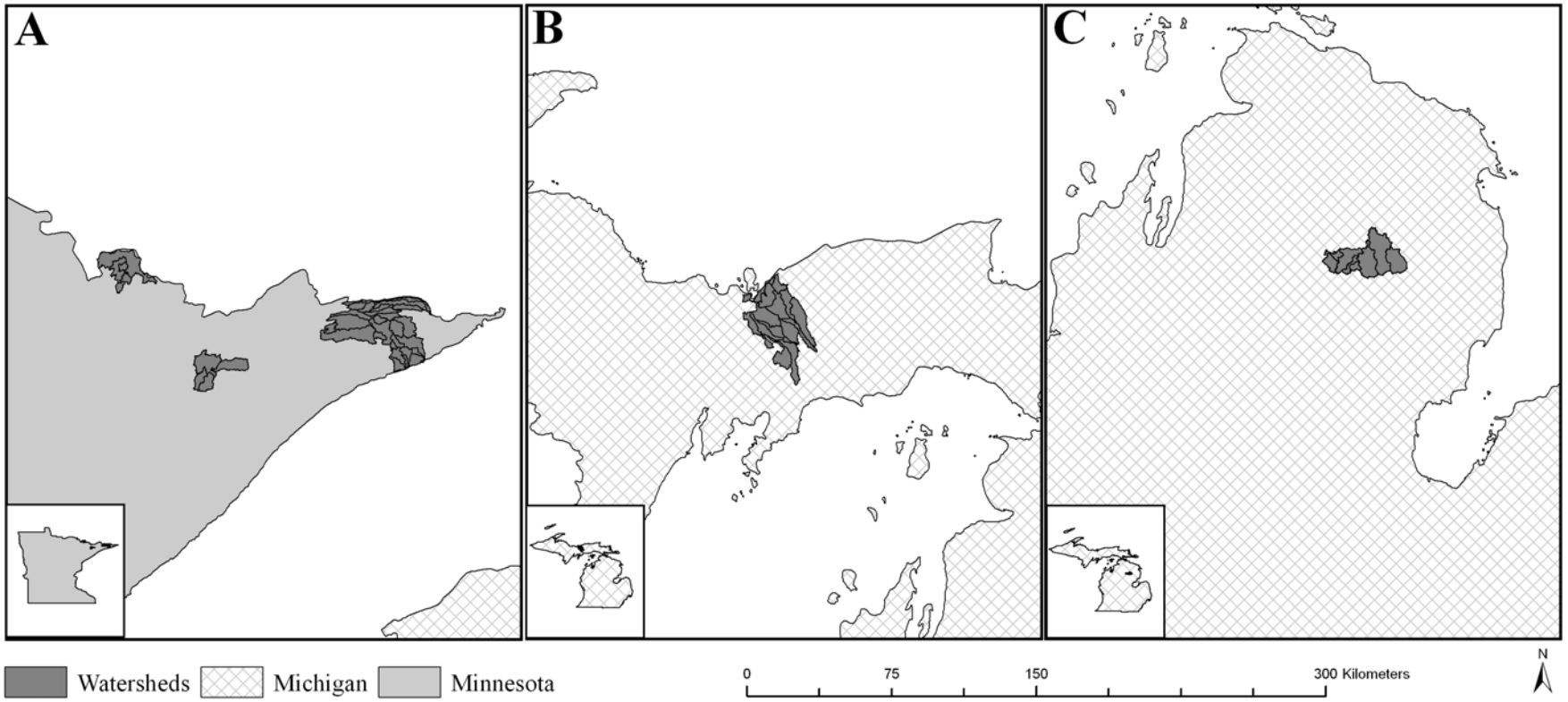

Figure 2: A) Northeastern Minnesota study areas (MN Plot). This area includes 3 noncontiguous areas with groups of 5, 4 and 27 adjacent watersheds. B) Upper Peninsula of Michigan study area (UP Plot). This area is composed of 16 adjacent watersheds. C) Lower Peninsula of Michigan study area (Plot LP). The area is composed of 9 watersheds in and near Oscoda County.

ridges. The site covers two subsections including Cadillac (sandy end-moraine) towards the southwest and Grayling (ice-contact topography) to the northeast (Albert 1995).

Hydrologic Estimations

A determination of an appropriate 50-year flood height was necessary and vital information for the model. To estimate flood heights, data from 10 Minnesota sites and 8 Michigan sites which occurred within or 100 kilometers of the study areas were obtained from the USGS Real-Time Water Data website (USGS 2007). The data included the annual average stream flow rate and frequent measurements of flow rate, velocity and width.

Based on hydrologic principles from Bedient and Huber (2002) and the realtionship between stream measurements, flood heights were calculated. To do this, the annual average flow rate measurements were organized by year and sorted from fastest to slowest for each stream gauge location. After sorting, the annual flow rate measurements were ordinally ranked, so that the fastest flow rate received a value of 1. To calculate the recurrence interval, the rank number was divided by the number of measurements.

Recurrence Interval (Years) = (Total # of Data Points) / Rank

The flow rate was then plotted against the logarithmic recurrence interval to develop a flood occurrence regression (Bedient and Huber 2002). An example of an individual site regression is shown in Figure 3A. The cross-sectional area (flow rate divided by

velocity) is plotted against flow rate measurements, as illustrated in Figure 3B. Figure 3C shows the regression of the width versus the cross-sectional area. An R-squared value of 0.85 or higher was noted for all fitted equations. The width and cross-sectional area were determined from this series of regressions, and the stream height was calculated by dividing the cross-sectional area by the width.

Using the regression equations for each corresponding to each stream gauge, 1-year (to provide a baseline) and 50-year flood heights were determined. The flood height calculation results ranged between 0.3 and 1.75 m across all data sites. To simplify the initial model and shorten computing time we used, an average flood height of 1 m. However, this height might vary based on stream order or other information as the model has been written to utilize varying flood heights. The stream order information for each segment can be entered into the attribute table of the GIS vector layer and the 50-year flood heights corresponding to stream orders can be set as model parameters. This allows the model to utilize a different elevation change for each stream order level.

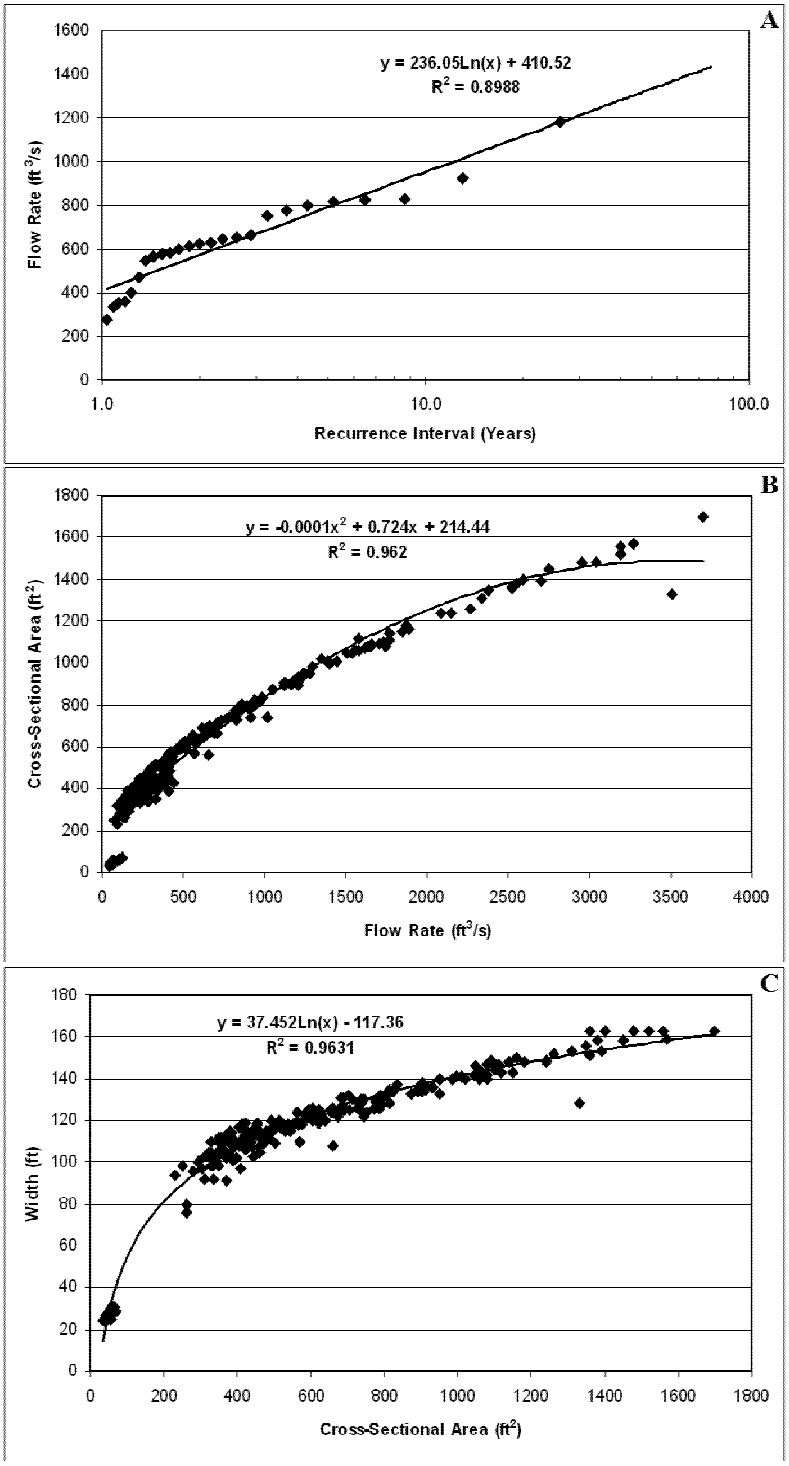

Figure 3: Regression graphs for one example site showing the hydrologic estimators for determining an approximate flood height. Calculations shown are A) flow rate vs. recurrence interval fitted lograthmically; B) cross-sectional area vs. flow rate; and C) width vs. cross-sectional area, stepping through these calculations results in an approximate flood height.

Model Development

The modeling language Python was used to develop the riparian delineation model. The model is composed of five modules to facilitate customization (Figure 4) and maximize computer processing efficiency. Inputs must be in ArcMap geodatabase format and the user must have access to ArcMap's Spatial Analyst extension. The five modules are defined as: 1) preparing input data and calculations the lake buffers; 2) building sample points along streams; 3) building transects along streams; 4) determining the outside edge of the variable-width buffer and reprocessing areas that did not find the flood height elevation; and 5) creating an easy to use riparian boundary polygon. The program does utilize built-in tools offered through the ArcGIS software, but most of the analysis functions were designed and coded specifically for this study. To aid in understanding model processing, example output from each module is provided. The example outputs are from the same geographic location.

The first module edits the streams and lakes feature classes for input (Figure 4, Module 1). Each stream length is typically composed of several stream segments designated with a reach code. To optimize transect building, the stream segments are dissolved by reach code to remove extraneous nodes, Figure 5. Next, stream segments delineated within a lake or other open water bodies are erased, as mapping of a riparian zone along these segments would be erroneous (Figure 6). Lastly, a 30.5 m (100 ft) buffer is computed around all lakes and other open water bodies based on the recommendations of Ilhardt and others (2000).

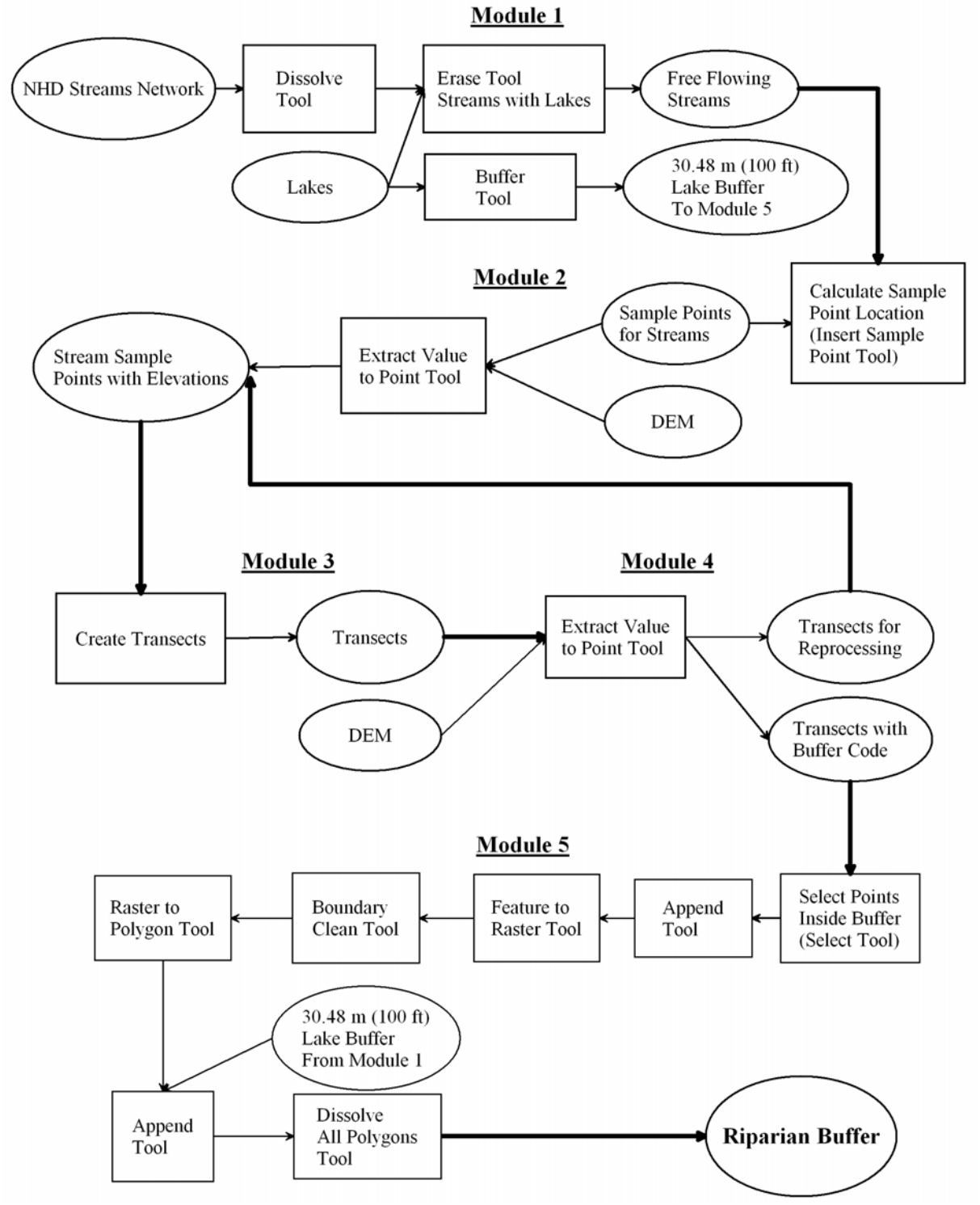

Figure 4: Flowchart of the riparian zone model. The model is composed of 5 modules to facilitate editing and customization. The heavy black line indicates the transition into the next module.

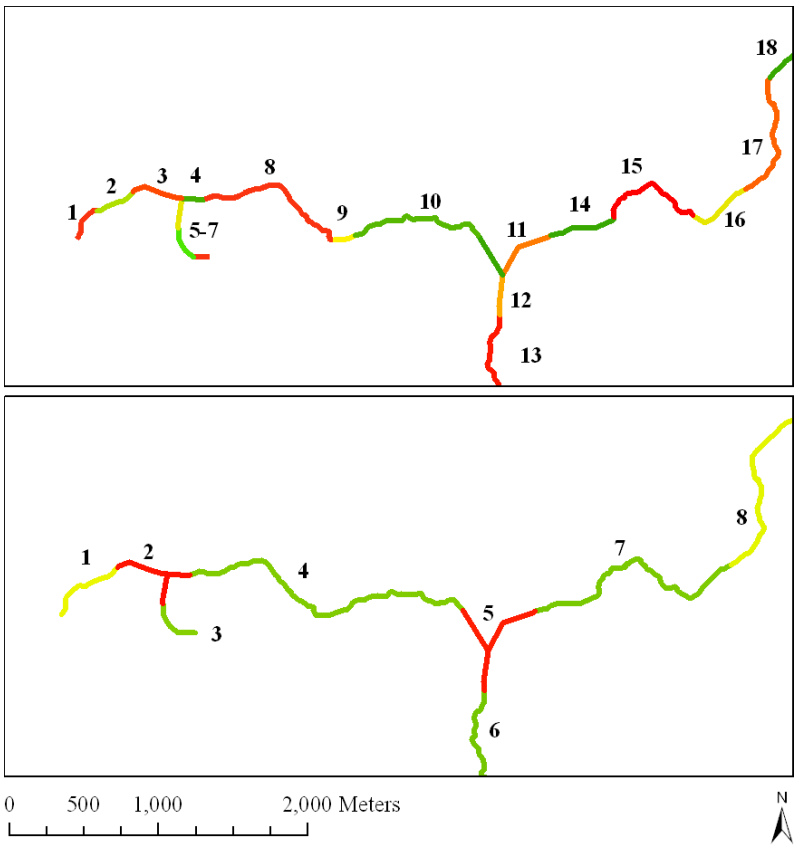

Figure 5: An example of how the Dissolve tool works in ArcGIS on a section of streams. In the upper box, there are 18 individual stream segments that are dissolved by reach code to create 8 stream segments in the lower box.

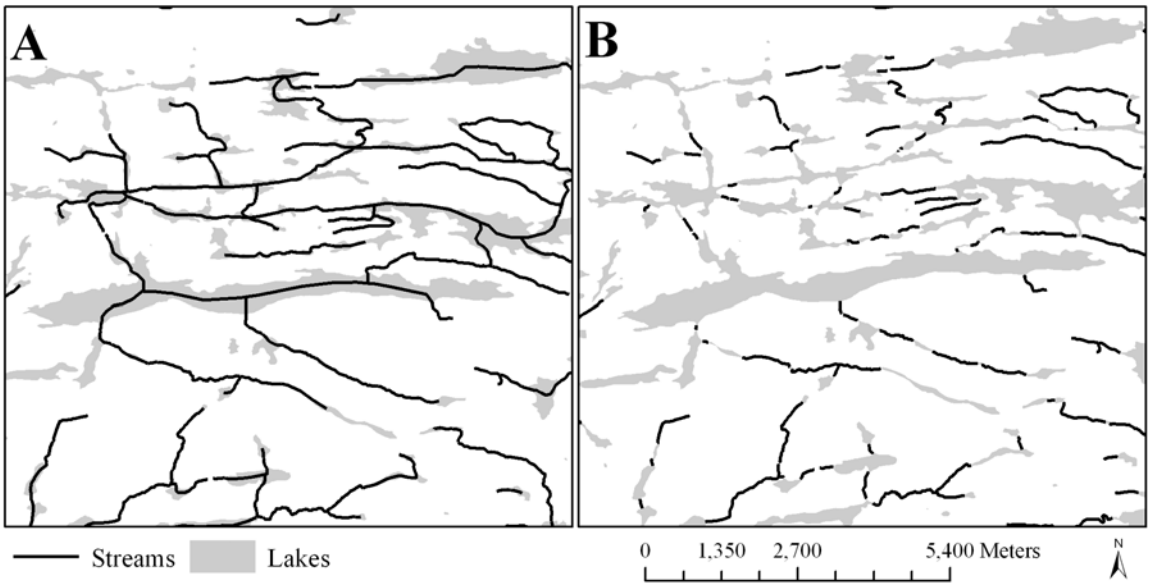

Figure 6: Example showing: A) streams and lakes layer input into Module 1; and B) the streams feature class after the erase function.

Module 2 calculates the x, y coordinates for the starting point of each transect (Figure 4, Module 2). Input parameters include the DEM's spatial resolution and a pixel ratio, expressed as a percentage of pixel size. The distance between sample points is set to a distance of 75% of the pixel's spatial resolution along each stream segment. This was done to minimize the influence of the DEM's spatial resolution on the distribution of the sample points along the stream course, but not assume a horizontal positional accuracy better than the DEM's accuracy standard (USGS 1997). Point spacing is calculated using Euclidean distance from one point to the next along the stream segment. The stream segments are treated as continuous features to avoid sampling gaps and to maintain a constant spacing distance, Figure 7.

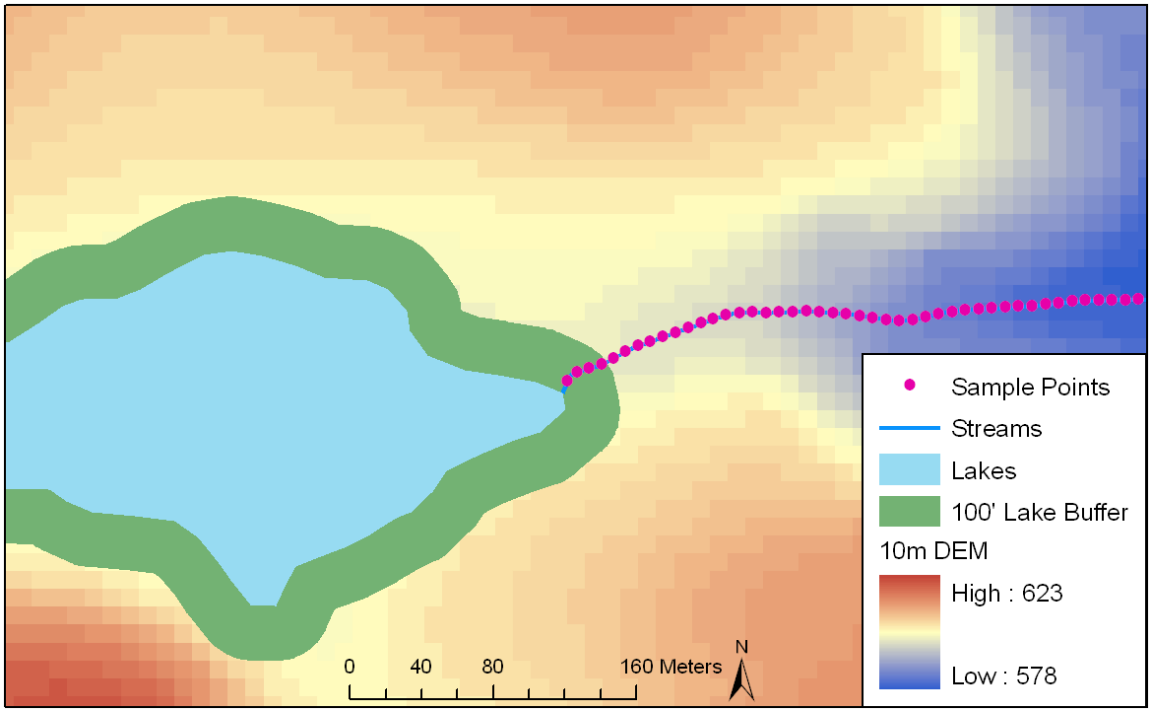

Figure 7: Sample points, shown in magenta, generated by Module 2 along all stream segements.

Upon completion of the stream sample point calculations, the program retrieves the elevation for each sample point from the DEM and writes the value to the sample point attribute table.

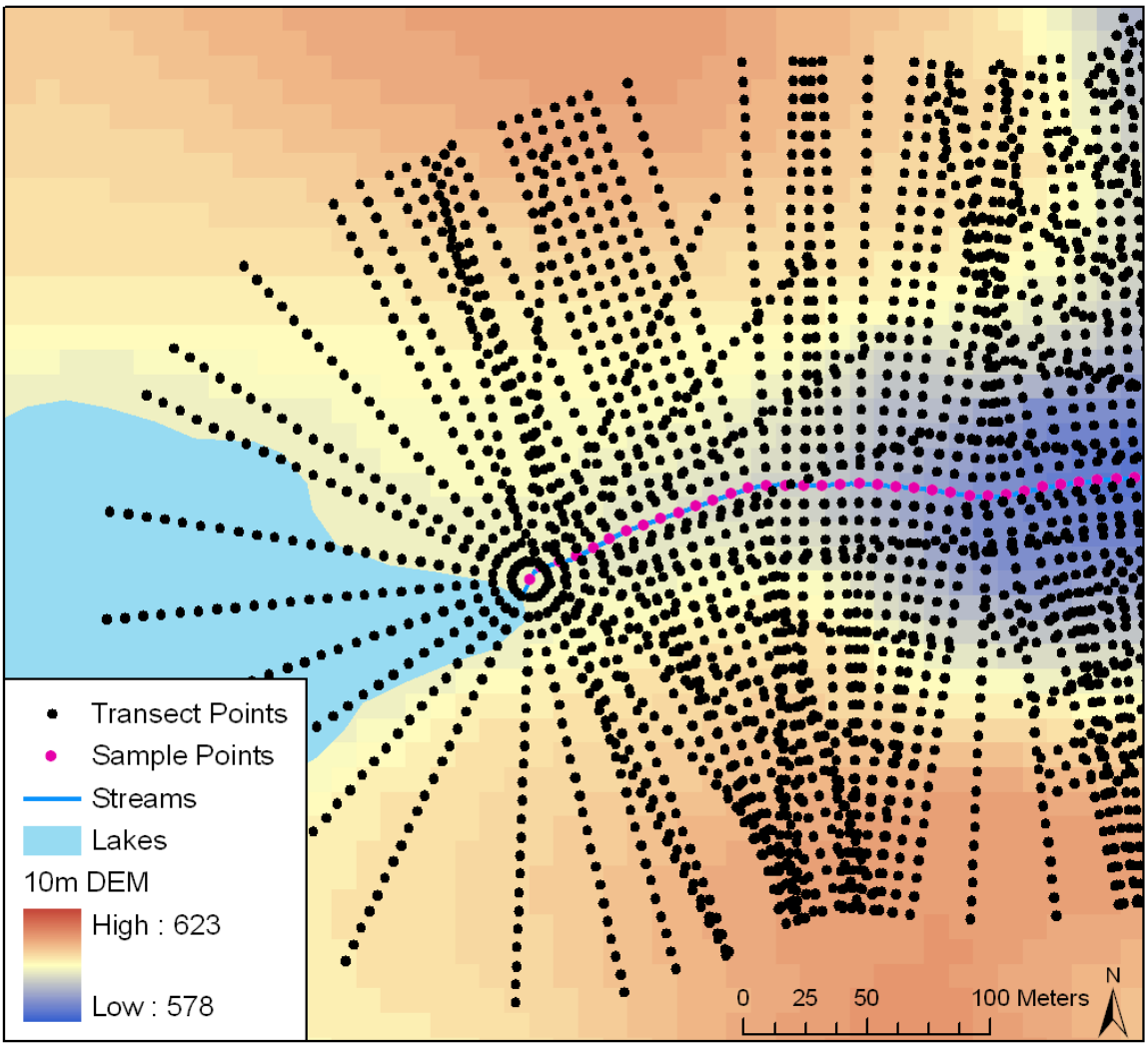

Figure 8: Black dots show the location of transect points generated in Module 3.

The third module (Figure 4**,** Module 3) produces transects perpendicular to the direction of the stream channel at each sample point, **Error! Reference source not found.**. At the origin and termination of streams, transects are created in a circle around the nodes to for additional sample points. This ensures a more realistic mapping of the riparian area as

opposed to an abrupt cutoff point, which is a data analyses artifact. To optimize processing time, a maximum transect length was imposedl; however, the user may change this. If the transect does not find an elevation change within the maximum specified transect length, it is flagged for reprocessing without a length limitation.

Based on elevation change, module 4 determines if the transect points are part of the riparian buffer (Figure 4, Module 4). If the elevation change is 1 m (the average calculated flood height) or less between the transect's starting point and the point in question, the point is included in the riparian zone (inbuffer = 1). If the elevation change is greater than a meter, the point is considered to be outside the riparian zone (inbuffer = 2). The last "in" point is flagged (inbuffer = 3) to establish the location of the riparian boundary (Figure 9). If an elevation change does not occur, the transect is flagged and the original transect point coordinates are written to an output file for reprocessing in modules 3 and 4. The reprocessing elongates the original transect until the required elevation change is met. The output file (transects with buffer transect code) contains all of the sample points from all transects.

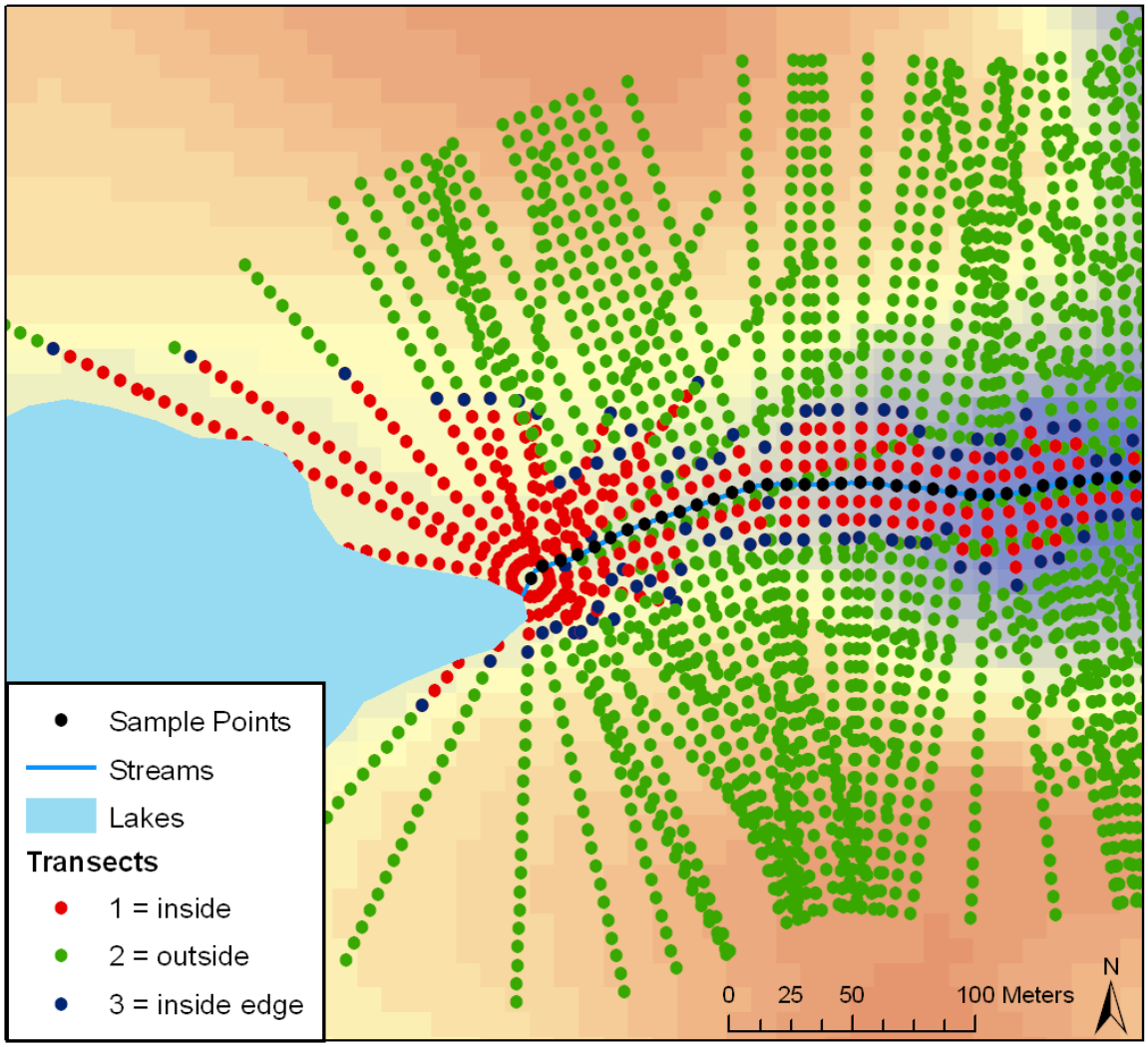

Figure 9: Boundary of the riparian area, shown as yellow points calculated in module 4 of the model. The magenta dots are the original sample points from which the transects originate and the red dots are transect points outside of the study area.

The riparian zone layer including all the buffer areas around streams and lakes is calculated in module 5 (Figure 4, Module 5). The program reads the transects with buffer code attribute table and selects only the points whose in buffer attribute equals 1 or 3. The original sample points along the streams are appended to the transect feature to prevent gaps in the center of the buffer. This composite of points is rasterized with a spatial resolution equal to the DEM (Figure 10). The raster is cleaned using the

Boundary Clean Tool for smoothing ragged edges along boundaries. The one-way sort option is selected to enable the sample points on the stream segment to remain in the buffer after processing. Otherwise, if the buffer is only one pixel wide, these individual pixels are not prioritized and are removed in a two-way sort. The cleaned raster is converted to a vector polygon. The final riparian area consists of the stream riparian zone (polygon) merged with the 30.5 m (100 ft) lake buffer created in module 1. The merged buffer is typically composed of many irregularly shaped, adjacent polygons at this point, so the Dissolve Tool is executed, with the "all" option selected, to create one contiguous riparian area.

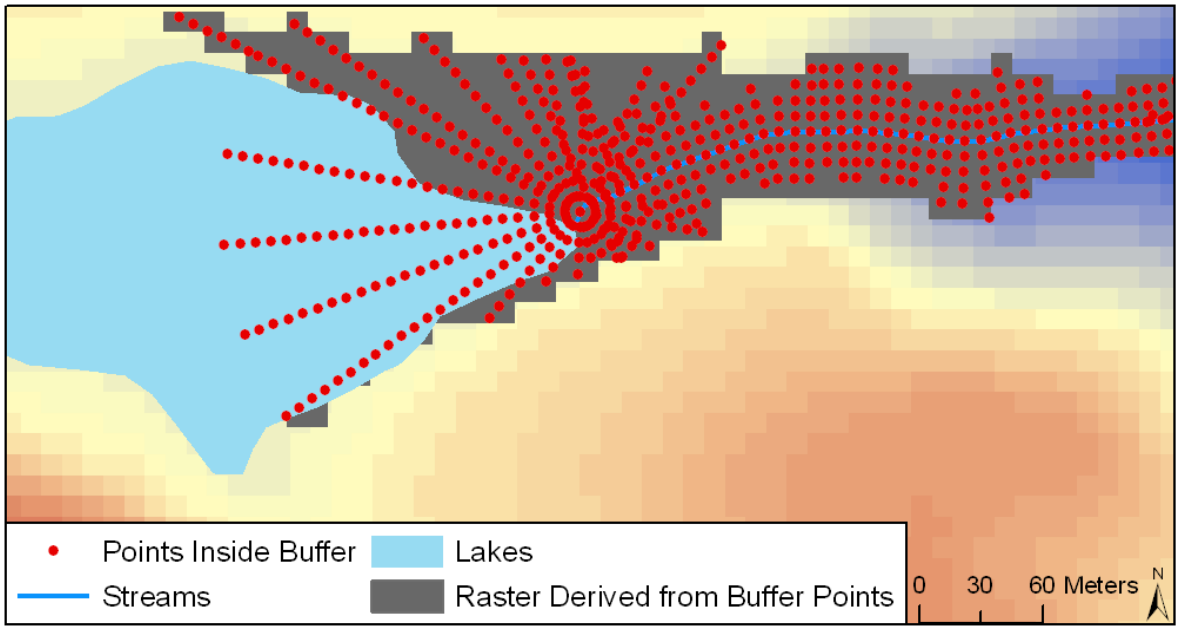

Figure 10: Rasterization of points within the riparian zone. The spatial resolution is equal to that of the DEM elevations used as input into the model. The raster is cleaned using the Boundary Clean Tool with a one-way sort.

Fixed-Width Riparian Zones

Fixed-width riparian zones of 30 m and 60 m were generated to compare to the riparian

area calculated by the model. These widths were chosen based on the recommendations by Palik and others (2004), and permits a direct comparison to their findings. The fixed-widths buffers were created using the Buffer Tool with the linear distance input 30 m for the first GIS layer created, and 60 m for the second layer created. The option to "dissolve all" was selected to create one continuous feature around all stream segments and prevent an overestimation of total riparian area.

Visual Assessment of Variable-Width Riparian Zones

After all of the study sites were processed through the model a visual assessment was made of the variable-width riparian boundaries. The visual assessment was to evaluate boundary placement. The model output boundaries were overlaid on digital infrared orthophotos and scrutinized. The expert was looking for matches with vegetation, development, logged areas and dark soils and water to match with the riparian boundaries.

Statistical Assessment Between Variable-Width and Fixed-Width Riparian Zones

The statistical assessment was was performed to test for a significant difference in area between the variable-width and fixed-width riparian areas. To evaluate the impact of different spatial resolutions, riparian zones were calculated for the same areas using both the 10 m and the 30 m resolution DEMs. In addition 30 m and 60 m fixed-width buffers were also calculated. The riparian zone total area was summed for multiple watersheds within each of the 3 study sites, excluding lake surface area. The statistical data table contained a unique ID for each watershed (1-61), the riparian area total in meters squared,

designation of which buffer type corresponds to that specific polygon (1 of 4), and then the landform the watershed was located in (1 of 3), this created 244 unique cases. This information was input into the program R for Statistical Computing (R Development Core Team 2005) and analyzed to ascertain if there was a statistically significant difference between the riparian areas delineated with the 10 m DEM, the 30 m DEM, the 30 m fixed-width, and 60 m fixed-width buffers.

An analysis of variance was used to test whether riparian zone delineation method or landform had a significant effect on estimated riparian area. In this study, the delineation was repeatedly applied to the same subject (i.e., the same watershed). Therefore, the appropriate analytical approach was to analyze the study as a repeated measures design (Kutner and others 2005). The corresponding linear, mixed-effects model included several components. The riparian area was the *response*, the *treatment effect*, which was the delineation method, is a change in the response variable due to the application of a treatment. The landform was the *block effect* that describes the change in the response variable due to membership of an experimental unit (watershed) in a given block (landform). The landform is not a treatment in this study because it was not assigned randomly to an experimental unit. A *block-treatment* interaction occurs when the treatment effect on experimental units was not independent of the block effect. In other words, the treatment effect differs by block. The *subject effect* (watershed) was treated as a random variable. The subject effect, essentially a block effect, is the change in the response variable due to the fact that the treatment (delineation method) was applied more than once to the same experimental unit (watershed).

Model estimation was performed in the R statistical environment (R Development Core Team 2005). Fitting used a linear mixed-effects function, which relies upon maximizing the restricted log-likelihood (Pinheiro and Bates 2000). This approach permitted straightforward accounting for lack of balance in the data because the number of watersheds in each landform, or block, is not the same (Kutner and others 2005). Normality was assessed using normal probability plots and assumptions of within-subject variance homogeneity and additivity were examined using scatterplots.

RESULTS AND DISCUSSION

The variable-width riparian areas calculated from the 10 m and 30 m DEMs produced quite different area totals and spatial extents (Table 3). For all of the watersheds in the three study areas, the riparian areas derived from the 30 m DEM are larger than those calculated using the 10 m data. Based on a visual qualitative assessment of key locations in the three study areas, portions of the riparian buffers generated with the 30 m DEM are located beyond the boundary of the actual riparian area. A representative area is illustrated in Figure 11 and Figure 12.

This result was anticipated given that the spatial resolution of the 30 m DEM is 9X larger than the 10 m. What is more important is the fact that we have effectively shown the inadequacies of the 30 m DEM to accurately map elevation changes in a landscape

Table 3: Area summaries for the 3 study sites using the variable-width buffer model and fixed-width buffers of 30 and 60 meters.

Study Sites			Minnesota	Upper Peninsula Michigan	Lower Peninsula Michigan
Total Watershed Area (Ha)			168,642	92,009	59,274
Variable-width Buffer	10m DEM	Buffer Area (Ha)	16,359	17,014	6,130
		% of Watershed	9.70%	18.49%	10.34%
	30m DEM	Buffer Area (Ha)	20,454	28,503	10,063
		% of Watershed	12.13%	30.98%	16.98%
Fixed-width Buffer	30m Buffer	Buffer Area (Ha)	4,279	3,563	2,880
		% of Watershed	2.54%	3.87%	4.86%
	60m Buffer	Buffer Area (Ha)	8,726	6,896	5,704
		% of Watershed	5.17%	7.49%	9.62%

heavily impacted by glaciation which has resulted in significant elevation differences over short distances (in this study, less than 30 m).

In Figure 11 an obvious elevation change is denoted by a white oval. When examining the same location in Figure 12 with the variable-width buffers overlaid, the 10 m DEM boundary notices the elevation and creates a boundary separating each side. But when looking at the 30 m DEM boundary, the pixel are so large, the elevation change has been averaged out to not create a change of more than the designated flood height. This creates a riparian area that is too large. Figure 13 shows the 60 m fixed-width buffer overlaid on the variable-width boundaries, and it is immediately recognizeable that it does not follow the landscape or characterize the watercourse.

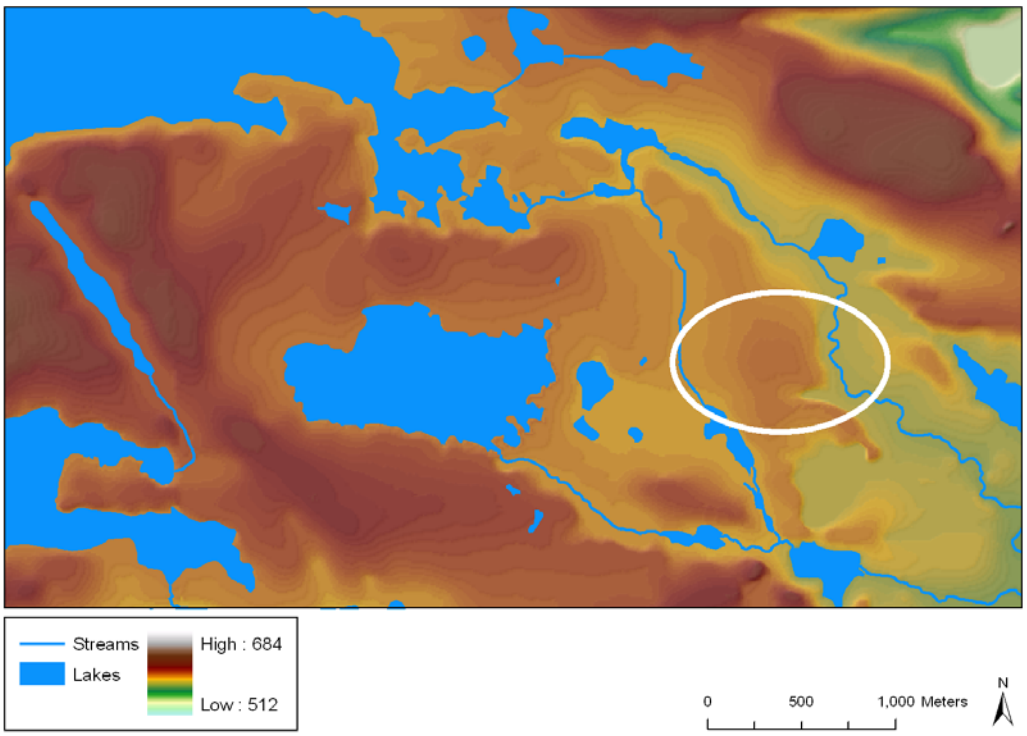

Figure 11: A close-up look at part of the Northern Minnesota study site showing elevation across the landscape, notice the elevation change inside the white oval.

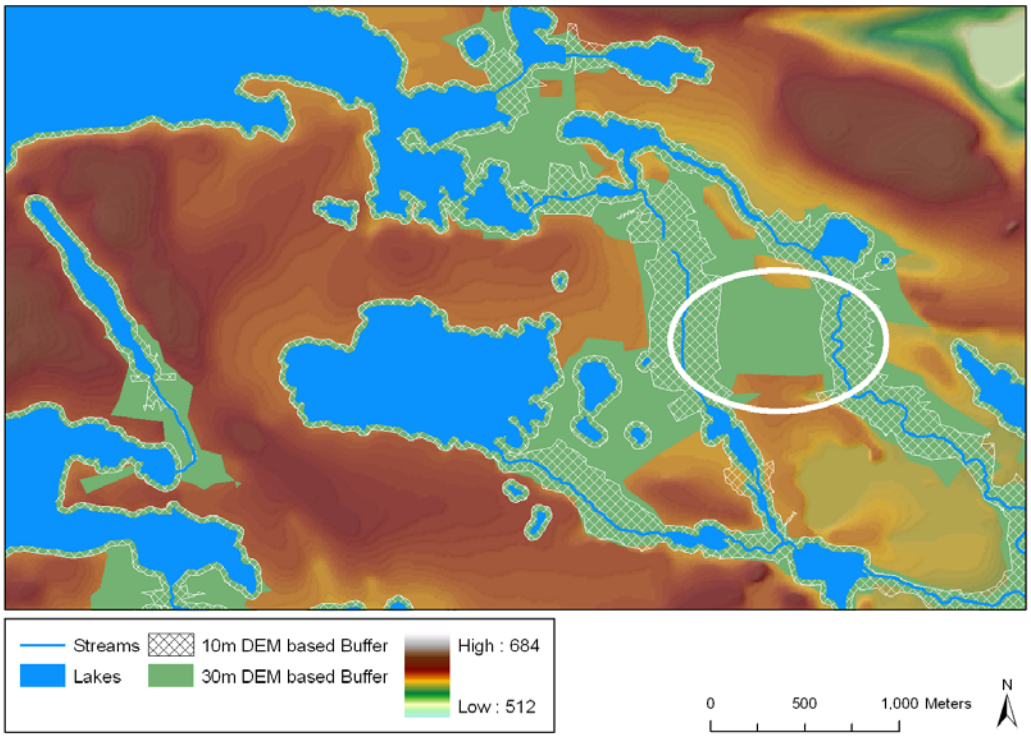

Figure 12: The same site with the 10 m and 30 m variable-width buffers overlaid. Notice the 30 m variable-width buffer did not recognize the elevation change inside the white oval.

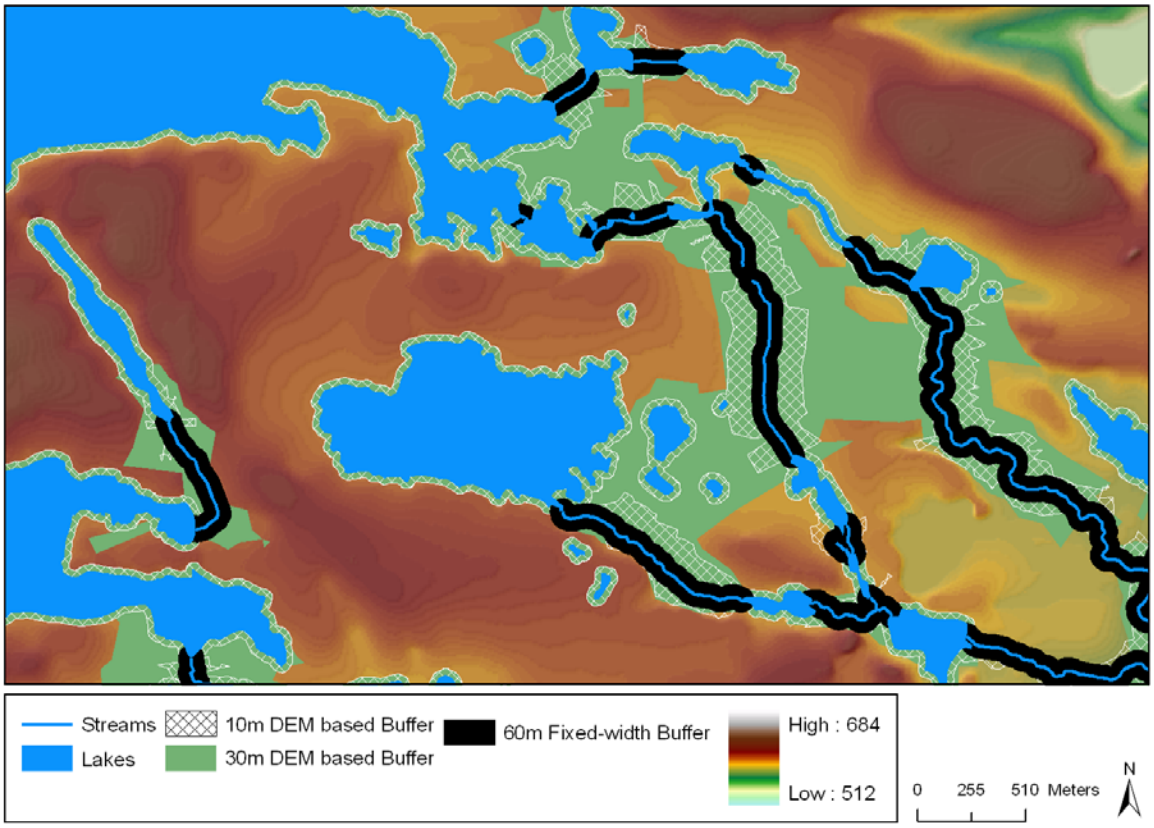

Figure 13: A close-up look at part of the Northern Minnesota study site, comparing the riparian variable-width model 10 m DEM resolution and 30 m DEM resolution results with a fixed-width buffer distance of 60 m. Note the 30 m fixed-width buffer (not shown) would be half as wide as the 60 m fixed-width buffer.

A visual assessment of the study areas was performed comparing the different delineation approaches overlaid on infrared orthophotos. It was apparent that the variable-width buffers followed vegetation and landform changes more closely than the fixed-width buffers, which are both major components of riparian areas. It was also observed that the fixed-width buffers failed to include well known riparian zone features such as ox bow lakes and the land between meandering stream channels, which can be noticed in the lower right hand corner of Figure 14. Also in Figure 15 the variable-width riparian boundary follows the vegetation change and the border where the forest had been clear-

cut. These are significant points because the riparian area should follow a change in vegetation types and density. Ideally, the riparian forest should not be logged, and something stopped the logging from occurring in Figure 15. The boundaries overlaid on the digital orthophotos also shows how complex the level of mapping can be using this model. There are many bends and curves to the boundary, and the model is capable of picking out islands or high spots on the landscape.

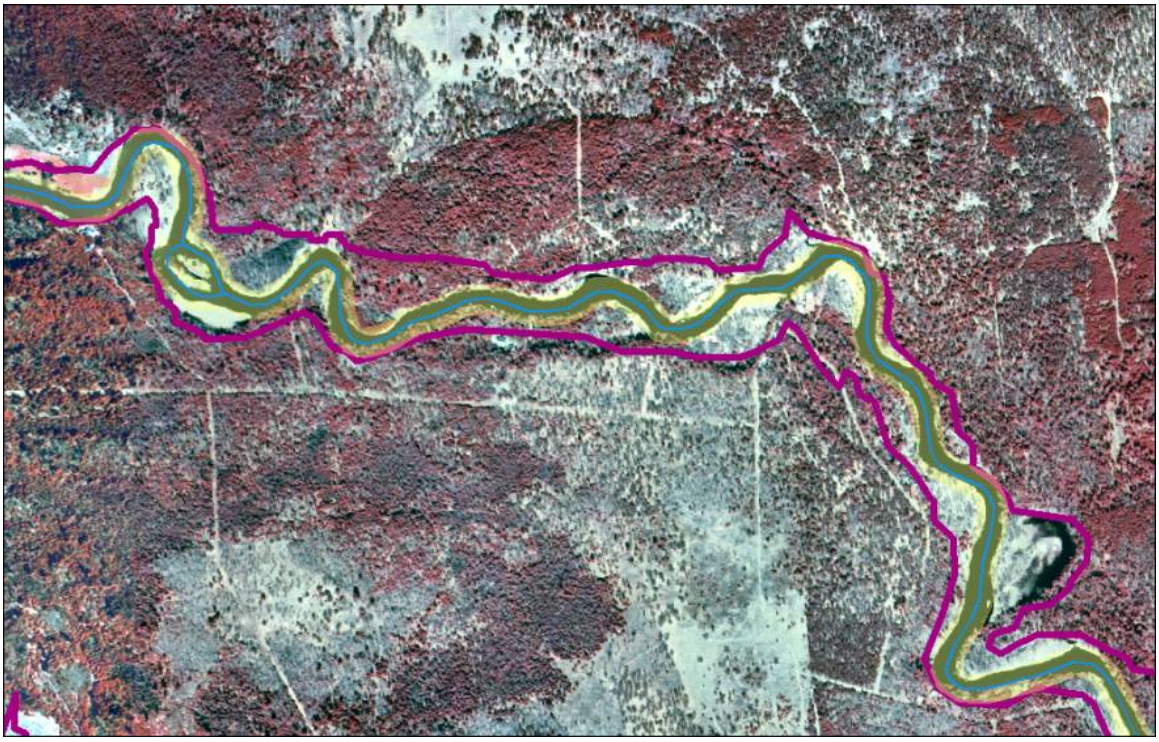

Figure 14: A digital infrared orthophoto close-up look at part of the Lower Peninsula of Michigan study site. The 10 m DEM variable-width riparian boundary is shown in magenta and the 30 m fixed-width buffer is shown in yellow.

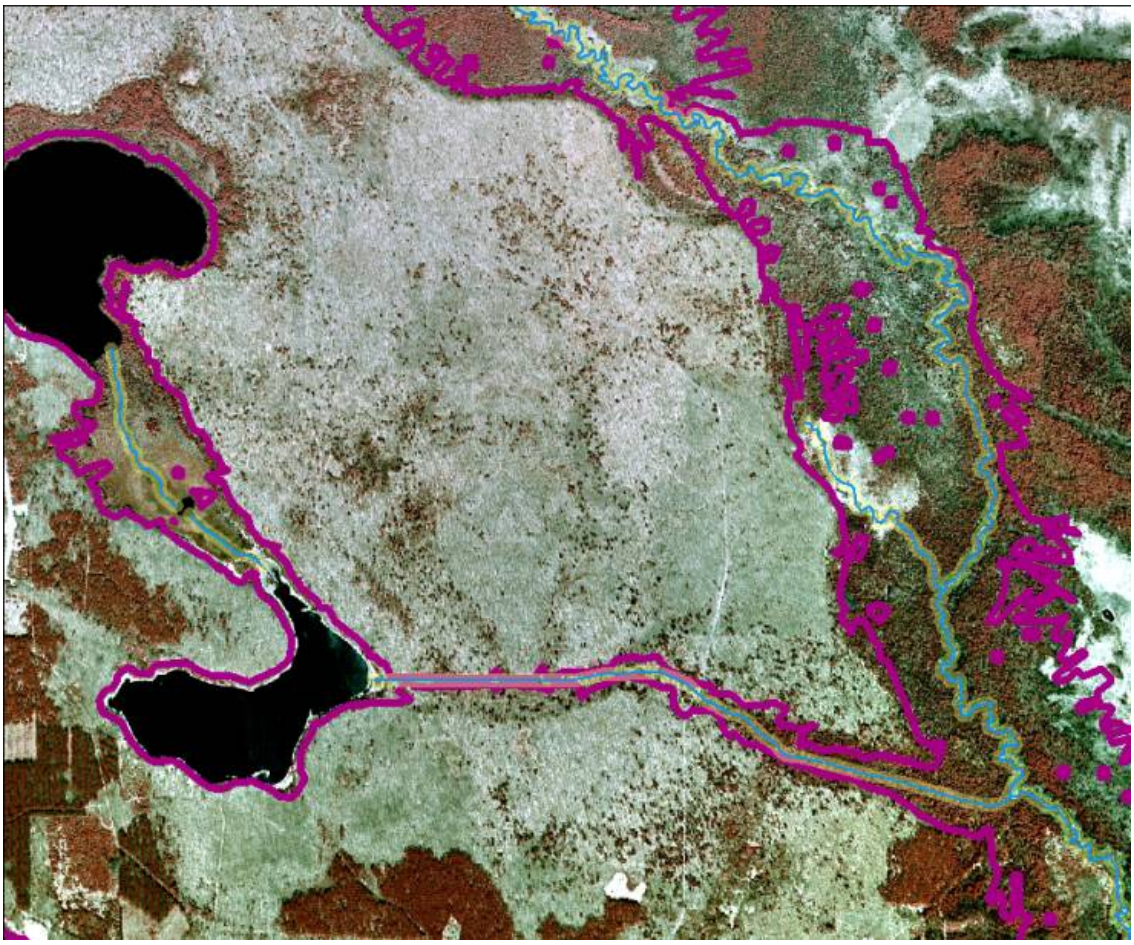

Figure 15: A digital infrared orthophoto close-up look at part of the Upper Peninsula of Michigan study site. The 10 m DEM variable-width riparian boundary is shown in magenta and the 30 m fixed-width buffer is shown in yellow.

Statistical assessment confirmed that the 4 riparian zone boundary delineation methods (10 m DEM variable, 30 m DEM variable, 30 m fixed and 60 m fixed) are significantly different. Using restricted maximum likelihood, during analysis of fixed and mixed effects within the linear-mixed effects model, found the likelihood ratio of the random effect to be significant ($p<0.05$). The statistical analysis was continued with a usual sequential F-test. The results are shown in Table 4. All 3 fixed effects (buffer type, landform and the interaction) were significant. Treatment contrasts were not evaluated using the full model because of the strong interaction, but the main effects were evaluated

seperately. In Table 5, the signifncance to a level of 0.10 is shown between all delineation methods across the 3 landforms.

Table 4: Results from sequential F-test of fixed effects, linear-mixed effects model of buffer delineation type and landform, for all study sites.

	Numerator Degrees of Freedom	Denominator Degrees of Freedom	F-value	p-value
Intercept	1	174	80.9	< 0.0000
Buffer Type	3	174	54.0	< 0.0001
Landform	2	58	7.7	0.0011
Buffer:Landform	6	174	10.0	< 0.0001

Table 5: Statistical significance test results of riparian delineation methods across 3 landforms at a level of 0.10. The delineation methods included: 1) 10 m DEM; 2) 30m DEM; 3) 30m fixed-width; and 4) 60m fixed-width, NS = not significant.

	Minnesota	UP-MI	LP-MI
1 vs. 2	*	*	*
1 vs. 3	*	*	*
1 vs. 4	*	*	NS
2 vs. 3	*	*	*
2 vs. 4	*	*	*
3 vs. 4	*	NS	*

The study also supports the conclusions of Palik and others (2004) that riparian areas determined via fixed width buffers do not accurately delineate riparian areas since they do not incorporate landscape features such as changes in elevation. The 30 and 60 m fixed-width buffers delineated around all streams of the three study areas consistently underestimated the total riparian area, and also did an inadequate job of accurately delineating the spatial location of the boundary. Buffers generated in this manner do not protect enough of the riparian ecotone to maintain natural corridors. The variable-width buffer characterized the stream better by considering the landform change around the stream and protecting that area which highly influenced the stream.

CONCLUSIONS

This study has shown that delineating a variable-width riparian area is possible utilizing current GIS technology coupled with digital elevation and hydrologic data. The modeling was computationally intensive, but can be accomplished within a reasonable amount of time. It is important to remember that the quality and accuracy of the delination is dependent on the quality of the inputs, especially the spatial resolution of the DEM. Other factors that must be considered include the age and quality of the stream digitization and the scale of the vector based stream data, which should be comparable to the spatial resolution of the DEM. The availability, through documentation, update formats and widespread use and knowledge of using the NHD as it is in geodatabase format cannot be discounted, and the quality of the data is consistent over large geographic areas.

Fixed-width riparian area delineation did not adequately map the actual riparian areas in any of the study sites. Our research supports previous work which noted that fixed width delineation does not begin to characterize the water course and its surrounding riparian area. In all cases, this approach consistently underestimates the riparian area. Therefore it is recommended that resource management agencies review their BMPs and consider revising their guidelines.

Careful consideration must also be given to the spatial resolution of the input DEM. While, the variable-width riparian area delineated with the 30m DEM was more accurate than the fixed-width buffers, there were inaccurate boundaries mapped that are directly related to the spatial resolution. The area averaged by the 30m DEM pixels results in a lose of important topographical features that impact boundary delineation. The 10m spatial resolution DEM appears to maintain a significant amount of this needed information.

As land development continues, and water resources become scarcer, it is important that riparian zones are accurately mapped, protected and maintained for future generations. This method of delineating riparian areas is fairly easy to implement and modify.

Plans are to conduct a quanitative accuracy assessment in the near future using digital orthophotos to manually delineate the riparian boundary by an expert aerial photograph interpreter. These delineations will be used as “truth” and comparisions made to the four

delineation methods used in our research. Planned improvements to the model include data smoothing to better approximation the topography and the ability to incorporate addition information into the model such a digital soils and wetlands maps.

LITERATURE CITED

Albert, D.A. 1995. Regional landscape ecosystems of Michigan, Minnesota, and Wisconsin: a working map and classification. Gen. Tech. Rep. NC-178. St. Paul, MN: U.S. Department of Agriculture, Forest Service, North Central Forest Experiment Station. 250 pp.

Aunan, T., B.J. Palik and E.S. Verry. 2005. A GIS approach for delineating variable-width riparian buffers based on hydrological function. Minnesota Forest Resources Council, Research Report 0105.

Bedient, P.B. and W.C. Huber. 2002. *Hydrology and Floodplain Analysis*, 3rd edition. Prentice Hall, NJ. 763 pp.

ESRI ArcDesktop 9.1. 1999-2005. Environmental Systems Research Institute. Redlands, CA.[CD-ROM]

GeoCommunity. 2007. Free GIS data available for download. http://data.geocomm.com/ [last accessed, 3/26/07].

Hanowski, J.M., N. Danz and J. Lind. 2007. Breeding bird response to riparian forest management: 9 years post-harvest. *Forest Ecology and Management* 241:272-277.

Hanowski, J.M., P.T. Wolter and G.J. Niemi. 2000. Effects of riparian buffers on landscape characteristics: implications for breeding birds. Pages 523-528 in Wignington, P.J., Jr. and R.B.L. Beschta (eds.). *Riparian ecology and management in multi-land use watersheds*. Proceedings of the Journal of American Water Resources Association.

Ilhardt, B.L., E.S. Verry and B.J. Palik. 2000. Defining Riparian Areas. Pages 23-45 in Verry, E.S., J.W. Hornbeck and C.A. Dolloff (eds.). *Riparian Management in Forests of the Continental Eastern United States*. Lewis Publishers, New York.

Kutner, M.H., C.J. Nachtsheim, J. Neter and W. Li. 2005. *Applied Linear Statistical Models* (5th ed). McGraw-Hill Irwin, Madison WI. 1396 pp.

Macdonald, S.E., C.J. Burgess, G.J. Scrimgeour, S. Boutin, S. Reedyk and B. Kotak. 2003. Should riparian buffers be part of forest management based on emulation of natural disturbance? *Forest Ecology and Management* 187:185-196.

Mitsch, W.J. and J.G. Gosselink. 2000. *Wetlands*, 3rd Edition. John Wiley & Sons, Inc., New York. 920 pp.

Naiman, R.J. and M.E. McClain. 2005. *Riparia: Ecology, Conservation and Management of Streamside Communities*. Elsevier Academic Press, Burlington, MA. 430 pp.

Palik, B., S.M. Tang and Q. Chavez. 2004. Estimating riparian area extent and land use in the Midwest. General Technical Report NC-248. St. Paul, MN: U.S. Department of Agriculture, Forest Service, North Central Research Station. 28 pp.

Palik, B.J., J. Zasada and C. Hedman. 2000. Ecological considerations for riparian silviculture. Pages 233-254 in Verry, E.S., J.W. Hornbeck and C.A. Dolloff (eds.). *Riparian Management in Forests of the Continental Eastern United States*. Lewis Publishers, New York.

Pinheiro, J.C. and D.M. Bates. 2000. *Mixed-effects models in S and S-Plus*. Springer, New York, NY. 528 pp.

R Development Core Team. 2005. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

Rosgen, D. 1996. *Applied river morphology*. Wildland Hydrology Press, Pagosa Springs, CO. 376 pp.

Skally, C. and E. Sagor. 2001. Comparing riparian management zones to riparian areas in Minnesota : a pilot study. Minnesota Forest Resources Council, Research Report 1001.

United States Geological Survey (USGS). 2007. Real-Time Water Data for the Nation. http://waterdata.usgs.gov/nwis/rt [last accessed, 3/25/07].

United States Geological Survey Geography (USGS). 2007. The National Map: The Nation's Topographic Map for the 21st Century [last accessed, 3/26/07].

United States Geological Survey (USGS). 2005. National Hydrography Dataset (NHD) [computer file]. U.S. Geological Survey, Reston, VA.

United States Geological Survey (USGS). 1997. Standards for digital elevation models, part I. National Mapping Program Technical Instructions, U.S. Department of Interior. 11 pp.

Verry, E.S., C.A. Dolloff, and M.E. Manning. 2004. Riparian ecotone: a functional definition and delineation for resource assessment. *Water, Air, and Soil Pollution: Focus* 4:67-94.

APPENDIX A: Watershed Area Raw Data

Case	WS_ID	Area(m^2)	Buffer	Total_Area(m^2)	Landform
1	1	11577117.81	1	182307825.47	1
2	1	13120879.33	2	182307825.47	1
3	2	5365542.31	1	53666731.57	1
4	2	6768768.11	2	53666731.57	1
5	3	1246273.09	1	16313209.29	1
6	3	1406687.76	2	16313209.29	1
7	4	1763331.28	1	20036217.55	1
8	4	2181869.51	2	20036217.55	1
9	5	1537847.19	1	16666331.52	1
10	5	2190957.64	2	16666331.52	1
11	6	14121713.46	1	91346773.44	2
12	6	18037428.50	2	91346773.44	2
13	7	15606701.11	1	59833885.37	2
14	7	18728504.57	2	59833885.37	2
15	8	6840011.74	1	58323066.40	2
16	8	8080235.70	2	58323066.40	2
17	9	10288765.20	1	44765675.42	2
18	9	12294536.43	2	44765675.42	2
19	10	12706782.27	1	74084500.19	2
20	10	15005112.47	2	74084500.19	2
21	11	19098712.51	1	82840839.57	2
22	11	24088220.03	2	82840839.57	2
23	12	5114797.88	1	35936743.37	2
24	12	7415885.92	2	35936743.37	2
25	13	1136185.01	1	7314039.75	2
26	13	1431111.33	2	7314039.75	2
27	14	8136764.25	1	68664309.17	2
28	14	12321531.40	2	68664309.17	2
29	15	2306844.62	1	22784653.94	2
30	15	3347365.79	2	22784653.94	2
31	16	7045290.11	1	54995629.50	2
32	16	20156284.32	2	54995629.50	2
33	17	13071279.94	1	82039303.33	2
34	17	28409792.80	2	82039303.33	2
35	18	1083257.49	1	12626840.40	2
36	18	1311039.01	2	12626840.40	2
37	19	21166186.13	1	110880304.57	2
38	19	47522730.57	2	110880304.57	2
39	20	20341276.65	1	42451739.65	2
40	20	34860510.74	2	42451739.65	2
41	21	14603714.36	1	71199655.04	2
42	21	25040277.36	2	71199655.04	2

43	22	19362022.12	1	173073449.96	3
44	22	28848387.73	2	173073449.96	3
45	23	18042049.32	1	125432613.70	3
46	23	27581647.32	2	125432613.70	3
47	24	10326681.16	1	99063988.29	3
48	24	18455086.13	2	99063988.29	3
49	25	10026104.29	1	85759131.72	3
50	25	16093262.12	2	85759131.72	3
51	26	1784144.34	1	22488569.84	3
52	26	3360457.43	2	22488569.84	3
53	27	144263.61	1	1230552.04	3
54	27	241437.63	2	1230552.04	3
55	28	1907087.76	1	42970870.16	3
56	28	3226265.69	2	42970870.16	3
57	29	457051.66	1	23861594.05	3
58	29	1839210.57	2	23861594.05	3
59	30	1184202.22	1	18858781.74	3
60	30	2459868.18	2	18858781.74	3
61	31	23173291.37	1	107789770.48	1
62	31	24822298.85	2	107789770.48	1
63	32	17795305.04	1	81730934.17	1
64	32	19631768.04	2	81730934.17	1
65	33	4055317.31	1	37810544.85	1
66	33	4779593.84	2	37810544.85	1
67	34	5605802.69	1	48397611.40	1
68	34	5993054.75	2	48397611.40	1
69	35	2038173.40	1	34922198.60	1
70	35	2824267.23	2	34922198.60	1
71	36	3060645.67	1	61889654.17	1
72	36	4396103.43	2	61889654.17	1
73	37	877052.53	1	16870359.88	1
74	37	1217637.99	2	16870359.88	1
75	38	1745909.51	1	27940010.46	1
76	38	2601465.98	2	27940010.46	1
77	39	1582545.73	1	26158097.76	1
78	39	1844020.87	2	26158097.76	1
79	40	1683006.88	1	35036575.24	1
80	40	1857476.46	2	35036575.24	1
81	41	1526718.95	1	21119252.98	1
82	41	1789757.16	2	21119252.98	1
83	42	2834545.45	1	35414448.01	1
84	42	3477473.62	2	35414448.01	1
85	43	5680158.05	1	44076637.60	1
86	43	7914866.14	2	44076637.60	1
87	44	2290238.97	1	31577873.51	1
88	44	4037985.40	2	31577873.51	1
89	45	12432210.51	1	119392850.23	1

90	45	16352361.88	2	119392850.23	1
91	46	2140403.79	1	20532291.80	1
92	46	2546789.09	2	20532291.80	1
93	47	5322301.76	1	69488336.14	1
94	47	6498111.23	2	69488336.14	1
95	48	7397769.07	1	77002355.22	1
96	48	8869837.74	2	77002355.22	1
97	49	4732674.19	1	48079961.25	1
98	49	5744648.42	2	48079961.25	1
99	50	1922505.67	1	22633927.77	1
100	50	3780796.74	2	22633927.77	1
101	51	12172281.50	1	160241478.26	1
102	51	17194111.95	2	160241478.26	1
103	52	884280.46	1	18357955.92	1
104	52	928802.85	2	18357955.92	1
105	53	880206.75	1	20553347.81	1
106	53	1110173.49	2	20553347.81	1
107	54	5932924.85	1	76790536.63	1
108	54	7127698.10	2	76790536.63	1
109	55	707659.29	1	8753466.57	1
110	55	1049772.48	2	8753466.57	1
111	56	3288132.84	1	27759776.81	1
112	56	4374624.11	2	27759776.81	1
113	57	4863475.98	1	35428084.51	1
114	57	6888537.59	2	35428084.51	1
115	58	503589.22	1	13618516.09	1
116	58	837227.25	2	13618516.09	1
117	59	1419554.49	1	31250427.23	1
118	59	2172536.67	2	31250427.23	1
119	60	1041216.98	1	8707111.16	1
120	60	1723390.67	2	8707111.16	1
121	61	2511221.28	1	28100643.98	1
122	61	4482632.16	2	28100643.98	1
123	1	2182500.48	3	182307825.47	1
124	2	2007163.83	3	53666731.57	1
125	3	276669.80	3	16313209.29	1
126	4	572877.17	3	20036217.55	1
127	5	699535.90	3	16666331.52	1
128	6	1942197.12	3	91346773.44	2
129	7	2344638.20	3	59833885.37	2
130	8	1405257.72	3	58323066.40	2
131	9	1430873.11	3	44765675.42	2
132	10	3349771.51	3	74084500.19	2
133	11	2897194.31	3	82840839.57	2
134	12	1729938.90	3	35936743.37	2
135	13	324048.30	3	7314039.75	2
136	14	2826586.14	3	68664309.17	2

137	15	1211735.01	3	22784653.94	2
138	16	2062924.86	3	54995629.50	2
139	17	3045328.89	3	82039303.33	2
140	18	223177.08	3	12626840.40	2
141	19	4778432.20	3	110880304.57	2
142	20	3705837.85	3	42451739.65	2
143	21	2282709.10	3	71199655.04	2
144	22	8261113.83	3	173073449.96	3
145	23	7389676.70	3	125432613.70	3
146	24	6446726.06	3	99063988.29	3
147	25	3615727.53	3	85759131.72	3
148	26	940082.67	3	22488569.84	3
149	27	134497.83	3	1230552.04	3
150	28	885839.38	3	42970870.16	3
151	29	355307.31	3	23861594.05	3
152	30	763461.68	3	18858781.74	3
153	31	1870166.74	3	107789770.48	1
154	32	2283347.61	3	81730934.17	1
155	33	894157.05	3	37810544.85	1
156	34	1542156.24	3	48397611.40	1
157	35	587607.90	3	34922198.60	1
158	36	602176.95	3	61889654.17	1
159	37	282734.38	3	16870359.88	1
160	38	595073.35	3	27940010.46	1
161	39	190037.64	3	26158097.76	1
162	40	181016.87	3	35036575.24	1
163	41	280586.88	3	21119252.98	1
164	42	484214.22	3	35414448.01	1
165	43	1307375.75	3	44076637.60	1
166	44	1043734.98	3	31577873.51	1
167	45	2972761.58	3	119392850.23	1
168	46	773433.98	3	20532291.80	1
169	47	1746422.93	3	69488336.14	1
170	48	243647.42	3	77002355.22	1
171	49	1753979.31	3	48079961.25	1
172	50	659873.87	3	22633927.77	1
173	51	4512113.43	3	160241478.26	1
174	52	365618.67	3	18357955.92	1
175	53	250950.87	3	20553347.81	1
176	54	1576777.77	3	76790536.63	1
177	55	337300.58	3	8753466.57	1
178	56	1535016.55	3	27759776.81	1
179	57	2082673.24	3	35428084.51	1
180	58	455505.79	3	13618516.09	1
181	59	747523.29	3	31250427.23	1
182	60	762829.65	3	8707111.16	1
183	61	1782500.74	3	28100643.98	1

184	1	4550381.21	4	182307825.47	1
185	2	4135627.01	4	53666731.57	1
186	3	574590.88	4	16313209.29	1
187	4	1175671.93	4	20036217.55	1
188	5	1383685.53	4	16666331.52	1
189	6	3794125.96	4	91346773.44	2
190	7	4491345.39	4	59833885.37	2
191	8	2791784.71	4	58323066.40	2
192	9	2767127.71	4	44765675.42	2
193	10	6498092.70	4	74084500.19	2
194	11	5473519.25	4	82840839.57	2
195	12	3354792.87	4	35936743.37	2
196	13	662409.34	4	7314039.75	2
197	14	5617201.50	4	68664309.17	2
198	15	2443797.01	4	22784653.94	2
199	16	4103565.48	4	54995629.50	2
200	17	6007815.07	4	82039303.33	2
201	18	446531.48	4	12626840.40	2
202	19	9132529.53	4	110880304.57	2
203	20	6823311.05	4	42451739.65	2
204	21	4278927.81	4	71199655.04	2
205	22	16354626.18	4	173073449.96	3
206	23	14695980.27	4	125432613.70	3
207	24	12718613.95	4	99063988.29	3
208	25	7170229.96	4	85759131.72	3
209	26	1845085.86	4	22488569.84	3
210	27	253942.92	4	1230552.04	3
211	28	1761872.08	4	42970870.16	3
212	29	708727.92	4	23861594.05	3
213	30	1480262.92	4	18858781.74	3
214	31	3760957.05	4	107789770.48	1
215	32	4599571.47	4	81730934.17	1
216	33	1784468.53	4	37810544.85	1
217	34	3027954.55	4	48397611.40	1
218	35	1192107.49	4	34922198.60	1
219	36	1250780.73	4	61889654.17	1
220	37	576544.95	4	16870359.88	1
221	38	1234118.14	4	27940010.46	1
222	39	402737.57	4	26158097.76	1
223	40	379668.71	4	35036575.24	1
224	41	584696.82	4	21119252.98	1
225	42	1010100.59	4	35414448.01	1
226	43	2698408.37	4	44076637.60	1
227	44	2171395.82	4	31577873.51	1
228	45	6211552.90	4	119392850.23	1
229	46	1569591.01	4	20532291.80	1
230	47	3569914.21	4	69488336.14	1

231	48	4974130.67	4	77002355.22	1
232	49	3547567.24	4	48079961.25	1
233	50	1377905.48	4	22633927.77	1
234	51	9226058.31	4	160241478.26	1
235	52	730897.18	4	18357955.92	1
236	53	513630.21	4	20553347.81	1
237	54	3203594.76	4	76790536.63	1
238	55	664621.18	4	8753466.57	1
239	56	3036122.91	4	27759776.81	1
240	57	4116633.40	4	35428084.51	1
241	58	905167.93	4	13618516.09	1
242	59	1518358.08	4	31250427.23	1
243	60	1523086.92	4	8707111.16	1
244	61	3543998.63	4	28100643.98	1

APPENDIX B: R-Script for Statistical Analysis of Watershed Area Data

```
#####
# Riparian experiment
#
# Robert Froese; froese@mtu.edu
# Created: November 14, 2006; Modified: March 9, 2007
#####

require(nlme)
require(gmodels)

# Processing the raw data
ld <- read.csv("RipAnalysis_V3.csv")
names(ld) <- c("Case","WS","Area","DEM","Landform","TArea")
ld$WS <- as.factor(ld$WS)
ld$DEM <- as.factor(ld$DEM)
ld$Landform <- as.factor(ld$Landform)
ld$AreaPCT <- ld$Area/ld$TArea
ld$TY <- "Variable"
ld$TY[ld$DEM==3|ld$DEM==4] <- "Fixed"
ld$TY <- as.factor(ld$TY)

# Graphical Summaries
plot.design(ld[,3:5])
interaction.plot(ld$TY,ld$Landform,ld$Area)
pdf(file="interact.pdf",height=5,width=6)
interaction.plot(ld$DEM,ld$Landform,ld$Area/10000,xlab="Delineation
Method",ylab="Mean Riparian Area (ha)",legend=F)
legend(3,1500,legend=c("MI-LP","MI-UP","MN"),lty=c(1,2,3))
dev.off()
# fit the mixed-effects model
fm <- lme(Area~DEM*Landform,random=~1|WS,data=ld)
anova(fm) # strong interaction

# since there's a strong interaction effect, we compare main effects separately
# DEM 1
i <- ld$DEM==1
fm <- lme(Area~Landform,random=~1|WS,data=ld[i,])
anova(fm)
estimable(fm,c(0,1,0)) # 1 vs 2
estimable(fm,c(0,0,1)) # 1 vs 3
estimable(fm,c(0,-1,1))# 2 vs 3
```

```
# DEM 2
i <- ld$DEM==2
fm <- lme(Area~Landform,random=~1|WS,data=ld[i,])
anova(fm)
estimable(fm,c(0,1,0)) # 1 vs 2
estimable(fm,c(0,0,1)) # 1 vs 3
estimable(fm,c(0,-1,1))# 2 vs 3

# DEM 3
i <- ld$DEM==3
fm <- lme(Area~Landform,random=~1|WS,data=ld[i,])
anova(fm)
estimable(fm,c(0,1,0)) # 1 vs 2
estimable(fm,c(0,0,1)) # 1 vs 3
estimable(fm,c(0,-1,1))# 2 vs 3

 # DEM 4
i <- ld$DEM==4
fm <- lme(Area~Landform,random=~1|WS,data=ld[i,])
anova(fm)
estimable(fm,c(0,1,0)) # 1 vs 2
estimable(fm,c(0,0,1)) # 1 vs 3
estimable(fm,c(0,-1,1))# 2 vs 3

# LF 1
i <- ld$Landform==1
fm <- lme(Area~DEM,random=~1|WS,data=ld[i,])
anova(fm)
estimable(fm,c(0,1,0,0)) # 1 vs 2
estimable(fm,c(0,0,1,0)) # 1 vs 3
estimable(fm,c(0,0,0,1)) # 1 vs 4
estimable(fm,c(0,-1,1,0))# 2 vs 3
estimable(fm,c(0,-1,0,1))# 2 vs 4
estimable(fm,c(0,0,-1,1))# 3 vs 4

# LF 2
i <- ld$Landform==2
fm <- lme(Area~DEM,random=~1|WS,data=ld[i,])
anova(fm)
estimable(fm,c(0,1,0,0)) # 1 vs 2
estimable(fm,c(0,0,1,0)) # 1 vs 3
estimable(fm,c(0,0,0,1)) # 1 vs 4
estimable(fm,c(0,-1,1,0))# 2 vs 3
estimable(fm,c(0,-1,0,1))# 2 vs 4
estimable(fm,c(0,0,-1,1))# 3 vs 4
```

```
# LF 3
i <- ld$Landform==3
fm <- lme(Area~DEM,random=~1|WS,data=ld[i,])
anova(fm)
estimable(fm,c(0,1,0,0)) # 1 vs 2
estimable(fm,c(0,0,1,0)) # 1 vs 3
estimable(fm,c(0,0,0,1)) # 1 vs 4
estimable(fm,c(0,-1,1,0))# 2 vs 3
estimable(fm,c(0,-1,0,1))# 2 vs 4
estimable(fm,c(0,0,-1,1))# 3 vs 4


# testing some assumptions
# parallelism

i <- ld$DEM==2
pd <-
data.frame(cbind(ld$Area[ld$DEM==1],ld$Area[ld$DEM==2],ld$Area[ld$DEM==3],ld
$Area[ld$DEM==4]))
names(pd) <- c("DEM1","DEM2","DEM3","DEM4")
plot(0,0,ylim=c(min(pd),max(pd)),xlim=c(1,4),type="n",xlab="Delineation
Method",ylab="Area")
for (i in 1:nrow(pd)) {
  lines(c(1,2,3,4),pd[i,])
}  # no "serious" departure from parallelism

# test for normality
qqnorm(residuals(fm))
# nothing too nasty

# test for within-subject variance homogeneity
plot(ld$TArea,residuals(fm))
# I argue that because residuals are basically symmetric
# around zero they suggest homogeneous within-subject var.

#####
# END
#####
```

APPENDIX C: Python Code for Variable-Width Delineation Riparian Model

```
#############################################################################
# Program: Module 1 – Stream Buffer (streambuffer.py)                       #
# Purpose: Create 100' buffer around lakes layer; clip streams with lakes layer.  #
# Inputs: streams, lakes                                                    #
# Outputs: lakebuffer, clipped streams (streamsnol)                         #
# Author: James C. Rivard, Michigan Technological University; jcrivard@mtu.edu  #
# Date: 9/19/2006                                                           #
#############################################################################

import win32com.client, sys, traceback
import math, os, string
gp = win32com.client.Dispatch("esriGeoprocessing.gpDispatch.1")
def delfile(delfile):
    outtempstream = outpath + "\\" + delfile
    if gp.Exists(outtempstream):
       gp.RefreshCatalog(outpath)
       gp.AddMessage("Deleting file " + delfile + "......\n")
       gp.delete(outtempstream)
def AddPrintMessage(msg, severity):
    print msg
    if severity == 0: gp.AddMessage(msg)
    elif severity == 1: gp.AddWarning(msg)
    elif severity == 2: gp.AddError(msg)
try:
  streamlayer  = sys.argv[1]
  lakeslayer = sys.argv[2]
  template = sys.argv[1]
  streamsnol = sys.argv[3]
  lakebuffer = sys.argv[4]
  tempstream = "tempstream"
  desc = gp.Describe
  template = sys.argv[1]
  fullpath = desc(streamlayer).CatalogPath
  outpath = (os.path.split(fullpath)[0])
  gp.Workspace = outpath
  ########################
  # Delete output files  #
  ########################
  delfile(streamsnol)
  delfile(lakebuffer)
  delfile(tempstream)
  fclist = streamlayer
  outSR = gp.CreateSpatialReference("#", streamlayer, "#", "#", "#",       fclist, 100)
  ############################################################
  #  Create a 100' (30.48 meters) buffer around the lakes.
  ############################################################
  gp.toolbox = "analysis"
```

```
 gp.AddMessage("Creating lake buffer......\n")
 gp.buffer(lakeslayer,"lakebuffer",30.48)
 ###############################################################
 #Create a feature class containing the stream data without the portion
 #  of the stream running through the lakes.
 ###############################################################
 gp.AddMessage("Dissolving Streams......\n")
 gp.Dissolve_management(streamlayer,tempstream,"REACHCODE;ORDER_")
 gp.AddMessage("Creating stream layer without lakes......\n")
 gp.erase(tempstream,lakeslayer,streamsnol)
 #End of Program

except:
    tb = sys.exc_info()[2]
    tbinfo = traceback.format_tb(tb)[0]
    pymsg = "PYTHON ERRORS:\nTraceback Info:\n" + tbinfo + "\nError Info:\n    " + \
            str(sys.exc_type)+ ": " + str(sys.exc_value) + "\n"
    AddPrintMessage(pymsg, 2)

    msgs = "GP ERRORS:\n" + gp.GetMessages(2) + "\n"
    AddPrintMessage(msgs, 2)

#############################################################################
# Program: Module 2 – Sample Point Generation (samplepoints.py)             #
# Purpose: Create sample points along stream segments at a distance         #
#          determined by input variables.  Obtain elevation for             #
#          each sample point from the DEM input.                            #
# Inputs: streamsnol (output from clipstreams), DEM                         #
#         pixel size, pixel ratio (distance between points)                 #
# Outputs:sample_points_elev                                                #
# Author: James C. Rivard, Michigan Technological University; jcrivard@mtu.edu #
# Date: 9/19/2006                                                           #
#############################################################################

#import relevant modules, create geoprocessing dispatch object
import win32com.client, sys, traceback
import math, os, string
gp = win32com.client.Dispatch("esriGeoprocessing.gpDispatch.1")
##########################################
## subroutine to delete files
##########################################
def delfile(delfile):
    outtempstream = outpath + "\\" + delfile
    if gp.Exists(outtempstream):
       gp.RefreshCatalog(outpath)
       gp.AddMessage("Deleting file " + delfile + "......\n")
       gp.delete(outtempstream)
##########################################
## subroutine to insert sample points into output.
##########################################
def insertsamplepoint(xi,yi,azimuthi,orderi,slopei,streamnumi,streampnti):
```

```
    pnti.x = xi
    pnti.y = yi
    feati.azimuth = float(azimuthi)
    feati.ORDER_ = orderi
    feati.slope = slopei
    feati.Shape = pnti
    feati.streamnum = streamnumi
    feati.streampnt = streampnti
    curi.InsertRow(feati)
##########################################
## subroutine to generate error msgs
##########################################
def AddPrintMessage(msg, severity):
    print msg
    if severity == 0: gp.AddMessage(msg)
    elif severity == 1: gp.AddWarning(msg)
    elif severity == 2: gp.AddError(msg)
##########################################
## subroutine to calculate distance between 2 points.
##########################################
def CartesianDist(x1,y1,x2,y2):
    cdist = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
    return cdist
##########################################
## subroutine to calculate azimuth between 2 points.
##########################################
def Azimuth(x1,y1,x2,y2):
    if ((x1 == x2) and (y1 < y2)):
       Azimuth = 0
    elif ((x1 == x2) and (y1 > y2)):
       Azimuth = 180
    elif ((y1 == y2) and (x1 < x2)):
       Azimuth = 90
    elif ((y1 == y2) and (x1 > x2)):
      Azimuth = 270
    elif ((x1 < x2) and (y1 < y2)):
      Azimuth = 180 * ((math.atan(math.fabs(x1 - x2) / math.fabs(y1 - y2))) / 3.14159)
    elif ((x1 < x2) and (y1 > y2)):
      Azimuth = 180 * ((math.atan(math.fabs(y1 - y2) / math.fabs(x1 - x2))) / 3.14159) + 90
    elif ((x1 > x2) and (y1 > y2)):
      Azimuth = 180 * ((math.atan(math.fabs(x1 - x2) / math.fabs(y1 - y2))) / 3.14159) + 180
    elif ((x1 > x2) and (y1 < y2)):
      Azimuth = 180 * ((math.atan(math.fabs(y1 - y2) / math.fabs(x1 - x2))) / 3.14159) + 270
    return Azimuth

try:
    gp.AddMessage("Starting program \n")

     ##########################################
    ## Constants and data clean-up
    ##########################################
```

```
infc = str(sys.argv[1])
indem = str(sys.argv[2])
pixelsize = long(sys.argv[3])
pixelratio = float(sys.argv[4])
                # constant that determines distance between sample points
sample_points = "sample_points"
sample_points_elev = sys.argv[5]
temp_points_elev = "temp_points_elev"
pointdist = pixelratio * pixelsize
template = sys.argv[1]
desc2 = gp.describe
desc = gp.describe(infc)
fullpath = desc2(infc).CatalogPath
outpath = (os.path.split(fullpath)[0])
gp.Workspace = outpath
 ########################################
#  Cleanup old output files
########################################
delfile(temp_points_elev)
delfile(sample_points_elev)
delfile("sample_points_elevc")
gp.Workspace = outpath
outlayer = outpath + "\\" + sample_points
delfile(sample_points)
########################################
#  Create output sample point set
########################################
#gp.AddMessage("Output file:" + outpath + " " + sample_points + " " + template + "\n")
fclist = infc
gp.Addmessage("infc=" + infc + "\n")
outSR = gp.CreateSpatialReference("#", infc, "#", "#", "#", fclist, 100)
gp.CreateFeatureclass(outpath,
sample_points,"POINT",template,"SAME_AS_TEMPLATE","SAME_AS_TEMPLATE",outSR)
gp.addfield(sample_points, "azimuth", "FLOAT", "6")
gp.addfield(sample_points, "slope", "FLOAT", "6")
gp.addfield(sample_points, "streamnum", "LONG")
gp.addfield(sample_points, "streampnt", "LONG")
gp.AddMessage("Created Output Layer:" + sample_points + "\n")
 ########################################################
# Generate sample points by following stream segments
########################################################
#rows = gp.searchcursor(infc,"[OBJECTID_1]>0 AND [OBJECTID_1] < 30")
rows = gp.searchcursor(infc)
row = rows.next()
rowcount = 0
curi = gp.InsertCursor(outlayer)
pnti = gp.CreateObject("point")
feati = curi.NewRow()
streamnum = 0              #stream segment number for correcting elevations
while row:
   streampnt = 0           #sample point num on stream segment
```

```
        streamnum = streamnum + 1
        #get the geometry of the feature in question.
        feature = row.GetValue(desc.ShapeFieldName)
        #if the feature class is of type polyline
        if desc.ShapeType.lower() == "polyline":
            rowcount = rowcount + 1
            partcount = feature.partcount
            neworder = row.GetValue("ORDER_")   #save for output to sample points
            gp.AddMessage("Processing Next stream segement:" + str(rowcount) + "\n")
            #gp.AddMessage("OBJECTID,ORDER,NUMPART=" +
str(row.GetValue("OBJECTID_12")) + ", " + str(neworder) + ", " + str(partcount) + "\n")
            partnumber = 0
            #cycle through the parts
            while partnumber < partcount:
              #gp.AddMessage("New partnumber=" + str(partnumber) + "\n")
              part = feature.getpart(partnumber)
              pnt = part.next()
              pointnumber = 0
              #cycle through the points in each part
              leftdist = 0
                                  #leftdist is distance left over between line segments
              while pnt:
                  lastx = pnt.x
                  lasty = pnt.y
                  #Get the next point in the part
                  pnt = part.next()
                  if pnt:
                     dist = CartesianDist(lastx, lasty, pnt.x, pnt.y)
                     azim = Azimuth(lastx, lasty, pnt.x, pnt.y)
                    ################################################
                    ##  Calculate the sample point locations
                   ################################################
                     totdist = dist      # total length of segment
                     numsamppoints = 1
                                                #number of sample points on line segment
                                        newpointratio = ((pointdist*numsamppoints)-
                                        leftdist)/totdist
                                        #pointdist is a constant based on pixel size
                     while (newpointratio <= 1):
                       #we have enough distance to get a point in
                       newx = ((pnt.x - lastx)*newpointratio) + lastx
                       newy = ((pnt.y - lasty)*newpointratio) + lasty
                       run = pnt.x - lastx
                       if run == 0:
                          run = .00000000000000000001
                       slope = (pnt.y - lasty)/run
                                         #gp.Addmessage("(x1,y1),(x2,y2)=" + str(lastx)   + ","
                                        + str(lasty) + "::" + str(pnt.x) + "," + str(pnt.y) + "\n")
                                        #gp.AddMessage("newx,newy=" + str(newx) + "," +
                                        str(newy) + "\n")
                      streampnt = streampnt + 1
```

```
                                        insertsamplepoint(newx,newy,azim,neworder,slope,strea
                                        mnum,streampnt)
                    numsamppoints = numsamppoints + 1
                                        newpointratio = ((pointdist*numsamppoints)-
                                        leftdist)/totdist
                                            #pointdist - constant based on pixel size
                                    leftdist = totdist - ((numsamppoints-1)*pointdist) + leftdist
                                            #use on next line segment
                        pointnumber += 1
                    #end of while pnt
                    partnumber = partnumber + 1
        else:
              gp.AddMessage("Input file must be of type polyline.\n")
        row = rows.next()
        if streampnt == 0:          #stream segment was too short to get a point in
          streamnum = streamnum - 1
         #gp.AddMessage("Streamnum/numpoints=" + str(streamnum) + "/" + str(streampnt) + "\n")
AddPrintMessage("\n", 0)
#########################################################
#  End of Sample point generation section
#########################################################
#  Now get the elevation of the sample points
#########################################################
   gp.AddMessage("Getting elevation of sample points....\n")
   gp.CheckOutExtension("Spatial")
   gp.ExtractValuesToPoints(sample_points, indem, sample_points_elev,      "NONE",
"VALUE_ONLY")
#########################################################
# Now correct elevations of sample points that do not consistently rise or fall when going down
# the stream.
#########################################################
   gp.AddMessage("Creating feature layer to sort sample points for elevation correction...\n")
   gp.MakeFeatureLayer(sample_points_elev,temp_points_elev,"[STREAMNUM]>0 ORDER
BY [STREAMNUM],[STREAMPNT]")
   rows = gp.searchcursor(temp_points_elev)
   row = rows.next()
   hstreamnum = row.GetValue("STREAMNUM")
   totstreampnts = 0.000000
   totelev = 0.0000000
   aList = []          #Array to hold average stream elevations - indexed by streamnum
   aTuple = (0,.0000001)
   avgelev = 0.0000001
   gp.AddMessage("Getting Average elevation of steam segments...\n")
   while row:
    if hstreamnum != row.GetValue("STREAMNUM"):
      avgelev = float(totelev)/float(totstreampnts)
      #gp.AddMessage("Avgelev=" + str(avgelev) + "\n")
      aTuple = (hstreamnum,avgelev)
      #aList.insert(hstreamnum-1,aTuple)
      aList.append(aTuple)
```

```
        hstreamnum = row.GetValue("STREAMNUM")
        totstreampnts = 0
        totelev = 0
      totstreampnts = totstreampnts + 1
      totelev = totelev + row.GetValue("RASTERVALU")
      row = rows.next()
    avgelev = totelev/totstreampnts          #Add the average elev for last stream segment
    aTuple = (hstreamnum,avgelev)
    aList.append(aTuple)
    #aList.insert(hstreamnum-1,aTuple)
#######################################################
# Now loop through the temp_points_elev file and update the elevations that are in error.
# Note: By using the in memory layer for work, the
# update cursor actually updates the disk copy that is behind the memory layer or view.
#######################################################
    gp.AddMessage("Correcting anomalies in stream elevations.....\n")
    rows = gp.updatecursor(temp_points_elev)
    row = rows.next()
    hstreamnum = 0
    while row:
      thiselev = row.GetValue("RASTERVALU")
      if hstreamnum != row.GetValue("STREAMNUM"):
        hstreamnum = row.GetValue("STREAMNUM")
        helev = thiselev
        #gp.AddMessage("hstreamnum=" + str(hstreamnum) + "; helev=" + str(helev) + "\n")
        #gp.AddMessage("alist(0)=" + str(aList[hstreamnum-1][0]) + "alist(1)=" +
str(aList[hstreamnum-1][1]) + "\n")
        avgelev = aList[hstreamnum-1][1]
        if thiselev > avgelev:    # Descending stream segment
          streamdir = 0
        elif thiselev < avgelev:  # Ascending stream segment
          streamdir = 1
        else:                 # Flat stream segment - should not be any problems with these since avg
would not be 0
          streamdir = 2
      if (streamdir == 0) and (thiselev > helev):
        #gp.AddMessage("Bad elevation for stream:" + str(hstreamnum) + "; point:" +
str(row.GetValue("STREAMPNT")) + "\n")
        thiselev = helev       # Use elevation from previous point
        row.rastervalu = thiselev
        rows.updaterow(row)
      elif (streamdir == 1) and (thiselev < helev):
        #gp.AddMessage("Bad elevation for stream:" + str(hstreamnum) + "; point:" +
str(row.GetValue("STREAMPNT")) + "\n")
        thiselev = helev       # Use elevation from previous point
        row.rastervalu = thiselev
        rows.updaterow(row)
      elif (streamdir == 2) and (thiselev != helev):
        #gp.AddMessage("Bad elevation for stream:" + str(hstreamnum) + "; point:" +
str(row.GetValue("STREAMPNT")) + "\n")
        thiselev = helev       # Use elevation from previous point
```

```
        row.rastervalu = thiselev
        rows.updaterow(row)
      helev = thiselev          # update hold elevation variable
      row = rows.next()
    del row
    del rows

except:
    tb = sys.exc_info()[2]
    tbinfo = traceback.format_tb(tb)[0]
    pymsg = "PYTHON ERRORS:\nTraceback Info:\n" + tbinfo + "\nError Info:\n    " + \
            str(sys.exc_type)+ ": " + str(sys.exc_value) + "\n"
    AddPrintMessage(pymsg, 2)

    msgs = "GP ERRORS:\n" + gp.GetMessages(2) + "\n"
    AddPrintMessage(msgs, 2)

# ---------------------------------------------------------------------------
# Program: Module 3 – Transects (transects.py)                                #
# Purpose: Create transects at 90 degrees from sample points                  #
#          produced in samplepoints script.                                   #
# Inputs: sample_points_elev (output from samplepoints),                      #
#         pixel size, pixel ratio (distance between points)                   #
#         numpixels (how long is the transect)                                #
# Outputs:transects                                                           #
# Author: James C. Rivard, Michigan Technological University; jcrivard@mtu.edu #
# Date: 9/19/2006                                                             #
# ---------------------------------------------------------------------------

#import relevant modules, create geoprocessing dispatch object
import win32com.client, sys, traceback
import math, os, string
gp = win32com.client.Dispatch("esriGeoprocessing.gpDispatch.1")
##########################################
## subroutine to delete files
##########################################
def delfile(delfile):
    outtempstream = outpath + "\\" + delfile
    if gp.Exists(outtempstream):
       gp.RefreshCatalog(outpath)
       gp.AddMessage("Deleting file " + delfile + "......\n")
       gp.delete(outtempstream)
#################################################################
##  This routine will create transect points at angles of
##  +/- 90 degrees of the sample point based on slope.
#################################################################
def gettransectpoint(xi,yi,slopei,totdisti,streamlevi,orderi,trani,pointi,streamnumi,streampnti):
    if (slopei == 0):
      slopei = .000000000001
    perpslope = -1/slopei  #slope of line perpendicular to stream
    angle = math.atan(perpslope)
```

```
    newx = xi + (totdisti * math.cos(angle))   #create point at +90 degrees to sample point
    newy = yi + (totdisti * math.sin(angle))
    pnti.x = newx
    pnti.y = newy
    feati.ORDER_ = orderi
    feati.STREAMNUM = streamnumi
    feati.SLOPE = slopei
    feati.streamx = xi
    feati.streamy = yi
    feati.stream_ele = streamlevi
    feati.Shape = pnti
    feati.transect_n = trani              #transect number
    feati.point_num = pointi              #point number on transect
    feati.transect_l = 1                  #which side of stream
    curi.InsertRow(feati)
    newx = xi - (totdisti * math.cos(angle))  #create point at -90 degrees to sample point
    newy = yi - (totdisti * math.sin(angle))
    pnti.x = newx
    pnti.y = newy
    feati.Shape = pnti
    feati.transect_l = 2                  #which side of stream
    curi.InsertRow(feati)
    if (streampnti == 1):                 #create a cone around beginning of stream segment
      extrapnts = 10
      i = 0
      while (i <= extrapnts):
        p1angle = math.atan(slopei) + (360/extrapnts)*i
        newx = xi + (totdisti * math.cos(p1angle))
        newy = yi + (totdisti * math.sin(p1angle))
        pnti.x = newx
        pnti.y = newy
        feati.Shape = pnti
        feati.transect_n = i + 1              #transect number
        feati.transect_l = 3                  #which side of stream - somewhat arbitrary in this case
        curi.InsertRow(feati)
        newx = xi - (totdisti * math.cos(p1angle))
        newy = yi - (totdisti * math.sin(p1angle))
        pnti.x = newx
        pnti.y = newy
        feati.Shape = pnti
        feati.transect_l = 4                  #which side of stream - somewhat arbitrary in this case
        curi.InsertRow(feati)
        i = i + 1
      #end of while
    #end of if
##########################################
## subroutine to generate error msgs
##########################################
def AddPrintMessage(msg, severity):
    print msg
    if severity == 0: gp.AddMessage(msg)
```

```
    elif severity == 1: gp.AddWarning(msg)
    elif severity == 2: gp.AddError(msg)

try:
    gp.AddMessage("Starting program \n")
    gp.AddMessage("Checking out spatial analyst extension...\n")
    gp.CheckOutExtension("Spatial")
    ###########################################
    ## Constants - get this values from input or determine for final program
    ###########################################
    infc = str(sys.argv[1])
    pixelsize = long(sys.argv[2])
    pixelratio = float(sys.argv[3])        # constant that determines distance between sample points
    numpixels = long(sys.argv[4])          #number of pixels to go out on transect
    startpoint = long(sys.argv[5])
    endpoint = long(sys.argv[6])
    transects = str(sys.argv[7])
    ###########################################
    #Get Inputs (feature class and feature id)
    ###########################################
    #infc = gp.GetParameterAsText(0)
    #startpoint = gp.GetParameterAsText(1)
    #endpoint = gp.GetParameterAsText(2)
    #indem = gp.GetParameterAsText(3)
    ########################################
    #  Cleanup old output files
    ########################################
    template = infc
    desc2 = gp.describe
    desc = gp.describe(infc)
    fullpath = desc2(infc).CatalogPath
    outpath = (os.path.split(fullpath)[0])
    gp.Workspace = outpath
    outlayer = outpath + "\\" + transects
    delfile(transects)
    ########################################
    #  Create output transect set
    ########################################
    gp.AddMessage("Output file:" + outpath + " " + transects + " " + template + "\n")
    fclist = infc
    gp.Addmessage("infc=" + infc + "\n")
    outSR = gp.CreateSpatialReference("#", infc, "#", "#", "#", fclist, 100)
    gp.CreateFeatureclass(outpath, transects,"POINT","#","#","#",outSR)
    gp.addfield(transects, "ORDER_", "FLOAT", "6")
    gp.addfield(transects, "stream_ele", "FLOAT", "6")
    gp.addfield(transects, "streamx", "DOUBLE", "12", "3")        #x coordinate of stream
    gp.addfield(transects, "streamy", "DOUBLE", "12", "3")        #y coordinate of stream
    gp.addfield(transects, "SLOPE", "FLOAT", "6")          #carry through for short transects
    gp.addfield(transects, "transect_n", "LONG")           #identifies transect
    gp.addfield(transects, "point_num", "LONG")            #identifies which point in transect
    gp.addfield(transects, "transect_l", "LONG")          #identifies which side of stream
```

```
   gp.addfield(transects, "in_buffer", "LONG")             #flag for in the buffer or not
   gp.addfield(transects, "streamnum", "LONG")             #for use later on
   gp.AddMessage("Created Output Layer:" + transects + "\n")
   ##########################################################
   # Generate transects for desired sample points
   ##########################################################
   query = "[OBJECTID_1] >= " + str(startpoint) + " AND [OBJECTID_1] <= " + str(endpoint)
   gp.AddMessage("Generating transects, please be patient.....\n")
   transectnum = 0           #this field will track the transect number later used in determining
buffer
   rows = gp.searchcursor(infc,query)
   pnti = gp.CreateObject("point")
   curi = gp.InsertCursor(outlayer)
   feati = curi.NewRow()
   row = rows.next()
   while row:
      #get the geometry of the feature in question.
      feature = row.GetValue(desc.ShapeFieldName)
      if desc.ShapeType.lower() == "point":
            pnt = feature.getpart(1)
            neworder = row.GetValue("ORDER_")   #save for output to sample points
            newstreamnum = row.GetValue("STREAMNUM")   #save for output to sample points
            newstreampnt = row.GetValue("STREAMPNT")   #which point on the stream segment
            newstream_elev = row.GetValue("RASTERVALU")
            new_slope = row.GetValue("SLOPE")   #slope of line passing through stream
            trandist = numpixels * pixelsize          #length of transect
            tranpointdist = pixelratio * pixelsize     #distance between points on transect
            totdist = 0                          #distance used so far
            transectnum = transectnum + 1             #which transect are we on
            point_num = 0                        #which point on the transect
            if (transectnum % 100 == 0):
              gp.AddMessage("Transect Number: " + str(transectnum) + " generated.....\n")
#update user with progress
            while (totdist <= trandist):
              totdist = totdist + tranpointdist
              point_num = point_num + 1

gettransectpoint(pnt.x,pnt.y,new_slope,totdist,newstream_elev,neworder,transectnum,point_num,
newstreamnum,newstreampnt)   #create transect point
            #end of while
      else:
            gp.AddMessage("Input file must be of type point.\n")
      row = rows.next()
   AddPrintMessage("\n", 0)
   del row
   del rows
   del curi
   del pnti
   del feati
```

```
    ######################################################
    #  End of Sample point generation section
    ######################################################
    gp.AddMessage("Program complete.\n")
except:
    tb = sys.exc_info()[2]
    tbinfo = traceback.format_tb(tb)[0]
    pymsg = "PYTHON ERRORS:\nTraceback Info:\n" + tbinfo + "\nError Info:\n    " + \
            str(sys.exc_type)+ ": " + str(sys.exc_value) + "\n"
    AddPrintMessage(pymsg, 2)

    msgs = "GP ERRORS:\n" + gp.GetMessages(2) + "\n"
    AddPrintMessage(msgs, 2)

# ---------------------------------------------------------------------------
# Program: Module 4 – Transects Last Part (transectslast.py)                  #
# Purpose: Determine if transect points are in or out of the buffer           #
# Inputs: sample_points_elev (output from samplepoints),                      #
#         transects, starttransect, 1st increment, increment, dem             #
# Outputs:transexxxx, transxxxx (completed transects and short trans.         #
# Author: James C. Rivard, Michigan technological University; jcrivard@mtu.edu #
# Date: 9/19/2006                                                             #
# ---------------------------------------------------------------------------
#import relevant modules, create geoprocessing dispatch object
import win32com.client, sys, traceback
import math, os, string
gp = win32com.client.Dispatch("esriGeoprocessing.gpDispatch.1")
#################################################################
##  NOTE:  Per ESRI support web site, try changing the JET database max records to calculate
##         from the default of 9500 to something like 10 million.
##         This is done via the Arcmap advanced utilities setup program under the editor sections.
##         Doing this may eliminate the need to process the data in chunks to avoid
##         extractvaluesbypoints from locking up.
#################################################################
##  This routine will insert points once the position in the buffer has been determined.
#################################################################
def
updatetransectpoint(xi,yi,transectl,transectn,pointnum,rastervalu,streamele,inbuffer,streamnum):
    pnti.x = xi
    pnti.y = yi
    feati.transect_l = transectl
    feati.stream_ele = streamele
    feati.Shape = pnti
    feati.transect_n = transectn            #transect number
    feati.point_num = pointnum              #point number on transect
    feati.in_buffer = inbuffer
    feati.streamnum = streamnum
    if rastervalu > 0:
      feati.rastervalu = rastervalu
    curi.InsertRow(feati)
```

```
########################################
## subroutine to delete files
########################################
def delfile(delfile):
   outtempstream = outpath + "\\" + delfile
   if gp.Exists(outtempstream):
     gp.RefreshCatalog(outpath)
     gp.AddMessage("Deleting file " + delfile + "......\n")
     gp.delete(outtempstream)
########################################
## subroutine to generate error msgs
########################################
def AddPrintMessage(msg, severity):
   print msg
   if severity == 0: gp.AddMessage(msg)
   elif severity == 1: gp.AddWarning(msg)
   elif severity == 2: gp.AddError(msg)

try:
   ###############################################
   #Get Inputs and set constants
   ###############################################
   infc = str(sys.argv[1])         #should be sample_points_elev
   transects = str(sys.argv[2])
   tranmin = long(sys.argv[3])
   tranmax = long(sys.argv[4])
   tranincrement = long(sys.argv[5])
   indem = str(sys.argv[6])
   tottran = gp.GetCount(infc)
   gp.AddMessage("Starting program \n")
   gp.AddMessage("Checking out spatial analyst extension...\n")
   gp.CheckOutExtension("Spatial")
   gp.AddMessage("Total Number of Transects = " + str(tottran) + "\n")
   template = infc
   desc2 = gp.describe
   desc = gp.describe(infc)
   fullpath = desc2(infc).CatalogPath
   outpath = (os.path.split(fullpath)[0])
   gp.Workspace = outpath
   elevlimit = 1                    #elevation above stream level to draw buffer at
   transects_efl = "transects_efl"
   transects_elevt = "transects_elevt"
   trandone = 0
   while (trandone == 0):
    ###############################################
    ## Constants - get this values from input or determine for final program
    ###############################################
    transects_elev = "trane" + str(tranmax)
    transects_short = "trans" + str(tranmax)
    transectt = "trant"
```

```
    ########################################
    #  Cleanup old output files
    ########################################
    gp.AddMessage("Compacting Geodatabase....\n")
    #gp.Compact(outpath)
    delfile(transects_short)
    delfile(transects_elev)
    delfile(transectt)
    delfile(transects_elevt)
    if gp.Exists(transects_efl):
      gp.RefreshCatalog(outpath)
      gp.AddMessage("Deleting file " + transects_efl + "......\n")
      gp.delete(transects_efl)
    outlayer = outpath + "\\" + transects

     #query = "[transect_n]>=1 AND [transect_n]<=200"
    query = "[TRANSECT_N]>=" + str(tranmin) + " AND [TRANSECT_N]<=" + str(tranmax)
    gp.AddMessage(str(query) + "; " + str(transects) + "; " + str(transectt) + "\n")
    gp.AddMessage("Selecting transects: " + str(tranmin) + " - " + str(tranmax) + "\n")
    gp.Select(transects,transectt,query)
    #gp.SelectLayerByAttribute(transects,"NEW_SELECTION",query)

    ########################################################
    #  Now get the elevation of the transect points and
    #  create a feature layer to sort the data - note
    #  this method of sorting is not supported, but works.
    #  ArcGIS 9.2 is supposed to support ORDER BY on
    #  SQL SELECTS, but this is quicker than reading all
    #  recs into memory and then sorting.
    ########################################################
    gp.AddMessage("Getting elevation of transect points.....\n")
    gp.ExtractValuesToPoints(transectt, indem, transects_elevt, "NONE", "VALUE_ONLY")
    gp.AddMessage("Creating fudged feature layer to sort transect elevation data...\n")
    gp.MakeFeatureLayer(transects_elevt,transects_efl,"[TRANSECT_L]>0 ORDER BY
[STREAMNUM],[TRANSECT_L],[TRANSECT_N],[POINT_NUM]")

    ########################################################################
    # Delete the transect_elev feature class now that it's in the feature layer - transects_efl
    ########################################################################
    delfile(transects_elev)
    outlayer = outpath + "\\" + transects
    ########################################################################
    #  Create output transect set (again) to be filled by updated list
    ########################################################################
    gp.AddMessage("Recreating transect_elev file....\n")
    fclist = infc
    outSR = gp.CreateSpatialReference("#", infc, "#", "#", "#", fclist, 100)
    gp.CreateFeatureclass(outpath, transects_elev,"POINT","#","#","#",outSR)
    gp.addfield(transects_elev, "rastervalu", "FLOAT", "6")
    gp.addfield(transects_elev, "transect_n", "LONG")          #identifies transect
```

```
    gp.addfield(transects_elev, "point_num", "LONG")              #identifies which point in
transect
    gp.addfield(transects_elev, "transect_l", "LONG")            #identifies which side of stream
    gp.addfield(transects_elev, "in_buffer", "LONG")             #flag for in the buffer or not
    gp.addfield(transects_elev, "stream_ele", "FLOAT", "6")
    gp.addfield(transects_elev, "streamnum", "LONG")
    gp.AddMessage("Created Output Layer:" + transects_elev + "\n")

    ######################################################################
    #  Create output fc to hold sample point nums for which transects were not long enough
    ######################################################################
    gp.AddMessage("Creating fc for short transects....\n")
    fclist = infc
    outSR = gp.CreateSpatialReference("#", infc, "#", "#", "#", fclist, 100)
    gp.CreateFeatureclass(outpath, transects_short,"POINT","#","#","#",outSR)
    gp.addfield(transects_short, "RASTERVALU", "FLOAT", "6")   #stream elevation
    gp.addfield(transects_short, "ORDER_", "LONG")             #stream order
    gp.addfield(transects_short, "OBJECTID_1", "LONG")         #point num
    gp.addfield(transects_short, "SLOPE", "FLOAT", "6")        #slope of line at stream
    gp.addfield(transects_short, "TRANSECT_L", "LONG")         #transect leg
    gp.addfield(transects_short, "streamnum", "LONG")        #stream segment
    gp.AddMessage("Created Output Layer:" +  transects_short + "\n")
    ######################################################################
    # Loop through the points in the list and determine if in buffer
    # Write output to feature class as we go along
    ######################################################################
    ##aTuple =
(row.GetValue("TRANSECT_L"),row.GetValue("TRANSECT_N"),row.GetValue("POINT_NU
M"),row.GetValue("RASTERVALU"),row.GetValue("STREAM_ELE"),pnt.x,pnt.y,row.GetVal
ue("STREAMX"),row.GetValue("STREAMY"),row.GetValue("SLOPE"))
    rows = gp.searchcursor(transects_efl)
    row = rows.next()

    gp.AddMessage("Determining buffer position...\n")
    htrans = row.GetValue("TRANSECT_N")
    #htrans = 1
    leg1done = "No"
    totshort = 0
    stop_elev = row.GetValue("STREAM_ELE") + elevlimit
    #stop_elev = 1
    outlayer = outpath + "\\" + transects_elev
    curi = gp.InsertCursor(outlayer)
    pnti = gp.CreateObject("point")
    feati = curi.NewRow()
    outlayersh = outpath + "\\" + transects_short
    cursh = gp.InsertCursor(outlayersh)
    featsh = cursh.NewRow()
    pntsh = gp.CreateObject("point")
    lastpointinbuffer = 0
    while row:
      if (row.GetValue("POINT_NUM") == 1):
```

```
        ru = -1                 #previous point elevation - need to initialize for each transect
      if htrans != row.GetValue("TRANSECT_N"):
        runagain = 2                                        #default value
        if (leg1done == "No"):                                  #never found an end to buffer
          gp.AddMessage("Transect: " + str(htrans) + " was not long enough.\n")
          totshort = totshort + 1
          pntsh.x = sx
          pntsh.y = sy
          featsh.SLOPE = sl
          featsh.Shape = pntsh
          featsh.OBJECTID_1 = totshort
          featsh.RASTERVALU = se
          featsh.TRANSECT_L = tl
          featsh.STREAMNUM = sn
          cursh.InsertRow(featsh)
          #gp.AddMessage("Point Inserted to short transect file\n")
        stop_elev = row.GetValue("STREAM_ELE") + elevlimit
        htrans = row.GetValue("TRANSECT_N")
        leg1done = "No"
        lastpointinbuffer = 0
        if (row.GetValue("POINT_NUM") != 1):                                    #ignore out of
sequence points
          leg1done = "Yes"
          gp.AddMessage("Out of order point found on transect: " +
str(row.GetValue("TRANSECT_L")) + ", " + str(row.GetValue("TRANSECT_N")) + ";  point
will be disregarded.\n")
      if (row.GetValue("RASTERVALU") > stop_elev) and (row.GetValue("POINT_NUM") ==
1):     # 1st point on transect is > elevation limit; include in buffer
        leg1done = "No"
        inbuffer = 1
      elif ((row.GetValue("RASTERVALU") > stop_elev) or (row.GetValue("RASTERVALU")
< ru)):      #point is higher than limiting elevation OR We have jumped into the next watershed
(over the hill)
        leg1done = "Yes"
        inbuffer = 3                #3 indicates last "in" point on transect;  add point with data from
previous record
        if lastpointinbuffer == 0:
          updatetransectpoint(pnt.x,pnt.y,tl,tn,pn,ru,se,inbuffer,sn)
          lastpointinbuffer = 1          # flag field
        inbuffer = 2
      elif leg1done == "No":            #point is in buffer so update record
        inbuffer = 1
      else:
        inbuffer = 2                                    #rest of points outside buffer
      feature = row.shape
      #if row.GetValue("TRANSECT_L") > 2:
       #gp.AddMessage("Leg1done=" + str(leg1done) + "\n")
       #gp.AddMessage("Leg=" + str(tl) + "; inbuffer=" + str(inbuffer) + "\n")
       #gp.AddMessage("ru" + str(ru) + "; rastervalu=" + str(row.GetValue("RASTERVALU"))
+ "\n")
        #gp.AddMessage("stopelev=" + str(stop_elev) + "\n")
```

```
        pnt = feature.getpart(1)
        tl = row.GetValue("TRANSECT_L")
        tn = row.GetValue("TRANSECT_N")
        pn = row.GetValue("POINT_NUM")
        ru = row.GetValue("RASTERVALU")
        se = row.GetValue("STREAM_ELE")
        sn = row.GetValue("STREAMNUM")
        sx = row.GetValue("STREAMX")
        sy = row.GetValue("STREAMY")
        sl = row.GetValue("SLOPE")
        updatetransectpoint(pnt.x,pnt.y,tl,tn,pn,ru,se,inbuffer,sn)
        row = rows.next()
      #end of while
      gp.AddMessage("Phase complete.\n")
      ################################################
      ## update counters for next set of transects
      ################################################
      del pnt
      del feature
      del row
      del rows
      del curi
      del pnti
      del feati
      del pntsh
      del featsh
      del cursh
      tranmin = tranmax + 1
      if tranmin > tottran:
        trandone = 1
      tranmax = tranmin + tranincrement
      if tranmax > tottran:
        tranmax = tottran
      #### End of while
    ##end of try
except:
    tb = sys.exc_info()[2]
    tbinfo = traceback.format_tb(tb)[0]
    pymsg = "PYTHON ERRORS:\nTraceback Info:\n" + tbinfo + "\nError Info:\n    " + \
            str(sys.exc_type)+ ": " + str(sys.exc_value) + "\n"
    AddPrintMessage(pymsg, 2)
    msgs = "GP ERRORS:\n" + gp.GetMessages(2) + "\n"
    AddPrintMessage(msgs, 2)
#############################################################################
# Program:Module 5 – Reprocess (reprocess.py)                               #
# Purpose: Create longer transects for ones that were not long enough to start.  #
# Inputs: dem,any point feature class,pixelsize,pixelratio, startpixel, numpixels  #
# Outputs:transects_reprocess                                               #
# Author: James C. Rivard, Michigan Technological University; jcrivard@mtu.edu  #
# Date: 9/19/2006                                                           #
#############################################################################
```

```
#import relevant modules, create geoprocessing dispatch object
import win32com.client, sys, traceback
import math, os, string
gp = win32com.client.Dispatch("esriGeoprocessing.gpDispatch.1")
#################################################################
##  This program will reprocess the short transects out to a max distance set by numpixels.
##  This routine will create transect points at angles of
##  +/- 90 degrees of the sample point based on slope.
#################################################################
def gettransectpoint(xi,yi,slopei,totdisti,streamlevi,orderi,trani,pointi,streamnumi,new_legi):
   if (slopei == 0):
     slopei = .000000000001
   perpslope = -1/slopei  #slope of line perpendicular to stream
   angle = math.atan(perpslope)
   if new_legi == 1:                        #first leg of transect
     newx = xi + (totdisti * math.cos(angle))   #create point at +90 degrees to sample point
     newy = yi + (totdisti * math.sin(angle))
   else:                                    #2nd leg of transect
     newx = xi - (totdisti * math.cos(angle))  #create point at -90 degrees to sample point
     newy = yi - (totdisti * math.sin(angle))
   pnti.x = newx
   pnti.y = newy
   feati.ORDER_ = orderi
   feati.STREAMNUM = streamnumi
   feati.SLOPE = slopei
   feati.streamx = xi
   feati.streamy = yi
   feati.stream_ele = streamlevi
   feati.Shape = pnti
   feati.transect_n = trani                 #transect number
   feati.point_num = pointi                   #point number on transect
   feati.transect_l = new_legi              #which side of stream
   curi.InsertRow(feati)
   #end of subroutine

#########################################
## subroutine to delete files
#########################################
def delfile(delfile):
   outtempstream = outpath + "\\" + delfile
   if gp.Exists(outtempstream):
     gp.RefreshCatalog(outpath)
     gp.AddMessage("Deleting file " + delfile + "......\n")
     gp.delete(outtempstream)
#########################################
## subroutine to generate error msgs
#########################################
def AddPrintMessage(msg, severity):
   print msg
   if severity == 0: gp.AddMessage(msg)
   elif severity == 1: gp.AddWarning(msg)
```

```
    elif severity == 2: gp.AddError(msg)

try:
    gp.AddMessage("Starting program \n")
    gp.AddMessage("Checking out spatial analyst extension...\n")
    gp.CheckOutExtension("Spatial")
    ##############################################
    ## Constants - get this values from input or determine for final program
    ##############################################
    indem = str(sys.argv[1])
    anyfc = str(sys.argv[2])       # to get working geodatabase
    pixelsize = long(sys.argv[3])
    numpixels = long(sys.argv[4])                    #number of pixels to go out on transect
    startpixel = long(sys.argv[5])
    pixelratio = float(sys.argv[6])
    transects = str(sys.argv[7])      #output file
    outfc = "tran_short_merged"       #working file
    pointdist = pixelratio * pixelsize
    elevlimit = 1                     #elevation above stream level to draw buffer at

    ##############################################
    #Data Clean-up
    ##############################################
     indem = gp.GetParameterAsText(0)
    anyfc = gp.GetParameterAsText(1)       # to get working geodatabase
    desc2 = gp.describe
    desc = gp.describe(anyfc)
    fullpath = desc2(anyfc).CatalogPath
    outpath = (os.path.split(fullpath)[0])
    outlayer = outpath + "\\" + outfc
    gp.Workspace = outpath
    gp.AddMessage("Workspace=:" + str(outpath) + "\n")
    delfile(outfc)
    delfile(transects)

    ##############################################
    # Merge the short transect files into 1 file
    ##############################################
    outSR = gp.CreateSpatialReference("#", anyfc, "#", "#", "#", anyfc, 100)
    gp.CreateFeatureclass(outpath, outfc,"POINT","#","#","#",outSR)
    gp.addfield(outfc, "RASTERVALU", "FLOAT", "6")
    gp.addfield(outfc, "ORDER_", "LONG")
    gp.addfield(outfc, "OBJECTID_1","LONG")         #x coordinate of stream
    gp.addfield(outfc, "SLOPE", "FLOAT", "6")          #carry through for short transects
    gp.addfield(outfc, "transect_l", "LONG")           #identifies which side of stream
    gp.addfield(outfc, "streamnum", "LONG")            #for use later on
    listofinputs = ""
    i = 0
    while i < 10:
      loopvar = "trans" + str(i) + "*"
      fcs = gp.listfeatureclasses(loopvar, "Point")
```

```
    fcs.reset()
    fc = fcs.next()
    while fc:
      if listofinputs == "":          # 1st iteration - don't add semicolon
        listofinputs = str(fc)
      else:
        listofinputs=listofinputs + ";" + str(fc)
      fc=fcs.next()
    i = i + 1
  gp.AddMessage("FC=" + listofinputs + "\n")
  gp.Append_management(listofinputs,outfc,"TEST")
  #gp.Append("trans17202;trans200;trans25703",outfc)
  infc = outfc                  # so that we don't have to change the code much
  template = infc
  outlayer = outpath + "\\" + transects
  ########################################
  #  Create output transect set
  ########################################
  gp.AddMessage("Output file:" + outpath + " " + transects + " " + template + "\n")
  fclist = infc
  gp.Addmessage("infc=" + infc + "\n")
  outSR = gp.CreateSpatialReference("#", infc, "#", "#", "#", fclist, 100)
  gp.CreateFeatureclass(outpath, transects,"POINT","#","#","#",outSR)
  gp.addfield(transects, "ORDER_", "FLOAT", "6")
  gp.addfield(transects, "stream_ele", "FLOAT", "6")
  gp.addfield(transects, "streamx", "DOUBLE", "12", "3")        #x coordinate of stream
  gp.addfield(transects, "streamy", "DOUBLE", "12", "3")        #y coordinate of stream
  gp.addfield(transects, "SLOPE", "FLOAT", "6")          #carry through for short transects
  gp.addfield(transects, "transect_n", "LONG")           #identifies transect
  gp.addfield(transects, "point_num", "LONG")             #identifies which point in transect
  gp.addfield(transects, "transect_l", "LONG")           #identifies which side of stream
  gp.addfield(transects, "in_buffer", "LONG")            #flag for in the buffer or not
  gp.addfield(transects, "streamnum", "LONG")             #for use later on
  gp.AddMessage("Created Output Layer:" + transects + "\n")

  ########################################################
  # generate additional transect points
  ########################################################
  #query = "[OBJECTID_1] >= " + startpoint + " AND [OBJECTID_1] <= " + endpoint
  query = "[OBJECTID_1] >=0 AND [TRANSECT_L] <3"       #don't reprocess stream end
point transects
  #gp.AddMessage("Query=" + query + "\n")
  tottranr = gp.GetCount(infc)
  gp.AddMessage("Total Records to Process = " + str(tottranr) + "\n")
  gp.AddMessage("Generating transects, please be patient.....\n")
  transectnum = 0          #this field will track the transect number later used in determining
buffer
  rows = gp.searchcursor(infc,query)
  pnti = gp.CreateObject("point")
  curi = gp.InsertCursor(outlayer)
  feati = curi.NewRow()
```

```
   row = rows.next()
   while row:
      #gp.AddMessage("new row\n")
      #get the geometry of the feature in question.
      feature = row.GetValue(desc.ShapeFieldName)
      #gp.AddMessage("Feature type=" + str(desc.ShapeType.lower()) + "\n")
      #if the feature class is of type point
      if desc.ShapeType.lower() == "point":
            pnt = feature.getpart(1)
            #pnt = part.next()
            neworder = row.GetValue("ORDER_")   #save for output to sample points
            newstreamnum = row.GetValue("STREAMNUM")   #save for output to sample points
            newstream_elev = row.GetValue("RASTERVALU")
            new_slope = row.GetValue("SLOPE")   #slope of line passing through stream
            new_leg = row.GetValue("TRANSECT_L")   #leg of transect we are reprocessing
            #gp.AddMessage("Point(x,y)=" + str(pnt.x) + ", " + str(pnt.y) + "\n")
            trandist = numpixels * pixelsize          #length of transect
            tranpointdist = pixelratio * pixelsize     #distance between points on transect
            totdist = startpixel * pixelsize          #distance used so far
            transectnum = transectnum + 1             #which transect are we on
            point_num = 0                          #which point on the transect
            if (transectnum % 100 == 0):
              gp.AddMessage("Reprocessed transect Number: " + str(transectnum) + "
generated.....\n")   #update user with progress
            while (totdist <= trandist):
              totdist = totdist + tranpointdist
              point_num = point_num + 1

gettransectpoint(pnt.x,pnt.y,new_slope,totdist,newstream_elev,neworder,transectnum,point_num,
newstreamnum,new_leg)   #create transect point
              #gp.AddMessage("New Point = " + str(newpnt.x) + ", " + str(newpnt.y) + "\n")
            #end of while
      else:
            gp.AddMessage("Input file must be of type point.\n")
      row = rows.next()
   AddPrintMessage("\n", 0)
   del row
   del rows
   del curi
   del pnti
   del feati
   ######################################################
   #  End of reprocessing transects
  ######################################################
    gp.AddMessage("Program complete.\n")

except:
   tb = sys.exc_info()[2]
   tbinfo = traceback.format_tb(tb)[0]
   pymsg = "PYTHON ERRORS:\nTraceback Info:\n" + tbinfo + "\nError Info:\n    " + \
         str(sys.exc_type)+ ": " + str(sys.exc_value) + "\n"
```

```
    AddPrintMessage(pymsg, 2)

    msgs = "GP ERRORS:\n" + gp.GetMessages(2) + "\n"
    AddPrintMessage(msgs, 2)

#############################################################################
# Program: Module 6 – Transects Reprocess Part 2 (reprocess_part2.py)       #
# Purpose: Determine is reprocessed transect points are in the buffer.  Similar #
#                          to transects last part                           #
# Inputs: transects_reprocess,dem                                           #
# Outputs:tranr_merged                                                      #
# Author: James C. Rivard, Michigan Technological University; jcrivard@mtu.edu #
# Date: 9/19/2006                                                           #
#############################################################################
#import relevant modules, create geoprocessing dispatch object
import win32com.client, sys, traceback
import math, os, string
gp = win32com.client.Dispatch("esriGeoprocessing.gpDispatch.1")
#################################################################
##  This routine will insert a point into the output file
##  after the buffer position is determined.
#################################################################
def
updatetransectpoint(xi,yi,transectl,transectn,pointnum,rastervalu,streamele,inbuffer,streamnum):
    pnti.x = xi
    pnti.y = yi
    feati.transect_l = transectl
    feati.stream_ele = streamele
    feati.Shape = pnti
    feati.transect_n = transectn            #transect number
    feati.point_num = pointnum              #point number on transect
    feati.in_buffer = inbuffer
    feati.streamnum = streamnum
    if rastervalu > 0:
      feati.rastervalu = rastervalu
    curi.InsertRow(feati)

########################################
## subroutine to delete files
########################################
def delfile(delfile):
    outtempstream = outpath + "\\" + delfile
    if gp.Exists(outtempstream):
      gp.RefreshCatalog(outpath)
      gp.AddMessage("Deleting file " + delfile + "......\n")
      gp.delete(outtempstream)

########################################
## subroutine to generate error msgs
########################################
def AddPrintMessage(msg, severity):
```

```
    print msg
    if severity == 0: gp.AddMessage(msg)
    elif severity == 1: gp.AddWarning(msg)
    elif severity == 2: gp.AddError(msg)

try:
    ###########################################
    #Get Inputs and housekeeping
    ###########################################
    infc = str(sys.argv[1])     #This should be tran_short_merged
    anyfc = str(sys.argv[1])
    indem = str(sys.argv[2])
    transects = str(sys.argv[3]) #should be transects_reprocess
    outfc = str(sys.argv[4])     #should be tran_r_merged
    transects_efl = "transects_efl" #working file
    transects_elevt = "transects_elevt" #working file
    transectt = "trant"
    elevlimit = 1                       #elevation above stream level to draw buffer at
    tottran = gp.GetCount(infc)
    gp.AddMessage("Starting program \n")
    gp.AddMessage("Checking out spatial analyst extension...\n")
    gp.CheckOutExtension("Spatial")
    gp.AddMessage("Total Number of Transects = " + str(tottran) + "\n")
    template = infc
    desc2 = gp.describe
    desc = gp.describe(infc)
    fullpath = desc2(infc).CatalogPath
    outpath = (os.path.split(fullpath)[0])
    gp.Workspace = outpath
    trandone = 0
    tranmin = 1
    tranmax = 200
    increment = 8500
    while (trandone == 0):

      ###########################################
      ## build filename for working files
      ###########################################
      transects_elev = "tranr_" + str(tranmax)
      ########################################
      #  Cleanup old output files
      ########################################
      gp.AddMessage("Compacting Geodatabase....\n")
      #gp.Compact(outpath)
      delfile(transects_elev)
      delfile(transects_efl)
      outlayer = transects_efl
      if gp.Exists(outlayer):
        gp.RefreshCatalog(outpath)
        gp.AddMessage("Temp output Layer " + transects_efl + " already exists, deleting.....\n")
        gp.delete(outlayer)
```

```
    delfile(transectt)
    delfile(transects_elevt)
    outlayer = outpath + "\\" + transects

    ####################################################
    ##  Select the data to process
    ##  extractvaluestopoints does not work with
    ##  large datasets in ArcGIS 9.1.
    ####################################################
    query = "[TRANSECT_N]>=" + str(tranmin) + " AND [TRANSECT_N]<=" + str(tranmax)
    gp.AddMessage(str(query) + "; " + str(transects) + "; " + str(transectt) + "\n")
    gp.AddMessage("Selecting transects: " + str(tranmin) + " - " + str(tranmax) + "\n")
    gp.Select(transects,transectt,query)
    #gp.SelectLayerByAttribute(transects,"NEW_SELECTION",query)

    ####################################################
    #  Now get the elevation of the transect points
    ####################################################
    gp.AddMessage("Getting elevation of transect points.....\n")
    gp.ExtractValuesToPoints(transectt, indem, transects_elevt, "NONE", "VALUE_ONLY")
    gp.AddMessage("Creating fudged feature layer to sort transect elevation data...\n")
    gp.MakeFeatureLayer(transects_elevt,transects_efl,"[TRANSECT_L]>0 ORDER BY
[TRANSECT_L],[TRANSECT_N],[POINT_NUM]")
    #gp.SaveToLayerFile(transects_efl,"test_efl.lyr")

    ##########################################################################
    ### Delete the transect_elev feature class now that it's in the feature layer - transects_efl
    ##########################################################################
    delfile(transects_elev)
    outlayer = outpath + "\\" + transects
    ##########################################################################
    #  Create output transect set (again) to be filled by updated list
    ##########################################################################
    gp.AddMessage("Recreating transect_elev file....\n")
    fclist = infc
    outSR = gp.CreateSpatialReference("#", infc, "#", "#", "#", fclist, 100)
    gp.CreateFeatureclass(outpath, transects_elev,"POINT","#","#","#",outSR)
    gp.addfield(transects_elev, "rastervalu", "FLOAT", "6")
    gp.addfield(transects_elev, "transect_n", "LONG")          #identifies transect
    gp.addfield(transects_elev, "point_num", "LONG")           #identifies which point in
transect
    gp.addfield(transects_elev, "transect_l", "LONG")          #identifies which side of stream
    gp.addfield(transects_elev, "in_buffer", "LONG")           #flag for in the buffer or not
    gp.addfield(transects_elev, "stream_ele", "FLOAT", "6")
    gp.addfield(transects_elev, "streamnum", "LONG")
    gp.AddMessage("Created Output Layer:" + transects_elev + "\n")

    ##########################################################################
    ### Loop through the points in the list and determine if in buffer
    ### Write output to feature class as we go along
    ##########################################################################
```

```
    rows = gp.searchcursor(transects_efl)
    row = rows.next()
    gp.AddMessage("Determining buffer position....\n")
    htrans = row.GetValue("TRANSECT_N")
    leg1done = "No"
    totshort = 0
    stop_elev = row.GetValue("STREAM_ELE") + elevlimit
    outlayer = outpath + "\\" + transects_elev
    curi = gp.InsertCursor(outlayer)
    pnti = gp.CreateObject("point")
    feati = curi.NewRow()
    lastpointinbuffer = 0
    while row:
      if (row.GetValue("POINT_NUM") == 1):
        ru = -1                 #previous point elevation - need to initialize for each transect
        if (htrans % 100 == 0):
          gp.AddMessage("At transect: " + str(htrans) + "\n")
      if htrans != row.GetValue("TRANSECT_N"):
        runagain = 2                                        #default value
        if (leg1done == "No"):                                  #never found an end to buffer
          #gp.AddMessage("Transect: " + str(htrans) + " was not long enough.\n")
          totshort = totshort + 1
        stop_elev = row.GetValue("STREAM_ELE") + elevlimit
        htrans = row.GetValue("TRANSECT_N")
        leg1done = "No"
        lastpointinbuffer = 0
      if (row.GetValue("RASTERVALU") > stop_elev) and (row.GetValue("POINT_NUM") ==
1):     # 1st point on transect is > elevation limit; include in buffer
        leg1done = "No"
        inbuffer = 1
      elif ((row.GetValue("RASTERVALU") > stop_elev) or (row.GetValue("RASTERVALU")
< ru)):      #point is higher than limiting elevation
        leg1done = "Yes"
        inbuffer = 3
                  #3 indicates last "in" point on transect;  add point with data from previous record
        if lastpointinbuffer == 0:
          updatetransectpoint(pnt.x,pnt.y,tl,tn,pn,ru,se,inbuffer,sn)
          lastpointinbuffer = 1          # flag field
        inbuffer = 2
      elif leg1done == "No":            #point is in buffer so update record
        inbuffer = 1
      else:
        inbuffer = 2                                    #rest of points outside buffer
      feature = row.shape
      pnt = feature.getpart(1)
      tl = row.GetValue("TRANSECT_L")
      tn = row.GetValue("TRANSECT_N")
      pn = row.GetValue("POINT_NUM")
      ru = row.GetValue("RASTERVALU")
      se = row.GetValue("STREAM_ELE")
      sn = row.GetValue("STREAMNUM")
```

```
    sx = row.GetValue("STREAMX")
    sy = row.GetValue("STREAMY")
    sl = row.GetValue("SLOPE")
    updatetransectpoint(pnt.x,pnt.y,tl,tn,pn,ru,se,inbuffer,sn)
    row = rows.next()
  #end of while
  gp.AddMessage("Total Short on the 2nd go:" + str(totshort) + "\n")
  gp.AddMessage("Phase complete.\n")
  ################################################
  ## update counters for next set of transects
  ################################################
  del pnt
  del feature
  del row
  del rows
  del curi
  del pnti
  del feati
  tranmin = tranmax + 1
  if tranmin > tottran:
    trandone = 1
  tranmax = tranmin + increment
  if tranmax > tottran:
    tranmax = tottran
  #### End of while
##end of try
############################################
# Merge the final transect files into 1 file
############################################
delfile(outfc)
outSR = gp.CreateSpatialReference("#", anyfc, "#", "#", "#", anyfc, 100)
gp.CreateFeatureclass(outpath, outfc,"POINT","#","#","#",outSR)
gp.addfield(outfc, "rastervalu", "FLOAT", "6")
gp.addfield(outfc, "transect_n", "LONG")            #identifies transect
gp.addfield(outfc, "point_num", "LONG")              #identifies which point in transect
gp.addfield(outfc, "transect_l", "LONG")            #identifies which side of stream
gp.addfield(outfc, "in_buffer", "LONG")             #flag for in the buffer or not
gp.addfield(outfc, "stream_ele", "FLOAT", "6")
gp.addfield(outfc, "streamnum", "LONG")
gp.AddMessage("Created Output Layer:" + outfc + "\n")
listofinputs = ""
i = 0
while i < 10:
  loopvar = "tranr_" + str(i) + "*"
  fcs = gp.listfeatureclasses(loopvar, "Point")
  fcs.reset()
  fc = fcs.next()
  while fc:
    if listofinputs == "":          # 1st iteration - don't add semicolon
      listofinputs = str(fc)
    else:
```

```
        listofinputs=listofinputs + ";" + str(fc)
      fc=fcs.next()
    i = i + 1
  gp.AddMessage("FC=" + listofinputs + "\n")
  gp.Append_management(listofinputs,outfc,"TEST")
  ################################################
  ## Append the original trane files
  ################################################
  listofinputs = ""
  i = 0
  while i < 10:
    loopvar = "trane" + str(i) + "*"
    fcs = gp.listfeatureclasses(loopvar, "Point")
    fcs.reset()
    fc = fcs.next()
    while fc:
      if listofinputs == "":          # 1st iteration - don't add semicolon
        listofinputs = str(fc)
      else:
        listofinputs=listofinputs + ";" + str(fc)
      fc=fcs.next()
    i = i + 1
  gp.AddMessage("FC=" + listofinputs + "\n")
  gp.Append_management(listofinputs,outfc,"TEST")
except:
  tb = sys.exc_info()[2]
  tbinfo = traceback.format_tb(tb)[0]
  pymsg = "PYTHON ERRORS:\nTraceback Info:\n" + tbinfo + "\nError Info:\n    " + \
          str(sys.exc_type)+ ": " + str(sys.exc_value) + "\n"
  AddPrintMessage(pymsg, 2)
  msgs = "GP ERRORS:\n" + gp.GetMessages(2) + "\n"
  AddPrintMessage(msgs, 2)

#############################################################################
# Program: Module 7 – Final Processing (z_final.py)                         #
# Purpose: Final step in processing.  Create final output theme for buffer  #
# Inputs: sample_points_elev,output pixel size, lakebuffer,                 #
#        sample_points_elev,tran_r_merged                                   #
# Outputs:buffer                                                            #
# Author: James C. Rivard, Michigan Technological University; jcrivard@mtu.edu #
# Date: 9/19/2006                                                           #
#############################################################################
#import relevant modules, create geoprocessing dispatch object
import win32com.client, sys, traceback
import math, os, string
gp = win32com.client.Dispatch("esriGeoprocessing.gpDispatch.1")

########################
# procedure to delete single file
########################
def delfile(delfile):
```

```
    outtempstream = outpath + "\\" + delfile
    if gp.Exists(outtempstream):
      gp.RefreshCatalog(outpath)
      gp.AddMessage("Deleting file " + delfile + "......\n")
      gp.delete(outtempstream)
#########################
# error messages
#########################
def AddPrintMessage(msg, severity):
    print msg
    if severity == 0: gp.AddMessage(msg)
    elif severity == 1: gp.AddWarning(msg)
    elif severity == 2: gp.AddError(msg)

try:
    ###############################################
    #Get Inputs (feature class and feature id)
    ###############################################
    anyfc = str(sys.argv[1])            #sample_points_elev
    infc2 = anyfc
    template = anyfc
    pixelsize = long(sys.argv[2])       #pixel size of output raster
    lakebuffer = str(sys.argv[3])       #lakebuffer from step 1
    infc = str(sys.argv[4])             #output from previous script
    outfc = str(sys.argv[5])            #final output buffer
    gp.AddMessage("Starting program \n")
    gp.AddMessage("Checking out spatial analyst extension...\n")
    gp.CheckOutExtension("Spatial")
    temp1 = "buffer_inpoints"
    temp2 = "buffer_raster"
    temp3 = "buffer_raster_clean"
    temp4 = "buffer_notdissolve"
    desc2 = gp.describe
    desc = gp.describe(anyfc)
    fullpath = desc2(anyfc).CatalogPath
    outpath = (os.path.split(fullpath)[0])
    gp.Workspace = outpath

    #  Cleanup old output files
    gp.AddMessage("Compacting Geodatabase....\n")
    gp.Compact(outpath)
    delfile(outfc)
    delfile(temp1)
    delfile(temp2)
    delfile(temp3)
    delfile(temp4)

    #  Select only points in buffer
    query = "[IN_BUFFER]=1"
    gp.AddMessage("Selecting points in buffer.\n")
    gp.Select(infc,temp1,query)
```

```
    #  Add stream sample points so no holes in buffer #
    gp.AddMessage("Adding stream points back in.\n")
    gp.Append_management(infc2,temp1,"TEST")

    # Convert to   raster
    gp.AddMessage("Convert to raster.\n")
    gp.FeatureToRaster_conversion(temp1,"STREAMNUM",temp2,pixelsize)

    # Clean raster
    gp.AddMessage("Cleaning raster file.\n")
    #gp.BoundaryClean_sa(temp2,temp3,"NO_SORT","TWO_WAY")
    gp.BoundaryClean_sa(temp2,temp3,"ASCEND","ONE_WAY")
    #gp.BoundaryClean_sa(temp2,temp3,"ASCEND","TWO_WAY")

    # Convert to polygons
    gp.AddMessage("Converting raster buffer to polygon buffer.\n")
    gp.RasterToPolygon_conversion(temp3,temp4,"SIMPLIFY","VALUE")

    #  Add lakebuffer and (wetland) polygons
    #appendlist = lakebuffer + ";" + wetlands
    appendlist = lakebuffer
    gp.AddMessage("Adding lakebuffer and (maybe) wetlands to final buffer.\n")
    gp.Append_management(appendlist,temp4,"TEST")

    #  Dissolve Polygons
    gp.AddMessage("Dissolving polygons for cleaner buffer.\n")
    gp.Dissolve_management(temp4,outfc)
except:
    tb = sys.exc_info()[2]
    tbinfo = traceback.format_tb(tb)[0]
    pymsg = "PYTHON ERRORS:\nTraceback Info:\n" + tbinfo + "\nError Info:\n    " + \
            str(sys.exc_type)+ ": " + str(sys.exc_value) + "\n"
    AddPrintMessage(pymsg, 2)
    msgs = "GP ERRORS:\n" + gp.GetMessages(2) + "\n"
      AddPrintMessage(msgs, 2)
```