Stream Correction for Local Government GIS: Project Analysis

Nicholas McKenny, GEOG 596B, Dec 14, 2015
Adviser: James O'Brien

Overview

In the briefest terms, this capstone project was to improve the spatial accuracy of stream network lines using freely-available data and software. More particularly, the aim was to take the surface water flow lines of the U.S. Geological Survey's National Hydrography Dataset (USGS's NHD; nhd.usgs.gov) and, using free, open-source software and high-resolution digital elevation models (DEMs), re-align those line features to a scale and form suitable for tax parcel boundary work while retaining attribute information and topological relationships.

For aesthetic appeal and survey compatibility, this project initially aimed to smooth stream flow lines using tangential arcs and line segments. The reasoning for this consideration is covered in the following section.

Furthermore, the intent of this project was to provide local governments with a reproducible, adaptable, and easy-to-follow process guide.

Ultimately, I was unable to complete this project in the expected time frame. Indeed, the goals of "simplicity and accessibility" may be ultimately at odds with some more complicated low-level considerations of using basic tools and with issues at processing large datasets. On a lesser note, I determined the goal of smoothing lines using arcs to be unlikely worth the effort required. Project implementation never progressed beyond the creation of a hydrologically-enforced terrain. The intermediate outputs produced showed promise but were not without issues and some room for improvement. The incomplete guide can be found in Appendix A & Appendix B.

Image A: Left, NHD flow lines from a 1:24K scale. Right, NHD flow lines at a local map resolution, with corrected stream lines for reference. [Local data from the County of Prince George, VA. Imagery courtesy of the Commonwealth of Virginia.]

Value and Potential Applications

The following is a brief introduction to the value and potential applications of improved stream lines along with a few statements regarding what was this project's scope.

As a scale-appropriate reference for neighborhoods, lots, and intersections; updated stream lines would provide a better reference for set-back considerations in building permits and evaluating run-off buffers. But my chief focus was on surface water flow lines that are of a scale and accuracy suitable to define the geometry of, and align with, certain natural boundaries in land records that are defined as stream centerlines. In my experience in the GIS department of Prince George County, Virginia, many parcel boundaries are defined as "along centerline Ward's Creek" or "center of Black Water Swamp" or "shoreline at low tide" (see Image B).

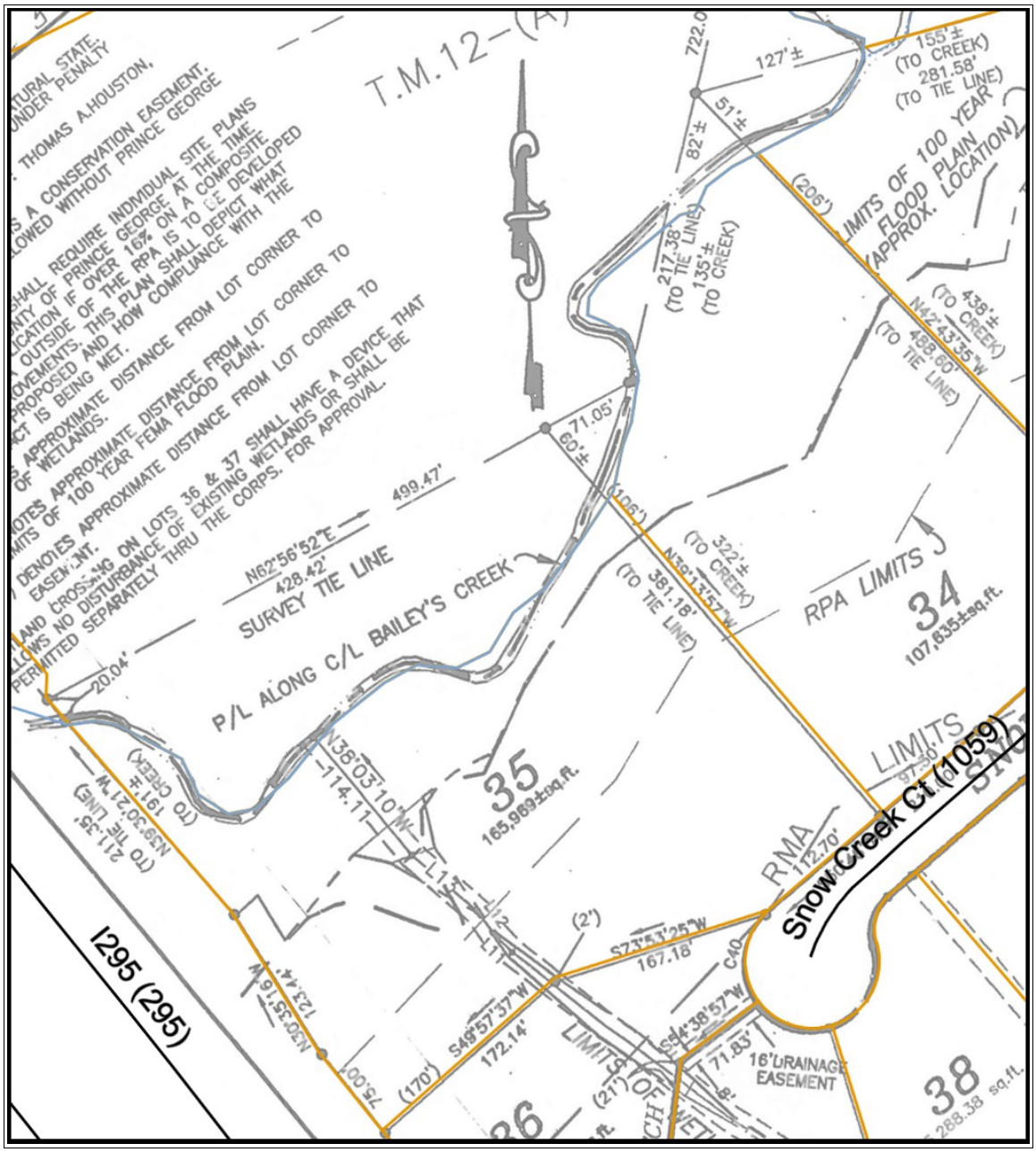


Image B: Tax parcel boundaries and corrected stream lines, over a relevant survey, along I-295 in Prince George County, VA. [Data from the County of Prince George, VA.]

In metes and bounds, qualitative, mathematical representations define surveyed runs and arcs. The bounds, however, are subject to a certain amount of hand-waving and with good reason: a river shoreline or a centerline of a swamp, while understandable concepts, defy a precise, fixed definition. Most natural features do not possess clean, distinct edges at the scale of property definitions. In addition, natural features vary and change over time. In surveys, the survey-defined lines are connected across natural bounds through a single tie line or a chain of

measurements. While modern land surveys may be internally precise to distances of less than an inch, the fuzzy nature of natural features cannot be not adequately represented through survey tie lines. Digital stream lines based on high-resolution terrain data, however, may work.

The USGS NHD FAQ states that positional accuracy standard of well-defined features at the "high resolution" 1:24,000 scale dataset should be within 40' ("NHD FAQ"), but, even strictly adhering to this standard, surveyed property boundaries are unlikely to be compatible with a creek centerline that crosses a road forty feet north of the bridge. The maximum accuracy of any updated hydrologic line features will be limited by that of the elevation data used to perform the correction, but it is hoped that, given the somewhat loose and malleable nature of natural boundary lines and the actual widths of streams, an accuracy of +/- 2.5 to 5 feet would be acceptable. The fuzzy nature of natural boundaries works in favor of a reasonable accuracy goal: unlike with survey-defined lines, here, a horizontal precision of a fraction of an inch would be pointless.

Property lines that conform to these stream lines would be immediately recognizable as following a natural boundary by their shape alone. Applying smoothing to create a continuous, undulating line may even better communicate, at a glance, the presence of a natural feature. Bézier curves in typical GIS line smoothing may be incompatible with common editing tools and land record software in particular. To address this, I intend to explore options to smooth stream lines using tangential arcs and straight lines, as is used in survey measurements.

I find it important to note that GIS property lines do not directly determine the amount of taxable land. In Prince George County, real estate taxes are based primarily on the stated acreage in recorded legal documents: deeds and plats. That said, land found to be encumbered or unencumbered by natural features may affect the taxes due.

I had hoped this project would provide knowledge and an established workflow for counties to follow or to build upon at little cost. Rural counties are typically responsible for a large area

relative to their population and, thus, have a small workforce and budget. With an accessible process, these counties might be free to improve the quality of their GIS features where they might not otherwise choose to do so. The guide was to be presented as a downloadable PDF guide for public use.

This project was intended to focus on the cartographic value of water flow lines and not on hydrographic analysis. Furthermore, this process aimed to only update the stream network data and not the entire NHD data model ("NHD Model"). While the attributes and topology of the hydrographic flow lines, NHDFlowline, were to be retained where deemed possible and worthwhile; maintaining spatial relationships with other existing points, lines, and polygons within the NHD data model was beyond the scope of this project from the start. I had hoped that retaining NHDFlowline attributes would be sufficient to map back to the original features and data model.

Research Approach Overview

In the broadest terms, this project sought to create a fine-scaled stream network from a lower resolution stream network and a high-resolution elevation surface. From a high view, the most apparent approach appeared to be:

1. Identify sources of public GIS data.
2. Generate a hydrographic flow network using high-resolution DEMs.
3. Populate the generated network's features with attributes.

   3a. Apply cartographic generalization and smoothing to stream polylines using lines and arcs.
4. Evaluate results for accuracy and visual appeal.
5. Document how to perform steps two, three, and three-a using free, open-source tools.

Individual Research and Production Steps:

Below, I will explore these proposed research and production steps in more detail.

Step 1: Identify Sources of Public GIS Data

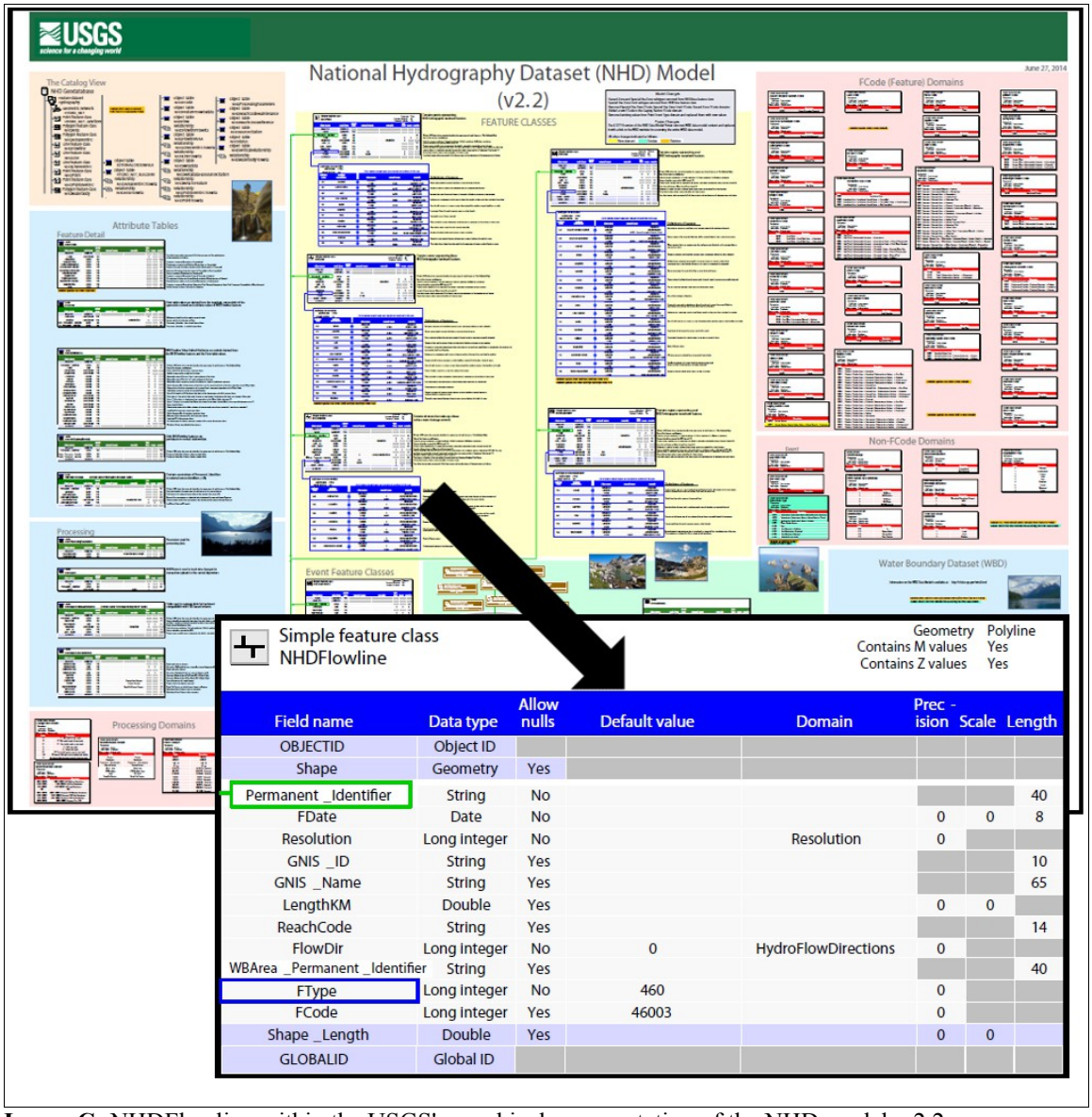
As the particulars of data quality and nature greatly influence the possible analysis results and required processes, the identification of available GIS data and their characteristics should be the basis of any workflow guide. The data required for this process was to be (1) a line network of surface water flow and (2) a surface of high-resolution elevation data. In addition, reference data in the form of waterbody boundaries and aerial orthoimagery were optional but important. This reference data should be current to the date of the elevation data collection and of comparable accuracy.

The hydrographic flow network used are the surface flow lines from the USGS National Hydrography Dataset (NHD). For reference, this linear feature class, NHDFlowline, and its relationship with the rest of the NHD can be found on the poster-sized USGS NHD model (v2.2) ("NHD Model") (see Image C). A more detailed description can be found in the NHD User Guide (*NHD User Guide*).

While much of NHD is intended for map scales no smaller than 1:24,000, this line network possesses valuable traits that should be maintained if possible. The NHDFlowline polylines have attribute connections to the greater NHD model, and their inclusion may useful for easily mapping relationships back to the original dataset. Beyond other tables and classes, any local extraction of NHDFlowline will have topological connections to a larger, Federally-maintained dataset that is national in scope.

Within the NHDFlowline feature class itself, each line segment has attributes and geometry describing both itself and its relationship within a topologically-sound flow network. The vertices of NHDFlowline lines possess m-values and z-values according to the v2.2 data model ("NHD Model"). NHD measures, m-values, are used for linear referencing and identifying positions

along a length of water ("NHD FAQ"). However, z-values, which represent elevation in this context, are not mentioned in the FAQ and do not appear to be present in my local extraction of NHDFlowline.



Field name	Data type	Allow nulls	Default value	Domain	Precision	Scale	Length
OBJECTID	Object ID						
Shape	Geometry	Yes					
Permanent_Identifier	String	No					40
FDate	Date	No			0	0	8
Resolution	Long Integer	No		Resolution	0		
GNIS_ID	String	Yes					10
GNIS_Name	String	Yes					65
LengthKM	Double	Yes			0	0	
ReachCode	String	Yes					14
FlowDir	Long Integer	No	0	HydroFlowDirections	0		
WBArea_Permanent_Identifier	String	Yes					40
FType	Long Integer	No	460		0		
FCode	Long Integer	Yes	46003		0		
Shape_Length	Double	Yes			0	0	
GLOBALID	Global ID						

Image C: NHDFlowline within the USGS's graphical representation of the NHD model, v2.2.

Again, I did not expect all of these attributes, model connections, and spatial relationships to survive this update intact and in a state that can be seamlessly dropped back into the existing NHD, but I hoped enough would be maintained for some automated re-connections to be feasible.

The NHD is freely available through a few methods through the USGS website at

nhd.usgs.gov.

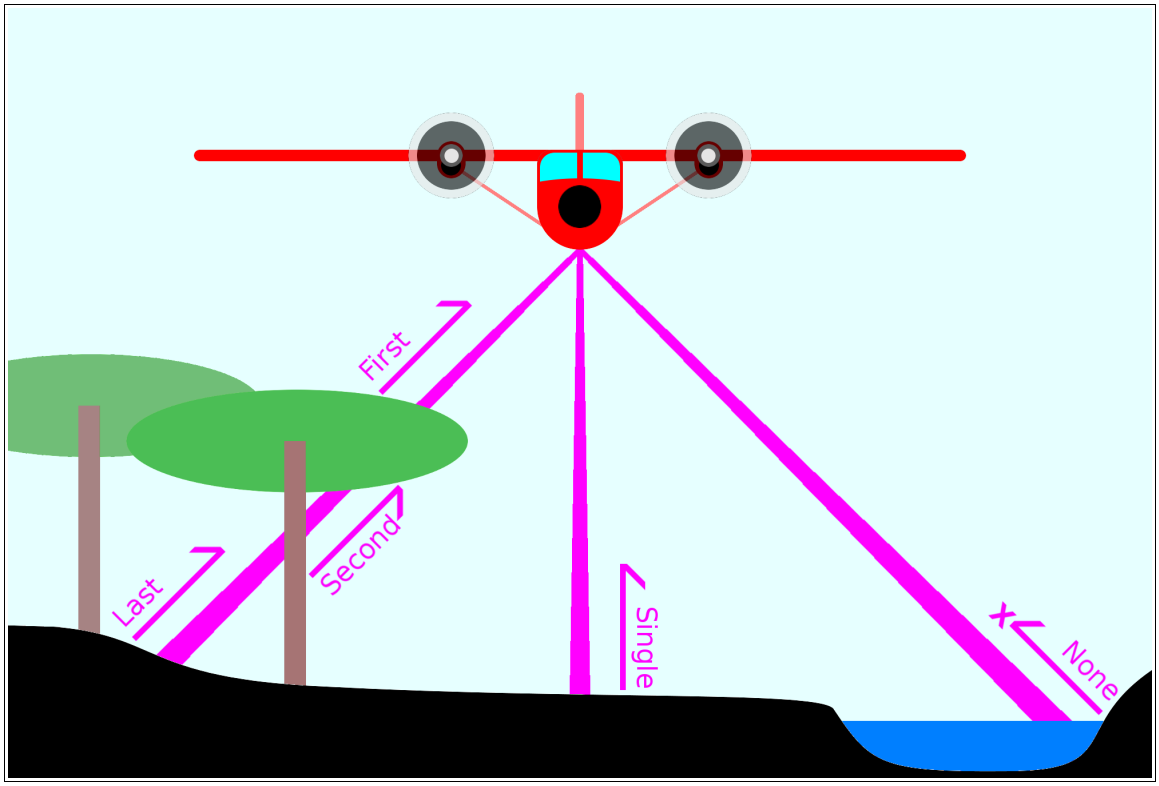


Image D: Airplane-based topographic lidar blankets swaths of terrain with laser pulses. Return pulses create a dense and detailed 3-D terrain dataset. From bare earth, a single pulse is captured; from forest, multiple; from water bodies, one or none.

High-resolution elevation surface data is less universal in availability. For complete coverage within continental United States, the USGS's National Elevation Dataset (NED, ned.usgs.gov) offers, at best, 1/9 arc second resolution (~3 meter/10 foot) elevation data ("NED FAQ"). While this is decent, I preferred a one meter resolution, at least.

For large areas, newer high-resolution DEMs are likely created from topographic lidar. Lidar, a portmanteau of "laser" and "radar" (Merriam-Webster), is a technology wherein relative positions are determined, like radar, like sonar, by the direction and return travel time of reflected signals, in this case, a laser (ASPRS). In typical topographic lidar set-up, laser pulses, from a light aircraft of a known attitude, GPS position, and flight path, blankets swaths of terrain below (see Image D). The resulting product is a dense and accurate 3-D "point cloud" from which other products like DEMs can be generated. Because of the size of the pulse footprint and the transmissivity of some materials, a single laser pulse may produce multiple returns (ex: tree top, branch, trunk,

and ground). This feature of topographic lidar can reveal features such as forest floors that might be obscured in other methods such as photogrammetry. Individual points are classified as bare earth, buildings, vegetation, noise, and such. However, the commonly-used infrared frequency is absorbed by water. As a result, return signals "drop out", and waterbodies are mostly empty

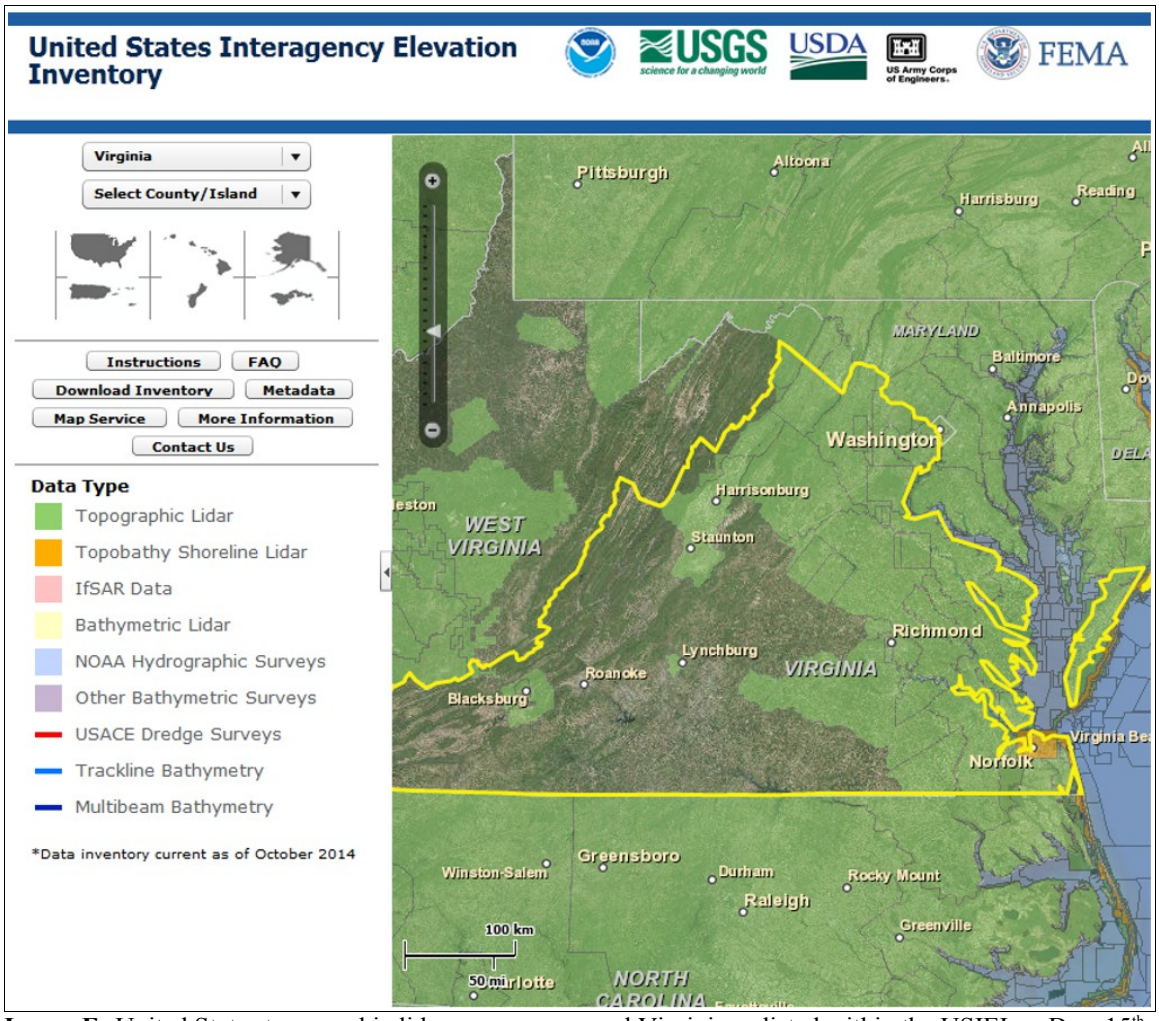


Image E: United States topographic lidar coverage around Virginia as listed within the USIEI on Dec. 15th, 2014.

spaces within the lidar point cloud.

Lidar-derived DEMs, despite being a processing step removed from the point cloud, was the data format to be used in the guide. While working as directly as possible with the source data might be preferable to working with a derived product, some interpolation is required to form a continuous surface from what is a collection of points with no defined measurements between them. Compared to the relatively new subject of lidar 3-D point cloud processing, I hoped the

more established subject of raster-based geoprocessing would provide a more familiar and accessible approach in terms of research, established procedures, and available software.

There appears to be no central clearinghouse for US public lidar data. The United States Interagency Elevation Inventory (USIEI, coast.noaa.gov/inventory/) appears to be the most complete representation of the national coverage of available Federal lidar data (NOAA). In addition, I've found a Wikipedia page, "National Lidar Dataset (United States)", to be a valuable resource, listing some additional state and local datasets (Wikipedia).

Where detailed, high-resolution DEMs are not available, local governments may consider contracting its collection. Given an application for the data, doing so might be found to be worthwhile.

On the subject of reference data, appropriate waterbody features are likely to be found within a lidar dataset as “breaklines”, manually delineated shapes and boundaries to, in this case, compensate for areas of empty data due to water's low reflectivity. The delivery of a lidar product typically includes all the data and documentation needed to reproduce that product (ASPRS).

Aerial orthoimagery is regularly collected by the federal and state governments.

Step 2: Generate a New Hydrographic Flow Network

This step and the one following was to make up the bulk of the end product guide: generating a new hydrographic flow network based on high-resolution DEMs, and then populating those new lines with attributes.

The process of generating stream flow networks from elevation rasters is fairly well established, researched, and supported by software. In the interests of accessibility, I intended to support the most common approach possible.

In exploring the subject of generating a stream network using sub-meter resolution, lidar-derived DEMs, I found a series of publications by Poppenga et al. of the USGS. In particular, I

found two publications to be valuable resources on the issues and utility of high-resolution DEMs in surface flow networks: a 2013 paper published in the Journal of the American Water Resources Association (JAWRA) on detecting areas of hydrographic change in the NHD ("Hydrography Change"), and a 2010 USGS report on creating continuous hydrographic networks ("Selective Drainage"). While these publications focus on correcting NHD flow lines using high-resolution elevation data, and not specifically on performing an area-wide resolution update, I followed Poppenga's et al. process to the best of my ability and understanding.

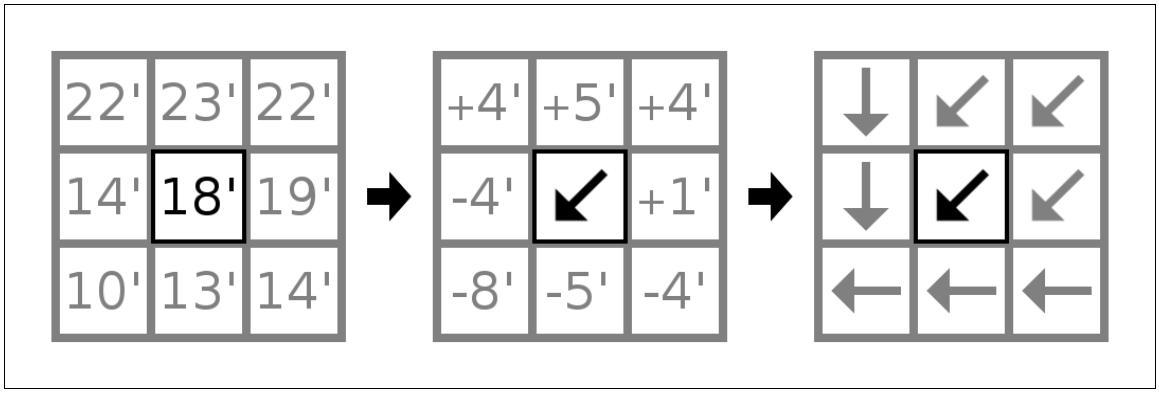


Image F: D8 method for generating a flow direction raster from a DEM.

The very basic concept of generating a stream network on a DEM is being able to determine where, from any given cell in the DEM, a drop of water will flow next. A method is applied to a DEM to create a grid of flow directions from which vector channels can be determined. Perhaps the most common, simple means of doing this centers around the D8 method ("Hydrography Change" 373). In the D8 method, for a given cell, the surrounding eight cells are considered, and a hypothetical drop of water is assumed to flow in the direction that represents the biggest negative elevation change (see Image F). Other methods exist, including the D-infinity method, but other assessments indicate that the increased complexity offers little benefit for the computational complexity over the simpler D8 method, except perhaps in delineating the complex flow channels of low, flat swampy areas ("Hydrography Change" 373).

The method used by Poppenga et al. is to extract the upstream end points from the NHD flow lines and then trace downstream ("Hydrography Change" 375-6). This approach seeds the new stream

network with the upper ends of the existing network. Generated lines should then correspond to the existing NHD delineations within the NHDFlowline feature class.

While existing stream flow line delineation methods have been developed using coarse 30m (~100ft) DEMs ("Hydrography Change" 372), the finer, lidar-based DEMs have some additional considerations and challenges.

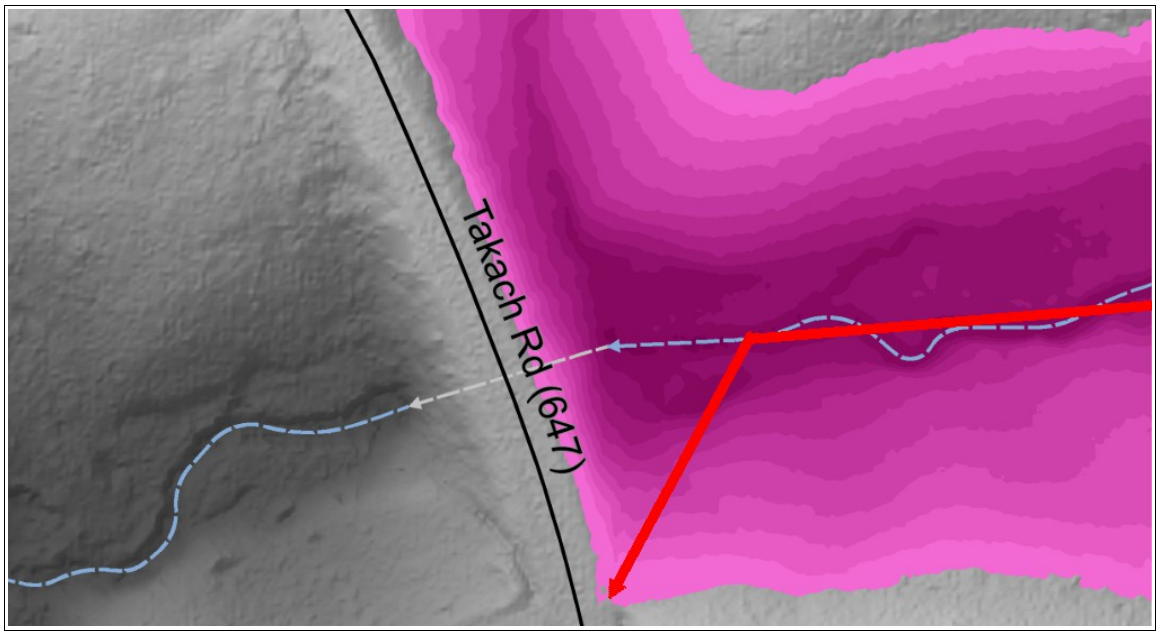


Image G: Actual path of water flow through culvert (blue/gray) compared to the erroneous breach path (red) found by filling a depression. [Local data from the County of Prince George, VA. DEM from USGS.]

One issue was how to handle sinks, or cells for which there are no lower adjacent cells. A downhill flow trace would stop at a sink yet, in reality, a surface water flow will almost always progress all the way to the ocean. With older, small scale DEMs, sinks were treated as errors and filled-in to preserve a continuous surface flow ("Selective Drainage"). At finer resolutions, dams appear, but the sinks behind these may not be errors but instead the product of unrepresented pipes and culverts. Underground features are, after all, beyond the detection of airborne topographic lidar. To ensure continuous flow on finer DEMs, the surface is lowered above underground conduits, breaching those dams and creating special "hydrologically enforced" DEMs. By filling-in depressions, as is done on coarser resolution DEMs, water paths are erroneously created through the lowest point of the leveled surface, rather than at the lowest

point of the sink where an underground conduit would normally lie (see Image G).

Poppenga et al. address this problem by creating a hydrologically-enforced DEM from the standard DEM. The 2010 report describes the following process ("Selective Drainage"):

1. Creating a hydrologically-conditioned DEM by filling-in sinks.
2. Extract these sinks by taking the difference between the original DEM and the conditioned DEM.
3. In each sink, identify the lowest point.
4. From that lowest point, perform a least-accumulative-cost path analysis to identify a path through that sink's dam.
5. Along these paths, lower the cells of the original DEM to that of the end point's elevation, creating a hydrologically-enforced DEM.

A second concern is how to address flat areas and water bodies. The local nature of the D8 method (searching eight surrounding cells) may fail to represent the bigger picture, and manual correction may be required in these situations.

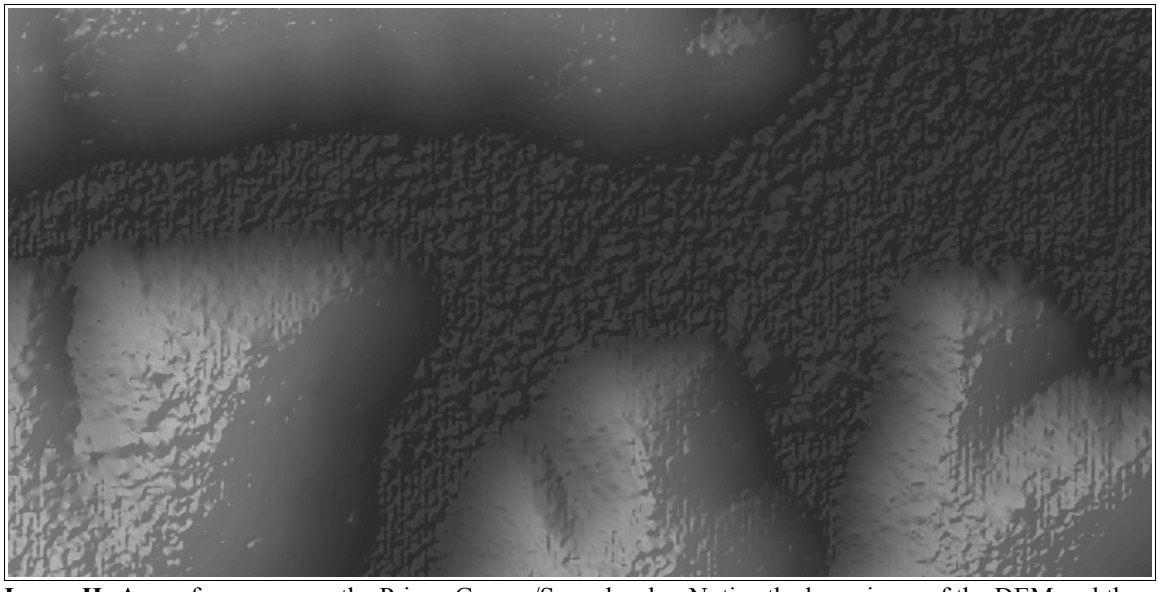


Image H: Area of swamp near the Prince George/Surry border. Notice the bumpiness of the DEM and the absence of a clear channel path. [DEM from USGS.]

In low, swampy areas, paths of water flow may be complex, braided, and very changeable with water level and time. It seems that both low vegetation classified as ground and minor

imprecisions may become dominant in this environment, creating areas with no clear path for stream delineations (see Image H).

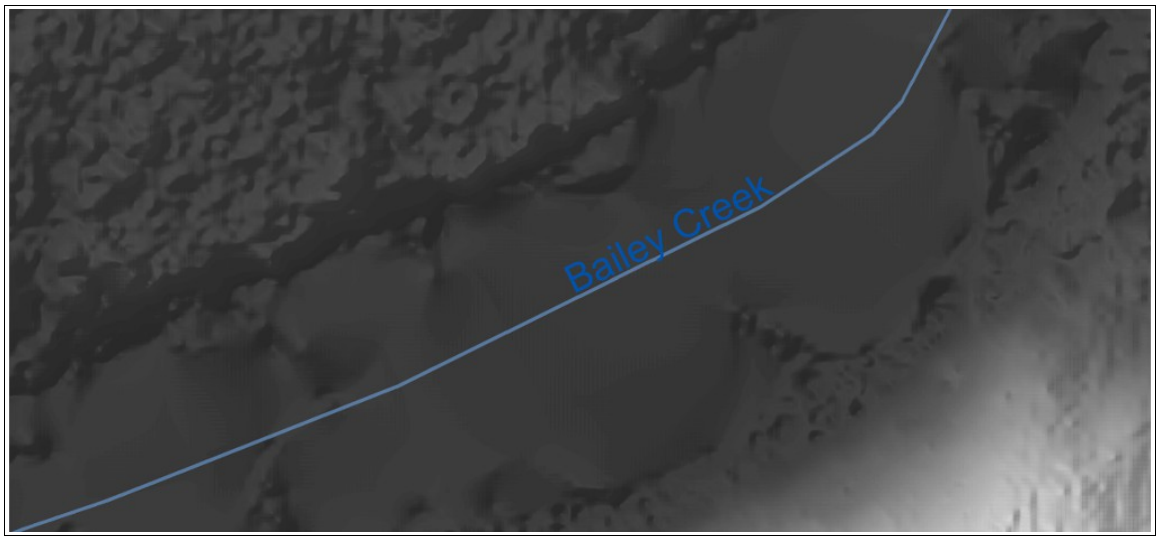


Image I: Wide stream on a DEM, smoothed, but uncorrected by breaklines, producing a lumpy surface. [Data from USGS.]

Due to lidar drop-out over water, when creating a triangulated irregular network (TIN) from the lidar point cloud and then directly creating a DEM, slight variations from stray points and a slightly irregular shoreline creates large facets and not the expected smooth surface (see Image I). A downstream trace will then follow the valleys of this non-existent topography. As mentioned in an earlier section, lidar producers typically create breakline lines and areas to ensure clean edges, flat surfaces on ponds, and even downstream surfaces on rivers (ASPRS 289-91), but an uneven surface can still manifest on smaller bodies of water.

Even where waterbodies are represented as flat, the downhill trace may not perform as expected. In an examination of the utility of dense lidar for extracting hydrographic features (Anderson 89), Anderson indicated that such an algorithm, lacking any local elevation differences, may continue in its prior bearing, bouncing off the shorelines of lakes and larger streams.

A general-purpose solution might be to align such network lines to waterbody centerlines, using lines and polygons or, otherwise, by manual correction using orthoimagery. A one-dimensional line through a two-dimensional feature (ex: lake, river) does not describe passage

across a volume along a linear path. Where such an inexact representation of water flow and network connectivity is required, the NHD classifies such a line as “Abstract” ("NHD Model"). Such a classification would be appropriate to distinguish lines in these areas from clear surface channels.

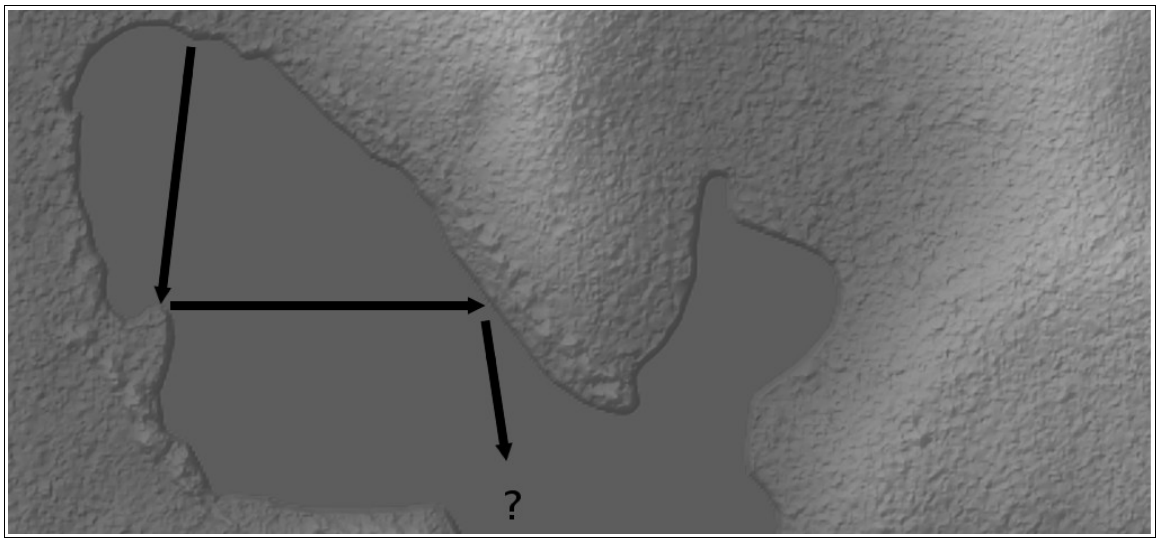


Image J: How a downhill trace might appear to "bounce" from shoreline to shoreline on a flat waterbody. [DEM from USGS.]

One final issue is that the NHD does not represent every stream branch and extent within the United States. Rather, the content of NHDFlowline is limited to what is deemed significant at that scale, a scale updated lines will surpass. While these missing streams, ditches, and canals may appear on surveys, they would not be generated from any network seeded from NHDFlowline headwaters. The inclusion of these non-NHD flow lines was not a priority. Creating and managing lines from two origins would require a modicum of additional steps and user judgment Still, the simpler route would be to forgo deviating from the NHD source and instead flag the issue for interested readers to explore further.

Step 3: Apply Attributes to the Features of the Generated Network

This was the second key step to address.

A single node-to-node polyline in the generated network might be represented in NHDFlowline by multiple segments, split according attribute classifications such as flow

persistence (ex: intermittent, perennial), type (ex: surface, abstract, underground), edit date, or watershed subsection. How the generated stream network is attributed depends on, among other things, the user's needs. The NHD is a large data model and may not be suited to a user's aims. This is the process I intended to expand upon:

First, unimportant divisions (ex: NHD edit date), should be conflated with adjacent and similarly-attributed segments.

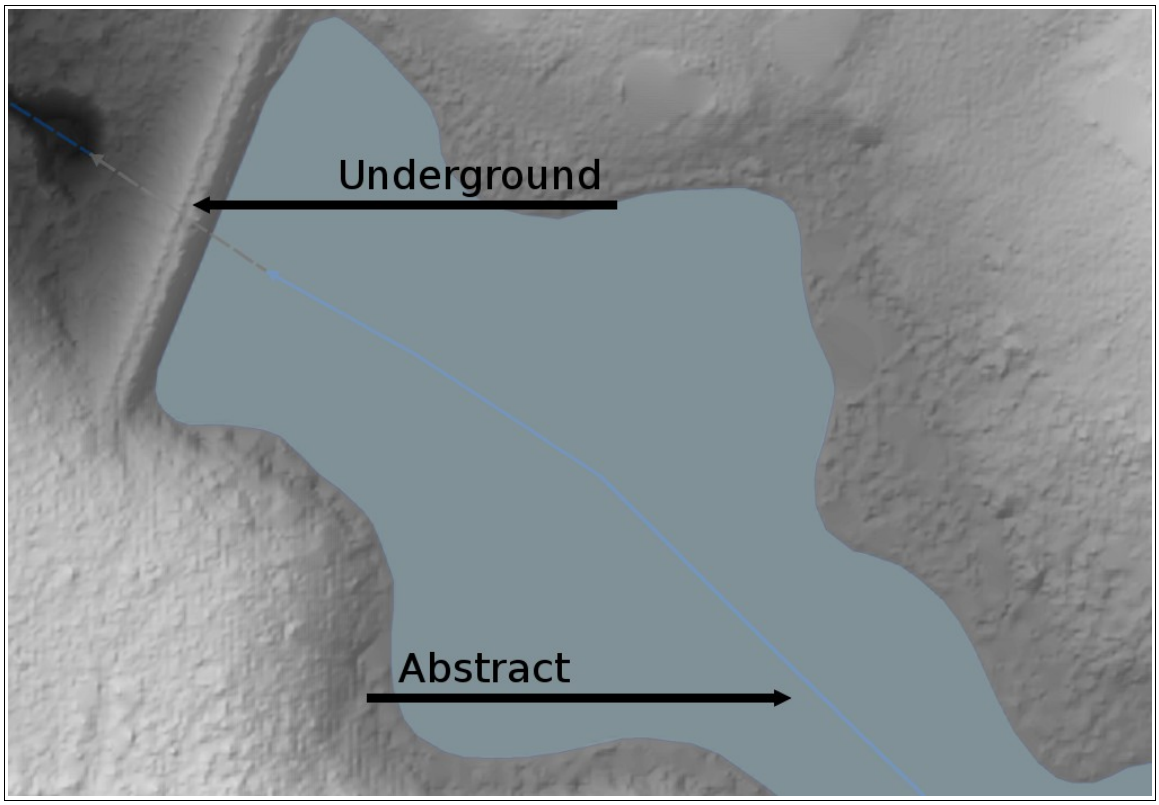


Image K: The passage of water flow, manually classified according to terrain. The lines represent an abstract, non-linear connection through the pond, the subsurface passage of the drain through the dam, and the surface flow beyond. [Stream data from the County of Prince George, VA. Terrain data from USGS.]

Second, wherever possible, the newly-generated polylines should be split and classified based on high-resolution data. Otherwise, mapped classifications will reflect the coarser resolution of the original NHD lines. Water passages through underground culverts may be identified, split-off, and classified as such by comparing the generated network against the difference of the original DEM and the hydrologically-enforced DEM. Similarly, “abstract” segments of the flow network can be identified by overlaying delineated waterbody polygons.

Using appropriate reference data is important here because, with the increase in resolution, some features that might have been represented as lines at lower resolutions may be more appropriately modeled as polygons. Existing polygon water features may have significantly different boundary lines at the higher resolution as well, altering where the stream classification changes.

As a side note, after classifying stream lines by type would be the appropriate point to implement smoothing and cartographic generalization to the generated stream network. This will be covered in the following Step 3a.

Third, map from the NHD any attributes that might be difficult to discern by other means or that might be valuable references. Such attributes include flow persistence, feature names and ID numbers, names and ID numbers of associated bodies of water, and reach codes shared by a continuous stretch one or more lines.

Relating existing NHD lines and attributes with the newly generated stream network geometry might be accomplished in a couple ways: aligning existing NHD line to new or mapping existing attributes to the new lines. Alignment might be possible using ArcGIS's Conflation toolset. For attribute mapping, I originally considered the USGS's NHD GeoConflation tool ("NHD Tools") but have since found it to be not available for this project.

In both approaches, pairs of lines between datasets would be identified within mutual buffers of a width based on the horizontal accuracy. Poppenga's et al. 2013 publication ("Hydrography Change" 376-7) uses this same approach to identify stream line discrepancies.

Regardless of the approach, issues stemming from the resolution change and existing data structure would need to be addressed. Again, some attributes may not be easy or worthwhile to copy from the NHD. How data is modeled can change as resolution increases, including topology. A point of intersection in the NHD may appear very different in the new stream network.

In addition, there may be errors or unrepresented changes within the NHD. As of yet, I've seen no fully automated means of addressing these.

Step 3a: Apply Smoothing to Stream Polylines Using Lines and Arcs

As mentioned previously, smoothing network polylines using tangential arcs and line segments may further improve the aesthetic value while retaining compatibility with a potentially broad base of tools and generally reflecting the nature of survey measurements.

Smoothing may only be appropriate for parts of the flow network that model surface sections of the hydrographic network best represented lines and not polygons. Abstract lines representing an unspecified flow connection through an area such as ponds and wide streams may best be represented as a series of straight segments approximating centerlines. Underground pipes and culverts may be best represented as straight connections as might any irrigation ditches (see Image L).

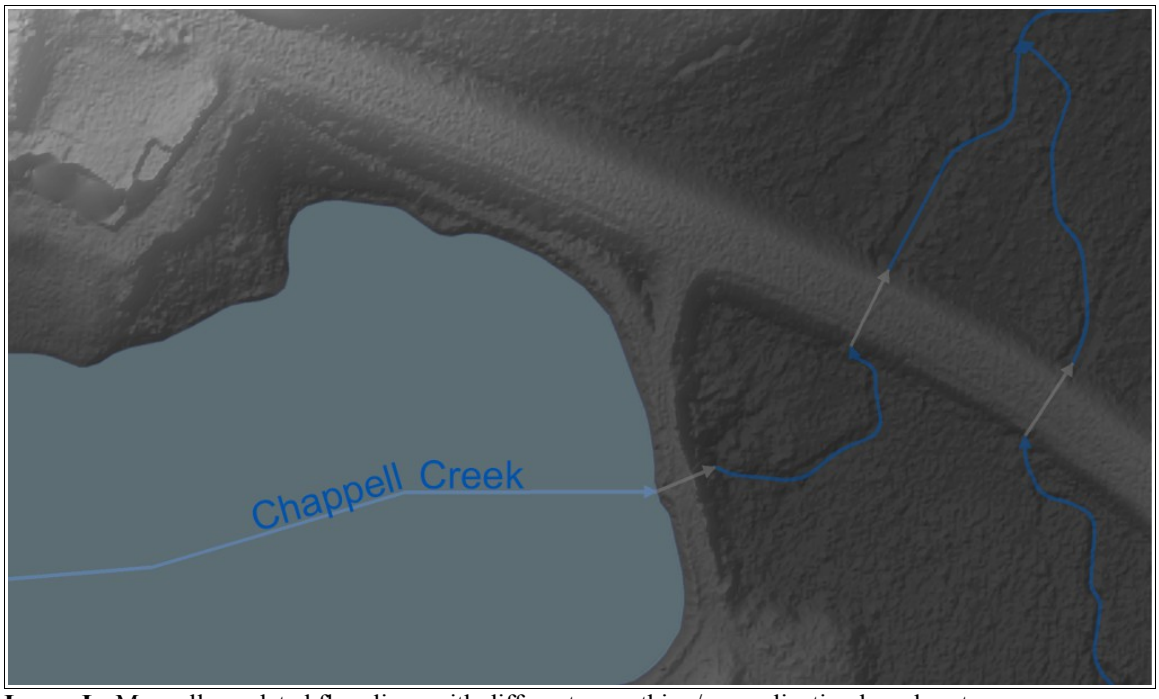


Image L: Manually-updated flow lines with different smoothing/generalization based on type classification. Dark blue surface lines are smooth arcs and lines. Lighter abstract lines are a series of long, straight centerline segments. Through the dam and culverts, gray underground lines are straight, single segments [Stream data from the County of Prince George, VA. Terrain data from USGS.]

I had originally considered smoothing surface lines using tangent curves under the

assumption they would be compatible with land record software. I've since learned curves would complicate their use within ESRI Parcel Fabrics, negating their value for the effort.

Step 4: Evaluate Results for Accuracy and Visual Appeal

At this point, I had intended to evaluate the improvements offered by my workflow.

To evaluate the accuracy improvement of my process, I intended to compare the lines of the new hydrographic network against existing photogrammetric lines of comparable accuracy.

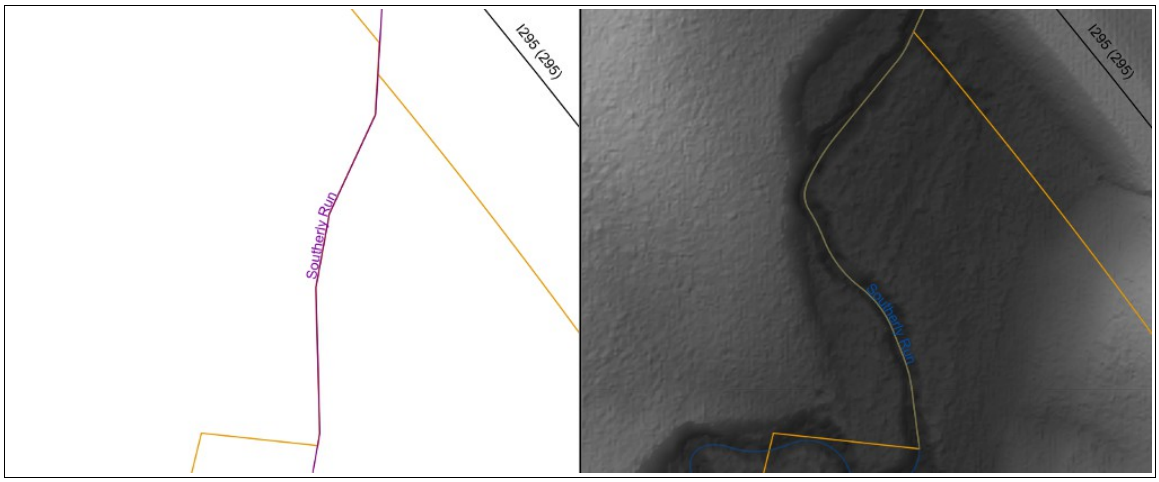


Image M: New stream network lines (right) with smoothing and classifications compared to original NHD lines (left). [Local data from the County of Prince George, VA. Terrain and NHD data from USGS.]

Compared to quantitative analysis, the qualitative quality of visual appeal was to be best evaluated in a side-by-side comparison (see Image M).

Step 5: Document Process Guide using Free, Open Source Tools

In writing the guide, the ultimate product of this project, I aimed to keep the workflow as general as possible. While instructions reference specific tools and pieces of software, the steps and the reasoning behind the steps should stand on their own and not be overly dependent on a particular version of a particular product being available. My intent was to avoid creating step-by-step instructions that, while precise, do nothing to explain the bigger picture.

For this reason, I organized the guide into a hierarchy of instructions. Software-specific "Steps" are grouped under a "Task" overview. Multiple "Tasks" then make up a few very high-level, general "Job" overviews. Since a guide cannot cover all cases and all users' needs, I've

added asides under the title “Go Further...” to highlight opportunities for the interested to explore other options (ex: working with lidar point clouds or using other stream tracing methods).

The first pass at developing the process was with a student copy of ArcGIS 10.2, but with the aim of also developing an open-source alternate set of steps. To keep the guide semi-independent of any specific piece of software, I aimed to constrain steps to relatively common tools. I originally intended to perform the open-source development in QGIS, under the misguided impression it was like ArcGIS, but open-source. Since then, I've realized the true strength of QGIS lies in the individual libraries of tools that it functions as a front-end interface for.

Implementation Retrospective

My original project timeline was to develop the ArcGIS version of the workflow in two months, adapt this to an open-source implementation within a month, and complete a PDF guide at the end of month four. This did not happen. The project, marred by multiple restarts and unexpectedly long processing times, stretched to eleven months, even as it eroded in scope, before being largely suspended. Still, the experience, while frustrating, was educational.

First and foremost, nearly every major aspect of this project lain beyond the realms of my previous experience and education. While I now feel I've learned quite a bit on the subjects of open-source GIS tools, working with rasters, processing very large datasets, and multiprocessing in Python; the time taken in false starts and the learning process quickly and repeatedly proved my initial timeline unrealistically optimistic in relation to my initial skill set.

Attempts to process large datasets can fail quietly or with an unspecific “Unexpected Error” when the demands exceed the resources of the computer and software. The absence of meaningful error messages adds a “poking around in the dark” aspect to identifying the problem's source. Additionally, software documentation, examples, and tutorials are often focused on very small datasets, and thus don't tend to address issues with larger datasets or even processing limits, scaling, and performance.

Processing the area of a county at a fine resolution requires much in terms of computer resources and time. The elevation surface requires gigabytes of storage, multiplied by the needs of output and intermediate data. Depending on the step, processing can take over a week.

For any readers possibly encountering issues with large datasets, I'd suggest first testing the process on a smaller dataset to confirm the issue and to explore alternatives. If data size is indeed the issue and alternative tools are not identified, then consider breaking the dataset into smaller pieces for individual processing. Be aware doing so will require additional steps: breaking-up the data; performing the operation on each piece, including seams; and merging the pieces back into a whole.

Despite an easy-to-explain overview, adapting this approach to a step-by-step implementation uncovered numerous complexities. Many were minor and concessions to existing tools and data: multiple tools where a single tool does not exist, or additional checks to verify and classify data. These amount to multiple steps and checks for tasks that, at high level, can be described as in a single sentence.

A wrinkle with far more serious consequences was that, in the creation of a hydrologically-enforced DEM, each sink drain path cannot be evaluated simultaneously. Normally, a great advantage of raster processing is that a single operation can be applied to the entire surface. However, the buffers of potential drain outlet locations overlap and can even include other sinks. This means it becomes difficult, if not impossible, to keep drain start and end locations both distinct from each each other and associated with its partner. Thus, identifying a drain path is most easily processed one sink at a time. Furthermore, sinks contain sinks, requiring iteration until no sinks remain.

While I briefly considered dividing sinks into subsets of non-overlapping drain areas, I rejected this as being too complicated for the guide's purpose.

Unfortunately, this sequential processing then required deviating from step-by-step

instructions to requiring a Python script. Even then, it slowly became apparent that, to reasonably process ~270 thousand sinks in less than three weeks or so, I had to learn and implement multiprocessing within the Python script. Though all this, I concluded creating a hydrologically-enforced DEM from a topographic one is complicated enough to be organized into it's own high-level job within the guide.

This is where the project currently sits: with a multiprocessing Python script that (mostly) works and a single iteration of sink draining (mostly) completed. However, I must question if a simple, accessible guide is possible if the approach requires a 700+ line multiprocessing Python script. In fact, I'd wager that my dedication to simple, common tools ironically made my process more complicated and my goals much harder to achieve. At the beginning, I reasonably expected a step-by-step manual approach to be feasible; and I had little way of anticipating both, that terrain sinks would have to be individually drained, and the consequences of that. This is on top of the care needed in creating a guide for a general audience. Anyone attempting a similar goal using any tools available may very well have an easier (and more successful!) time of it, provided they keep in mind considerations in processing large datasets.

As a footnote, I'd like to add that open-source software is difficult to develop for, if one is unfamiliar with the tools. For any readers just beginning to explore open-source GIS tools, I'd recommend one not be distracted by pretty user interfaces and instead focus on trying-out the individual command-line tools in open-source libraries. These are the true muscle of any open-source implementation; QGIS, for example, often simply calls these tools with a text string. If you do try QGIS, remember command-lines have a character limit. I found myself editing the command string to be more concise. The open-source aspect is not present in the current version of my guide. I changed my focus to ESRI software early on, and I never progressed far enough to pick it up again.

Output Analysis

At the time of this writing, I've produced one iteration of drain path identifications. On the whole, I'd describe the results of my script as decent but with room for refinement.

First and foremost, there are some situations that cannot be correctly addressed with a wholly programmatic approach because the underlying data cannot support it. Consider the close-up from Image A (see Image N). The manually drawn path is obvious from aerial imagery and programmatic lines. However, the cut-in by the culvert entrance is not represented in the elevation data. Furthermore, the cell with the lowest elevation in the sink (highlighted) is abnormally low: not by enough to be caught in quality control but by enough to skew the pool of drain start cells.

Image N: Left: Apparent stream path compared to aerial imagery and photogrammetric lines. Right: Drain path against terrain data, with the lowest sink cell highlighted. [Local data from the County of Prince George, VA. Imagery courtesy of the Commonwealth of Virginia. Terrain data from USGS.]

Second, there exist connections that may be trivial for a reader to recognize as a drainage channel, but requires more thought and analysis to identify programmatically. Specifically, I have in mind chains of shallow sinks that are minor stream channels or drainage paths (see Image O). These appear either minor enough to not contribute to a stream network or difficult to distinguish from noise. I mention them only to highlight another reasonable limitation on delineating drainage paths.

The most reasonably obtainable approaches toward a moderate improvement in selecting drain start and end cells are adjusting vertical and horizontal leeway, and considering slope and elevation in the immediate vicinity of the potential drain endpoints.

My existing process already selects the drain start from all in-sink cells whose value is within the vertical accuracy of the lowest elevation, and I think this works fairly well. My motivating concern was that the single sink cell with the lowest elevation, perhaps by a fraction of an inch, may not be the logical one for a drain start considering data imprecision.

A similar approach make work for evaluating horizontal differences, treating paths within some fraction of the cell size as the same length. While this might lessen the influence of the raster grid blockiness, I can't claim I have a solid idea how to implement such a change.

Image O: Chain of shallow sinks, representing a minor, roadside drainage path. The right path correctly connects under the driveway, but the left misses the other pipe and paths back into the yard. [Local data from the County of Prince George, VA. Imagery courtesy of the Commonwealth of Virginia.]

A more sophisticated pathing approach might yield the greatest benefits at the cost of processing time. The current path delineation script is simplistic, only considering the nearest connection between two zones of cells. Where multiple possible connections exist with the same length, the one selected is the first identified. Instead, one might favor the lowest elevation start and end cells, or the cells with the sharpest neighboring uphill elevation difference, such as what

might exist adjacent to a culvert or canal wall. That said, in a visual check, I struggled to find any cases where such a refinement would significantly change the delineated output location.

A more cosmetic issue with the initial results is that the Least Cost Path tool, on a flat cost surface, does not produce straight lines in a project where many of these connections should be direct connections (see Image P). Perhaps, instead of raster paths, a more desirable script output would be a feature class of path end point pairs or of path line segments. Alternately, this might be corrected much later as part of simplifying and smoothing the hydrographic network.

Image P: Crooked drainage path from an inlet (top left) to a forested gully (bottom right). Notice the drainage grate partway between the path endpoints. [Local data from the County of Prince George, VA. Imagery courtesy of the Commonwealth of Virginia. Terrain data from USGS.]

Conclusion

And so, here this project currently sits: incomplete and just short of a usable output. That said, I have found this entire process to be enormously instructive in a way that only first-hand, hands-on experience can be. While I do not intend to continue development of this process or guide, the basic idea is sound. I welcome any expanding on this work, either to complete a guide or to develop one's own project.

Appendix A: The Incomplete Guide

Stream Correction for Local Government GIS: An Incomplete Guide

Nicholas McKenny, Dec 14, 2015

Introduction

This is a guide for local governments to create a stream line network based on the USGS National Hydrography Dataset (NHD), updated to a resolution appropriate for maps and applications at the scale of a house lot or road intersection.

In addition, and central to this guide, is the creation of a hydrologically-enforced elevation surface wherein apparent sinks are drained.

Furthermore, this guide aims to provide understandable instruction on the reasoning behind each step so to encourage the adaption of this process to local circumstances.

That said, be aware that this guide is most definitely far from complete. In it's current state, it almost produces a hydrologically-enforced DEM.

Format

The process described in this guide will be organized as following:

Job 1: First Job

Job Overview

Task 1.1: First Task

Task Overview

Step 1.1.1: First Step

Step 1.1.2: Second Step

Task 1.2: Second Task

The seven or eight “jobs” are large, self-contained steps. Each job will start with an overview

section, looking ahead in broad terms.

Within each job are "tasks": lower-level, non-software specific steps within that job. Each task will be a somewhat high-level of description.

Below tasks will be, as needed, software-specific "steps". The implementation described within will be in terms of an ArcGIS 10.2 Standard license with the Spatial Analyst extension.

Within the process description, certain key names and values will be formatted in the following way:

- Tool or software name: Add Rasters To Mosaic Dataset tool (Data Management toolbox, Raster toolset)
- Data body, layer, or dataset: `[StreamUpdateDB]`
- Tool parameter name: `Add New Datasets Only`
- Variable or parameter value: `[CVA]` or `OVERWRITE_DUPLICATES`

This guide will also include frames as informational asides, such as below:

Go Further...

Occasionally through this text, you will see frames such as this one briefly mentioning alternate approaches or additional steps. While they will not be explored in any detail, they are included should you wish to explore them further.

Or, you may see:

Take Note

This guide will also sometimes include a frame highlighting a point or caveat of particular importance.

Disclaimers

First, while following this guide will increase the spatial resolution of existing NHD lines, the scale of the existing NHD means some features are not represented and thus will also not be present in new feature class based directly on the NHD. That is, the intended scale of a feature

class determines not only the fineness of feature details, but also what features are included and how they're represented. At the relative low resolution and small scale of the NHD, some minor tributaries and areas of complex branching will be too small or fine to be represented.

Second, while this guide offers a particular series of steps, this is not the only, or even the best, way to go about this.

Hardware and Software Requirements

This process requires ArcGIS, Standard license or better, plus the Spatial Analyst extension. This guide was written using ArcGIS 10.2. The computer used will require tens of gigabytes of storage free, depending on the area to be processed, plus ample space for scratch work. An external drive may be helpful to free-up needed space.

Go Further...

If the costs of licensed software is a barrier, consider open-source. However, if you are new to open-source libraries, I suggest you experiment on test data first. Be aware adapting this process to open-source tools may be a time-intensive one.

Process Overview

1. Identify and acquire the necessary data for this project.
2. Prepare the data and work environment.
3. Created modified copies of the DEMs that allow for continuous water flow.
4. Extract headwater points from the NHD stream network.
5. Regenerate NHD stream network using DEMs.
6. Classify stream features based on terrain data.
7. Generalize stream features.
8. Apply NHD attributes to stream features.

Job 1: Collect Data

Job Overview

First things first. Locate and acquire the data you'll be working with, plus any associated metadata and documentation.

Task 1.1: NHD Stream Network

Task Overview

The NHD stream network is found in the NHDFlowline feature class, within the larger NHD data model. The value of using NHDFlowline is that it's a part of a maintained dataset with national (U.S.) coverage.

Go Further...

This guide will not deal with any other features within the National Hydrography Dataset, but you may consider updating them or applying their attributes and relationships to your own data.

The NHD website offers multiple ways to download state or sub-region dataset extractions at nhd.usgs.gov/data.html.

Take Note

Be aware that the NHD may contain better-than-1:24,000 scale or even local resolution data for some areas. Check your area: it's unlikely, but maybe someone has already contributed updated water flow lines to the NHD. If so, your work can stop right here!

Or maybe you can be the one to contribute updated lines to the NHD. The USGS is open to local contributions.

Task 1.2: Elevation Data and Waterbody Delineations

Task Overview

To update NHDFlowline, high-resolution terrain data will be needed. The terrain data covered in this guide will be lidar-derived DEMs and associated breaklines.

Airborne topographic lidar is capable of producing dense, highly accurate datasets in the form of point clouds. These point clouds can be interpolated into a continuous, ground-level surface, and the raster-based DEM is a common (and commonly-supported) format. Furthermore, lidar products (ex: DEMs) are often delivered with all the base data (ex: point clouds, breaklines) and documentation needed to reproduce the delivered products. Among these are breaklines, delineated feature boundaries from which suitable waterbody boundaries may be extracted.

While there appears to be no central clearinghouse for US public lidar data as of this typing, the United States Interagency Elevation Inventory (USIEI, coast.noaa.gov/inventory/) appears to be the most complete representation of the national coverage of available Federal lidar data (NOAA). In addition, I've found a Wikipedia page, "National Lidar Dataset (United States)", to be a valuable resource, listing some additional state and local datasets (Wikipedia).

Go Further...

The use of lidar-derived DEMs is a convenience of availability and simplicity. Other options for data and geoprocessing exist, but, in the interests of space and readability, only one common case will be covered herein.

Instead of working with lidar-derived DEMs, you may consider working directly with the lidar point cloud.

Task 1.3: Aerial Orthoimagery

Task Overview

Aerial orthoimagery is regularly collected by the federal and state governments. Original data can be readily available to local governments.

Don't confuse resolution for spatial accuracy. Check metadata.

Remember that for orthoimagery, and all other reference data, a comparable accuracy,

resolution, and collection date to that of the elevation data is valuable.

Job 2: Prepare and Organize Data and Environment

Job Overview

Organize and, if needed, re-project your collected data to match the spatial reference of the DEM. This includes importing the base data into the storage structure you'll be using in the steps ahead, and making sure you have a back-up created.

Take Note

A number of layers will be created in this process. To keep them organized, you may consider naming them using the prefix identifying their step or task of origin (ex: “s243_” for “Step 2.4.3”).

Task 2.1: Install Software Tools

Task Overview

Install the necessary software, tools, and licenses for this project. Import and ready the base data for processing.

Note that this guide will assume vector feature layers are stored within a local ESRI file geodatabase.

This project will require an ArcGIS Standard license level plus the Spatial Analyst extension. Installation help, if needed, is provided through ESRI resources.

Task 2.2: Determine Project Values

Task Overview

The source data, as well as the intended application, define and limit the product. Finer and higher resolution is not always better, meaningful, useful, or desired. For example, the NHD data horizontal accuracy standard is that 90% of points are within 40 feet of their true location. While some of these values may have to be determined by experimentation and guesswork, it's best to determine and note these ahead of time.

Check metadata and documentation. Remember that accuracy and resolution are both important, but not equivalent.

Take Note

This guide assumes a plus/minus (±) value measure of accuracy. If you find accuracy expressed as a normalized root-mean-square error (RMSE), multiply by 1.96 to produce will produce an accuracy range that 95% of points should be within (ex: 1.9600*$RMSE_z$ => $Accuracy_z$ or 0.31 RMSE => 0.61 ft).

Lidar documentation will list multiple measures of vertical accuracy. This guide will use Consolidated Vertical Accuracy (CVA).

Lidar documentation doesn't always list horizontal accuracy. Check the metadata and flight documentation. For example, in developing this guide, I used 2.5 ft resolution DEMs from a lidar collection with roughly a 3.28 ft (1 m) horizontal resolution.

Finally, be sure your values are those associated with the data product you'll be using and not with contributing data.

Feel free to record project values below for your own reference:

`[DEM Tiles]`

1. Collection Date Ranges:
2. Resolution/Grid Size:
3. Pixel Type:
4. Horizontal Coordinate System and Units:
5. Horizontal Accuracy:
6. Vertical Coordinate System and Units:

7. Fundamental Vertical Accuracy (`[FVA]`):
8. Consolidated Vertical Accuracy (`[CVA]`):

`[Waterbody Delineations]`

1. Collection Date Ranges:
2. Horizontal Coordinate System and Units:
3. Horizontal Accuracy:

`[NHD Extraction]`

1. Horizontal Coordinate System and Units:
2. Horizontal Accuracy:

`[Aerial Orthoimagery]`

1. Collection Date Ranges:
2. Horizontal Coordinate System and Units:
3. Horizontal Accuracy:

Task 2.3: Set-up Work Environment

Task Overview

Set-up the needed parameters and workspace.

Create a work folder and a folder connection to it.

Create a file geodatabase, `[StreamUpdateDB]`.

Task 2.4: Lidar-Based DEM

Task Overview

Organize or transform the acquired DEM tiles so they may be processed as a single unit. If you haven't already done so (and you really should have), verify that the tiles are seamless and that they have a set and consistent spatial reference.

Step 2.4.1: Create Mosaic Dataset

Create `[TopoDEMMosaic]`, a mosaic dataset, within `[StreamUpdateDB]`. Set the horizontal **and** vertical coordinate systems to that of `[DEM Tiles]`; import the coordinate system directly from the tiles if possible. Optionally, set the mosaic pixel type to match `[DEM Tiles]`.

Step 2.4.2: Populate Mosaic Dataset

Populate `[TopoDEMMosaic]` with references to `[DEM Tiles]`. Use the Add Rasters To Mosaic Dataset tool (Data Management toolbox, Raster toolset). Easily open this tool with a right-click on `[TopoDEMMosaic]` in the Catalog window and the selection of "Add Rasters...". Use the following parameters: leave `Raster Type` on its default; select the folder containing `[DEM Tiles]` as the Input Data workspace; check `Update Overviews`. Under advanced options, set `Add New Datasets Only` option to `OVERWRITE_DUPLICATES` and check `Calculate Statistics` if the source rasters lack statistics. Check the other settings according to your need.

> **Take Note**
>
> Be aware that, depending on the options enabled, populating the mosaic may take tens of minutes to complete.

Double-check `[TopoDEMMosaic]` for any errors or unusual values. If, under Layers, you can zoom to Boundary and Footprint but not Image, and, under Layer Properties, "Columns and Rows" are blank; then refresh `[TopoDEMMosaic]` through the right-click menu. If statistics have not been created, do so within a Catalog window using "right-click > Enhance > Calculate Statistics...".

Verify that `[TopoDEMMosaic]` isn't missing any rasters. Overviews will need to be rebuilt after adding new rasters.

Step 2.4.3: Export Mosaic to Single Raster Dataset

Create [TopoDEM], a single raster dataset from [TopoDEMMosaic]. In the coming steps, a single raster may work better than a mosaic. Use the Copy Raster tool (Data Management toolbox, Raster toolset), opened with a right-click on [TopoDEMMosaic] and selection of Export > Raster to Different Format.

Create and save a map document for this project (ex: StreamUpdateProj), if you haven't done so already. Set the data frame's coordinates system to match [TopoDEM]. Set the layer symbology as you wish. Remove [TopoDEMMosaic] after the following task.

Task 2.5: NHD NHDFlowline class

Task Overview

From the [NHD Extraction], extract a subset of the NHDFlowline features, [NHDFlowline_Local], in the vicinity of your area of focus.

Step 2.5.1: Select NHD Lines

Add the [NHDFlowline] feature class from the [NHD Extraction] to your map document.

Be sure this subset, [NHDFlowline Selection], has the same spatial reference as [TopoDEM].

Create [NHDFlowline_Selection], the subset of [NHDFlowline] lines that intersect [TopoDEMMosaic]. Use either the Select By Location menu option or the Select Layer By Location tool (Data Management toolbox, Layers and Table View toolset) to select features from [NHDFlowline] that intersect the boundary of [TopoDEMMosaic. Export these selected features into [StreamUpdateDB] as a new feature class [NHDFlowline_Selection]. Don't change the feature the coordinate system until the next step.

Step 2.5.2: Extract Selected NHD Lines

Create [NHDFlowline_Local], a copy of [NHDFlowline_Selection] projected into the coordinate system of [TopoDEM]. Use the Project tool (Data Management toolbox, Projections and Transformations toolset).

Adjust the symbology of [NHDFlowline_Local] or import that of [NHDFlowline]. Remove [NHDFlowline] and [NHDFlowline_Selection] from the map document

Task 2.6: Reference Data

Task Overview

Similarly organize and re-project (as needed) the other reference data. On-the-fly reprojections may be sufficient for [Aerial Orthoimagery], but not for data that will participate in any geoprocessing.

Set or bookmark a project extent.

Step 2.6.1: Create an Orthoimagery Mosaic

In the example of the previous steps, import your project's reference data (ex: [Waterbody Delineations] and [Aerial Orthoimagery]).

For [Aerial Orthoimagery], create and populate another mosaic dataset, [AerialOrtho], within [StreamUpdateDB].

Step 2.6.2: Import Waterbody and Other Reference Data

Import [Waterbody Delineations] into [StreamUpdateDB], as the polygon layer [WaterbodyPolygons], matching the coordinate system of [TopoDEM]. You may need to merge and process [Waterbody Delineations] to create a single polygon layer of just waterbodies.

A custom full extent can be set through the Data Frame Properties dialog on the Data Frame tab. Access it by right-clicking on the data frame background or, under

the Table of Contents, on the default “Layers” data frame.

Job 3: Create Hydrologically-Enforced DEM

Job Overview

Create a hydrologically-enforced DEM, `[HydroEnfDEM]`, from `[TopoDEM]`. A hydrologically-enforced DEM accommodates the modeling of interrupted downhill flow. While topographic terrain data represents accurate surface elevations, continuous downhill water flow requires channels corresponding to culverts and underground pipes to drain larger sinks (areas with no apparent downhill outlet). Smaller sinks can be filled.

> **Take Note**
>
> This job is iterative. Consider adding a numbered suffix (ex. “_01”) to datasets created in this job.

Task 3.1: [If Needed] Remove Tidal Sinks

Task Overview

Sometimes minor depressions can appear on the borders of tidal waterbodies. Since these sinks will not be able to be drained to a lower elevation, the DEM within the tidal waterbody must be lowered slightly. If this doesn't apply, skip ahead to the next task.

A visual check may be enough to identify the issue. Other options would be either to preemptively lower any tidal waterbodies to some value below the lowest tide, or to proceed with the process and perform this step if sinks are unable to be addressed by the following steps withing this job.

Step 3.1.1: [If Needed] Extract Tidal Waterbodies

If the equivalent does not already exist, create `[Waterbodies_Tidal]` by selecting features from `[Waterbody Delineations]` and saving them as a new feature class or raster dataset.

Step 3.1.2: Lower Tidal Waters to Slightly Below Low Tide

Create [TopoDEM_Lowered], a new version of [TopoDEM] with artificially lowered sinks. Use Raster Calculator (Spatial Analyst toolbox, Map Algebra toolset) with the following expression:

```
Con("[TopoDEM]", Con("[Waterbodies_Tidal]",
"[TopoDEM]" - 10, "[TopoDEM]"))
```

Take Note

Outside of ArcGIS Desktop, this expression requires a full path for each referenced raster.

While I've lowered the DEM by 10 feet in this example, use your own judgment to determine a relative or absolute elevation. Also, in the following steps, treat [TopoDEM_Lowered] as [TopoDEM].

Task 3.2: Create Raster of Sink Depths

Task Overview

By filling sinks and calculating the difference from the original DEM, create a raster where the value of each cell is the depth below the spill height of any enclosing sink. This is set-up for creating a raster map of sink location in the next task.

Step 3.2.1: Create a Hydrologically-Conditioned DEM

Create [HydroCondDEM_01], a hydro-conditioned copy of [TopoDEM]. Use the Fill tool (Spatial Analyst toolbox, Hydrology toolset).

Take Note

This tool can take hours to complete. A 2.5' grid of Prince George County, VA on a 2011 laptop took 3 hours.

This version [TopoDEM] has every sink removed by filling to a spill point. The

difference will be used to identify sinks.

Step 3.2.2: Find Difference Between DEMs

Create `[SinkDifference_01]`, a raster of the depth of individual cells filled the creation of `[HydroCondDEM_01]`. Use the Minus tool (Spatial Analyst toolbox, Math toolset) with the following parameters: `Input raster 1` as `[HydroCondDEM_01]`; `Input raster 2` as `[TopoDEM]`; `Output raster` as `[SinkDifference_01]`.

Task 3.3: Create Sink Map (using Region Group)

Task Overview

Create `[SinkMap_01]`, an attributed raster identifying each contiguous group of individual filled cells within `[SinkDifference_01]` as a sink with a unique ID number. The preferred way, using the Region Group tool, breaks down when adding an attribute table to a raster with a very large number of unique values (over 5 million, at least). For any sizable area, the initial iteration within this job may have many more than this count. To get past this initial hurdle, a second option, using the Sink tool, is offered as the following task.

Go Further...

If you have building footprint or undrainable feature boundary data and would like to remove them from any potentially drained sinks, this would be the time.

Step 3.3.1: Create Single Zone Raster of Non-Zero Cell Differences

Create `[SinkDifferenceMask_01]`, an integer raster of all filled cells in `[SinkDifference_01]`, classified as a single zone. Use the Con tool (Spatial Analyst toolbox, Map Algebra toolset) with the following parameters: `Input conditional raster` as `[SinkDifference_01]`; `Expression` as "`>0`"; `Input true`

`constant value` as "`1`"; `Output raster` as `[SinkDifferenceMask_01]`.

Step 3.3.2: Split Single Zone into Contiguous Sinks

Create `[SinkMap_01]`. Use the Region Group tool (Spatial Analyst toolbox, Generalization toolset) with the following parameters: `Input raster` as `SinkDifferenceMask_01]`; `Output raster` as `[SinkMap_01]`; `Number of neighbors` to use as `EIGHT`; `Zone grouping method` as `WITHIN`; `Add link field to output` as unchecked.

This tool defines an attribute table with Value and Count fields. If the attribute table is missing, use the Build Raster Attribute Table tool (Data Management toolbox, Raster Properties toolset). If attempts to create the attribute table ends in an "unexpected error", try the alternate approach below.

Step 3.3.3: Clean-up

Delete `[SinkDifferenceMask_01]`.

Task 3.4: [Alternate] Create Sink Map (using Sink Tool)

Task Overview

If the preceding task will not work, create `[SinkMap_01]` using the Sink tool.

Step 3.4.1: Create Artificially Lower Filled Sinks

Create `[TempLoweredSinks]`, a version of `[HydroCondDEM_01]` with artificially lowered sinks. Use Raster Calculator (Spatial Analyst toolbox, Map Algebra toolset) with the following expression:

```
Con("[SinkDifference_01]" > 0, "[HydroCondDEM_01]" -
10, "[HydroCondDEM_01]")
```

> **Take Note**
>
> Outside of ArcGIS Desktop, this expression requires a full path for each referenced raster.

Step 3.4.2: Create Flow Direction Raster

Create `[TempLoweredFlow]`, a flow direction raster for `[TempLoweredSinks]`. Use the Flow Direction tool (Spatial Analyist toolbox, Hydrology toolset) with the following parameters: `Input surface raster` as `[TempLoweredSinks]`; `Output flow direction raster` as `[TempLoweredFlow]`.

Step 3.4.3: Identify Sinks

Create `[SinkMap_01]`. Use the Sink tool (Spatial Analyst toolbox, Hydrology toolset) with following parameters: `Input flow direction raster` as `[TempLoweredFlow]`; `Output raster` as `[SinkMap_01]`.

If the attribute table is missing and the Build Raster Attribute Table tool (Data Management toolbox, Raster Properties toolset) fails, explore other means to divide the input dataset into smaller units.

Step 3.4.4: Clean-up

Delete `[SinkDifferenceMask_01]`, `[TempLoweredSinks]`, `[TempLoweredFlow]`.

Task 3.5: Create Raster Attribute Fields

Task Overview

Create raster fields to store sink attribute data. These values will be the means by which sinks are classified and processed.

Step 3.5.1: Create Field to Store Sink Type Classifications

Create the short integer field `"Drain"` in the `[SinkMap_01]` raster attribute

table. This field will be contain your classification for each sink as to if they are to be filled or to be drained and how. This roughly corresponds to types of sinks.

- `<null>` = unclassified
- `0` = false / "to be filled". Sinks with neither surface in-flow or out-flow (ex: most quarries). Also, insignificant sinks (i.e. noise) or those otherwise not worth consideration.
- `1` = true / "to be drained via dam breaching". Sinks with both surface in-flow and out-flow (ex: most "false" sinks).
- `2` = true / "to be drained through bottom". Sinks with surface in-flow but no surface outflow. Rarer than the other two cases, but examples might be some flooded quarries, sink holes, or dry lakes. Rare, and either ignored or manually drained.
- `3` = complex/"requiring additional steps". Don't ever actually use this as a classification value; it is listed here for ease of reference and to raise awareness of an important, but rare situation.

Take Note

Sinks can (and do) exist within sinks such that, when the containing sink is drained (`1` or `2`) and not filled (`0`), new, smaller sinks are revealed. Normally, these new sinks are handled on the next iteration of the process. However, if the enclosing sink is to be drained via dam breaching (`1`) but the internal sink is to be filled (0) **and** it contains the lowest elevations, interfering with the dam-breaching pathing, the internal sinks will have to be dealt with separately in an additional step. This should be a very rare case.a

Go Further...

You may wish to consider a more complicated classification scheme, such as by type.

Take Note

The goal here is not to classify every sink, but to filter-out irrelevant features and to mark important sinks appropriately. Unclassified sinks can be treated as `1`, “to be drained via dam breaching”.

Step 3.5.2: Create Additional Sink Attribute Fields

Add the following attribute fields to `[SinkMap_01]`: `"ElevMin"`, `"ElevSpill"`, `"DepthMax"`, `"DepthMean"`. In selecting a data type, consider the raster type but understand that Zonal Statistics as Table will return double-precision floating point fields in the coming steps.

Go Further...

Include any other fields you deem valuable for filtering and classifying sinks or for draining them.

Task 3.6: [If Needed] Process Sink Map In Chunks

Task Overview

If the following tasks to collect sink attribute data fail with an unexpected error, break `[SinkMap_01]` into smaller chunks that ArcGIS can handle, run the step on each `[SinkMap_01]` subset, and then merge the subset step results together.

Step 3.6.1: Split Sink Map

To create subset of `[SinkMap_01]`, use the Con tool (Spatial Analyst toolbox, Map Algebra toolset) with parameters similar to the following: `Input conditional raster` as `[SinkMap_01]`; `Expression` as "`(Value >= x) AND (Value <`

y)"; Input true raster as [SinkMap_01]; Output raster as [SinkMap_yofz].

Here, x is initially 0, y is initially the number of zones in a single division, z is the number of zones in [SinkMap_01], and "_yofz" is a suffix of your choosing to keep the subsets distinct. After the first subset is created, run the tool again with updated parameters until all of [SinkMap_01] is divided.

Pick a value of x that minimizes the number of subsets required. Picking a round number will make count errors obvious to a visual check. Limited to 32-bit processing, I found 5000000 to be within the upper limit of what Zonal Statistics as Table can handle. Some experimentation may be needed to determine an acceptable value for x.

Keep the suffix readable and able to be sorted. For example, I used [SinkMap_01_05of18] for my roughly 18 million initial sinks.

Step 3.6.2: Merge Sink Map Chunks

To merge the step results back together, use the Merge tool (Data Management toolbox, General toolset) with parameters similar to the following: Input Datasets as ZonalSt_Topo_05of18, ZonalSt_Topo_10of18, ZonalSt_Topo_15of18, ZonalSt_Topo_20of18; Output Dataset as ZonalSt_Topo.

Task 3.7: Create Sink Statistics Temporary Tables

Task Overview

Generate temporary tables containing sink statistics corresponding to the statistics fields previously created in the [SinkMap_01] raster attribute table.

Step 3.7.1: Get Sink Minimum Elevations

Create [ZonalSt_Topo], a temporary table of the minimum elevation of each

sink. Use the Zonal Statistics as Table tool (Spatial Analyst toolbox, Zonal toolset) with the following parameters: Input raster zone data as [SinkMap_01]; Zone field as Value; Input value raster as [TopoDEM]; Output table as [ZonalSt_Topo]; Statistics type as MINIMUM.

Step 3.7.2: Get Sink Spill Elevations

Create [ZonalSt_Hydro], a temporary table of the spill elevation of each sink. Use the Zonal Statistics as Table tool (Spatial Analyst toolbox, Zonal toolset) with the following parameters: Input raster zone data as [SinkMap_01]; Zone field as Value; Input value raster as [HydroCondDEM_01]; Output table as [ZonalSt_Hydro]; Statistics type as MAXIMUM.

Step 3.7.3: Get Sink Mean and Maximum Depths

Create [ZonalSt_Diff], a temporary table of the maximum and mean depth of each sink. Use the Zonal Statistics as Table tool (Spatial Analyst toolbox, Zonal toolset) with the following parameters: Input raster zone data as [SinkMap_01]; Zone field as Value; Input value raster as [SinkDifference_01]; Output table as [ZonalSt_Diff]; Statistics type as MIN_MAX_MEAN.

Task 3.8: Copy Statistics from Temporary Tables

Task Overview

Copy sink statistics from temporary table to the [SinkMap_01] raster attribute table. For each temporary sink statistics table, do the following:

Step 3.8.1: Define Inner Join

Define an inner join on [SinkMap_01] to the temporary sink statistics table. In ArcMap, use the Join Data window or use the Add Join tool (Data Management toolbox, Joins toolset) with the following parameters: Layer Name as [SinkMap_01]; Input

`Join Field` as `Value`; `Join Table` as the sink statistics table (ex: `[ZonalSt_Diff]`); `Output Join Field` as `Value`; `Keep All Target Features` unchecked.

Step 3.8.2: Copy Values

Copy the values from the joined table to the appropriate field (ex: `MEAN` to `VAT_SinkMap_01.DepthMean`) using the Field Calculator window.

Step 3.8.3: Clean-up

Remove the join. Double-check that no sinks have blank values. Remove the temporary sink statistics tables.

Task 3.9: Classify Sinks: Minor by Attribute

Task Overview

Classify sinks in `[SinkMap_01]` via updating the `"Drain"` field. This task and the following will use three approaches to classifying sinks: by attributes, by location relative to other features and raster zones, and by manual classification.

Go Further...

Before proceeding, consider both additional classification options and the order in which the are best executed. Some options include:

1. ["Drain" = F] "DepthMax" < level of vertical inaccuracy.
2. ["Drain" = F] Sink notably outside area of interest (ex: Across a waterbody that serves as the county boundary).
3. ["Drain" = T] Sink overlaps waterbodies.
4. ["Drain" = T] Area or volume >= X value. (Using "Count", "DepthMean")
5. OR ["Drain" = F] Area or volume <= X value.
6. ["Drain" = T] Sink within 40' or 80' from NHD lines.
7. OR ["Drain" = F] Sink beyond 40' or 80' from NHD lines.
8. ["Drain" = F] "ElevMin" < sea level or minimum elevation.
9. ["Drain" = F] Location matches undrainable location (ex: quarries, sinkholes)

Step 3.9.1: Classify Minor Sinks To Be Filled

Set "Drain" = 0 for any sinks that are indistinguishable from noise (i.e. those with extremely small areas or maximum depths). Use the Field Calculator dialog directly or use the Calculate Field tool (Data Management toolbox, Fields toolset) with the following parameters: Input Table as [SinkMap_01]; Field Name as Drain; Expression Type as Python. Expression and the optional Code Block will vary, as explained below.

Set Expression as follows, where "ClassifySink" is the name of a function and the names between the exclamation points are those of the fields to be checked:

```
ClassifySink(!Count!, !DepthMax!)
```

In `Code Block`, define the function that's called in `Expression`:

```
def ClassifySink(Count, DepthMax):
  # Set Drain = 0 if area or maximum depth is noise.
  if Count <= 1 or DepthMax <= [CVA]:
    return 0
  # Otherwise, reset Drain to NULL.
  else:
    return None
```

Be aware that the dialog will sometimes reset `Expression Type` to `VB`.

Eliminating single-cell sinks and those with depths not greater than your data's `[CVA]` is a good minimum. If you'd like to eliminate more, start with small values and work up. Remember, the goal is not to classify all sinks, but to filter-out those not worth consideration.

Go Further...

Adapt the above code block or define your own scripts. If you're not familiar with Python, peruse the documentation at python.org. See the following page for comparisons in Python:

https://docs.python.org/2/reference/expressions.html#not-in

Take Note

The most important thing to know about copying any Python examples in this guide is that leading spaces define blocks of code. Leading spaces, however, are not always retained when copying and pasting.

Task 3.10: Classify Sinks: Waterbodies by Overlap

Task Overview

Classify sinks in `[SinkMap_01]` that overlap `[WaterbodyPolygons]` as to be drained (`"Drain"` = `1`).

Take Note

Not all waterbodies contribute to downhill surface flow, such as evaporation ponds and flooded quarries. Afterwards, these features may be classified as `0` or `2`.

Step 3.10.1: Identify Sinks with Waterbodies

Use the Extract by Mask tool (Spatial Analyst toolbox, Extraction toolset) with the following parameters: `Input raster` as `[SinkMap_01]`; `Input mask data` as `[WaterbodyPolygons]`; `Output raster` as `[SinkMap_01Waterbodies]`.

In my attempt, this step ran for 1h 45m on three feature classes that made up `[WaterbodyPolygons]`.

Step 3.10.2: Limit Sink Map Identified Sinks

Define an inner join on `[SinkMap_01]` to the `[SinkMap_01Waterbodies]` attribute table. In ArcMap, use the Join Data window or use the Add Join tool (Data Management toolbox, Joins toolset) with the following parameters: Layer Name as `[SinkMap_01]`; `Input Join Field` as `Value`; `Join Table` as `[SinkMap_01Waterbodies]`); `Output Join Field` as `Value`; `Keep All Target Features` unchecked.

Step 3.10.3: Classify Identified Sinks

Set `"Drain"` = `1` for matched sinks. Use the Field Calculator dialog directly or use the Calculate Field tool (Data Management toolbox, Fields toolset), as in the previous step. Set `Expression` to `1`; leave `Code Block` empty.

Step 3.10.4: Clean-Up

Remove join from `[SinkMap_01]`.

Remove `[SinkMap_01Waterbodies]`.

Task 3.11: Manually Classify Undrainable Sinks

Task Overview

Spot check and manually classify sinks that cannot be drained through the dam-breaching process (ex: those whose `"ElevMin"` < sea level or minimum elevation, or those that match an undrainable location (ex: quarries, sinkholes). This will be done similarly the step where sinks are matched to waterbodies, except you will be actively defining and editing the associated polygon. The intent is to have polygons with `Drain` classification intersect existing raster zones such that no raster zone is overlapped by two polygons with different `Drain` classifications. Having a polygon with the bounds of a sink but significantly smaller is fine as the classified areas remain that of the raster zones. Having a polygon larger than the target sink, can cause unexpected issues.

Step 3.11.1: Extract Very Low Elevation Sinks

Extract sinks with anomalous minimum elevations (ex: below sea level) as `[SinkUnusual]`. Use the Extract by Attributes tool (Spatial Analyst toolbox, Map Algebra toolset) with the following parameters: `Input raster` as `[SinkMap_01]`; `Where clause` as "`ElevMin < -2.585`"; `Output raster` as `[SinkMap_01Unusual]`.

> **Take Note**
>
> The above example takes the minimum drainable elevation on my data (-2.875 feet, not 0) and subtracts the vertical accuracy value I used previously. The value you use should be appropriate to your data.

Manual classification should only be done on a select handful of sinks. Check the number of extracted sinks. If your criteria extracts too many sinks, change or add to the parameters.

Step 3.11.2: Create Polygon Class for Manual Classifications

Create [SinkManualCheck], a polygon version of [SinkMap_01Unusual]. Use the Raster to Polygon tool (Conversion toolbox, Raster toolset) with the following parameters: Input raster as [SinkUnusual]; Field as Value; Output polygon features as [SinkManualCheck]; Simplify polygons as unchecked.

Be aware that Raster to Polygon will both split raster zones into separate polygons and also not retain the raster attribute table; both these issues will be addressed in the next few steps.

Step 3.11.3: Merge Split Sinks in Manual Check Class

Merge split sink polygons back together. Use the Dissolve tool (Data Management toolbox, Generalization toolset) with the following parameters: Input Features as [SinkManualCheck]; Output Feature Class as [SinkManualCheck2]; Dissolve_Field(s) with gridcode checked; Create multipart features checked.

Step 3.11.4: Restore Raster Attributes to Manual Check Class

Add the prior raster attribute fields to [SinkManualCheck2]. Use the Join Field tool (Data Management toolbox, Joins Toolset) with the following parameters: Input Table as [SinkManualCheck2]; Input Join Field as gridcode; Join Table as [SinkMap_01Unusual]; Output Join Field as Value; Join Fields as all fields selected.

Step 3.11.5: Symbolize Manual Check Class

Symbolize the features of [SinkManualCheck2] by the Drain field. Add the populated values (0, 1, 2, &, optionally, 3) if they're not already present, and edit the labels to be meaningful. Let the <NULL> value be symbolized as <all other

values> and select a contrasting, garish, awful color and pattern. Set the transparency to 50% or so, and adjust the layer draw order so you can easily compare against any reference data.

Step 3.11.6: Classify Identified Sinks

Start editing in the source containing [SinkManualCheck2]. Under the Create Features window, check that templates each of the populated Drain values exist and are correct. Delete and recreate the templates under the Organize Templates button if needed.

Inspect and categorize the Drain field for each polygon in [SinkManualCheck2]. For each row in [SinkManualCheck2], select and pan to the polygon. Select and edit the Drain field using the Attribute window. Visually inspect against imagery, terrain data, and any other reference data. For sinks containing other sinks, classify so to eliminate one of the sinks, leaving the others to be drained in following iterations. Remember that classifying a sinks as 0 will fill all of the target sink. For sinks that won't be correctly addressed by this approach, consider drawing a polygon mask to create a copy of [TopoDEM] with the deepest part of the sink either copied from [HydroCondDEM_01], filling it, or set as NoData, draining it.

Step 3.11.7: Delineate Additional Sinks

If appropriate and if you have reference data, draw and classify new polygons inside any other sinks that require special attention (ex: water treatment plants, outdoor pools, waste retention ponds).

Step 3.11.8: Apply Classifications Back To Sink Map

Update the Drain field in [SinkMap_01] to match those of the overlapping polygons in [SinkManualCheck2].

For existing polygons, create an inner join on `[SinkMap_01]` to `[SinkManualCheck2]` and copy `Drain`.

> **Take Note**
>
> This repeatedly failed, apparently due to the size of `[SinkMap_01]`. Since manual classifications should apply to a handful of sinks, individual manual edits will also work outside of a join, just be sure to double-check values entered.

If you created new polygons, then, for each classification in `[SinkManualCheck2]`, do the following:

Create a new feature class or layer of `[SinkManualCheck2]` filtered to that single classification of `Drain`.

Extract sinks from `[SinkMap_01]` that are overlapped by these polygons using the Extract by Mask tool (Spatial Analyst toolbox, Extraction toolset).

Create an inner join on `[SinkMap_01]` to this extract, and set `Drain` to match the intended classification

Task 3.12: Remove To-Be-Filled Sinks and Update Sink Map

Task Overview

In this task, create a copy of the elevation surface, removing all sinks you marked to be filled, and create a new copy of the sink map, updated to reflect theses changes. This will create the necessary inputs required for creating sink drains in the next task.

Step 3.12.1: Create Map Of Sinks To Be Filled

Create `[SinkMap_01Fill]`, a copy of `[SinkMap_01]` with all sinks classified as `Drain` = `0` removed. Use the Extract by Attributes tool (Spatial Analyst toolbox, Map Algebra toolset) with the following parameters: `Input raster` as `[SinkMap_01]`;

Where clause as "(Drain = 0) AND (Drain IS NOT NULL)"; Output raster as [SinkMap_01Fill].

Step 3.12.2: Create Map Of Sinks To Be Drained

Create [SinkMap_01Smooth], a copy of [SinkMap_01] with all sinks classified as Drain = 0 removed. Use the Extract by Attributes tool (Spatial Analyst toolbox, Map Algebra toolset) with the following parameters: Input raster as [SinkMap_01]; Where clause as "(Drain <> 0) OR (Drain IS NULL)"; Output raster as [SinkMap_01Smooth].

Step 3.12.3: Create Raster Of Cells To Be Filled

Create a raster of the elevation values in the sinks that are to be filled, [TopoDEM_01FilledCells]. Use the Extract by Mask tool (Spatial Analyst toolbox, Extraction toolset) with the following parameters: Input raster as [HydroCondDEM_01]; Input raster mask data as [SinkMap_01Fill]; Output raster as [TopoDEM_01FilledCells].

This will be used in the following step to create an elevation surface with these sinks removed.

Step 3.12.4: Create Terrain With To-Be-Filled Sinks Removed

Create [TopoDEM_01Smooth], a copy of [TopoDEM] with sinks classified as Drain = 0 filled with cells from [HydroCondDEM_01]. Use the Mosaic to New Raster tool (Raster Dataset toolset, Data Management toolbox)owing parameters: Input rasters as [TopoDEM] and [TopoDEM_01FilledCells]; Raster Dataset Name as [TopoDEM_01Smooth]; Pixel Type as [Pixel Type] (ex: 32_BIT_FLOAT); Number of Bands as 1; Mosaic Method as MAXIMUM.

Step 3.12.5: Rebuild Raster Attribute Table

Rebuild the attribute table on `[SinkMap_01Smooth]` using the Build Raster Attribute Table tool (Data Management toolbox, Raster Properties toolset).

Step 3.12.6: Repopulate Raster Attribute Table

Repopulate the attribute table in `[SinkMap_01Smooth]` with the fields from `[SinkMap_01]`. Use the Join Field tool (Data Management toolbox, Joins Toolset) with the following parameters: `Input Table` as `[SinkMap_01Smooth]`; `Input Join Field` as `Value`; `Join Table` as `[SinkMap_01]`; `Output Join Field` as `Value`; `Join Fields` as all fields selected except those already present in `[SinkMap_01Smooth]`.

Task 3.13: Generate Sink Drain Data

Task Overview

Run the Python script "DrainSink.py" (found in Appendix B) generating path drain data.

> **Take Note**
>
> The development of this guide did not progress beyond this task (11/2015). The Python script mostly works, but it lacks polish and does not merge individual sink drains into two outputs, despite what the guide says.

Step 3.13.1: Run "DrainSink.py" Python Script

Create `[Drain_01PathElevations]`, a raster of sink drain path elevations; `[Drain_01InternalMap]`, a raster map of sink internal drains; and `[Drain_01Errors]`, a tab-separated value file of undrained sinks. Use the Python script "DrainSink.py" in Appendix B, edited to use the following parameters: `sink_map` as `[SinkMap_01Smooth]`; `topo_dem` as `[TopoDEM_01Smooth]`; `v_accuracy` as `[CVA]`; `d_limit` as `200 meters` or `626.168 ft`, depending on

Take Note

This script will be processor-intensive and take days to complete. For example, my processing of ~270K sinks on a 2011 8-core laptop took ~8 days.

Also, this script has an issue with multiprocessing conflicts wherein one or more processes will fail early on. However, if one waits a bit and then restarts the script, my experience is that, after a few attempts, all the processes spawned by the script will remain functional.

the horizontal units of `[TopoDEM_01Smooth]`.

Step 3.13.2: Address Undrained Sinks

Inspect and address any sinks listed within `[Drain_01Errors]`. Either re-run the script with a temporarily larger `d_limit` value, or plan to manually drain or exclude the sink on the following iteration of this job.

To quickly inspect undrained sinks, try defining an inner join on the polygon version of `[SinkMap_01Smooth]`, created as a scratch feature class by "DrainSink.py".

Step 3.13.3: Apply Drains To Terrain Surface

Create `[TopoDEM_01Drained]` by applying `[Drain_01PathElevations]` elevations and `[Drain_01InternalMap]` mask as NoData values to `[TopoDEM_01Smooth]`. Use Raster Calculator (Spatial Analyst toolbox, Map Algebra toolset) with the following expression:

```
Con("[TopoDEM_01Smooth]", Con("[Drain_01InternalMap]",
NoData, Con("[Drain_01PathElevations]",
"[Drain_01PathElevations]", "[TopoDEM_01Smooth]")))
```

Alternately, use the following expression:

```
Con("[TopoDEM_01Smooth]" IS NOT NULL,

Con("[Drain_01InternalMap]" IS NOT NULL, NoData,

Con("[Drain_01PathElevations]" IS NOT NULL,

"[Drain_01PathElevations]", "[TopoDEM_01Smooth]")))
```

Step 3.13.4: Clean-up Script Scratch Data

The Python script creates a very large amount of scratch data. Delete or move it to an external drive.

Task 3.14: Repeat Job

Task Overview

Repeat this job, incrementing names from “01” to “02” and so on, until no sinks remain. At that point, re-name `[TopoDEM_01Drained]` as `[HydroEnfDEM]`, remove the intermediate data, and proceed to the next job.

Job 4: Generate Stream Network

Job Overview

Generate a new linear stream network from `[HydroEnfDEM]`.

Job 5: Classify Stream Network By Terrain

Job Overview

Split, classify, and applying attributes to the generated stream network. It's important to do so first based on data of similar accuracy, resolution, and temporal provenance, for obvious reasons. In this job, you will split and classify stream network segments based on your reference terrain data.

Job 6: Simplify Stream Network

Job Overview

Generating a stream network may produce a point dense stream network that should be

generalized, removing extraneous points and smoothing.

Different line classifications may be generalized in slightly different ways. Artificial paths through water bodies should approximate the feature's centerline and use only straight segments. Underground conduits should, by default, be single, straight line segments.

Job 7: Populate Stream Network With NHD Attributes

Job Overview

Populate the simplified stream network with attributes from the NHD, such as names and reach codes .

Closing Comments

While this guide is incomplete, I would encourage any readers to explore other tools that can create a hydro-enforced DEM.

One worth a look is the Optimized Tool for DEM Pit Removal, found in a March 5th, 2013 ESRI blog post. For reference, the related term paper can be found at this URL: www.ce.utexas.edu/prof/maidment/giswr2012/termpaper/jackson.pdf

This tool will still require undrainable sinks to be filtered-out. In addition, the maximum raster size it can process is 50,000,000 cells, so you will need a plan to split your terrain data, process it, and merge the pieces back into a processed whole. Finally, this only works the older USGS ASCII DEM format; search for a raster-to-ASCII tool for the conversion.

Appendix B: DrainSink.py

Below is the Python code for the script "DrainSink.py". To assemble it, copy and paste the lines below into a Python editing environment such as PyScripter. Be careful about preserving leading spaces!

Be sure to update the paths to ones appropriate to your data before executing!

```
# -*- coding: utf-8 -*-
# ---------------------------------------------------------------------#------#
# DrainSink_v0_09.py
# Created on: 2015-11-12 (still beta)
# Original Author: Nicholas McKenny, County of Prince George, Virginia
# Usage: DrainSink_v0_09.py
# Description: Generate drain_paths, a raster of drain elevations, and
#                generate drain_holes, a raster mask of future drain holes.
#              Intermediate output include sink_map_poly and two directories
#                of individual sink drain rasters.
#              WARNING: Does not yet successfully output merged sink drains.
# User Notes:
#   Lines limited to 79 characters; 72 where possible.
#   Users should set own default values in main() function under
#     "Settings: User parameters".
#   Users should also double-check any modifications (ex: field names).
#   Users should be careful about existing drain_paths and drain_holes files.
#   This script has an issue with multiprocessing conflicts wherein one or more
#     processes will fail early on. However, if one waits a bit and then
#     restarts the script, my experience is that, after a few attempts, all the
#     processes spawned by the script will remain functional. (v0_09)
# Analysis Notes (v0_08):
#  2015.11.01: In multiprocessing, persistent error around ExtractByMask()
#                in process_row():
#                  "FATAL ERROR (INFADI)"
#                  "MISSING DIRECTORY"
#                Appears stable after a few restarts.
#              Multiprocessing changes make each drain take ~2-4x as long.
#              On restarts, skipping rows creates incorrect time estimates.
#  2015.11.10: Process took 8 days. Complete and crashed python.exe instead of
#              waiting for acknowledgment
# Analysis Notes (v0_09):
#  2015.11.12: Python.exe failure at drain_path merge; likely too large.
# v0_10 Revision Goals:
# - Exclude skipped sinks from time estimates. Variable ideas:
#   - [adjusted_start_time]:
#   - [skipped row count]:
# - Fix "ExtractByMask()" multiprocessing conflict.
#   - If processes fail, script becomes stable after a few restarts.
#   - Potential to detect and restart processes automatically
# ---------------------------------------------------------------------#------#

# Import modules
import multiprocessing, os.path, time
import arcpy

# Check out Spatial Analyst license
arcpy.CheckOutExtension("spatial")

# ---------------------------------------------------------------------#------#
```

```
def main():
    # Main script body

    # Script settings
    settings = {}

    # Settings: Debugging and Configuration
    settings["test_debug"] = 0 # TESTING: Debugging off
    ##settings["test_debug"] = 1 # TESTING: Debugging on, no output
    ##settings["test_debug"] = 2 # TESTING: Debugging on, test output only
    settings["test_kill_switch"] = 3 # TESTING: Iteration limit
    settings["test_name_base"] = "C:\\Users\\Nicholas\\Documents\\ArcGIS\\" \
        "Scratch\\Scratch.gdb\\test_" # TESTING: Test output name base
    ##settings["use_mp"] = 0 # CONFIG: Multiprocessing off (slow)
    settings["use_mp"] = 1 # CONFIG: Multiprocessing on (mostly not broken)

    # Settings: User parameters, input
    settings["sink_map"] = "C:\\Users\\Nicholas\\Desktop\\Capstone Project\\" \
        "Capstone GIS Data\\StreamUpdateArcGIS\\StreamUpdateDB.gdb\\" \
        "SinkMap_01Smooth" # SinkMap_01Smooth
    settings["topo_dem"] = "C:\\Users\\Nicholas\\Desktop\\Capstone Project\\" \
        "Capstone GIS Data\\StreamUpdateArcGIS\\StreamUpdateDB.gdb\\" \
        "TopoDEM_01Smooth" # TopoDEM_01Smooth
    settings["v_accuracy"] = 0.61 # VAccuracy, vertical accuracy value
    settings["d_limit"] = 626.168 # DLimit, max drain path length
        # (default: 200 m in ft)

    # Settings: User parameters, output
    settings["sink_map_poly"] = "C:\\Users\\Nicholas\\Documents\\ArcGIS\\" \
        "Scratch\\Scratch.gdb\\SinkMap_Poly" # sink_map_poly, polygon sink_map
    settings["tmp_dir"] = "C:\\Users\\Nicholas\\Documents\\ArcGIS\\" \
        "Scratch\\Drain_Intermediate" # Script-managed directory
    settings["drain_paths"] = "C:\\Users\\Nicholas\\Documents\\ArcGIS\\" \
        "Scratch\\Scratch.gdb\\Drain_PathElevations" # drain_paths, drain paths
    settings["drain_holes"] = "C:\\Users\\Nicholas\\Documents\\ArcGIS\\" \
        "Scratch\\Scratch.gdb\\Drain_InternalMap" # drain_holes, internal drain
    settings["drain_errors"] = "C:\\Users\\Nicholas\\Documents\\ArcGIS\\" \
        "Scratch\\Drain_Errors.txt" # TSV file of undrained sinks
    # NB: Other settings added below, after sink_map_poly is checked/created.


    # Verify input exists (sink_map, topo_dem)
    if not arcpy.Exists(settings["sink_map"]):
        print ("Aborting: 'sink_map' not found at '{0}'." .format(
            settings["sink_map"]))
        return
    elif not arcpy.Exists(settings["topo_dem"]):
        print ("Aborting: 'topo_dem' not found at '{0}'." .format(
            settings["topo_dem"]))
        return

    # Create sink_map_poly if it does not already exist
    if not arcpy.Exists(settings["sink_map_poly"]):
        scratch_1_path = arcpy.CreateUniqueName("Scratch1",
                                                arcpy.env.scratchGDB)
        scratch_2_path = arcpy.CreateUniqueName("Scratch2",
                                                arcpy.env.scratchGDB)
        arcpy.RasterToPolygon_conversion(settings["sink_map"], scratch_1_path,
                                         "NO_SIMPLIFY", "Value")
        arcpy.Dissolve_management(scratch_1_path, scratch_2_path, "gridcode")
        arcpy.JoinField_management(scratch_2_path, "gridcode",
```

```
                                settings["sink_map"], "Value")
    arcpy.Sort_management(scratch_2_path, settings["sink_map_poly"],
                          [["ElevMin", "DESCENDING"]])
    arcpy.Delete_management(scratch_1_path)
    arcpy.Delete_management(scratch_2_path)

# Add "DrainStatus" field if not already in sink_map_poly
if len(arcpy.ListFields(settings["sink_map_poly"], "DrainStatus")) == 0:
    arcpy.AddField_management(settings["sink_map_poly"], "DrainStatus",
                              "LONG")

# Create tmp_dir if it does not already exist
if not arcpy.Exists(settings["tmp_dir"]):
    arcpy.CreateFolder_management(*os.path.split(settings["tmp_dir"]))

# Create/Reset drain_errors file and verify creation.
f = open(settings["drain_errors"], "w")
f.write("OBJECTID\tDrain\n")
f.close
if not arcpy.Exists(settings["drain_errors"]):
    print ("Aborting: 'drain_errors' not created at '{0}'." .format(
        settings["drain_errors"]))
    return


# Settings: Environment and Other settings
pcode = {'U1':'1_BIT','U2':'2_BIT','U4':'4_BIT',
         'U8':'8_BIT_UNSIGNED', 'S8':'8_BIT_SIGNED',
         'U16':'16_BIT_UNSIGNED', 'S16':'16_BIT_SIGNED',
         'U32':'32_BIT_UNSIGNED','S32':'32_BIT_SIGNED',
         'F32':'32_BIT_FLOAT', 'F64':'64_BIT'}

arcpy.env.workspace = "in_memory" # Prevent unneeded file creation
arcpy.env.overwriteOutput = True # Careful...
arcpy.env.cellSize = arcpy.Raster(settings["topo_dem"]).meanCellHeight
arcpy.env.outputCoordinateSystem = \
    arcpy.Describe(settings["topo_dem"]).spatialReference

settings["lock"] = multiprocessing.Lock()
settings["process_num"] = 0 # Process number (main)
settings["cell_size"] = float(arcpy.env.cellSize)
settings["spref"] = arcpy.env.outputCoordinateSystem
settings["d_limit_grid"] = int(settings["d_limit"] /
                               float(arcpy.env.cellSize)) \
                           * float(arcpy.env.cellSize)
settings["cur_fields"] = ["OID@", "SHAPE@", "Drain", "ElevMin"]
settings["cur_where"] = "({0} IS NULL) AND ({1} IN (NULL, 1, 2)".format(
    arcpy.AddFieldDelimiters(settings["sink_map_poly"], "DrainStatus"),
    arcpy.AddFieldDelimiters(settings["sink_map_poly"], "Drain"))
settings["rcount_this"] = 0 # Rows processed
settings["rcount_total"] = 0 # Total row count (placeholder)
settings["tprocess_start"] = 0 # Process start time (placeholder)
settings["tloop_start"] = 0 # Row process start time (placeholder)
settings["tmp_gdb_template"] = "{0}\\{1}-{2}.gdb" # Temp file gdb template
settings["tmp_file_template"] = "{0}\\{1}-{2}" # Temp file template
settings["tmp_data_template"] = "" # Temp data template (placeholder)


# Set-up output: Create drain_paths if it does not already exist.
if not arcpy.Exists(settings["drain_paths"]):
    # NB: Copy Raster avoids creation issue where, later, Append fails.
    scratch_3_path = "in_memory\\Scratch3"
    arcpy.CreateRasterDataset_management(
```

```
        "in_memory", "Scratch3", arcpy.env.cellSize,
        pcode[arcpy.Raster(settings["topo_dem"]).pixelType],
        arcpy.Raster(settings["topo_dem"]).spatialReference, 1)
    arcpy.CopyRaster_management(
        scratch_3_path, settings["drain_paths"],
        pixel_type = pcode[arcpy.Raster(settings["topo_dem"]).pixelType])
    arcpy.Delete_management(scratch_3_path)


# Set-up output: Create drain_holes if it does not already exist
if not arcpy.Exists(settings["drain_holes"]):
    # NB: Copy Raster avoids creation issue where, later, Append fails.
    scratch_4_path = "in_memory\\Scratch4"
    arcpy.CreateRasterDataset_management(
        "in_memory", "Scratch4", arcpy.env.cellSize,
        "32_BIT_SIGNED",
        arcpy.Raster(settings["topo_dem"]).spatialReference, 1)
    arcpy.CopyRaster_management(
        scratch_4_path, settings["drain_holes"],
        pixel_type = "32_BIT_SIGNED")
    arcpy.Delete_management(scratch_4_path)


# ---------------------------------------------------------------------#------#
# For each sink in sink_map_poly, seek a drain path

# Count total rows. Format temporary output template strings.
arcpy.MakeTableView_management(settings["sink_map_poly"],
                               "in_memory\\Scratch5",
                               "")
settings["rcount_total"] = int(
    arcpy.GetCount_management("in_memory\\Scratch5").getOutput(0))
arcpy.Delete_management("in_memory\\Scratch5")
settings["tmp_data_template"] = \
    "{0}\\{1}_{2:0" + str(len(str(settings["rcount_total"]))) + "}"

# Set-up multiprocessing (if settings["use_mp"] == 1)
if settings["use_mp"] == 1:
    # Multiprocessing: Set number of processes to be utilized.
    process_count = multiprocessing.cpu_count() - 1 # Unhandled error?
    if process_count < 2:
        process_count = 2

    # Multiprocessing: Create row number ranges list in single-item tuples.
    # NB: Assumes row IDs match row count from 1 to rcount_total.
    # Example: r_ranges = [([1, 4559],), ([4560, 9118],), ([9119, 13677],)]
    r_ranges = []
    i = 1
    while i <= process_count:
        range_f = (
            (i - 1) * int(settings["rcount_total"] / process_count)
            ) + 1

        if i == process_count:
            range_t = settings["rcount_total"]
        else:
            range_t = i * int(settings["rcount_total"] / process_count)

        r_ranges.append(([range_f, range_t],))
        i += 1

    # Multiprocessing: Assign each row range to a child process.
    mp_processes = []
```

```
    for r_range in r_ranges:
        settings["process_num"] += 1
        temp_process = multiprocessing.Process(
            target=process_range,
            args=r_range,
            kwargs=settings)
        temp_process.start()
        mp_processes.append(temp_process)
    settings["process_num"] = 0

    # Multiprocessing: Block this process until child processes complete.
    for this_process in mp_processes:
        this_process.join()

else:
    # Run as single process.
    process_count = 0
    r_range = [1, settings["rcount_total"]]
    process_range(r_range, **settings)

# ----------------------------------------------------------------#------#

# OUTPUT: Update "DrainStatus" field. Create output data from temp data.
if settings["test_debug"] == 0:
    # Merge "drain_errors" contents of tmp_dir to drain_errors
    settings["lock"].acquire()
    print("Processes complete. Merging drain_errors files.")
    settings["lock"].release()

    f = open(settings["drain_errors"], "w")
    f.write("OBJECTID\tDrain\n")
    i = min(1, process_count)
    while i <= process_count:
        g = open(settings["tmp_file_template"].format(
            settings["tmp_dir"], "tmp_errors", i))
        tmp_error_lines = g.readlines()
        if len(tmp_error_lines) > 1:
            f.writelines(tmp_error_lines[1:])
        g.close()
        i += 1
    f.close()

    # DEVELOPMENT: Mark sink_map_poly "DrainStatus" as 1 where processed.
    # Try Calculate Field tool (see Step 3.9.1)
    #arcpy.CalculateField_management(), maybe on a table view.
    pass

    # DEVELOPMENT: WorkspaceToRasterDataset_management can't hack it.
    # Merge "internal drain" contents of tmp_dir to drain_holes
    settings["lock"].acquire()
    print("Merging drain_errors files complete. Merging drain_holes.")
    settings["lock"].release()

    i = min(1, process_count)
    while i <= process_count:
        settings["lock"].acquire()
        print("Merging drain_holes from process {0}.".format(i))
        settings["lock"].release()

        arcpy.WorkspaceToRasterDataset_management(
            settings["tmp_gdb_template"].format(
                settings["tmp_dir"], "tmp_dhole", i),
            settings["drain_holes"],
```

```
                "INCLUDE_SUBDIRECTORIES", "MINIMUM")
            i += 1

        # DEVELOPMENT: WorkspaceToRasterDataset_management can't hack it.
        # Merge "drain path" contents of tmp_dir to drain_paths
        settings["lock"].acquire()
        print("Merging drain_paths complete. Merging drain_paths.")
        settings["lock"].release()

        i = min(1, process_count)
        while i <= process_count:
            settings["lock"].acquire()
            print("Merging drain_paths from process {0}.".format(i))
            settings["lock"].release()

            arcpy.WorkspaceToRasterDataset_management(
                settings["tmp_gdb_template"].format(
                    settings["tmp_dir"], "tmp_dpath", i),
                settings["drain_paths"],
                "INCLUDE_SUBDIRECTORIES", "MINIMUM")
            i += 1


    # Await user acknowledgment
    try:
        settings["lock"].acquire()
        input("Script reached end. Press Enter/Return to exit.")
        settings["lock"].release()
    except:
        pass
    # End main


def process_range(r_range, **kwargs):
    # Process a range of rows as a child process. "settings[]" in kwargs.

    # Get settings[] and set arcpy environment
    settings = kwargs
    arcpy.env.workspace = "in_memory" # Prevent unneeded file creation
    ##arcpy.env.scratchWorkspace = "in_memory" # NEVER use in_memory as scratch
    arcpy.env.overwriteOutput = True # Careful...
    arcpy.env.cellSize = settings["cell_size"]
    arcpy.env.outputCoordinateSystem = settings["spref"]


    # Update settings. Create intermediate output File GDBs and workspaces.
    settings["rcount_total"] = r_range[1] - r_range[0] + 1

    # Define process-unique scratch in attempt to avoid this error:
    #   "FATAL ERROR (INFADI)" "MISSING DIRECTORY".
    settings["scratch_dir"] = "{0}\\Scratch-{1}".format(
        settings["tmp_dir"], settings["process_num"])
    settings["scratch_gdb"] = "{0}\\scratch.gdb".format(
        settings["scratch_dir"])
    if not arcpy.Exists(settings["scratch_dir"]):
        arcpy.CreateFolder_management(
            *os.path.split(settings["scratch_dir"]))
    if not arcpy.Exists(settings["scratch_gdb"]):
        arcpy.CreateFileGDB_management(
            *os.path.split(settings["scratch_gdb"]))
    arcpy.env.scratchWorkspace = settings["scratch_dir"]

    settings["tmp_dpath_gdb"] = settings["tmp_gdb_template"].format(
```

```
    settings["tmp_dir"], "tmp_dpath", settings["process_num"])
if not arcpy.Exists(settings["tmp_dpath_gdb"]):
    arcpy.CreateFileGDB_management(
        *os.path.split(settings["tmp_dpath_gdb"]))
settings["tmp_dhole_gdb"] = settings["tmp_gdb_template"].format(
    settings["tmp_dir"], "tmp_dhole", settings["process_num"])
if not arcpy.Exists(settings["tmp_dhole_gdb"]):
    arcpy.CreateFileGDB_management(
        *os.path.split(settings["tmp_dhole_gdb"]))
settings["tmp_errors_file"] = settings["tmp_file_template"].format(
    settings["tmp_dir"], "tmp_errors", settings["process_num"])
f = open(settings["tmp_errors_file"], "w")
f.write("OBJECTID\tDrain\n")
f.close()


# TESTING: Output settings{} as used by this process.
if settings["test_debug"] > 0:
    settings["lock"].acquire()
    print("TESTING-{0}: r_range[{1}, {2}] with these settings:".format(
        settings["process_num"], r_range[0], r_range[1]))
    print("settings{")
    tmp_settings_keys = sorted(settings.iterkeys())
    for key in tmp_settings_keys:
        print("    {0}: {1}".format(key, settings[key]))
    print("}")
    settings["lock"].release()


# For each sink_map_poly sink in range, seek a drain path.
# ---------------------------------------------------------------#------#
child_where = "{0} AND ({1} BETWEEN {2} AND {3})".format(
    settings["cur_where"],
    arcpy.AddFieldDelimiters(settings["sink_map_poly"], "OBJECTID"),
    r_range[0], r_range[1])

with arcpy.da.SearchCursor(settings["sink_map_poly"],
                           settings["cur_fields"], child_where,
                           settings["spref"]) as cursor:

    settings["tprocess_start"] = time.time()

    for row in cursor:
        # Update progress variables
        settings["rcount_this"] += 1
        settings["tloop_start"] = time.time()

        # TESTING: Short-circuit increment and exit. Print row data.
        if settings["test_debug"] > 0:
            if settings["test_kill_switch"] <= 0:
                break
            else:
                settings["test_kill_switch"] -= 1

            settings["lock"].acquire()
            print("TESTING-{0}: "
                  "Entering iteration {1} of {2} on this row:".format(
                  settings["process_num"],
                  settings["rcount_this"], settings["rcount_total"]))
            print(row)
            settings["lock"].release()

        # Process row
```

```
            process_row(row, settings)
    # ---------------------------------------------------------------#------#
    # End process_range


def process_row(row, settings):
    # Process row. Save any output to individual file.

    try:
##    if 1 == 1: # TESTING: Skip 'try:'
        # Create temp file subdirectory and path string
        drain_processed = 0 # Success (1); Existing (0); Failure (-1)
        if (row[settings["cur_fields"].index("Drain")] == 2):
            drain_output_type = 2
            tmp_data_path = settings["tmp_data_template"].format(
                settings["tmp_dhole_gdb"], "dhole",
                row[settings["cur_fields"].index("OID@")])
        else:
            drain_output_type = 1
            tmp_data_path = settings["tmp_data_template"].format(
                settings["tmp_dpath_gdb"], "dpath",
                row[settings["cur_fields"].index("OID@")])

        # Skip row if output already exists.
        if arcpy.Exists(tmp_data_path):
            return


        # NOTES: (1) Create drain_scells mask
        # Extract topo_dem cells within this sink mask
        sink_topo = arcpy.sa.ExtractByMask(
            settings["topo_dem"], row[settings["cur_fields"].index("SHAPE@")])
        # DEVELOPMENT: 2015.11.01 (v8). The above "ExtractByMask()" call is
        #   the center of a persistent error in multiprocessing, on two lines:
        #   "FATAL ERROR (INFADI)" "MISSING DIRECTORY"
        #   The affected process may also raise an error that a file already
        #   exists, similar to this "<user>\appdata\local\temp\t_t03", but
        #   this may be after and an effect of the original problem.
        # UPDATE: Setting the scratch workspace to a FGDB sees this error.
        #         Setting the scratch workspace to a folder seemingly works.

        # TESTING: Save sink_topo to hard drive as 00_sink_topo
        if settings["test_debug"] > 1:
            settings["lock"].acquire()
            print("TESTING-{0}: Exporting sink_topo as 00_sink_topo.".format(
                settings["process_num"]))
            settings["lock"].release()
            sink_topo.save(settings["test_name_base"] + "00_sink_topo")

        # Calculate maximum drain source elevation
        drain_smax = float(row[settings["cur_fields"].index("ElevMin")]) \
                     + settings["v_accuracy"]

        # Extract drain_scells_big, mask of sink_topo cells <= drain_smax
        drain_scells_big = arcpy.sa.Con(sink_topo <= drain_smax, 1)

        # Create drain_scells. Trim drain_scells_big; extent matches sink_topo.
        # NB: No, I never did find a less stupid way to do this.
        drain_scratch6 = "{0}\\Scratch6_{1}".format(
            arcpy.env.workspace,
            row[settings["cur_fields"].index("OID@")])
        arcpy.RasterToPolygon_conversion(drain_scells_big, drain_scratch6,
                                         "NO_SIMPLIFY", "Value")
```

```
arcpy.env.extent = arcpy.Describe(drain_scratch6).extent
arcpy.Delete_management(drain_scratch6)
drain_scells = arcpy.sa.Con(drain_scells_big, drain_scells_big)

# TESTING: Save drain_scells to hard drive as 01_drain_scells
if settings["test_debug"] > 1:
    settings["lock"].acquire()
    print("TESTING-{0}: "
          "Exporting drain_scells as 01_drain_scells.".format(
          settings["process_num"]))
    settings["lock"].release()
    drain_scells.save(settings["test_name_base"] + "01_drain_scells")


# OUTPUT: For internal drains, save drain_scells and continue.
if drain_output_type == 2:
    if settings["test_debug"] == 0:
        drain_scells.save(tmp_data_path) # OUTPUT
    else:
        settings["lock"].acquire()
        print("TESTING-{0}: rain_scells format = '{1}'.".format(
            settings["process_num"], drain_scells.format))
        settings["lock"].release()
    return


# NOTES: (2) Create drain_cost, the cost surface
arcpy.env.extent = arcpy.Extent(
    drain_scells.extent.XMin - settings["d_limit_grid"],
    drain_scells.extent.YMin - settings["d_limit_grid"],
    drain_scells.extent.XMax + settings["d_limit_grid"],
    drain_scells.extent.YMax + settings["d_limit_grid"])
drain_cost = arcpy.sa.Con(settings["topo_dem"], 1)

# TESTING: Save drain_cost to hard drive as 02_drain_cost
if settings["test_debug"] > 1:
    settings["lock"].acquire()
    print("TESTING-{0}: Exporting drain_cost as 02_drain_cost.".format(
        settings["process_num"]))
    settings["lock"].release()
    drain_cost.save(settings["test_name_base"] + "02_drain_cost")


# NOTES: (3) Create drain_dcells mask
arcpy.env.extent = drain_cost
drain_dcells = arcpy.sa.Con(
    (arcpy.sa.IsNull(sink_topo)) & \
    (arcpy.Raster(settings["topo_dem"]) <= drain_smax), drain_cost)

# TESTING: Save drain_dcells to hard drive as 03_drain_dcells
if settings["test_debug"] > 1:
    settings["lock"].acquire()
    print("TESTING-{0}: "
          "Exporting drain_dcells as 03_drain_dcells.".format(
          settings["process_num"]))
    settings["lock"].release()
    drain_dcells.save(settings["test_name_base"] + "03_drain_dcells")


# NOTES: (4) Create drain_elev, a drain path with lowered elevation
drain_backlink = "{0}\\drain_backlink_{1}".format(
    arcpy.env.workspace,
    row[settings["cur_fields"].index("OID@")])
```

```
drain_distance = arcpy.sa.CostDistance(
    drain_scells, drain_cost, settings["d_limit"], drain_backlink)
drain_lcpath_big = arcpy.sa.CostPath(
    drain_dcells, drain_distance, drain_backlink, "EACH_ZONE", "Value")
arcpy.Delete_management(drain_backlink)

# Create drain_lcpath. Extent of drain_lcpath_big == drain_cost; trim.
# NB: Again, I never did find a less stupid way to do this.
drain_scratch7 = "{0}\\Scratch7_{1}".format(
    arcpy.env.workspace,
    row[settings["cur_fields"].index("OID@")])
arcpy.RasterToPolygon_conversion(drain_lcpath_big, drain_scratch7,
                                 "NO_SIMPLIFY", "Value")
arcpy.env.extent = arcpy.Describe(drain_scratch7).extent
drain_lcpath = arcpy.sa.Con(drain_lcpath_big, drain_lcpath_big)
arcpy.Delete_management(drain_scratch7)

# Create drain_elev w/ min elevation of sink and along drain_lcpath
drain_scratch8 = arcpy.sa.ExtractByMask(settings["topo_dem"],
                                        drain_lcpath)
drain_emin = row[settings["cur_fields"].index("ElevMin")]
if (drain_scratch8.minimum < drain_emin):
    drain_emin = drain_scratch8.minimum
drain_elev = arcpy.sa.Con(drain_lcpath, drain_emin)

# If drain_elev is blank, raise Exception and skip
if (arcpy.GetRasterProperties_management(
        drain_elev, "ALLNODATA").getOutput(0) == "1"):
    raise Exception

# TESTING: Save drain_distance to hard drive as 04_drain_distance
if settings["test_debug"] > 1:
    settings["lock"].acquire()
    print("TESTING-{0}: "
          "Exporting drain_distance as 04_drain_dist.".format(
          settings["process_num"]))
    settings["lock"].release()
    drain_distance.save(settings["test_name_base"] + "04_drain_dist")

# TESTING: Save drain_lcpath to hard drive as 05_drain_lcpath
if settings["test_debug"] > 1:
    settings["lock"].acquire()
    print("TESTING-{0}: "
          "Exporting drain_lcpath as 05_drain_lcpath.".format(
          settings["process_num"]))
    settings["lock"].release()
    drain_lcpath.save(settings["test_name_base"] + "05_drain_lcpath")

# TESTING: Save drain_elev to hard drive as 06_drain_elev
if settings["test_debug"] > 1:
    settings["lock"].acquire()
    print("TESTING-{0}: Exporting drain_elev as 06_drain_elev.".format(
        settings["process_num"]))
    settings["lock"].release()
    drain_elev.save(settings["test_name_base"] + "06_drain_elev")


# OUTPUT: For internal drains, save drain_elev and continue.
arcpy.env.extent = None
if settings["test_debug"] == 0:
    drain_elev.save(tmp_data_path) # OUTPUT
drain_processed = 1
```

```
    except:
##    else: # TESTING: Skip 'except:'
        # Mark row as undrained.
        drain_processed = -1


    finally:
##    if 1 == 1: # TESTING: Skip 'finally:'
        arcpy.env.extent = None

        if drain_processed == 1:
            # Print progress message
            print_progress(settings["process_num"], settings["lock"],
                           row[settings["cur_fields"].index("OID@")],
                           settings["rcount_this"], settings["rcount_total"],
                           settings["tprocess_start"], settings["tloop_start"],
                           time.time())
        elif drain_processed == 0:
            # Print "already processed" message
            settings["lock"].acquire()
            print("SKIPPING-{0}: "
                  "({1}) Drain path already found for row {2}.".format(
                  settings["process_num"], drain_processed,
                  row[settings["cur_fields"].index("OID@")]))
            settings["lock"].release()
        else:
            # Record row number (OBJECTID) and drain type (Drain)
            f = open(settings["tmp_errors_file"], "a")
            f.write("{0}\t{1}\n".format(
                row[settings["cur_fields"].index("OID@")],
                row[settings["cur_fields"].index("Drain")]))
            f.close()

            # Print warning message
            settings["lock"].acquire()
            print("WARNING-{0}: "
                  "({1}) Drain path not found for row {2}.".format(
                  settings["process_num"], drain_processed,
                  row[settings["cur_fields"].index("OID@")]))
            settings["lock"].release()
    # End process_row


def print_progress(process_num, process_lock, row_ID, rcount_this,
                   rcount_total, tprocess_start, tloop_start, tloop_end):
    # Print progress after each sink is processed.

    rcount_percent = rcount_this / float(rcount_total) * 100
    tloop_elapsed = tloop_end - tloop_start
    tprocess_elapsed = tloop_end - tprocess_start
    tprocess_remain = (tprocess_elapsed / rcount_this) * \
                      (rcount_total - rcount_this)
    tprocess_elapsed_str = "{0:0.0f}d {1:02.0f}h {2:02.0f}m {3:05.2f}s".format(
        tprocess_elapsed // 86400, (tprocess_elapsed % 86400) // 3600,
        (tprocess_elapsed % 3600) // 60, tprocess_elapsed % 60)
    tprocess_remain_str = "{0:0.0f}d {1:02.0f}h {2:02.0f}m {3:05.2f}s".format(
        tprocess_remain // 86400, (tprocess_remain % 86400) // 3600,
        (tprocess_remain % 3600) // 60, tprocess_remain % 60)

    tloop_msg = ("PROCESSED-{0}: Row {1} in {2:0.2f}s. "
                 "{3:6.2f}% complete ({4} of {5}). "
                 "{6} elapsed; roughly {7} left.")
```

```
    process_lock.acquire()
    print(tloop_msg.format(process_num, row_ID, tloop_elapsed,
                           rcount_percent, rcount_this, rcount_total,
                           tprocess_elapsed_str, tprocess_remain_str))
    process_lock.release()
    # End print_progress


# ------------------------------------------------------------------------#------#

if __name__ == '__main__':
    main()

# ------------------------------------------------------------------------#------#
```

Works Cited and Referenced

Anderson, Danny L. "Detailed Hydrographic Feature Extraction from High-Resolution LIDAR Data." ProQuest, UMI Dissertations Publishing, 2012. Web. Accessed 2014-10-10. < http://search.proquest.com.ezaccess.libraries.psu.edu/docview/1017732181?pq-origsite=summon >

ASPRS. *Manual of Airborne Topographic Lidar*. Ed. Michael S. Renslow. Bethesda, MD: American Society for Photogrammetry and Remote Sensing, 2012. Print.

Dewberry. FEMA LiDAR: Middle Counties. USGS, 2012. Lidar dataset. Accessed 2014-07-14. Retrieved from Virginia Lidar < http://www.virginialidar.com >

Dewberry. *Project Report for the Middle Counties Acquisition and Classification for FEMA VA LiDAR*. Dewberry, 2012. PDF document. Updated 2012-08-31. Accessed 2014-07-23. < https://54ede8be3c8882517d1439d2ac5a1ec7a3b5e801.googledrive.com/host/0B_5XlZJJ2R5tUGtOY3dtR3I0NGs/Metadata/Project_Report/Dewberry_ProjectReport_MiddleCounties.pdf >

Merriam-Webster. "Lidar." *Merriam-Webster.com*. Merriam-Webster, n.d. Web. 5 Dec. 2014. < http://www.merriam-webster.com/dictionary/lidar >

NOAA. United States Interagency Elevation Inventory. National Oceanic and Atmospheric Administration. Web. Accessed 2014-11-26. < http://coast.noaa.gov/inventory/ >

Poppenga, S. K., D. B. Gesch, and B. B. Worstell. "Hydrography Change Detection: The Usefulness of Surface Channels Derived from Lidar DEMs for Updating Mapped Hydrography" *Journal of the American Water Resources Association* 49.2 (2013): 371-89. *ProQuest.* Web. Accessed 2014-09-25. < http://search.proquest.com.ezaccess.libraries.psu.edu/docview/1463114804?pq-origsite=summon >

Poppenga, S.K., B.B. Worstell, J.M. Stoker, and S.K. Greenlee. "Using Selective Drainage Methods to Extract Continuous Surface Flow From 1-Meter Lidar-Derived Digital Elevation Data". *U.S. Geological Survey Scientific Investigations Report 2010-5059*, 12 pp. Accessed 2014-10-11. < http://pubs.er.usgs.gov/publication/ofr20105059 >

United States Geological Survey. *Hydrography: National Hydrography Dataset, Watershed Boundary Dataset*. USGS, n.d. Web. Updated 2013-01-13 11:16:15 EST. Accessed 2014-09-14. < http://nhd.usgs.gov/ >

United States Geological Survey. *The National Hydrography Dataset (NHD) Model (v2.2)*. USGS, 2014-06-27. PDF. Accessed 2014-09-14. < http://nhd.usgs.gov/NHDv2.2_poster_062614.pdf >

United States Geological Survey. *NHD User Guide*. USGS, n.d. Web. Updated 2013-01-14 17:16:05 EST. Assessed 2014-09-14. < http://nhd.usgs.gov/userguide.html >

United States Geological Survey. “NHD Frequently Asked Questions.” *Hydrography: National Hydrography Dataset, Watershed Boundary Dataset*. USGS, 2011-02-01. Web. Updated 2013-02-14 11:16:15 EST. Accessed 2014-09-20. < http://nhd.usgs.gov/nhd_faq.html >

United States Geological Survey. “NHD Tools.” *Hydrography: National Hydrography Dataset, Watershed Boundary Dataset*. USGS. Web. Updated 2014-12-04 16:11:18 EST.

Accessed 2014-12-07. < http://nhd.usgs.gov/tools.html >

United States Geological Survey. "National Elevation Dataset - NED Frequently Asked Questions (FAQ) - U.S. Geological Survey" *National Elevation Dataset*. USGS, 2014-10-06. Web. Accessed 2014-11-21. < http://ned.usgs.gov/faq.html >

United States Geological Survey. "NHDH_VA.gdb\Hydrography\NHDFlowline" *National Hydrography Dataset* (Virginia State Extract). USGS, n.d. ESRI File Geodatabase. Updated 2014-08-27. Accessed 2014-09-14.

Virginia Geographic Information Network "VBMP Orthophotography" *Virginia Geographic Information Network.* Virginia Information Technologies Agency, Commonwealth of Virginia, n.d. Web. Accessed 2014-12-07. < http://www.vita.virginia.gov/isp/default.aspx?id=8412 >

Virginia Geographic Information Network. 2007 Imagery Dataset for Prince George County. *2006/2007 VBMP Orthoimagery*. Virginia Geographic Information Network (VGIN), Richmond, Virginia, 2007-02. Orthoimagery dataset.

Virginia Geographic Information Network. 2013 Imagery Dataset for Prince George County. *2013/2016 VBMP Orthoimagery*. Virginia Geographic Information Network (VGIN), Richmond, Virginia, 2013-10-31. Orthoimagery dataset.

Virginia Lidar. n.p., 2014. Website. Updated 2014-06-30. Accessed 2014-07-14. < http://www.virginialidar.com >

Wikipedia. "National Lidar Dataset (United States)" *Wikipedia*. Web. Updated 2014-11-04 05:59. Accessed 2014-11-26. < http://en.wikipedia.org/wiki/National_Lidar_Dataset_(United_States) >

County of Prince George, Virginia. "Geographic Information System" *Prince George County, Virginia*. Prince George County, Virginia, n.d. Web. Accessed 2014-12-15. < *http://princegeorgecountyva.gov/Index.aspx?page=160* >