GeoTree downloader

GIS Programming term project by Nikolay Golosov

Introduction

For a long time, GEOTree is hosting a LIDAR dataset for entire Iowa state. Access to the dataset is provided using a convenient web interface. Users can select areas of interest, download LIDAR tiles, bare earth XYZI files and derived products, including DEM, Hillshade and Contours datasets.

Despite those advantages, users should manually select areas of interest from the website, and then manually download files from the provided list of files. This approach could be not convenient for users, used to work with ArcGIS all the time. During this term project, I decided to develop a tool, allowing the user to download products of interest directly from ArcGIS. It eliminates manual work and minimizes chances of downloading of unnecessary tiles, as we are downloading tiles failing into current map extent or extent of the requested layer only, and the user does not need to manually select areas of interest anymore, which potentially improves his/her performance.

Methods

Source data and tile naming convention

Regarding the data, we're using datasets, published on http://geotree2.geog.uni.edu/lidar/ website. The website is providing access to LiDAR LAS and Bare earth XYZI files. All the data are in NAD 83 UTM Zone 15N spatial reference (EPSG:26915). LiDAR data were collected in 2007 by Sanborn Map Company using ALS50 system. Bare earth XYZI(ASCII) files were produced by removing all non-ground features by TerraScan software. By manual downloading of first data tiles, I was able to figure out their naming convention, as no information in that is given in the source metadata. Data was tiled to 2x2 km tiles. The naming convention for the tile files is *0 & first three digits of UTM15N X coordinate & 4 first digits of the Y coordinate* of the center point of each tile where ampersand sign (&) means concatenation. The left top coordinate of the first

tile is X=210000, 4822000, and all the tiles from this seed point are delineated to cover the entire Iowa border area. For instance, for tile with coordinates of its centers X=212000 Y=4820000 its base name will be 02124820 with file extension las.7z or xyzi.7z depending on the type of the data.

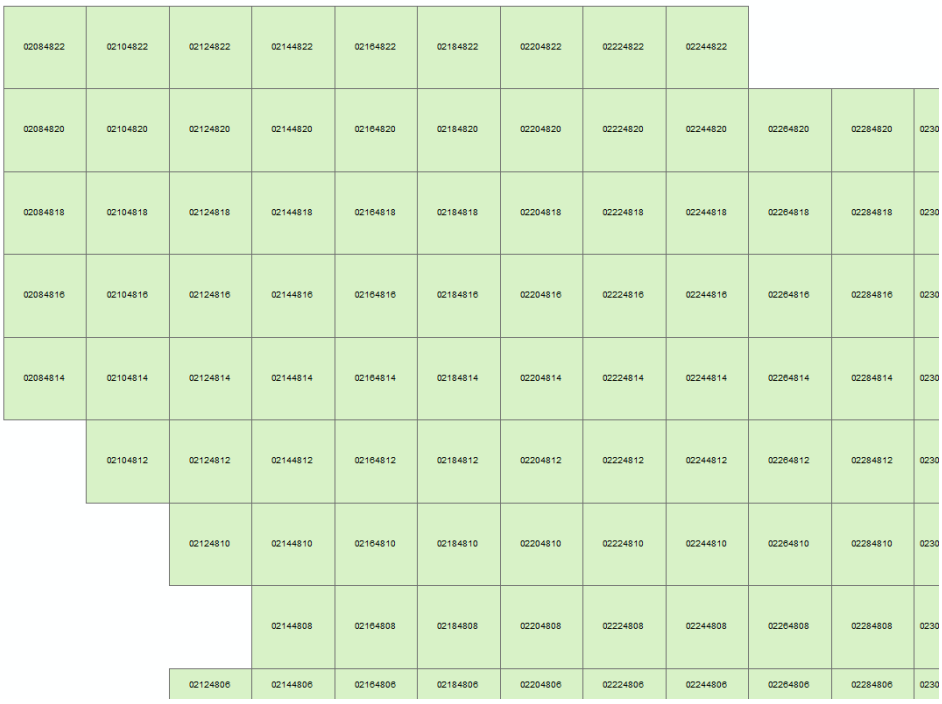


Fig 1. Fragment of the tile grid, which is used to locate tiles for downloading

The base URL for file download is http://geotree2.geog.uni.edu/IowaLidar/basename.ext.7z, so knowing tile base name and desired file extension, we can concatenate URL and download any file of interest. Data contains 36 946 tiles of LIDAR and XYZI data each.

Having figured out the tile naming convention, and using the considerations described above I have created an FME workflow to create tile grid inside Iowa state borders. Initially, tile grid was saved as a shape file. Then it was decided that to ship the tool with additional shape file will negatively affect user experience. Geometry and attributes of the tile grid layer were converted to Python tuple format. Each tuple element holds WKT representation of tile geometry and a string to store tile name. Such an approach allowed embedding the tile grid

layer into a Python script, so the user doesn't need to worry if he accidentally moves or deletes the shapefile layer.

Details of tool' implementation

To implement the tool, I used ESRI ArcPy API for ArcGIS 10.x product line. Regarding the Python and arcpy technologies used, it is a classic Python script tool. The user interface of the tool was created during embedding python script to a toolbox. The interface consists of thirteen controls, ten of them are checkboxes to allow user is selecting various options, and three of them – file selection controls to select layer of interest and specify paths for output files. Regarding the Python modules, I used *arcpy* module to access variables, passed to the script from the Tool interface and to run various geoprocessing tools, mainly from the 3D Analyst module. All the options read inside the script by arcpy.GetParameterAsText() for text fields or GetParameter functions for Boolean values(checkboxes). The other modules used in the script are *urllib*, function *urllib.urlretrieve* to retrieve the files over HTTP protocol, *subprocess* to run 7zip executable file during file extraction, and the *os* module to manipulate with files and paths.

Also, to control tool' GUI behavior, it was important to implement validation of the input data. Some of the options should be enabled dependent on the other options. For instance, if the user unchecks "Use current extent" checkbox, we should enable layer selector to allow select the layer of interest. If the user doesn't need to download LAS files, we cannot produce derived products. Thus all such checkboxes as Create DEM, Create Contours and so on should be disabled. Also, as to produce any of derived products, LAS files should be unpacked from downloaded 7z archives, so this checkbox should be automatically enabled if the user selects any checkbox from that group. For better usability, options to create DEM/XYZI derived products were wrapped into 'categories' inside validation script. It seems to be an effective option to improve the tool' GUI when it's too many controls.

Figure 2. GUI categories

To improve user experience, instead of supplying separate shape file with the tool, during the execution, script creating the tile grid layer right in memory. Each time when the user runs the script, we're creating empty polygon feature class in the *in_memory* workspace, inserting the geometry and attributes with *Insert cursor* and performing the remaining actions.

Listing 1. Geometry representation as WKT and attribute representation in the code.

```
row_values = [
('POLYGON ((202000 4732000,202000 4734000,204000 4734000,204000 4732000,202000 4732000))', '02024732'),
('POLYGON ((202000 4734000,202000 4736000,204000 4736000,204000 4734000,202000 4734000))', '02024734'),
```

Script' workflow

The general workflow depicted in appendix 1. Below we will review it a little more detailed. The first step of the algorithm is to read the input parameters and decide if a user wants to download data tiles covering the current extent or layer of interest. After that, we are programmatically creating the in-memory tile grid layer as explained above. Then we are creating a rectangle representing the current extent or using a specified layer. Then script running arcpy.SelectLayerByLocation_management() tool to find tile grid polygons from the tile grid layer, intersecting with the current extent or specified layer. Then we're using arcpy.SearchCursor() functionality to loop through selected cells, extract tile basenames, concatenate URL and download the files. Then, if any of the derived products options are selected, or if the user checked 'unpack and remove source files,' we're running extractAndRemove() subroutine to unpack and remove these files. The benefit of wrapping unpacking code to a subroutine is that we need to execute such code in different parts of the script, so this approach allows to make the code more compact and readable.

Then we are running 3D Analyst tools to produce derived products if the user selected appropriate options. In such case, we need to create LAS dataset, containing references to downloaded LAS files using arcpy.CreateLasDataset_management() and pre-populated list of the downloaded files. Then, we're running arcpy.SurfaceContour_3d(), arcpy.LasDatasetToRaster_conversion() or HillShade_3d() to produce contours, DEM or hillshade accordingly.

During the script execution, many things could go wrong, which will raise an exception and terminate the script. It could cause that temporary tile grid layer added to the current map document will not be removed and will prevent tool for running next time. To prevent that, handle the exceptions and ensure that cleanup code will be executed, I used *try…catch…finally* statement.

Results/Discussion

As a result of the development, a comprehensive script tool, allowing downloading XYZI/LAS files and produce derived data products(DEM, Contours, Hillshade) both from XYZI and LAS files was developed. It allows the user to obtain published Iowa LiDAR data right from ArcGIS without visiting the GeoTree LiDAR data distribution website.

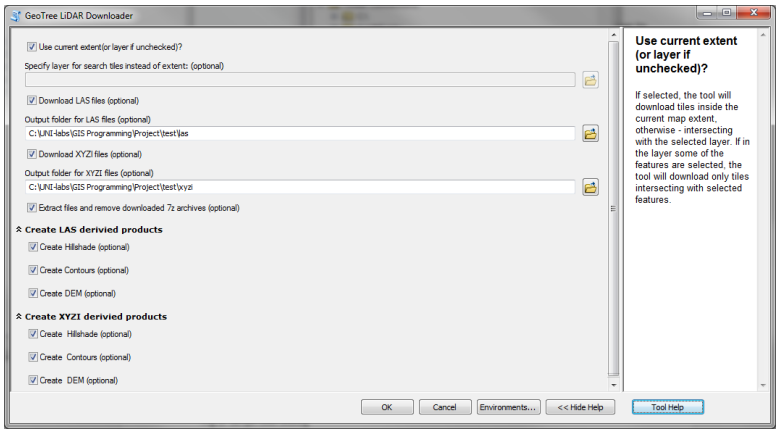
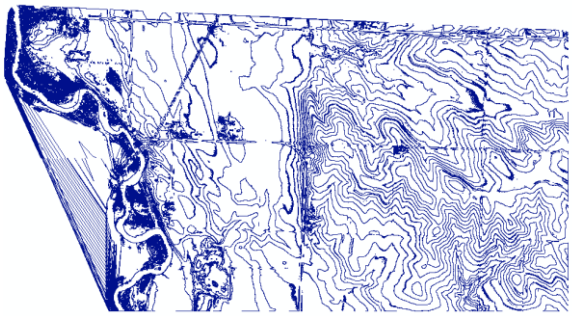
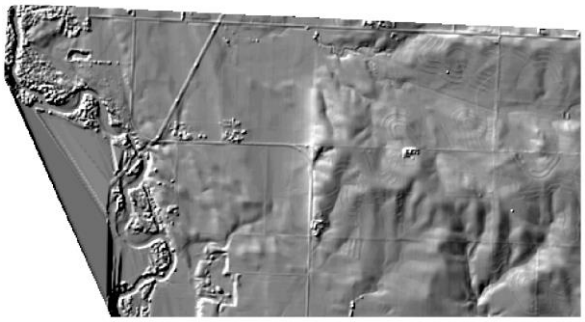
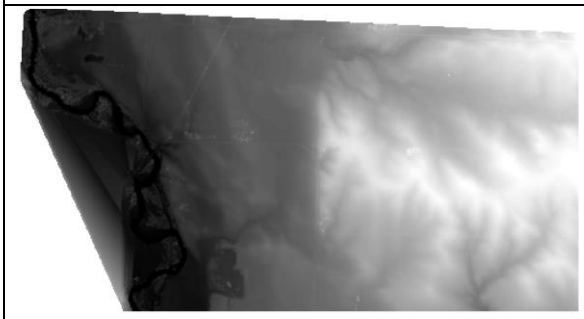
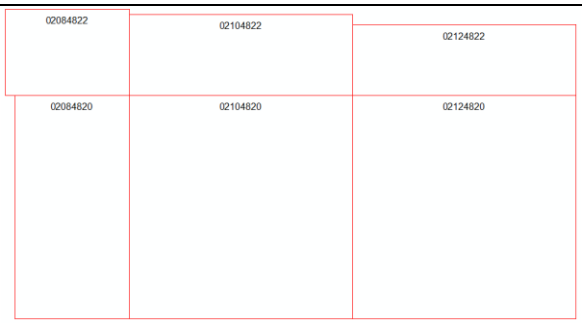


Fig 2. Script tool dialog

After tool execution, the user can obtain the following products, stored in the "output" subfolder of the folder used to save the downloaded LAS files:

 Contours(Contours.shp)	 Hillshade(hillshade ESRI GRID raster)
 Dem(las_dem ESRI GRID raster)	 Las dataset (las_dataset.lasd)

I would say that all the initial goals were reached and we got a working tool. I hope that the tool will be useful for others who work with Iowa LiDAR data. The tool doesn't require any significant configuration/installation efforts. The user needs to unpack the toolbox .tbx file and support files(7Za.exe executable) and run the tool from ArcMap.

Regarding the found problems, it was discovered, that tool can't work if:

- The user did not select any options in the tool dialog but pressed OK. Tried to handle it in the script, but it would be great to have an option to disable OK button in the validation class programmatically.
- Data frame in ArcGIS has an empty spatial reference (no layers are added). Added check and message to the user for spatial reference in the code.
- Current extent/extent of the specified layer is outside the Iowa area. Added check if a non-zero record number of tiles to download is selected.
- User tries to run the tool under ArcScene or ArcCatalog. Added check to allow the tool to be run from ArcMap only.

There could be other unforeseen limitations, but it is difficult to handle these issues without careful testing.

Conclusion and lessons learned

During work on the GIS Programming term project, a tool for downloading data products and producing derived data products was developed. It is a relatively portable tool, which could be re-used by others.

Regarding the limitations, the script was not rigorously tested. It is very difficult to predict what users could do wrong (not as designed) and be sure that we added all possible checks. For instance, I did not add any check to avoid derived products creation if a number of downloaded LIDAR tiles are too big and it would lead ArcGIS to crash due to lack of resources.

Regarding the issues and problems, I would say that initially, it was not clear how to organize input validation in the tool dialog, and it is the most tedious part of my project. Also, initially I confused GetParameterAsText and GetParameter functions. I was very surprised that GetParameterAsText is always returning 'True' Boolean value (as returned string always is not empty, and it ether equal 'true' or 'false'). I eventually figured out that I need to use a GetParameter function to get Boolean values.

If I had more time and programming experience, I would think more on code structure, like the overall design, and code organization like subroutines. Also, it would be great not to use a single block of try…except…finally statement but wrap individual "dangerous" parts of the code in these statements to provide detailed reporting what went wrong.

Appendix 1. Process flow diagram