Elevation Toolbox without Spatial Analyst Extension

Mike Bean
American River College, GEOG 375: Intro to GIS Programming; Spring 2020
Contact Information: w0701505@apps.losrios.edu

Introduction

For my Introduction to GIS Programming project, I wanted to learn more about some open source Python libraries and whether you could these libraries to emulate some of the elevation tools that are provided with Esri's licensed Spatial Analyst Extension. I created a Python Toolbox with code to generate slope, aspect, and hillshade rasters using Python NumPy library and a tool to create contour lines using Python GDAL (Geospatial Data Abstraction Library). Esri ArcGIS Pro installs (and uses) both libraries so no additional Python modules needed to be installed.

Project Tasks

- Combine open source Python code with Esri ArcPy functions
- Configure Python Toolbox template with parameters and calls
- Test to see if output matches output from Spatial Analyst tools
- Document tool parameters

Slope Aspect Hillshade Tool

- Load Elevation Raster
- Get Information about Elevation Raster
- Create 3x3 Windows over Elevation Values
- Creating the X and Y Gradient Arrays
- Calculate Slope
- Calculate Aspect
- Calculate Hillshade
- Saving an Output Raster
- Tool Screenshot

Loading Elevation Raster

```
arcpy.AddMessage("Opening elevation raster...")

inRaster = arcpy.Raster(elevation_input)


arcpy.AddMessage("Reading elevation values...")

elevation_values = arcpy.RasterToNumPyArray(in_raster, "", "", "", NODATA)

elevation_rows, elevation_columns = elevation_values.shape
```

Getting Information about Elevation Raster

```
# Spatial Reference from input raster, we use for output rasters

sr = inRaster.spatialReference


cell_width = in_raster.meanCellWidth

cell_height = in_raster.meanCellHeight

lower_left = in_raster.extent.lowerLeft
```

Create 3x3 Windows over Elevation Values

```
# Create 3x3 windows over elevation values, window array will contain 9 array views
after processing

window = []

    for row in range(3):

        for col in range(3):

            window.append(elevation_values[row:(row + elevation_rows - 2),

                                           col:(col + elevation_columns - 2)])
```

Creating the X and Y Gradient Arrays

```
# Calculate change in elevation in X direction

dz_dx = ((window[2] + window[5] + window[5] + window[8]) -

    (window[0] + window[3] + window[3] + window[6])) / (8. * cell_width)


# Calculate change in elevation in Y direction

dz_dy = ((window[6] + window[7] + window[7] + window[8]) -

    window[0] + window[1] + window[1] + window[2])) /  (8. * cell_height)
```

Calculate Slope

```
slope_rad = arctan(z_factor * sqrt(dz_dx * dz_dx + dz_dy * dz_dy))

slope = slope_rad * rad2deg


# todo: Need to determine how to deal with NODATA in a cell not on an edge!!!

# By using -9999 we get a slope value close to 90 degrees

slope[(slope > 89)] = NODATA
```

Calculate Aspect

```
aspect_rad = arctan2(dz_dx, -dz_dy)

# If terrain is flat, Esri outputs -1 as aspect, see:

#
https://desktop.arcgis.com/en/arcmap/10.7/tools/spatial-analyst-toolbox/how-aspect-
works.htm

aspect = np.where(slope_rad == 0., -1, aspect_rad * rad2deg + 180)
```

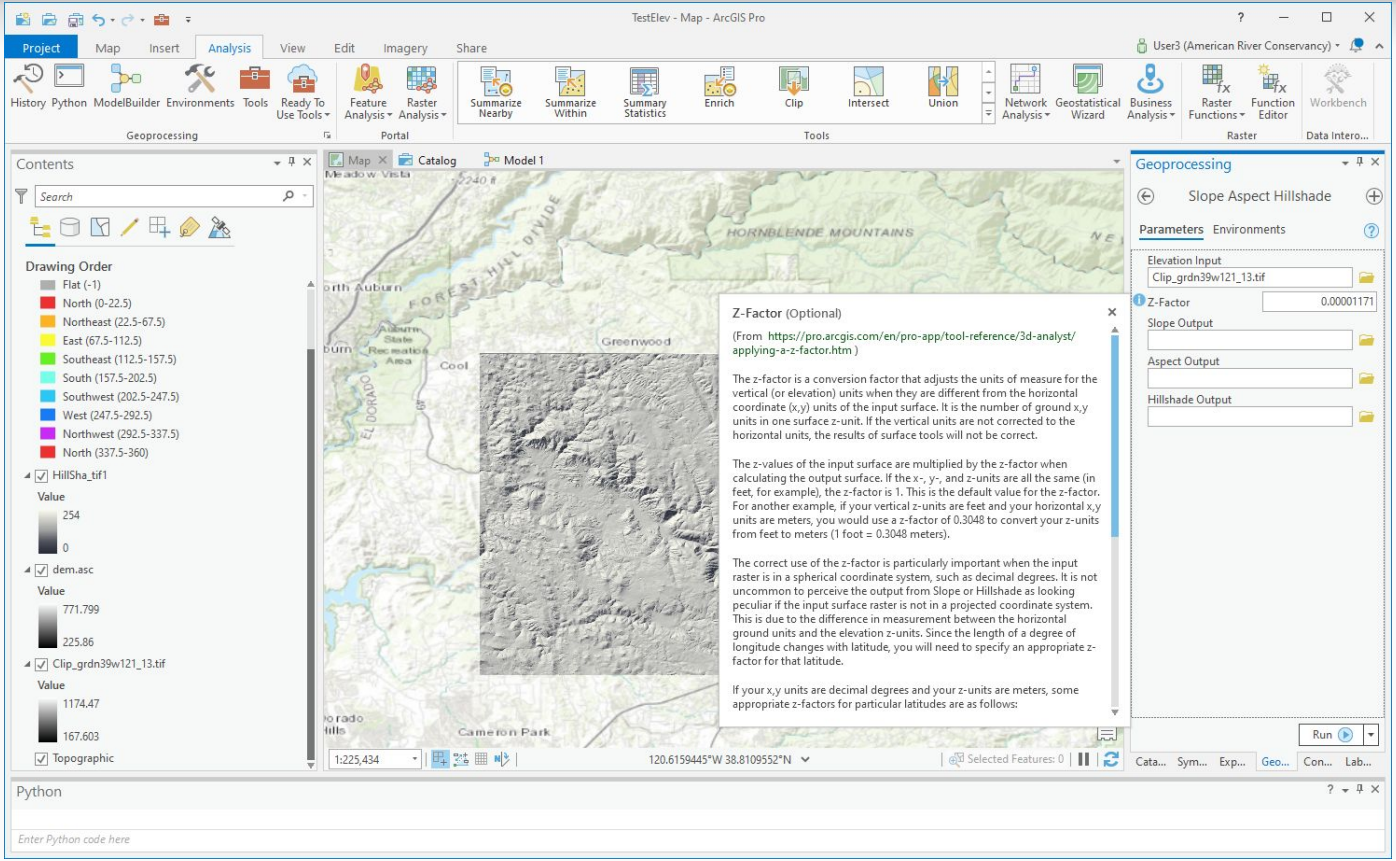
Calculate Hillshade

```
arcpy.AddMessage("Calculating hillshade raster...")

hillshade = np.clip(255 * ((cos(zenith_rad) * cos(slope_rad)) + (sin(zenith_rad)

    * sin(slope_rad) * cos(azimuth_rad - aspect_rad))),0, 255).astype(np.uint8)
```

Saving an Output Raster

```
if arcpy.Exists(hillshade_output):
    arcpy.Delete_management(hillshade_output)
hillshade_raster = arcpy.NumPyArrayToRaster(hillshade, lower_left, cell_width,
    cell_height)
arcpy.DefineProjection_management(hillshade_raster, sr)
hillshade_raster.save(hillshade_output)
arcpy.AddMessage("Saved hillshade raster")
```

Tool Screenshot



GDAL Contour Tool

- Load Elevation Raster
- Use Temporary Shapefile if Geodatabase
- Create Shapefile for Output
- Create Fields in Output Shapefile
- Generate the Contours
- Deleting Temporary Shapefile
- Tool Screenshot

Load Elevation Raster

```
# Get elevation values from elevation raster

arcpy.AddMessage("Opening elevation raster...")

ds = gdal.Open(elevation_input)

band = ds.GetRasterBand(1)

nodata = band.GetNoDataValue()

prj = ds.GetProjection()

srs = osr.SpatialReference(wkt=prj)
```

Use Temporary Shapefile if Geodatabase

```
if contour_feature_output.endswith(".shp"):
    arcpy.AddMessage("Creating shapefile for output...")
    scratch_name = None
    contour_shapefile = contour_feature_output
else:  # Create a temporary shapefile
    arcpy.AddMessage("Creating temporary shapefile for output...")
    scratch_name = arcpy.CreateScratchName("temp", data_type="Shapefile",
        workspace=arcpy.env.scratchFolder)
    contour_shapefile = scratch_name
```

Create Shapefile for Output

```
# Get OGR driver for shapefile

ogr_driver = ogr.GetDriverByName('ESRI Shapefile')



# Create shapefile for output

ogr_ds = ogr_driver.CreateDataSource(contour_shapefile)



# Create layer for new shapefile

    ogr_lyr = ogr_ds.CreateLayer(contour_shapefile, srs=srs,
geom_type=ogr.wkbLineString25D)
```

Create Fields in Output Shapefile

```
# Create ID field

field_defn = ogr.FieldDefn('ID', ogr.OFTInteger)

ogr_lyr.CreateField(field_defn)


# Create ELEV field

field_defn = ogr.FieldDefn('ELEV', ogr.OFTReal)

ogr_lyr.CreateField(field_defn)
```

Generate the Contours

```
arcpy.AddMessage("Generating contours...")
gdal.ContourGenerate(band, contour_interval, contour_base, [], 1, nodata, ogr_lyr,
    0, 1)
arcpy.AddMessage("Contours generated.")


# Close files so we can delete if temporary
del ogr_lyr, ogr_ds
```

Deleting Temporary Shapefile

```
if scratch_name:
    arcpy.AddMessage("Copying contours...")
    arcpy.CopyFeatures_management(contour_shapefile, contour_feature_output)


    # Delete temp shapefile
    arcpy.AddMessage("Deleting temporary shapefile...")
    arcpy.Delete_management(contour_shapefile)
```

Tool Screenshot