Complex Hillshade Tool in ArcGIS

Andrew Ashton, Lauren Collyer, Tyler Eldridge

For our final project in Geography 217, we decided to make a tool that creates a Hillshade map with a few extra features than the original Hillshade tool. Initially, our goal was to combine a few hillshade layers with lighting from different directions. In the real world, some sunlight that reaches Earth is diffused due to the atmosphere and cloudy conditions. The hillshade tool built into ArcGIS creates a harsh shadow on its topography and we wanted to improve it.

We added a function to our tool that could clip raster data to a polygon shapefile. Sometimes DEM data can be large, which leads to much longer processing time when applying the hillshade function, so clipping the data is a great way to streamline the process. This part of the code is the longest even though it is one of the less visually impressive functions of this script.

We then added a function that applies transparency to the layer. This is a relatively simple thing to do in ArcGis, but we thought it would be nice to be able to set an initial transparency of the layer through the tool.

The data input is very simple, you just need a DEM layer. For our specific trials in testing our tool, we ended up using the South Carolina DEM that we had used previously in class, as well as the 30 sec DEM layer that is available on the Q drive.

Approach

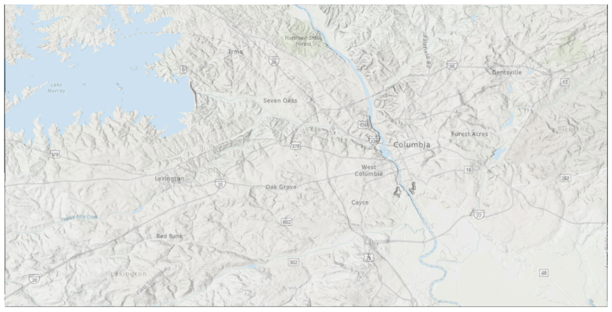
The first step was to decide what we wanted our tool to do. We talked about doing a hillshade layer with a light source from various different angles and then blending them. We also considered using a technique we have seen someone use, making multiple layers at various levels of blurriness for a nice hillshade effect. We also discussed some other possible things we wanted our tool to do, like cropping the DEM to a manageable size and making the layer transparent.

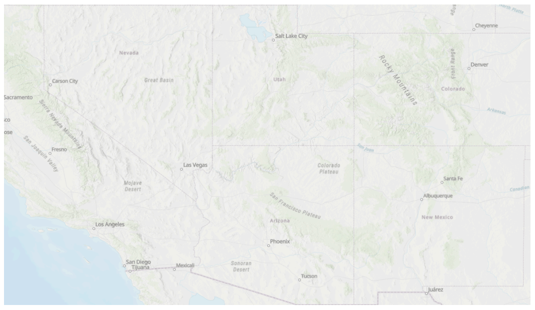
Next, we looked for documentation for the various functions we wanted to perform through ArcGIS's website. Here we learned that the python version of hillshade actually has a multishade option where the default tool does not. This was a great discovery, but it did mean that our program wouldn't be as revolutionary. Either way, it was nice to make a tool that allowed the user to take advantage of this built-in multishade tool without having to write code.

We were able to find some important documentation that gave us the foundation for our code. We attempted to find documentation for some of the other functions like blur and blending, but we had trouble finding any. Upon further research, these tools are pieces of other tools and the documentation was complicated and hard to find. In light of this, we turned more toward some of the other code designed to work with the raster, the cropping and transparency. We wrote the hillshade part of the code together and then broke off to work on these parts of the code. Using the documentation, class resources like Cade and Dr. Morgan, and chatGPT, we got all of the different pieces to work and began making a tool. From that point, we combined all of the different functions and then completed our tool. It required the user to input a DEM and determine a file path and name for the output hillshade file. The user can add a polyglot to be used to chrome the raster and a desired percent transparency, both optional. Our tests were successful and everything worked correctly, including when the optional values were provided and when they were not.

Results

When we finished our coding, we were able to create a tool that simply required the user to input a DEM, and our tool would produce a simple hillshade. Below are some examples. The image on the left shows a hillshade of the south carolina DEM, while the one on the right is a cropped hill shade from the 30 SEC DEM from the Q drive. While both are effective hillshades, we believe that our tool functions better when displaying smaller scale hillshades such as the South Carolina DEM.





Challenges and Lessons Learned

As we expected, we ran into lots of challenges as we were going about working on our project. For instance, finding documentation for certain ESRI applications was quite difficult. In particular, finding the documentation for raster calculator functions was very difficult to locate. Because it isn't really a geoprocessing tool, it is more difficult to find than finding documentation for a more particular tool.

Over the course of this process, we learned a lot about coding, and in particular how to create effective tools in arcPRO. In class, we have spent a decent amount of time working on coding different tools for arcPRO, but that was all in very structured situations. For this project, we didn;t really have any guidelines for what we were going to make, nor did we know how easy or difficult coding different features in our tool would be. For this particular tool, most of what we learned came down to figuring out how to find and implement ESRI documentation into our code, how to effectively code a clip function into our tool, and learning more about how the code for the ESRI hillshade tool works.

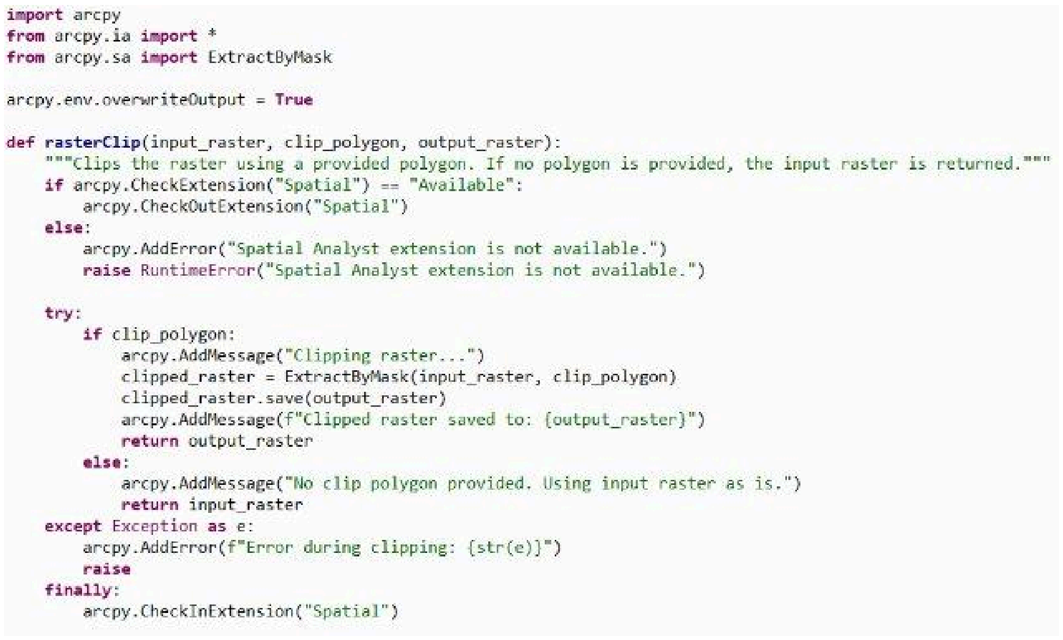
In addition to all of that, we learned that chatGPT and other AI are invaluable resources. None of us came into this project with a ton of coding experience, nor had we done anything similar to this before. Thus, when we needed help, we turned to chatGPT and the ESRI help pages, and more often than not, we were able to find what we were looking for. This is a valuable skill going forward in our careers working with GIS. GIS is constantly changing, and because of that, we need to be able to constantly learn and change with it. When we are working as professionals, we are going to be expected to face problems that no one around us is going to know how to solve, and it is up to us to know what resources to turn to to get help figuring out what is going on. That, one could argue, is the most important thing that we learned during this assignment. It was a great way for us to practice being faced with a problem that we didn't know how to solve, and then working together as a team in order to figure out what was going on and eventually solving it.

Conclusion

When we set out on this project, our goal was to create a hill shade tool that would be able to effectively display hill shade from a raster dataset. We wanted to make this tool easy for the user to use. In the end, we ended up creating a tool which required users to simply input a DEM and then the hill shade was generated. We were also able to give the user an option to clip the hill shade if they so desired. This process in some ways was easier than we thought it would be, and in other ways much more difficult. Overall though, it was a great learning experience, not just with coding in general, but in how to effectively apply our coding skills in arcPRO.

SOURCE CODE

(Let us know if you want the actual .py file.

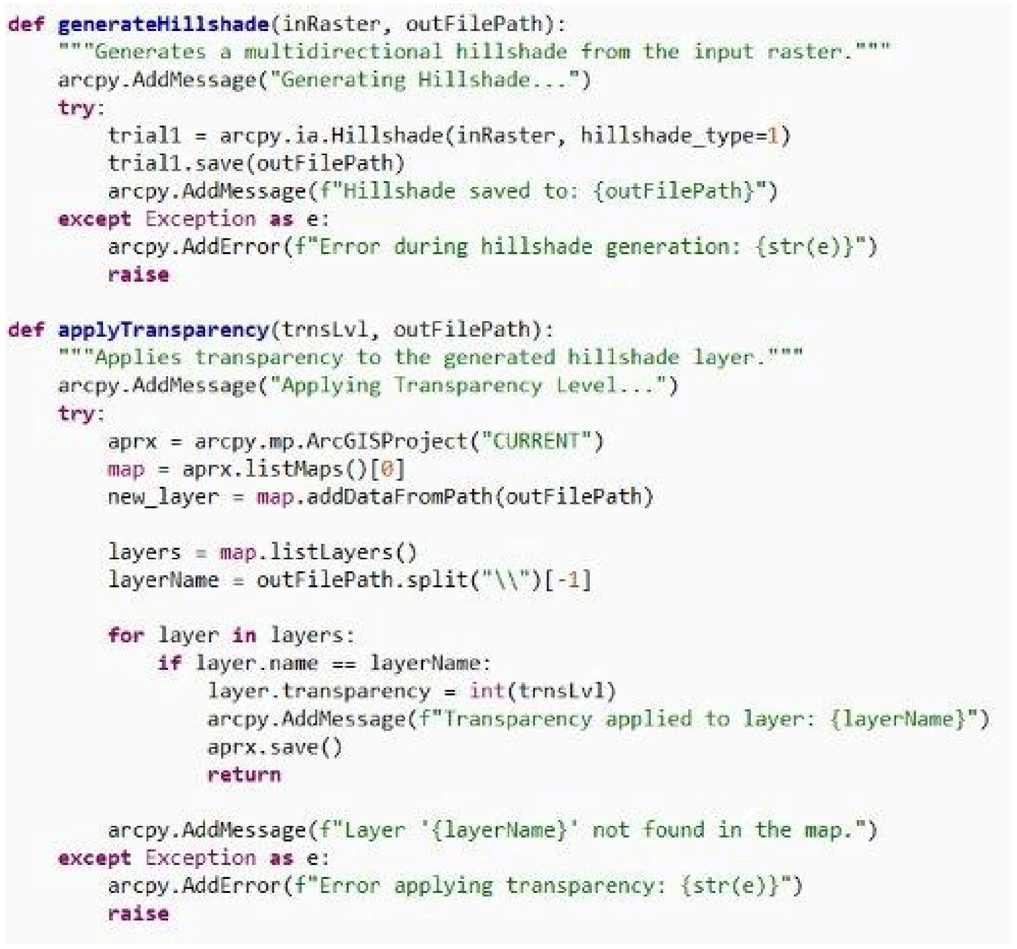


```python
import arcpy
from arcpy.ia import *
from arcpy.sa import ExtractByMask

arcpy.env.overwriteOutput = True

def rasterClip(input_raster, clip_polygon, output_raster):
    """Clips the raster using a provided polygon. If no polygon is provided, the input raster is returned."""
    if arcpy.CheckExtension("Spatial") == "Available":
        arcpy.CheckOutExtension("Spatial")
    else:
        arcpy.AddError("Spatial Analyst extension is not available.")
        raise RuntimeError("Spatial Analyst extension is not available.")

    try:
        if clip_polygon:
            arcpy.AddMessage("Clipping raster...")
            clipped_raster = ExtractByMask(input_raster, clip_polygon)
            clipped_raster.save(output_raster)
            arcpy.AddMessage(f"Clipped raster saved to: {output_raster}")
            return output_raster
        else:
            arcpy.AddMessage("No clip polygon provided. Using input raster as is.")
            return input_raster
    except Exception as e:
        arcpy.AddError(f"Error during clipping: {str(e)}")
        raise
    finally:
        arcpy.CheckInExtension("Spatial")
```



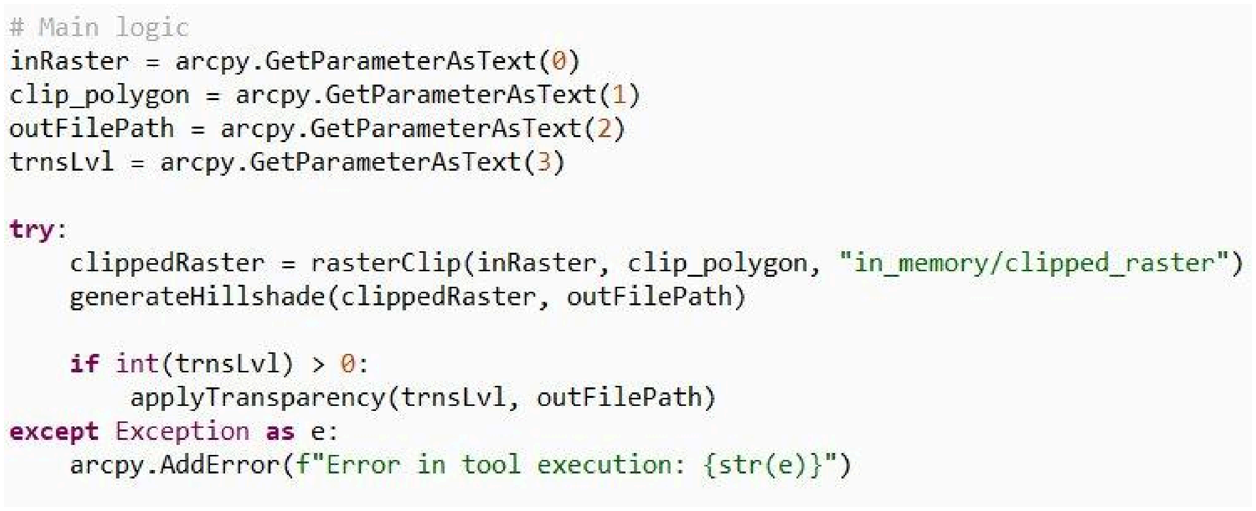
```
def generateHillshade(inRaster, outFilePath):
    """Generates a multidirectional hillshade from the input raster."""
    arcpy.AddMessage("Generating Hillshade...")
    try:
        trial1 = arcpy.ia.Hillshade(inRaster, hillshade_type=1)
        trial1.save(outFilePath)
        arcpy.AddMessage(f"Hillshade saved to: {outFilePath}")
    except Exception as e:
        arcpy.AddError(f"Error during hillshade generation: {str(e)}")
        raise

def applyTransparency(trnsLvl, outFilePath):
    """Applies transparency to the generated hillshade layer."""
    arcpy.AddMessage("Applying Transparency Level...")
    try:
        aprx = arcpy.mp.ArcGISProject("CURRENT")
        map = aprx.listMaps()[0]
        new_layer = map.addDataFromPath(outFilePath)

        layers = map.listLayers()
        layerName = outFilePath.split("\\")[-1]

        for layer in layers:
            if layer.name == layerName:
                layer.transparency = int(trnsLvl)
                arcpy.AddMessage(f"Transparency applied to layer: {layerName}")
                aprx.save()
                return

        arcpy.AddMessage(f"Layer '{layerName}' not found in the map.")
    except Exception as e:
        arcpy.AddError(f"Error applying transparency: {str(e)}")
        raise
```

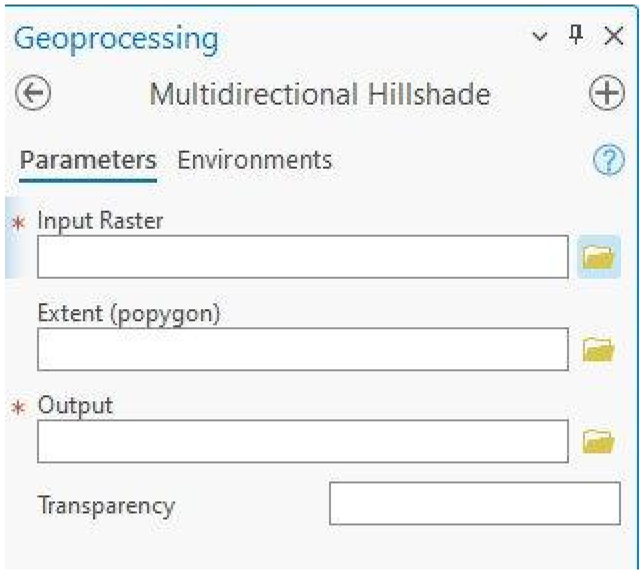


```
# Main logic
inRaster = arcpy.GetParameterAsText(0)
clip_polygon = arcpy.GetParameterAsText(1)
outFilePath = arcpy.GetParameterAsText(2)
trnsLvl = arcpy.GetParameterAsText(3)

try:
    clippedRaster = rasterClip(inRaster, clip_polygon, "in_memory/clipped_raster")
    generateHillshade(clippedRaster, outFilePath)

    if int(trnsLvl) > 0:
         applyTransparency(trnsLvl, outFilePath)
except Exception as e:
    arcpy.AddError(f"Error in tool execution: {str(e)}")
```

And here is what our tool looks like when it's in use.



* Input Raster

Extent (popygon)

* Output

Transparency