NYC Scaling - Midpoint Layer Derivation

By: Jason K Hawes - [jkhawes@umich.edu](mailto:jkhawes@umich.edu)

This notebook identifies the source data for NYC scaling and explains pre-processing to output the four basic layers on which scenarios are built:

1. Simplified Land Cover
2. Simplified Land Use
3. Slope
4. Sunlight availability

To replicate this work, first open a blank project in QGIS. I have tested most of this on QGIS 3.18 and 3.24, though anything past 3.00 should suffice. When possible, I will include a verbal explanation, a screenshot for the procedure described, and Python code to execute it - this should make it easier to replicate the process in varying versions where the syntax or appearance may change slightly. Theoretically, you could run this whole thing as one Python command, but I find it more accessible to break the whole thing out into chunks so folks can get a feel for the commands themselves. To help with this, I will include the Python script under each command. Copy-pasting this is going to be pretty limited, since the file structure on different computers is going to mean that you’ll need to edit every command, but I’ve tried to include enough detail that if you do want to build this as a Python script it shouldn’t be very hard.

The goal of this notebook is to translate the input layers into the four basic layers described above. The input layers come from several open databases furnished by New York City municipal authorities:

* NYC Land Cover from LiDAR and Orthoimagery - [download and metadata](https://data.cityofnewyork.us/Environment/Land-Cover-Raster-Data-2017-6in-Resolution/he6d-2qns)
* NYC Sidewalks dataset - [download and metadata](https://data.cityofnewyork.us/City-Government/Sidewalk/vfx9-tbb6/data?no_mobile=true)
* NYC Roadbeds dataset - [download and metadata](https://data.cityofnewyork.us/City-Government/Roadbed/xgwd-7vhd/data?no_mobile=true)
* NYC Buildings dataset - [metadata](https://github.com/CityOfNewYork/nyc-geo-metadata/blob/master/Metadata/Metadata_BuildingFootprints.md)
* PLUTO Land Use Clipped to Shoreline (similar to zoning) - [download and metadata](https://www1.nyc.gov/site/planning/data-maps/open-data/dwn-pluto-mappluto.page)
* NYC Parking lots dataset - [download and metadata](https://data.cityofnewyork.us/City-Government/Parking-Lot/h7zy-iq3d/data?no_mobile=true)
* NYC Parks dataset - [download and metadata](https://data.cityofnewyork.us/Recreation/Parks-Properties/enfh-gkve)
* NYC borough boundaries - [download and metadata](https://data.cityofnewyork.us/City-Government/Borough-Boundaries/tqmj-j8zm)
* NYC highest-hit model (DSM) - [metadata](https://gis.ny.gov/elevation/metadata/2017NYC-topobath-DSM.XML)
* NYC DEM - [metadata](https://gis.ny.gov/elevation/metadata/2017NYC-topobath-DEM-hef.XML)

At the end of this analysis, we will have tranformed these inputs into four aligned rasters at 1m resolution. All functions will be conducted in EPSG 102003 because it works well with solar irradiance actions and is a relatively painless transformation from the projections used by NYC staff. Based on those rasters, we are able to develop scenarios that capture the possible areas of expansion for urban agriculture in NYC.

# Land Cover Layer Derivation

This layer will identify open ground areas and rooftops. These open areas and rooftops will then be filtered by other qualifications from the other layers (e.g., slope). The final layer produced via this procedure will have the following codes:

* Impervious - 1
* Grass or dirt - 2
* Roof - 3
* Trees - 4
* Otherwise occupied - 0 (e.g., monument, water, railroad, road)

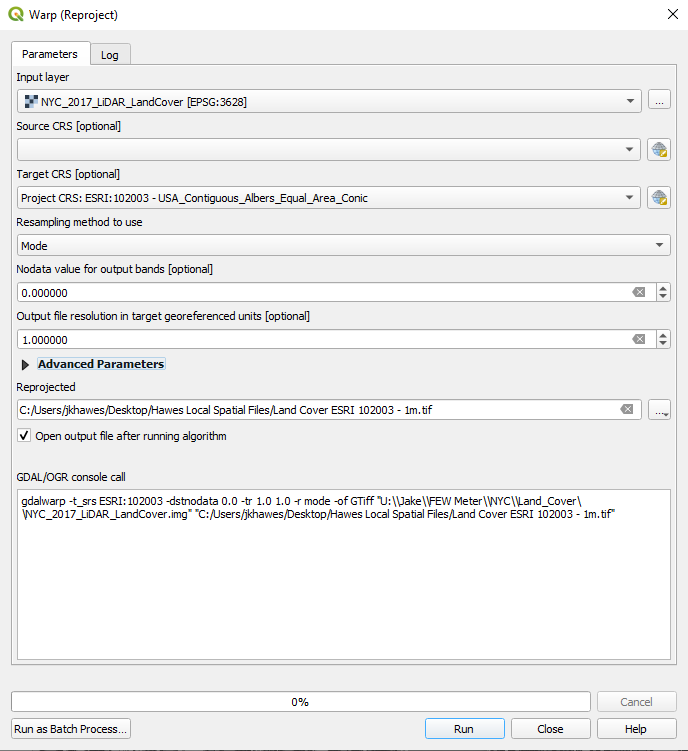
Overview: We have a land cover dataset that needs to be simplified to not have trees and needs to parcel out ineligible impervious areas like roads.

The simplest way to do this would be to use the land cover as a base layer, reclassify it to our needs, then add in the streets and sidewalks (to erase trees), then add in buildings. So, what we'll do is first reclassify land cover, then rasterize sidewalks and roads such that they equal zero and everything else equals 1. We'll also rasterize the building layer so that buildings = 3. Then we'll multiply those three layers together to get 0s everywhere we're not interested in. And finally, we'll add the buildings layer in equaling 3. Depending on the years and alignment of your data, there may be some cleaning to do with cells equalling 4 and 5, but those are easy to recode to roofs.

To begin, import the land cover, sidewalks, roads, and buildings datasets. We will reclassify and overlay these layers repeatedly to generate the final map of land cover in NYC. We will process each of these layers individually.

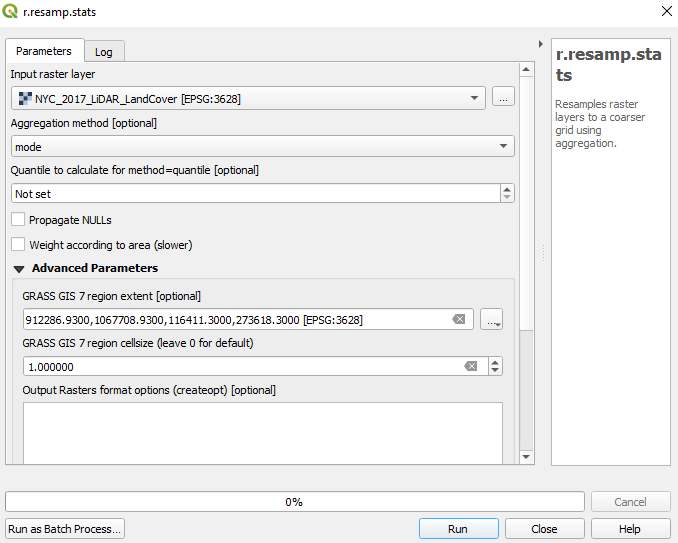
## Reproject, resample, and reclassify original land cover layer

First, since the Land Cover layer is the only raster we will use in this stage, we will resample it to our desired resolution. There are many ways to do this, but we will use either Warp or r.resamp.stats. You are most likely going to use Warp, since your Land Cover data is probably in EPSG:3628 - NAD83(NSRS2007) / New York Long Island (ftUS). Since we want this in ESRI:102003 / USA\_Contiguous\_Albers\_Equal\_Area\_Conic anyway, it makes most sense to both reproject and resample at the same time. You can use Warp with the command below. On our computer in the lab with the original layer downloaded, this took only 898 seconds.



If for some reason your Land Cover data is already in a meters format or you want to leave it in feet, you can accomplish the resampling with r.resamp.stats. Run it with the settings shown below, including "mode" and a set resampling distance of 1m. Again, be sure to check the projection of the layer. If it is in feet, then resampling to "1" will resample it to 1ft.

On our computer in the lab with the original layer downloaded, the command below took 6334 seconds to convert from 6 inch resolution to 1ft resolution. It would presumably be faster for 1m resolution.



Now, we just need to reclassify the Land Cover layer, which is easiest with r.reclass. We basically treat zero as our background noise and only classify things of importance as non-zero. So let’s start with land cover. Initial classes are: (1) Tree Canopy, (2) Grass\Shrubs, (3) Bare Soil, (4) Water, (5) Buildings, (6) Roads, (7) Other Impervious, and (8) Railroads. Our reclassify, in this case:

* We need 1 to be 4, since it’s a simple renumbering for trees.
* We need 2-3 to classify as 2, since that’s going to be our code for grass and soil.
* We need water (4) to go to 0.
* Even though we care about buildings, we also need buildings to go to zero, since we have a more robust measure of buildings (w/out trees) in our other raster and we’ll reclassify those as 3 next.
* We reclassify 6 as 0, again since we’ll be overlaying a buildings layers to get the best measure of that.
* We reclassify 7 as 1, since that’s our final code for other impervious, which appears to be mostly driveways and pathways and sidewalks (which we'll take care of next).
* Finally, 8 goes to 0.

In other words, the following code is fed to r.reclass:

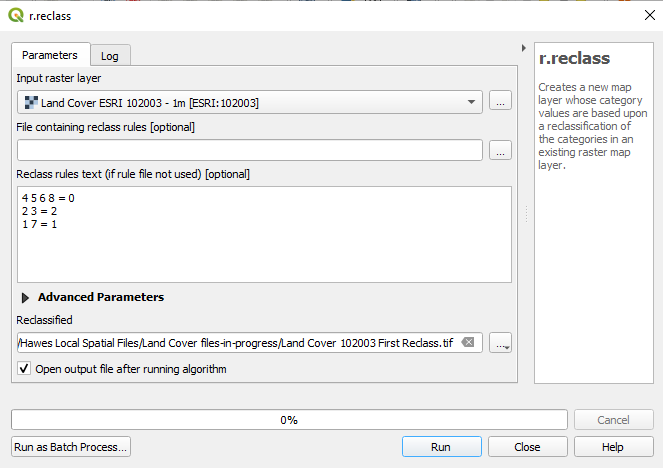
1 = 4

4 5 6 8 = 0

2 3 = 2

7 = 1

When running the command, you may run into CRS issues. If your final layer ends up somewhere odd (mine wound up in Cape Cod sometimes), try setting the CRS by right-clicking >> Layer CRS >> Set Layer CRS and changing it back to 102003.



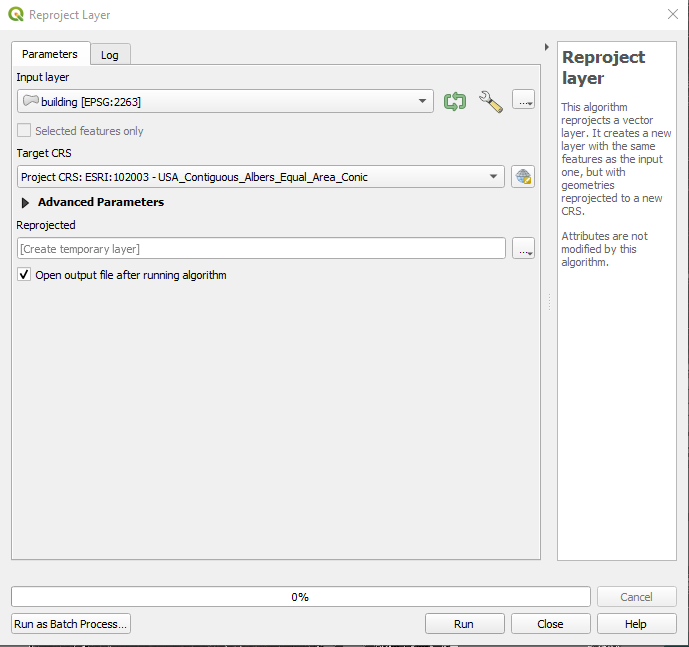
**Python code**

processing.run("grass7:r.reclass", {'input':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Land Cover ESRI 102003 - 1m.tif', 'rules':'', 'txtrules':'1 = 4\n4 5 6 8 = 0\n2 3 = 2\n7 = 1','output':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Land Cover/LC 102003 First Reclass.tif', 'GRASS\_REGION\_PARAMETER':None, 'GRASS\_REGION\_CELLSIZE\_PARAMETER':0, 'GRASS\_RASTER\_FORMAT\_OPT':'', 'GRASS\_RASTER\_FORMAT\_META':''})

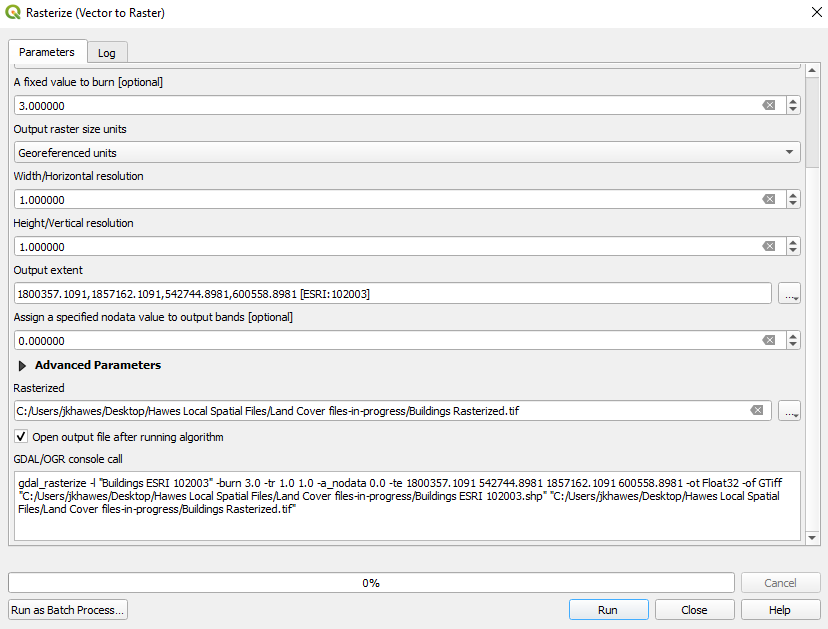
## Reproject and rasterize building, sidewalk, and roadbed layers

Once the land cover is correctly coded, we can begin rasterizing the other layers and overlaying them. In each case, we will reproject and rasterize the layer to the same resolution as the land cover and align it with the land cover. See the screenshots and code below for more details.

The first layer to convert is the Building Layer. First, we need to reproject it to the same CRS as the Land Cover layer. This should take barely more than a minute. See below for details.



Next, we can rasterize this layer. Since we already have it in m, it's simple enough to rasterize with the georeferenced units and let it run. Since we'll eventually need the Buildings raster to have a value distinct from other areas for the purposes of raster algebra, we can go ahead and set that now. The screenshot below show buildings set to 3 (the eventual value in the LC layer), but I recommend using 100 for the burn-in value to make raster algebra easier. This also helps with the land use because you can make sidewalks and roads distinct (below). Make sure also that NoData is set to 0. Finally, make sure that the Output Extent is set as "calculate from layer" with the land cover data. This should save us a step later on by aligning the rasters from the outset. On the computer in our lab, with the reprojected layer saved to the desktop, this took 27 seconds. See below for command details.



Proceed exactly as above with the Sidewalks and Roadbeds datasets. Reproject them, then rasterize them. Each one of these should be set to a binary layer upon rasterization, where the sidewalks are 40 and roads are 20 and everything else is 0. To accomplish this, set the burn in value to 40 or 20 and use r.null to set everything else to 0.

**Python code for all three**

processing.run("native:reprojectlayer", {'INPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/NYC Data/Building footprint/building.shp','TARGET\_CRS':QgsCoordinateReferenceSystem('ESRI:102003'),'OPERATION':'+proj=pipeline +step +proj=unitconvert +xy\_in=us-ft +xy\_out=m +step +inv +proj=lcc +lat\_0=40.1666666666667 +lon\_0=-74 +lat\_1=41.0333333333333 +lat\_2=40.6666666666667 +x\_0=300000 +y\_0=0 +ellps=GRS80 +step +proj=aea +lat\_0=37.5 +lon\_0=-96 +lat\_1=29.5 +lat\_2=45.5 +x\_0=0 +y\_0=0 +ellps=GRS80','OUTPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Land Cover/buildings 102003.shp'})

processing.run("native:reprojectlayer", {'INPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/NYC Data/Roadbeds 20230714/geo\_export\_9110f5a3-6121-41f8-bec0-2bb8a4e7d45e.shp','TARGET\_CRS':QgsCoordinateReferenceSystem('ESRI:102003'),'OPERATION':'+proj=pipeline +step +proj=unitconvert +xy\_in=deg +xy\_out=rad +step +proj=push +v\_3 +step +proj=cart +ellps=WGS84 +step +proj=helmert +x=0.9956 +y=-1.9013 +z=-0.5215 +rx=0.025915 +ry=0.009426 +rz=0.011599 +s=0.00062 +convention=coordinate\_frame +step +inv +proj=cart +ellps=GRS80 +step +proj=pop +v\_3 +step +proj=aea +lat\_0=37.5 +lon\_0=-96 +lat\_1=29.5 +lat\_2=45.5 +x\_0=0 +y\_0=0 +ellps=GRS80','OUTPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Land Cover/roads 102033.shp'})

processing.run("native:reprojectlayer", {'INPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/NYC Data/Sidewalk 20230714/geo\_export\_5e95bee1-18c6-45b6-a27f-95c3f6fbed9c.shp','TARGET\_CRS':QgsCoordinateReferenceSystem('ESRI:102003'),'OPERATION':'+proj=pipeline +step +proj=unitconvert +xy\_in=deg +xy\_out=rad +step +proj=push +v\_3 +step +proj=cart +ellps=WGS84 +step +proj=helmert +x=0.9956 +y=-1.9013 +z=-0.5215 +rx=0.025915 +ry=0.009426 +rz=0.011599 +s=0.00062 +convention=coordinate\_frame +step +inv +proj=cart +ellps=GRS80 +step +proj=pop +v\_3 +step +proj=aea +lat\_0=37.5 +lon\_0=-96 +lat\_1=29.5 +lat\_2=45.5 +x\_0=0 +y\_0=0 +ellps=GRS80','OUTPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Land Cover/sidewalks 102033.shp'})

processing.run("gdal:rasterize", {'INPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Land Cover/buildings 102003.shp','FIELD':'','BURN':100,'USE\_Z':False,'UNITS':1,'WIDTH':1,'HEIGHT':1,'EXTENT':None,'NODATA':-9999,'OPTIONS':'','DATA\_TYPE':5,'INIT':None,'INVERT':False,'EXTRA':'','OUTPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Land Cover/buildings 102003.tif'})

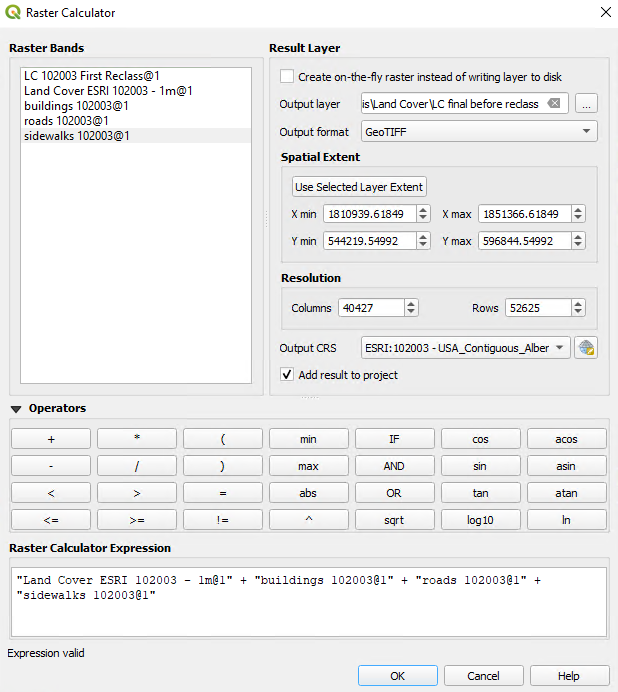
processing.run("gdal:rasterize", {'INPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Land Cover/roads 102003.shp','FIELD':'','BURN':20,'USE\_Z':False,'UNITS':1,'WIDTH':1,'HEIGHT':1,'EXTENT':None,'NODATA':-9999,'OPTIONS':'','DATA\_TYPE':5,'INIT':None,'INVERT':False,'EXTRA':'','OUTPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Land Cover/roads 102003.tif'})

processing.run("gdal:rasterize", {'INPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Land Cover/sidewalks 102003.shp','FIELD':'','BURN':40,'USE\_Z':False,'UNITS':1,'WIDTH':1,'HEIGHT':1,'EXTENT':None,'NODATA':-9999,'OPTIONS':'','DATA\_TYPE':5,'INIT':None,'INVERT':False,'EXTRA':'','OUTPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Land Cover/sidewalks 102003.tif'})

## Synthesize Land Cover Layer

Finally, we can bring all four layers together and do the final reclassification. Once all the rasterization is complete, we'll need to complete one last step before running the raster algebra. Before we do this, we can use r.null to set the null values to 0 on the new files. We can add everything together and reclassify. 0, 1, 2, 3, and 4 stay the same (although there should be no 3s). Anything in the ~20s or ~40s (or a combination of the two) becomes 1 and anything greater than 99 becomes 3. Unfortunately, there is no raster calculator python code, but you could theoretically do this with some fairly simple algebra in Python. Not going to, since we’re using the GUI and the Python is just to speed up re-running.

Because the output extent for the rasterization was the Land Cover dataset, we should not need to align the rasters, but it's a possibility. If there are obvious issues or if the raster calculation fails, try aligning the rasters first. Since we've already set the raster values where we want them, this should be a fairly straightforward raster calculation and should take about 10 minutes if everything is saved locally.



The resulting file should have values 0 thru 5 where 4 and 5 exist on the outskirts of buildings and in places where the trees overhung the buildings. Basically, this is an indication that the original land cover file minorly underestimated the buildings present in NYC, so anything greater than 3 can be safely called a building. So one final reclassify gets us to our final layer:

* 0=0 Otherwise occupied
* 1, 20 thru 99 =1 Impermeable
* 2=2 Low vegetation, grass or dirt
* 3, 100 thru 199 = 3 Buildings
* 4 = 4 Trees

Remove the commas and comment if you copy paste this - the words following the numbers will be used to label the layer, so you can take them or leave them as you see fit. You should now have a usable land cover file. Remember that all the forthcoming rasters must be aligned to this one so we can eventually add them all together.

processing.run("grass7:r.reclass", {'input':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Land Cover/LC final before reclass.tif','rules':'','txtrules':'0=0 Otherwise occupied\n1 20 thru 49 =1 Impermeable\n2=2 Grass or dirt\n3 50 thru 99 = 3 Buildings\n4 = 4 Trees\n','output':'C:/Users/jkhawes/Desktop/LocalSpatialData/Midpoint/LC.tif','GRASS\_REGION\_PARAMETER':None,'GRASS\_REGION\_CELLSIZE\_PARAMETER':0,'GRASS\_RASTER\_FORMAT\_OPT':'','GRASS\_RASTER\_FORMAT\_META':''})

# Land Use Layer Derivation This layer will describe the land use at a parcel level. These will be used primarily to sort the different types of gardens - e.g., it makes much more sense to assume an individual garden in a single-family backyard than a community garden. The final layer produced via this procedure will have the following codes:

* Residential - Single family -- 11 (PLUTO Code 1)
* Residential - Multi-family -- 12 (PLUTO Codes 2 and 3)
* Mixed Residential and Commercial Buildings -- 21 (PLUTO Code 4)
* Commercial Office Buildings - 22 (PLUTO Code 5)
* Industrial and Manufacturing - 23 (PLUTO Code 6)
* Parks and Playground – 31 (From Parks dataset)
* Other public green space - 32 (Original PLUTO Code 9, mostly cemeteries)
* Public Facilities and Institutions - 33 (hospitals, nursing homes, other buildings and open space - PLUTO Code 8)
* Parking lots -- 41 (From Parking Lot dataset)
* Transportation and Utility - 42 (PLUTO Code 7)
* Vacant land -- 70 (PLUTO Code 11)
* Roads – 80 (From roads layer)
* Sidewalks - 81 (From sidewalks layer)

Overview: We have PLUTO data that just need to be simplified and prepared for use in our coding scheme. We will also integrate a bit of additional information from other NYC data layers (adding in roads, sidewalks, parks, and parking lots).

To begin, import the following data sets:

1. PLUTO Land Use Clipped to Shoreline (similar to zoning) - [download and metadata](https://www1.nyc.gov/site/planning/data-maps/open-data/dwn-pluto-mappluto.page)
2. NYC Sidewalks dataset - [download and metadata](https://data.cityofnewyork.us/City-Government/Sidewalk/vfx9-tbb6/data?no_mobile=true)
3. NYC Roadbeds dataset - [download and metadata](https://data.cityofnewyork.us/City-Government/Roadbed/xgwd-7vhd/data?no_mobile=true)
4. NYC Parking lots dataset - [download and metadata](https://data.cityofnewyork.us/City-Government/Parking-Lot/h7zy-iq3d/data?no_mobile=true)
5. NYC Parks dataset - [download and metadata](https://data.cityofnewyork.us/Recreation/Parks-Properties/enfh-gkve) (actually a combination of parks and playgrounds)
6. An earlier version of this used the borough boundaries, and it can still be useful just to make sure that all your math stops at the water’s edge, but it’s not necessary anymore. NYC borough boundaries - [download and metadata](https://data.cityofnewyork.us/City-Government/Borough-Boundaries/tqmj-j8zm)

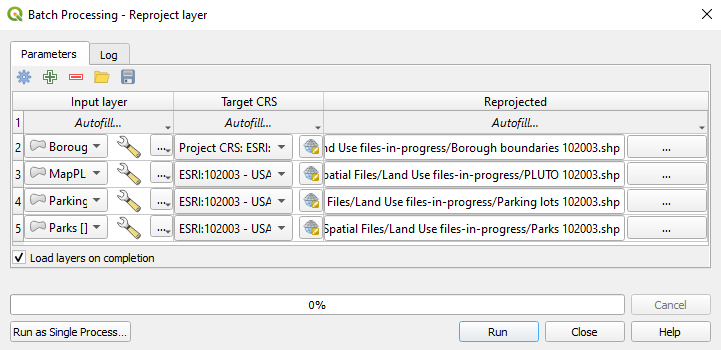
All of these are updated fairly regularly, so if it’s been a while since you ran this analysis (> 3 months), it might be helpful to refresh the data being used.

## 

## Reproject all layers

The first step in the process of deriving a land use layer which can be compared with the land cover data we just completed is to reproject all the land use layers to 102003. Since we already have a reprojected, rasterized, and aligned version of Roadbeds and Sidewalks, we can ignore those in these first two steps. So we need to reproject PLUTO, Parks, and Parking Lots. This was everything is in the same vector format to start, so we can better predict what will happen when we operate on them.

We can do this as a batch process to save a little time. I find it simplest to save them all as shapefiles, since occasionally different formats can interact with different algorithms in weird ways. To start a batch process, click the bottom-left button on the Reproject Layer screen. This will bring up a form where you can enter the parameters line-by-line. Fill it out for the three layers we’re analyzing (see below for my settings). Click run, and it should be done in a couple minutes on the computer in our lab.



**Python code example - just replace the file names (and remove the layername parameter for non-gdb):**

processing.run("native:reprojectlayer", {

'INPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/NYC Data/nyc\_mappluto\_23v1\_2\_fgdb/MapPLUTO23v1\_2.gdb|layername=MapPLUTO\_23v1\_2\_clipped',

'TARGET\_CRS':QgsCoordinateReferenceSystem('ESRI:102003'),

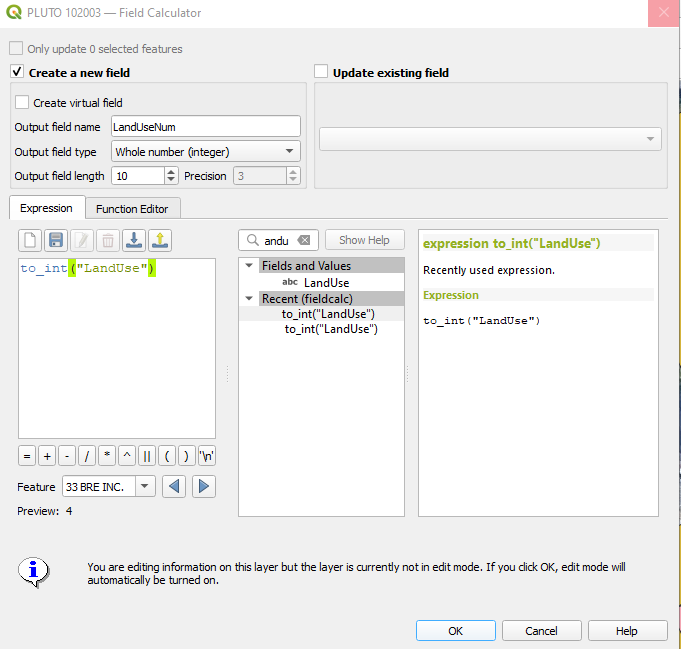
'OPERATION':'+proj=pipeline +step +proj=unitconvert +xy\_in=us-ft +xy\_out=m +step +inv +proj=lcc +lat\_0=40.1666666666667 +lon\_0=-74 +lat\_1=41.0333333333333 +lat\_2=40.6666666666667 +x\_0=300000 +y\_0=0 +ellps=GRS80 +step +proj=aea +lat\_0=37.5 +lon\_0=-96 +lat\_1=29.5 +lat\_2=45.5 +x\_0=0 +y\_0=0 +ellps=GRS80',

'OUTPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/LU/PLUTO 102003.shp'})

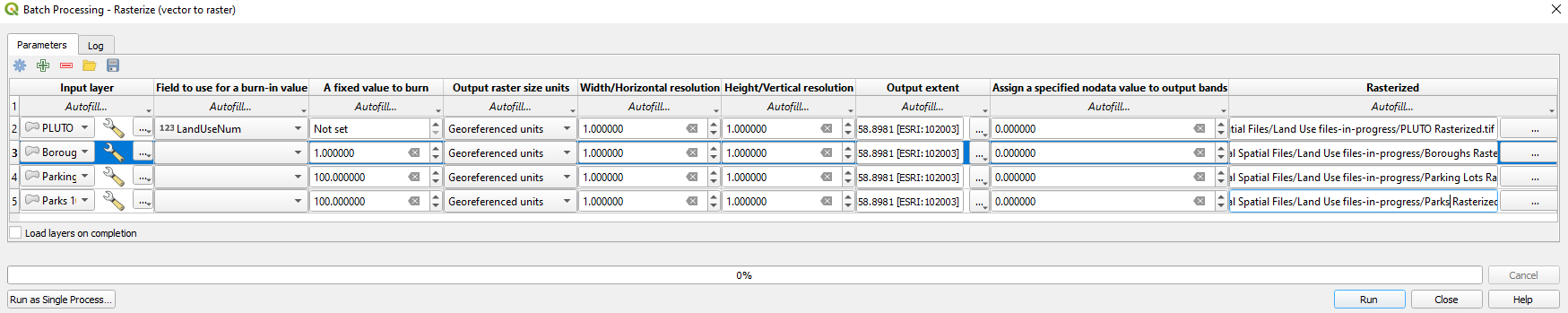
## Rasterize all layers

Once we have all the layers in the correct projection, we can move on to rasterizing the layers that remain to be rasterized. We already have a binary version of the roadbeds and sidewalks layers where everything is 1 except for them. That means we can use those in the same way we did before - multiplication to derive their extents. So now we just need PLUTO as a raster with the land use values as the code, Parks as a raster hardcoded to 200, and Parking lots hardcoded to 400. Always set no data to -9999.

Again, we can run this as a batch process. Before we can do this, we need to convert the PLUTO LandUse column to numeric with a field calculator operation.



Once this is ready, run rasterize. See below for the parameters, but change the burn-in values for parks and parking lots. Parking lots should be 400 and parks should be 200. This saves a long raster calculation. Make sure to use the land use raster as the output extent if you don’t want to have to align everything later. If LandUseNum doesn’t show up as a field, it’s because the change hasn’t saved to the file yet. Make sure to save the change and stop editing the PLUTO file. If this still doesn’t work, remove the PLUTO layer and reload it. Overall, this should only take a few minutes.



After this runs, check the results in the streets and other areas that PLUTO doesn’t cover. You will also need to run r.null and add 0s in place of No Data.

**Python code for rasterize**

**PLUTO:**

processing.run("gdal:rasterize", {'INPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/LU/PLUTO 102003.shp', 'FIELD':'LandUseNum', 'BURN':None, 'USE\_Z':False, 'UNITS':1,'WIDTH':1, 'HEIGHT':1, 'EXTENT':'1811199.222700000,1851362.222700000,544124.972700000,596830.972700000 [ESRI:102003]', 'NODATA':-9999 ,'OPTIONS':'', 'DATA\_TYPE':5 ,'INIT':None, 'INVERT':False, 'EXTRA':'', 'OUTPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/LU/PLUTO 102003.tif'})

**The others:**

processing.run("gdal:rasterize", {'INPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/LU/Parks 102003.shp', 'FIELD':'', 'BURN':200, 'USE\_Z':False, 'UNITS':1, 'WIDTH':1, 'HEIGHT':1, 'EXTENT':'1811199.222700000,1851362.222700000,544124.972700000,596830.972700000 [ESRI:102003]', 'NODATA':-9999, 'OPTIONS':'', 'DATA\_TYPE':5 ,'INIT':None, 'INVERT':False, 'EXTRA':'', 'OUTPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/LU/Parks 102003.tif'})

processing.run("gdal:rasterize", {'INPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/LU/Parking 102003.shp', 'FIELD':'', 'BURN':500, 'USE\_Z':False, 'UNITS':1, 'WIDTH':1, 'HEIGHT':1, 'EXTENT':'1811199.222700000,1851362.222700000,544124.972700000,596830.972700000 [ESRI:102003]', 'NODATA':-9999 ,'OPTIONS':'', 'DATA\_TYPE':5 ,'INIT':None, 'INVERT':False, 'EXTRA':'', 'OUTPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/LU/Parking 102003.tif'})

**Python code for r.null**

processing.run("grass7:r.null", {'map':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/LU/PLUTO 102003.tif','setnull':'','null':0,'-f':False,'-i':False,'-n':False,'-c':False,'-r':False,'output':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/LU/PLUTO rnull.tif','GRASS\_REGION\_PARAMETER':None,'GRASS\_REGION\_CELLSIZE\_PARAMETER':0,'GRASS\_RASTER\_FORMAT\_OPT':'','GRASS\_RASTER\_FORMAT\_META':''})

processing.run("grass7:r.null", {'map':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/LU/Parks 102003.tif','setnull':'','null':0,'-f':False,'-i':False,'-n':False,'-c':False,'-r':False,'output':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/LU/Parks rnull.tif','GRASS\_REGION\_PARAMETER':None,'GRASS\_REGION\_CELLSIZE\_PARAMETER':0,'GRASS\_RASTER\_FORMAT\_OPT':'','GRASS\_RASTER\_FORMAT\_META':''})

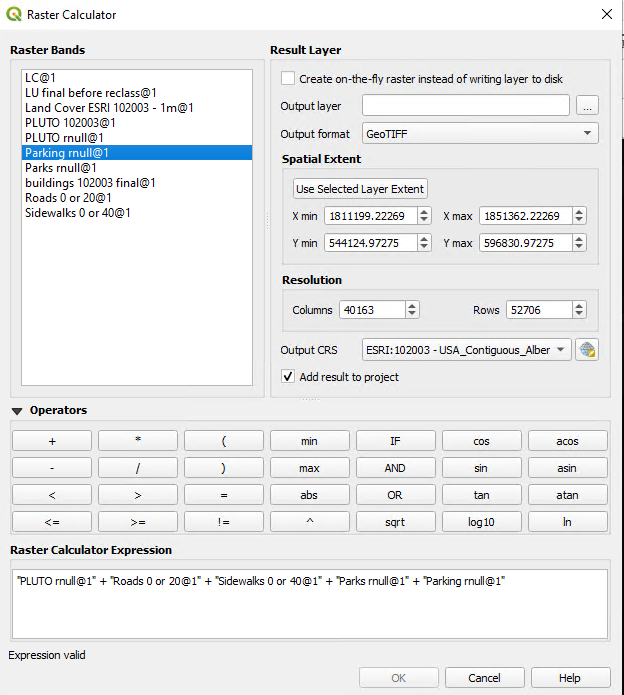
processing.run("grass7:r.null", {'map':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/LU/Parking 102003.tif','setnull':'','null':0,'-f':False,'-i':False,'-n':False,'-c':False,'-r':False,'output':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/LU/Parking rnull.tif','GRASS\_REGION\_PARAMETER':None,'GRASS\_REGION\_CELLSIZE\_PARAMETER':0,'GRASS\_RASTER\_FORMAT\_OPT':'','GRASS\_RASTER\_FORMAT\_META':''})

## Raster Calculations and Reclassification

The original PLUTO LandUse codes are:

1. 01 One & Two Family Buildings
2. 02 Multi-Family Walk-Up Buildings
3. 03 Multi-Family Elevator Buildings
4. 04 Mixed Residential & Commercial Buildings
5. 05 Commercial & Office Buildings
6. 06 Industrial & Manufacturing
7. 07 Transportation & Utility
8. 08 Public Facilities & Institutions
9. 09 Open Space & Outdoor Recreation
10. 10 Parking Facilities
11. 11 Vacant Land

In that case, we can do some simple raster algebra and reclassification to make this work. We’ll also add in parking lots and parks, since we coded them in the hundreds and can recode everything at once. We can also integrate roadbeds and sidewalks in that initial operation since we set them in the 200s and 400s, so everything should be well-spaced and avoid overlap.



This will take a little while, perhaps 20-30 minutes. Once this is complete, we should have the following file:

1. 00 = Water => 83
2. 01 = One & Two Family Buildings => 11
3. 02 = Multi-Family Walk-Up Buildings => 12
4. 03 = Multi-Family Elevator Buildings => 12
5. 04 = Mixed Residential & Commercial Buildings => 21
6. 05 = Commercial & Office Buildings => 22
7. 06 = Industrial & Manufacturing => 23
8. 07 = Transportation & Utility => 42
9. 08 = Public Facilities and Institutions => 39
10. 09 = Open Space & Outdoor Recreation => 32 (mostly cemeteries)
11. 10 = Parking Facilities => 41
12. 11 = Vacant Land => 70
13. 20-39 = Roads => 80
14. 40-59 = Sidewalks => 81
15. 60-99 = Overlap between roads and sidewalks, coded as roads => 80
16. 200 - 219 = Parks => 31
17. 220 - 239 = Roads in parks, coded as roads => 80
18. 240 - 299 = Sidewalks in parks, coded as parks => 31
19. 400 and up = Parking lots => 41 (includes some things in the 600s, parkings lots in parks. Decided to call these parking lots)

Once again we can use r.reclass. The text to copy-paste:

00 = 83

01 = 11

02 = 12

03 = 12

04 = 21

05 = 22

06 = 23

07 = 42

08 = 39

09 = 32

10 = 41

11 = 70

20 thru 39 = 80

40 thru 59 = 81

60 thru 99 = 80

200 thru 219 = 31

220 thru 239 = 80

240 thru 299 = 31

400 thru 999 = 41

Python code:

processing.run("grass7:r.reclass", {'input':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/LU/LU final before reclass.tif','rules':'','txtrules':'00 = 83 \n01 = 11 \n02 = 12 \n03 = 12\n04 = 21\n05 = 22\n06 = 23\n07 = 42\n08 = 39\n09 = 32\n10 = 41\n11 = 70\n20 thru 39 = 80\n40 thru 59 = 81\n60 thru 99 = 80\n200 thru 219 = 31\n220 thru 239 = 80\n240 thru 299 = 31\n400 thru 999 = 41','output':'C:/Users/jkhawes/Desktop/LocalSpatialData/Midpoint/LU.tif','GRASS\_REGION\_PARAMETER':None,'GRASS\_REGION\_CELLSIZE\_PARAMETER':0,'GRASS\_RASTER\_FORMAT\_OPT':'','GRASS\_RASTER\_FORMAT\_META':''})

Next, we can produce a slope layer to make sure we’re only looking at relatively flat spaces.

# Binary Slope Layer Derivation

This layer will describe the land use at a parcel level. These will be used primarily to sort the different types of gardens - e.g., it makes much more sense to assume an individual garden in a single-family backyard than a community garden. The final layer produced via this procedure will have the following codes:

* Not flat -- 0
* Flat ground – 1 (less than 15% slope)
* Flat roof – 2 (less than 5% slope)

Overview: We can use the LiDAR-derived DEM and DSM to identify flat ground and flat roofs throughout the city. We have to do these two things separately, since ground level varies across NYC and the DSM is reported in feet above sea level.

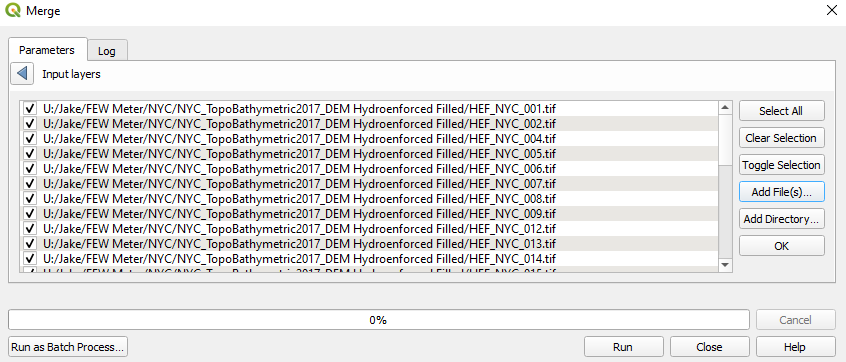
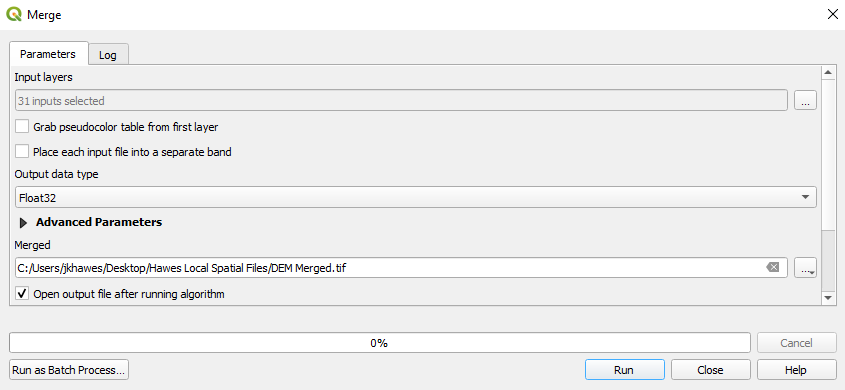
We will use the following data sets:

1. NYC highest-hit model (DSM) - [metadata](https://github.com/CityOfNewYork/nyc-geo-metadata/blob/master/Metadata/Metadata_HighestHitDigitalSurfaceModel.md)
2. NYC DEM - [metadata](https://elevation.its.ny.gov/arcgis/rest/services/NYC_TopoBathymetric2017_1_foot/ImageServer)
3. NYC Buildings dataset - [metadata](https://github.com/CityOfNewYork/nyc-geo-metadata/blob/master/Metadata/Metadata_BuildingFootprints.md) (in fact, we can use the NYC Buildings Raster created during the Land Cover derivation)

We will begin by preparing city-wide DEM and DSM layers, then we will proceed with the flat ground analysis. We will then identify flat roofs. Finally, we will mask buildings from the ground layer and add in the buildings results. In total, this should only take 30-40 minutes because so much of it is just big raster calculations.

## Create city-wide DEM and DSM layers

First, as a sort of step zero, we need to stitch together the DEM and DSM layers for NYC. They are delivered in tiles. This is fairly straightforward with the ***merge*** tool. We don’t even need to import the rasters in Q first (and we shouldn’t, because it would take a while) - instead we can select them directly from the merge tool by clicking on Add Files on the input layers option.

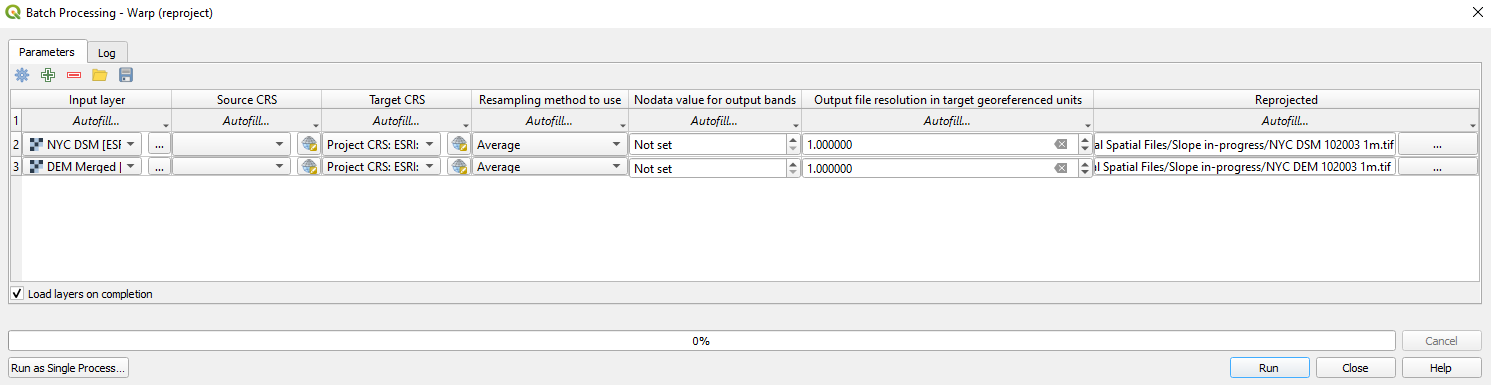


**Python code example:**

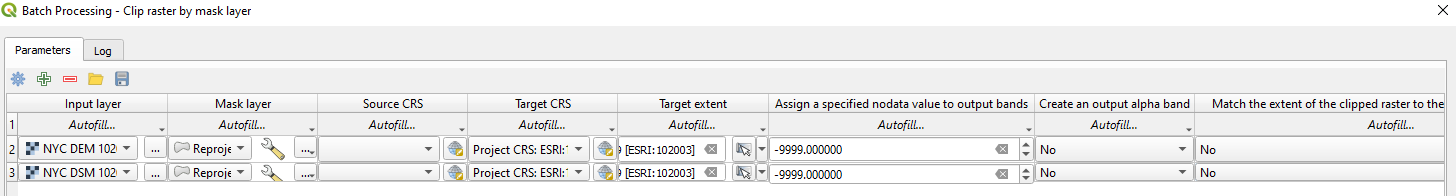
processing.run("gdal:merge", {'INPUT':['C:/Users/jkhawes/Desktop/LocalSpatialData/DEM/HEF\_NYC\_001.tif','C:/Users/jkhawes/Desktop/LocalSpatialData/DEM/HEF\_NYC\_002.tif'],'PCT':False,'SEPARATE':False,'NODATA\_INPUT':None,'NODATA\_OUTPUT':None,'OPTIONS':'','EXTRA':'','DATA\_TYPE':5,'OUTPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/DEM Example.tif'})

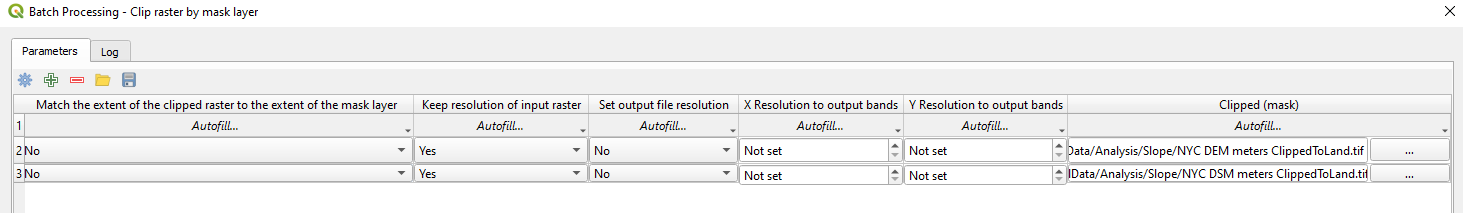
## Reproject DEM and DSM to ESRI: 102003 and convert height to meters

The two layers should be converted to 102003 to work with the rest of the derived data (at which point we can also convert them to 1m\*1m), and we need to make sure that the height values are also in meters. So the simplest solution is to ***Warp*** everything in batch mode, then divide both layers by 3.28084. The Warp procedure will be slow because the original file is very high-res, probably more than an hour to process both layers.

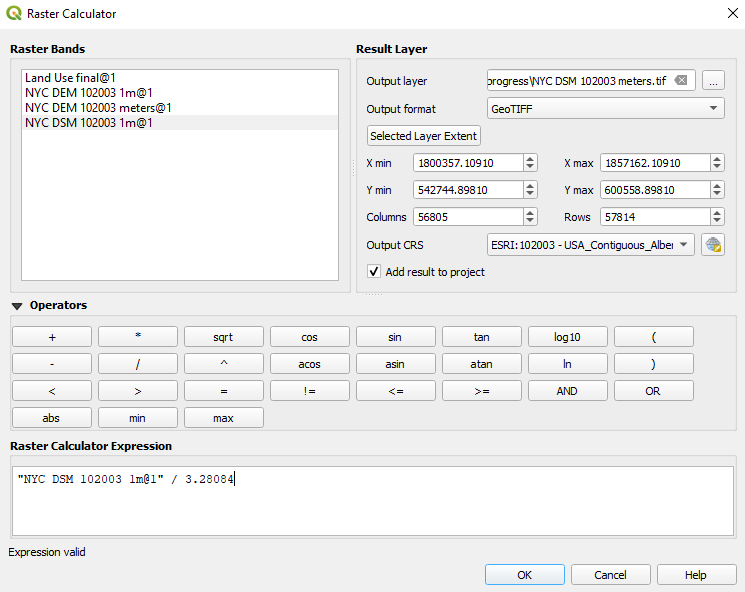


After the layers have been warped, it is useful to clip the rasters to the borough boundaries, since this will limit the computation time on the future layers. It is also convenient because it allows you to align the layers with the land use layer as you do this. Parameters below. As usual, I recommend setting the no data to -9999. Take care with the extent parameters and match below.



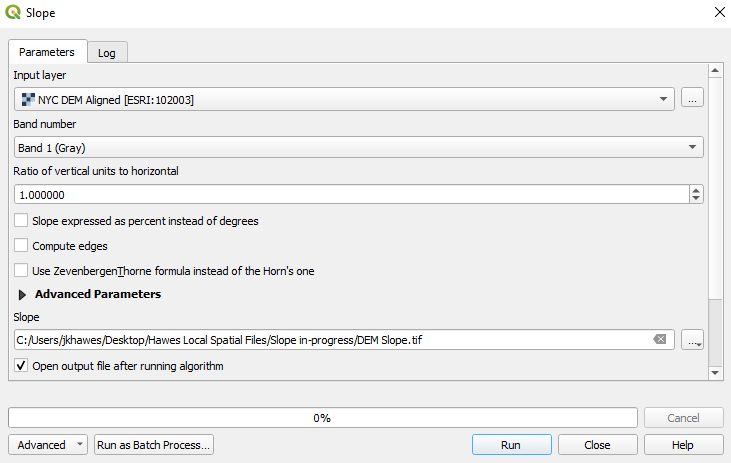


Once the raster has been shrunk down to size, we can proceed with the raster calculations that will give us the whole thing only in meters.



## Calculate Slope from the DEM

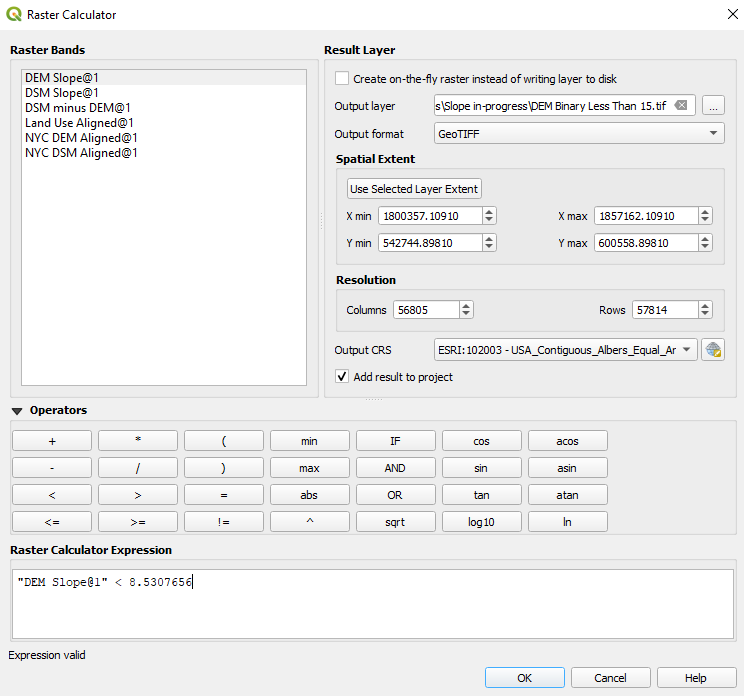
Now that we have clean layers to work with, we can derive slope on the ground. We can simply use the ***Slope*** function under Raster > Analysis on the hydroenforced DEM for NYC. This will yield a slope layer for all ground cells in the city. We’re interested in places where the slope is less than 15%. Unfortunately, the “Slope expressed as percent instead of degrees” function seems to return absolutely outrageous values, so I don’t recommend using that. Instead, it seems better just to convert the 15% to degrees and use that in the raster calculator in the next step. Note: you’ll need this same file for the DSM, so you can just run this as a batch process and get both.



**Python code:**

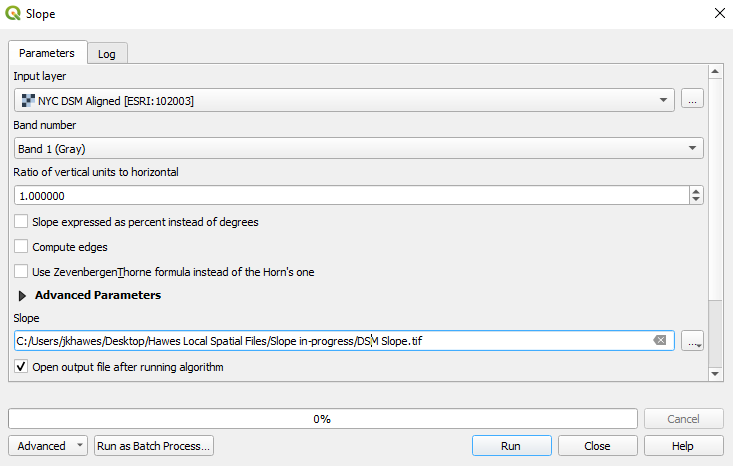
processing.run("gdal:slope", {'INPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Slope/NYC DEM meters ClippedToLand.tif','BAND':1,'SCALE':1,'AS\_PERCENT':False,'COMPUTE\_EDGES':False,'ZEVENBERGEN':False,'OPTIONS':'','EXTRA':'','OUTPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Slope/NYC DEM Slope.tif'})

Once we have this Slope file, we can do some simple raster algebra to determine where this is greater than and less than 15%. Expressed as degrees, a 15% slope is arc-tangent of 0.15, which is 8.5307656. So we want to find places where the slope layer is less than that.

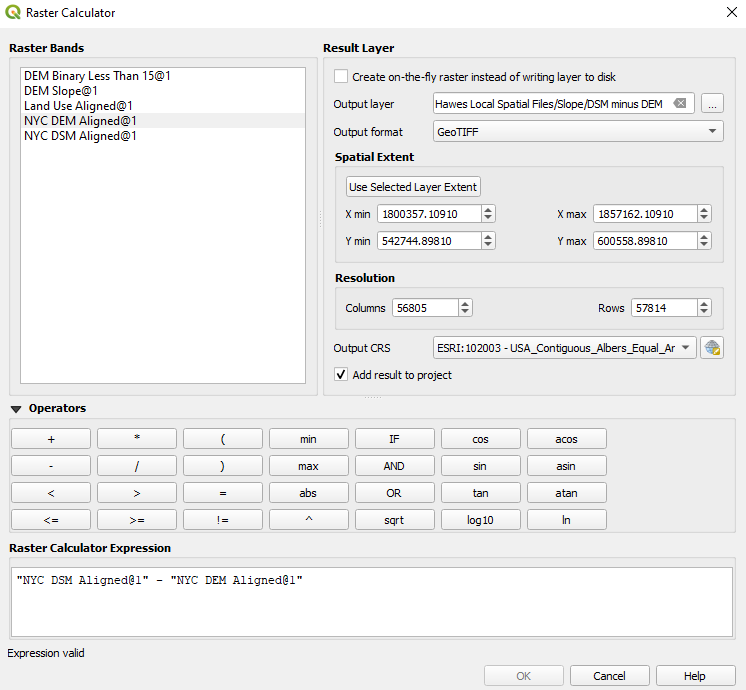


## Identify flat roofs from the DSM

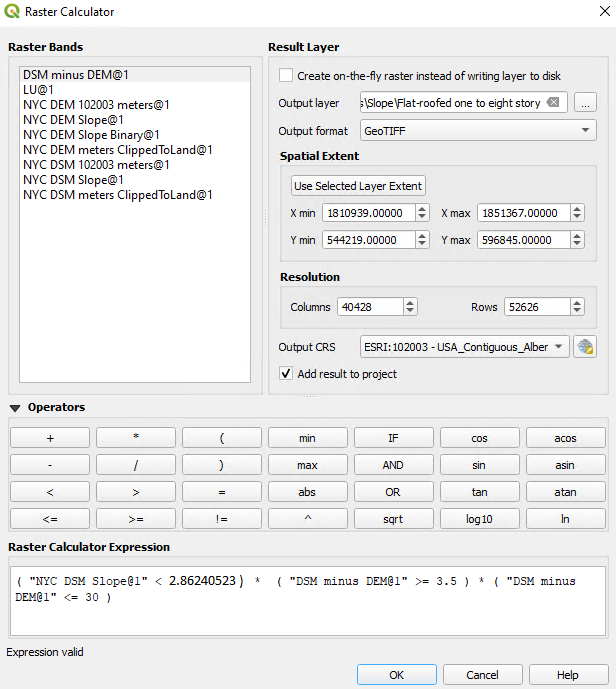
Next, we need to calculate the roof slopes - this is a bit trickier and requires several steps. First, we can run the Slope function under Raster > Analysis on the DSM (you might’ve already done this with batch processing above).



Next, when we need to subtract the DEM from the DSM to make sure the ground level is zero all over the map.

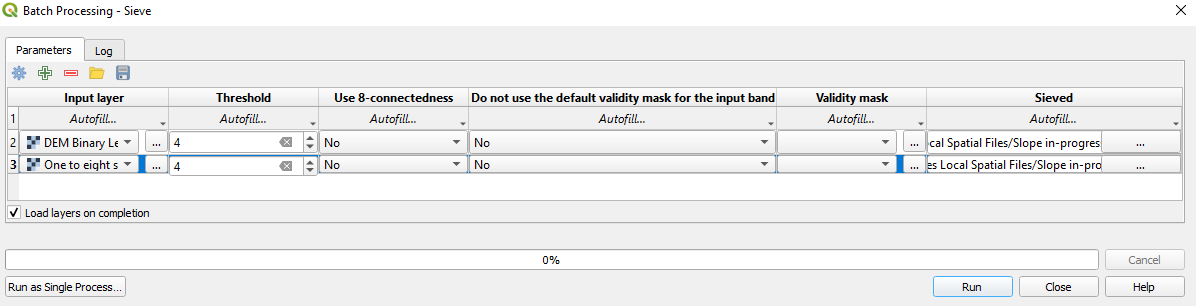


We can then run some simple raster calculations, finding the places where the Slope layer is less than 5% (less than arctangent of 0.05 = 2.86240523) and the Height of the DSM-DEM layer is less than 30m (8 stories or less) - note that this height is why we need to do the subtraction. If not, we can’t use 30 or something as a roof height, because ground level differs, so some roofs are below ground level in other places in the city. In some older versions of this, I also used 3.5 m as a minimum, but this really isn’t necessary since we use the building land cover classification below. Just need to limit the max so we’re not working on top of the Empire State building.



## Clean up layers with the sieve tool

Once we have useful results for both round and rooftops, we can use the sieve tool to clean it up a bit. I used 4 as a threshold because it tends to catch edge cases most effectively. Make sure to fix the symbology to 0-1 after running the function.



**Python code:**

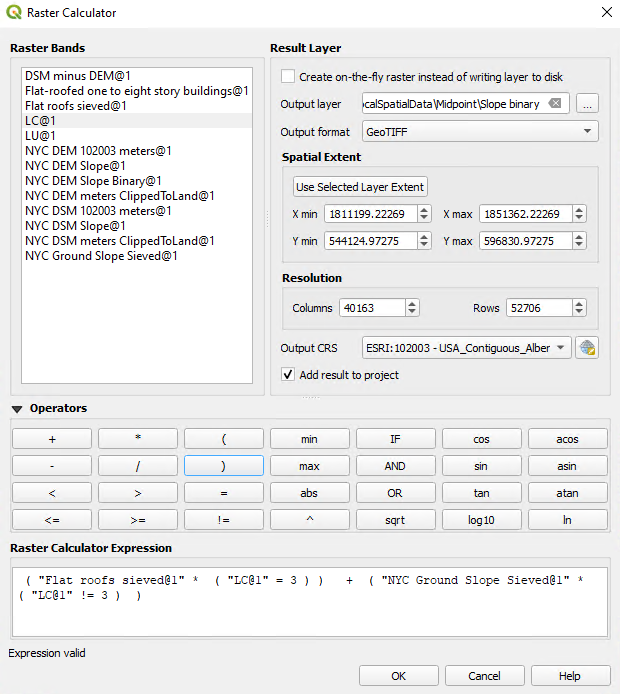
processing.run("gdal:sieve", {'INPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Slope/Flat-roofed one to eight story buildings.tif','THRESHOLD':4,'EIGHT\_CONNECTEDNESS':False,'NO\_MASK':False,'MASK\_LAYER':None,'EXTRA':'','OUTPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Slope/Flat roofs sieved.tif'})

processing.run("gdal:sieve", {'INPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Slope/NYC DEM Slope Binary.tif','THRESHOLD':4,'EIGHT\_CONNECTEDNESS':False,'NO\_MASK':False,'MASK\_LAYER':None,'EXTRA':'','OUTPUT':'C:/Users/jkhawes/Desktop/LocalSpatialData/Analysis/Slope/NYC Ground Slope Sieved.tif'})

## Final raster calculations

Lastly, we need to combine these layers into something intelligible - we will keep flat roofs and flat ground separate for now just for the sake of preserving information. We can always reclassify later. So our goal is: 0 = > 15% or 5% slope, 1 = flat ground, 2 = flat roof. How do we get there?

First, we’ll need to make all the building footprints zero in the ground slope layer and convert everything outside of buildings to zero in the roof slope layer. To keep the information discrete,, we can do one more raster calculation - RoofsLayer \* 2 + GroundLayer. In the end, we can do this all in one step. For the building layer conversion, we can import the building layer used in the previous steps. Since everything has been aligned, we should be able to use it directly. The raster calculation is exactly the same for the two layers, but inverted. See below.



After this calculation is complete, don’t move all the Slope files off the hard drive just yet - make sure to keep the aligned DSM - we will use it as our first input.

# Sunlight Availability

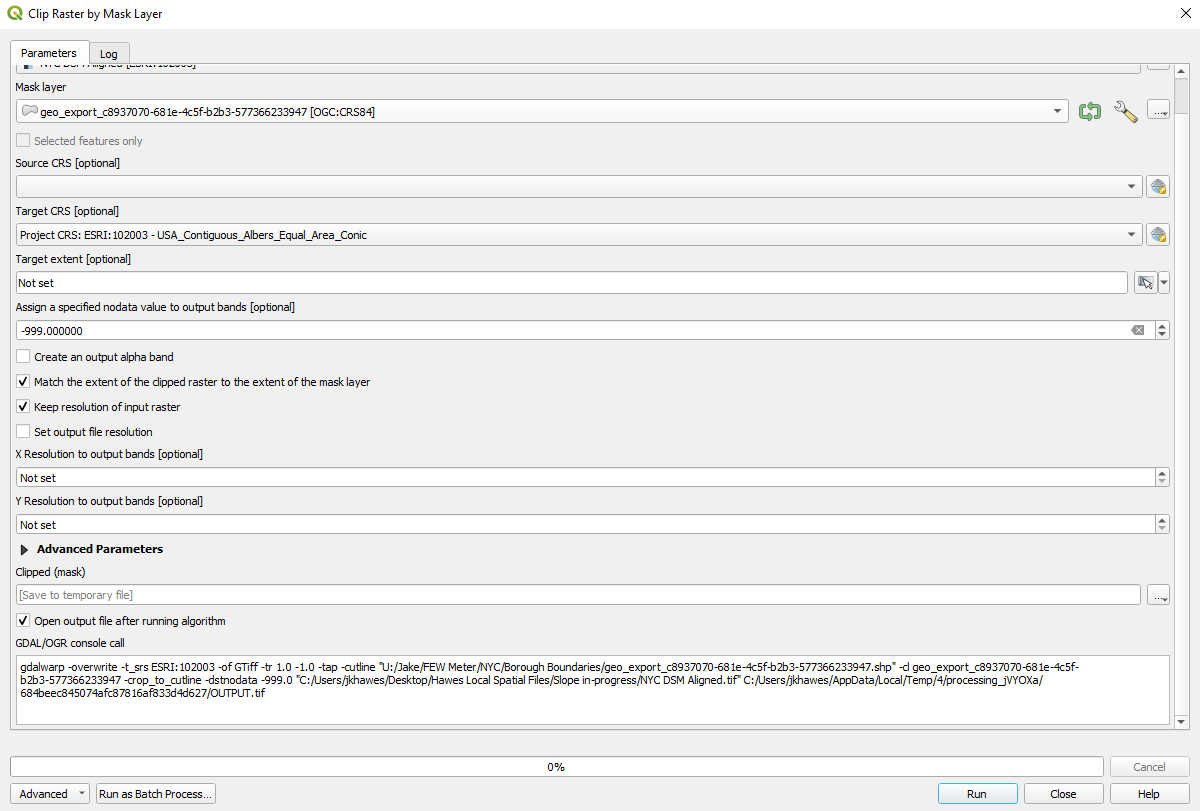
The final layer we will derive is a sunlight availability layer. This will take advantage of the r.sun package in GRASS, which takes the DSM and converts it to solar irradiance. This one is fairly complex, so I derive my process from an [example done in Canada](https://www.sciencedirect.com/science/article/pii/S0038092X10000812?via%3Dihub#fn15). Based on that paper, we will need a few inputs to make this work, including:

| Canadian layer/example | NYC layer/notes | Source link |
| --- | --- | --- |
| Digital Elevation Model (DEM) | DSM - include shading from buildings and trees |  |
| Slope/inclination | Derived from DSM |  |
| Aspect/orientation | Derived from DSM |  |
| Latitude | Not necessary if we use a proper projection (102003) |  |
| Albedo: the ratio of diffusely reflected radiation on a surface to its incident radiation. | Albedo can probably be calculated for each city with i.albedo function, but we can also use urban averages for generalization. For i.albedo, just need landsat imagery: https://grass.osgeo.org/grass78/manuals/i.albedo.html |  |
| Mean days and corresponding angular position of the sun. | Can use the same mean days if we do want to do the calculation for every month. “Table 1.6.1 in Duffie and Beckman (1991) readily provides the day of month, day of year and δ (sun declination) values to input into the simulation -- J.A. Duffie, W.A. Beckman, Solar Engineering of Thermal processes (second ed.), John Wiley & Sons (1991)” |  |
| Linke turbidity: a convenient approximation to model the atmospheric absorption and scattering of the solar radiation under clear skies. | If all we want is very high level stuff, we can get that from the same place the example paper did. Resolution is about the scale of NYC. Have three different raster cells for whole city, all the same value. Able to make a raster with the resolution of our DEMs and DSMs by downsampling. | <http://www.soda-pro.com/help/general-knowledge/linke-turbidity-factor> |
| Ground-measured values of global horizontal irradiation (GHI). | Available from NASA SSE POWER program - GHI is the first value (ALLSKY\_SFC\_SW\_DWN CERES SYN1deg All Sky Surface Shortwave Downward Irradiance (kW-hr/m^2/day)) while GHI under Clear-Sky conditions is the second value (CLRSKY\_SFC\_SW\_DWN CERES SYN1deg Clear Sky Surface Shortwave Downward Irradiance (kW-hr/m^2/day)) | Available at a 1x1 degree resolution. It claims to be ½ by ½ but doesn’t seem to output that for 2019 at least. – <https://power.larc.nasa.gov/data-access-viewer/> |
| Clear sky index Kc: “Ratio of the global horizontal irradiance to the global horizontal irradiance under clear sky conditions. It is important not to confuse and hence misuse this definition with those for insolation clearness index and clear sky insolation clearness index.” | Available in the POWER suite of indicators as ALLSKY\_KT. We do not use the normalized value - this transforms the Kc with the latitude - Kc = shortwave direct horizontal (GHI) / shortwave direct top-of atmos -- I haven’t figured out the use for the normalized parameter yet. Maybe comparing different locations? See [here](https://www.star.nesdis.noaa.gov/smcd/emb/radiation/solar_resource_definitions.php) for simple definitions. See [here](https://www.star.nesdis.noaa.gov/smcd/emb/radiation/documents/SRDB_1.0_Parameter_Definitions.pdf) for other details.  Definitions (rather unhelpful, except it explicitly mentions GHI): <https://power.larc.nasa.gov/#resources> | The regional data access panel at the website above allows “NetCDF” export, which can be imported as a raster in Q: <https://ereefs.aims.gov.au/ereefs-aims/help/how-to-open-a-NetCDF-file-with-ArcMap-and-QGIS>  So basically we turn this into a raster of the appropriate resolution and multiply to get a final value from r.sun. |

With this set of inputs, we should be able to calculate the shading effects in essentially any city around the world. The DSM is the hardest thing to find, and we should be able to simulate this with building height data, which is more often available. Even better, we don’t need the GHI or the Kc if we don’t care about the strength of the sunlight and just total sun hours on a typical day. See below for a step-by-step for just sun hours.

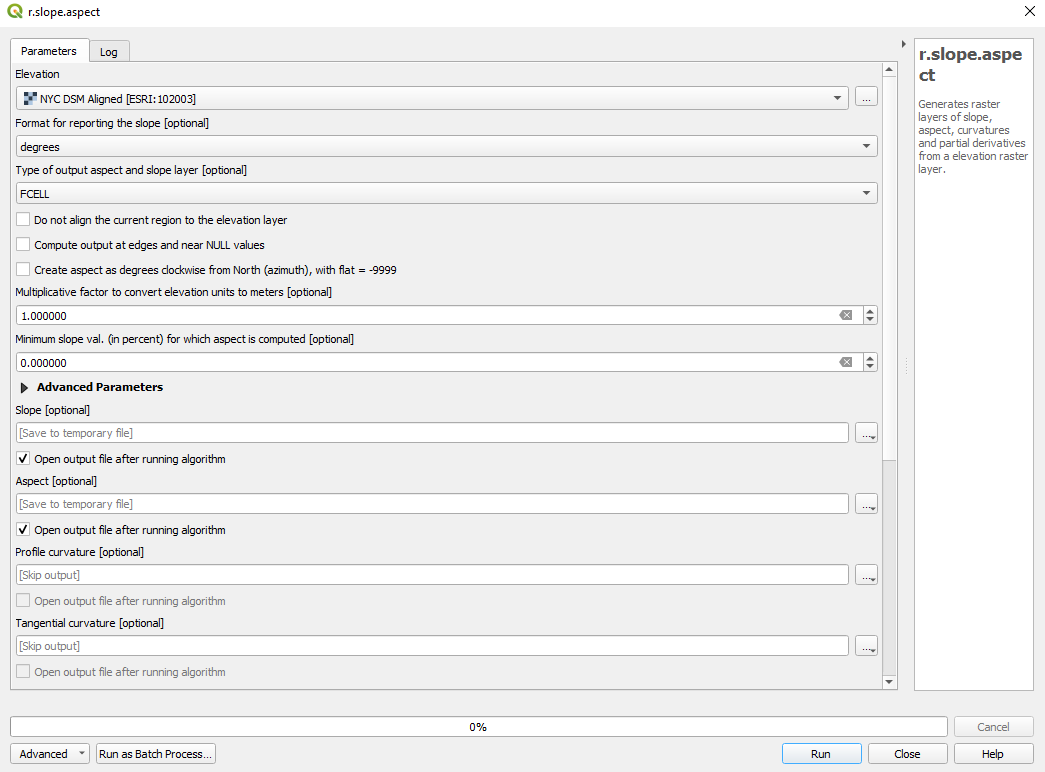
## If not already done: clip DSM to work better with GRASS

For some reason, of all the commands we use, the GDAL commands embedded in r.sun are the only ones that use the default maximum raster size, and they won’t save anything too big. So before moving forward, we have to clip the DSM to work with this limitation. This can be achieved by clipping to the borough boundaries. If you followed these instructions, you’ve already done this. If you’re just here for the sunlight stuff and skipped the earlier content, see the settings below.



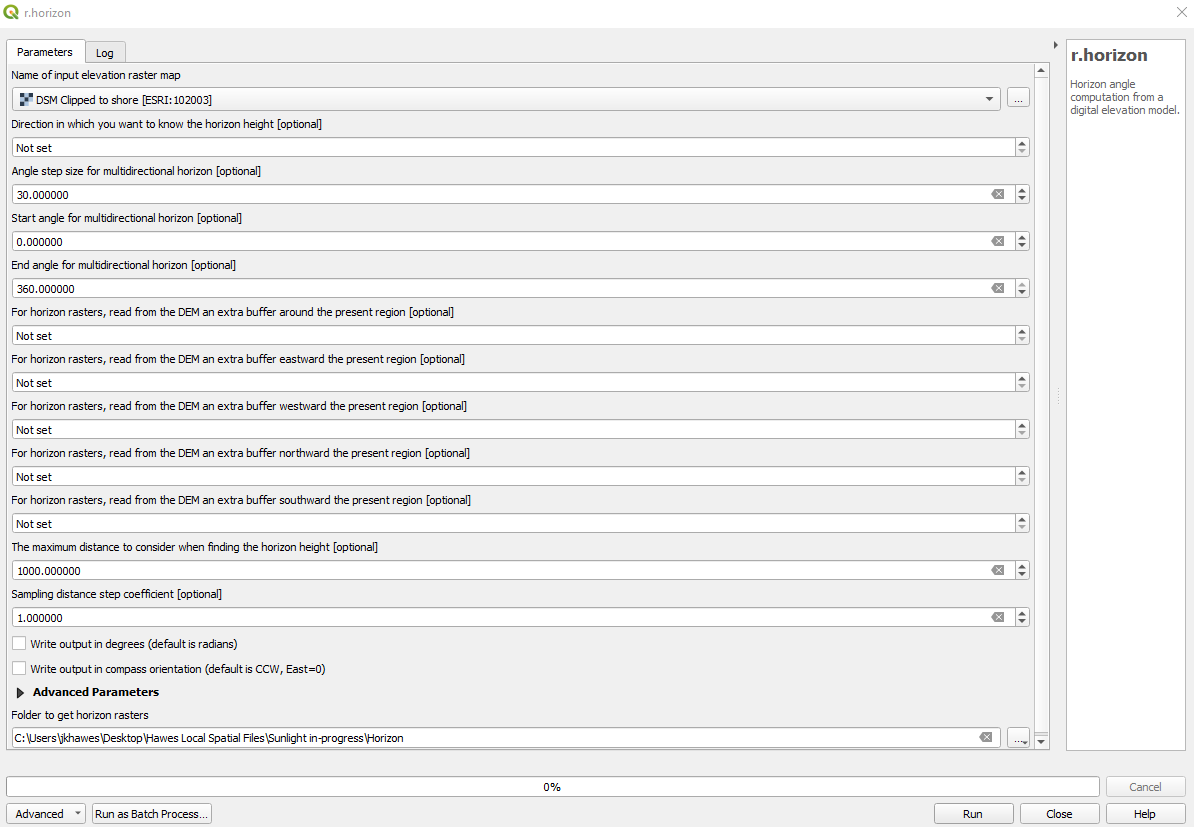
## r.slope.aspect

Once we have a useable DSM, we need to produce maps of the slope and the aspect based on our DSM. We could use the DSM slope map we already have, but it’s just as easy to just run it all within GRASS to make sure everything is formatted the way r.sun wants it to be. This command is fairly straightforward, only a couple things need to be customized. First, we need to uncheck the box that asks about aligning with the elevation region. We do want to align all of our calculations with that region. Second, we want to suppress the outputs other than slope and aspect. No reason to spend time calculating things we won’t use.

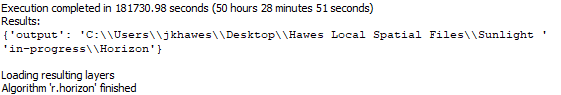


## Optional: r.horizon

If you are planning to use GRASS GIS more directly via commands, it can be very helpful to run r.horizon to determine the horizon height at all locations in the city. Unfortunately, if you plan to run the r.sun suite via QGIS, the interface does not play nice with loading an entire directory, which is the required format of the r.horizon output and the r.sun input, so we have to generate the horizon as part of the r.sun.insoltime below and don’t worry about it here.

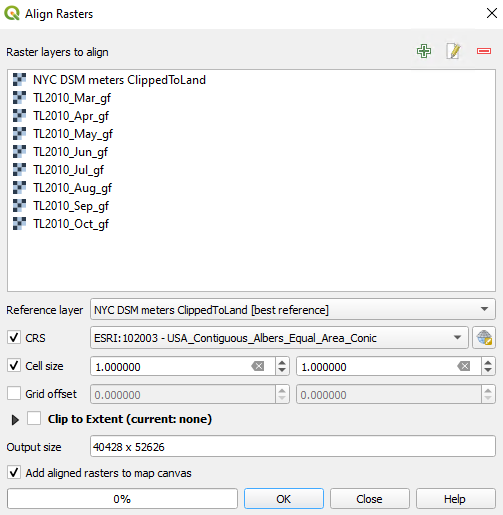


If you do go through GRASS to do this, it will take a very long time to run - on the order of 50 hours on the computer in our lab at 1m resolution. In fact, one of the first times I ran this, it ran for 50 hours and only saved 120-360, so you may have to run it twice to convince it to save everything correctly.



## Aligning Linke

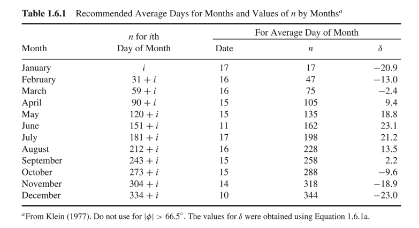
The last step before running the final command is to align the Linke values with the files we’ll be using. This will also clip the raster and will probably take several hours to run them all at once. I’d just plan to leave it overnight if you can.



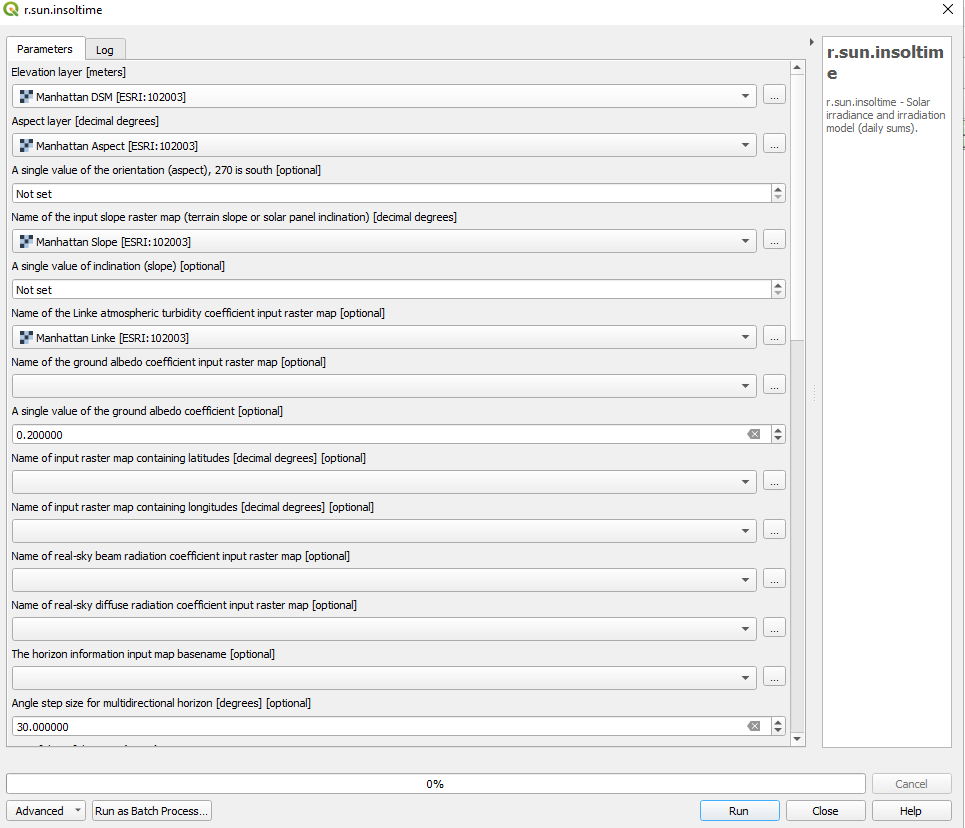
Once the Linke values have been converted to a 1x1m raster, we should be able to clip this to the borough boundaries, as we’ve done with the others. There’s no Python code for this one - much like raster calculator.

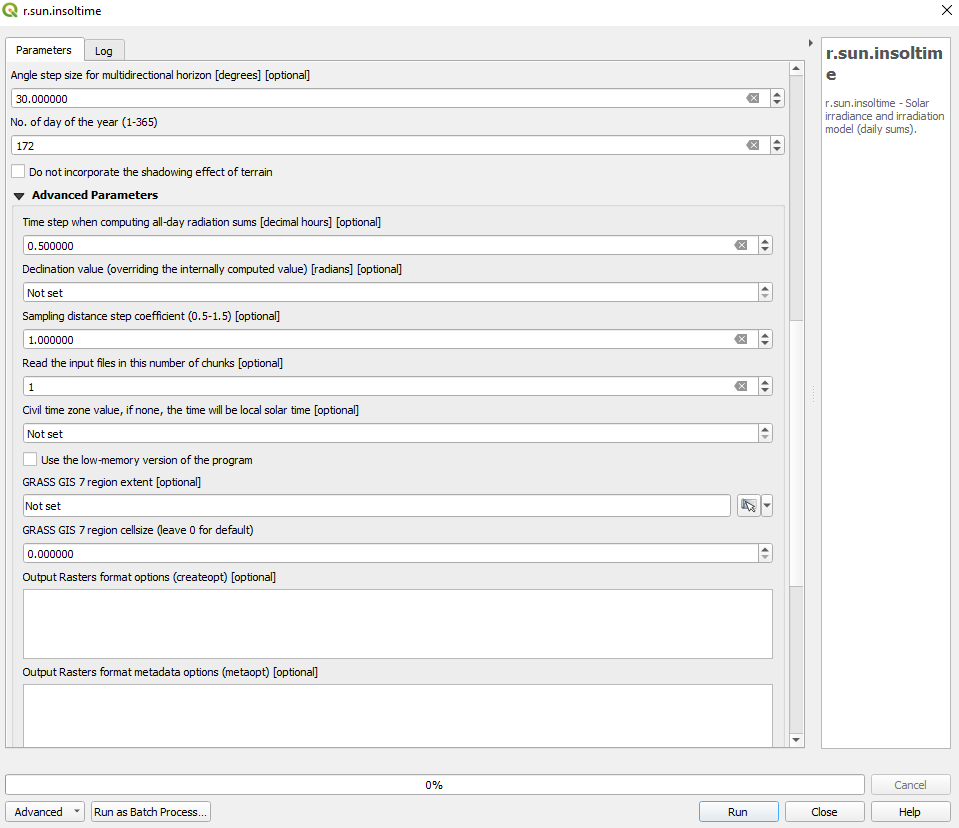
## r.sun.insoltime

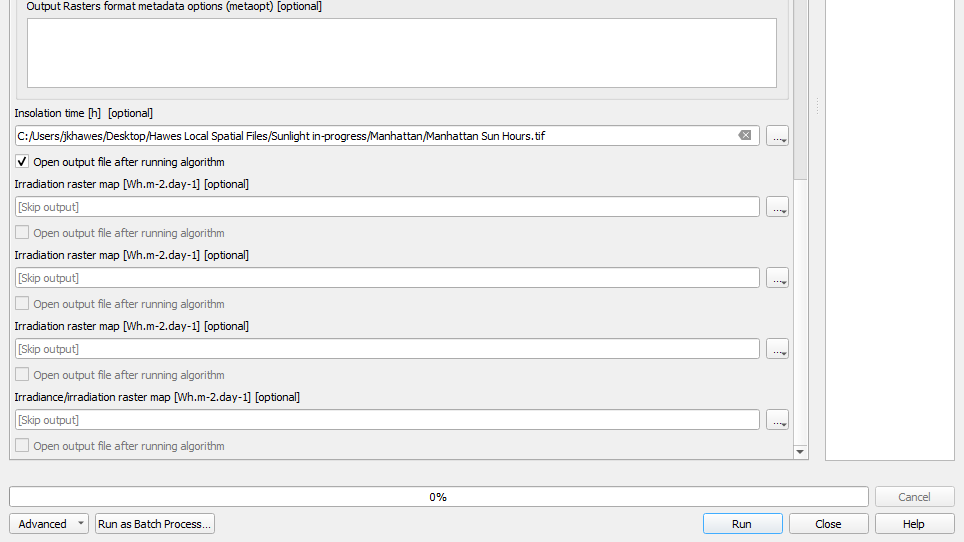
Since we are strictly interested in the number of hours of sunlight, we can simply ignore the more complex aspects like levels of radiation from the NASA data. We can retrieve average days from the book cited in the Canadian paper:



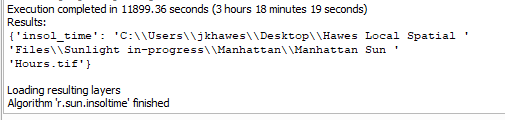
This means that the inputs look like this:



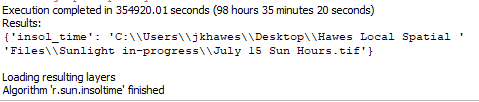




These commands run slowly, but stubbornly. I first ran it on just Manhattan and it took about 4 hours. This may be a useful experiment for you to test your inputs before letting it run for several days on the whole city.



I then ran it on the whole city, and it took about 4 days. This is not linear with the area used, but this makes sense, since the thin shape of Manhattan expedited the whole process a lot (and caused errors). Despite that extremely long calculation, it worked quite well.





Following this method, I calculate irradiation hours for March, July, and September and take the average across the growing season. From there, I am able to identify places that receive at least six hours of sunlight per day on average during the growing season.

# Old

## Align with Land Use final layer

Since the DEM and DSM came in as their own rasters and weren’t based on the land cover layer, they’re the first ones we need to manually align. THe Align Rasters tool can take all three rasters (including the land use layer) and produce an “aligned” version of each of them. Obviously we don’t need to worry about the new Land Use layer, but the new DEM and DSM are what we’ll use from here on.

