NYC Scaling - Facilities Layer Derivation

By: Anonymized for review

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 various QGIS versions 3.18 thru 3.32, though anything past 3.00 should suffice. When possible, I will include both a verbal explanation and a screenshot for the procedure described - this should make it easier to replicate the process in varying versions where the syntax or appearance may change slightly.

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.

These layers are next used in the [Scenario Derivation](https://docs.google.com/document/d/1Qqc_rs4k0YnrqffdWDM0vMUaHDXa2YlYBBzXgeXScZ0/edit), where we identify the parcels of interest for location-allocation. Once we have the parcel layers of interest, we’ll move on to final scenario generation via location-allocation optimization, a process described here.

# 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 or tree - 1 (trees assumed to be covering impervious so as not to overestimate capacity)
* Grass or dirt - 2
* Roof - 3
* Otherwise occupied - 0 (e.g., monument, water, railroad, road)

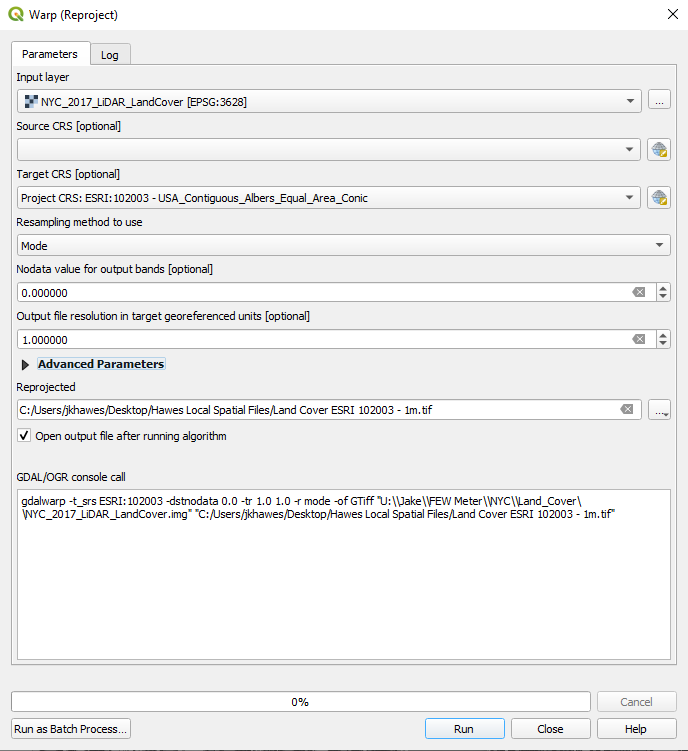
Overview: We have a land cover dataset that needs to be simplified to not have trees and needs to parcel out ineligible impervious areas like roads.

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.

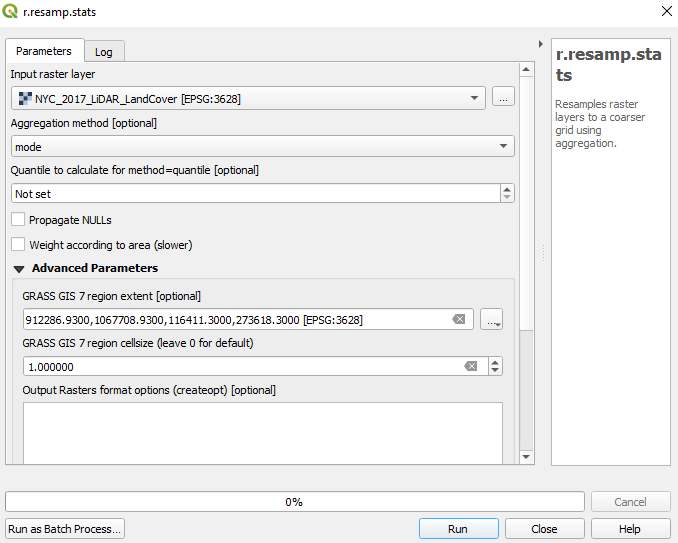
## Reproject, resample, and reclassify original land cover layer

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 remain 1, since we are assuming that anything under trees (on the ground) is just concrete.
* 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 PLUTO 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.

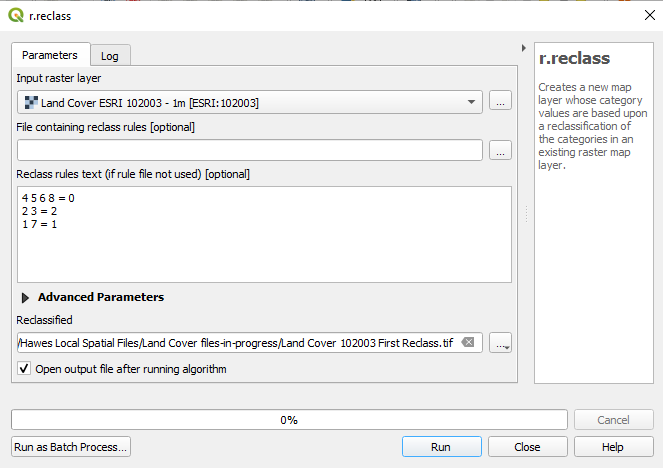
In other words, the following code is fed to r.reclass:

4 5 6 8 = 0

2 3 = 2

1 7 = 1

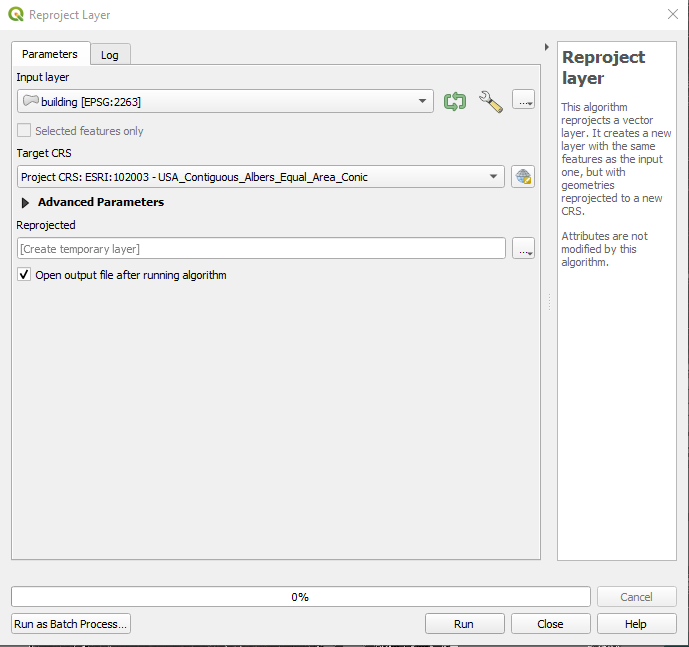
When running the command, you may run into CRS issues. If your final layer ends up somewhere odd (mine wound up in Cape Cod), try setting the CRS by right-clicking >> Layer CRS >> Set Layer CRS and changing it back to 102003.



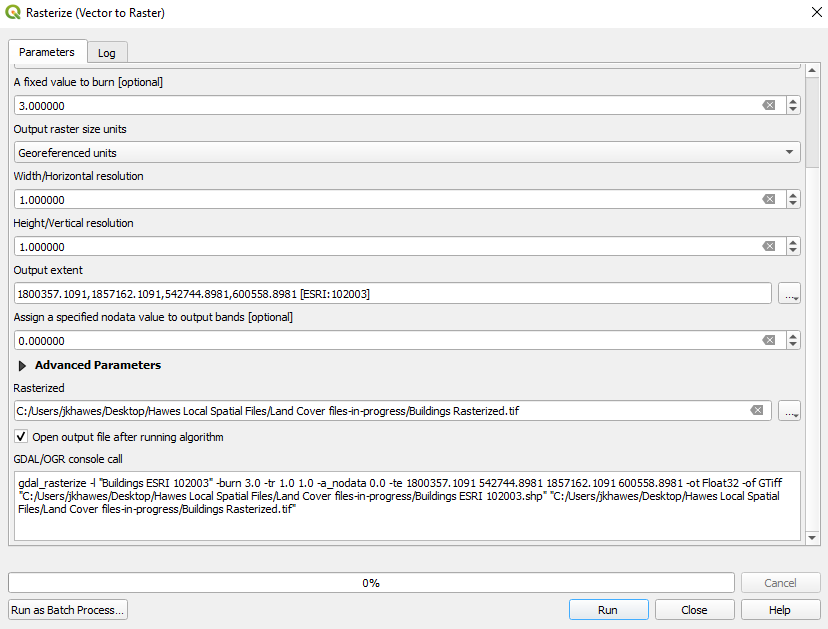
## Reproject and rasterize building, sidewalk, and roadbed layers

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 of 3 where the buildings are for the purposes of raster algebra, we can go ahead and set that now. 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.



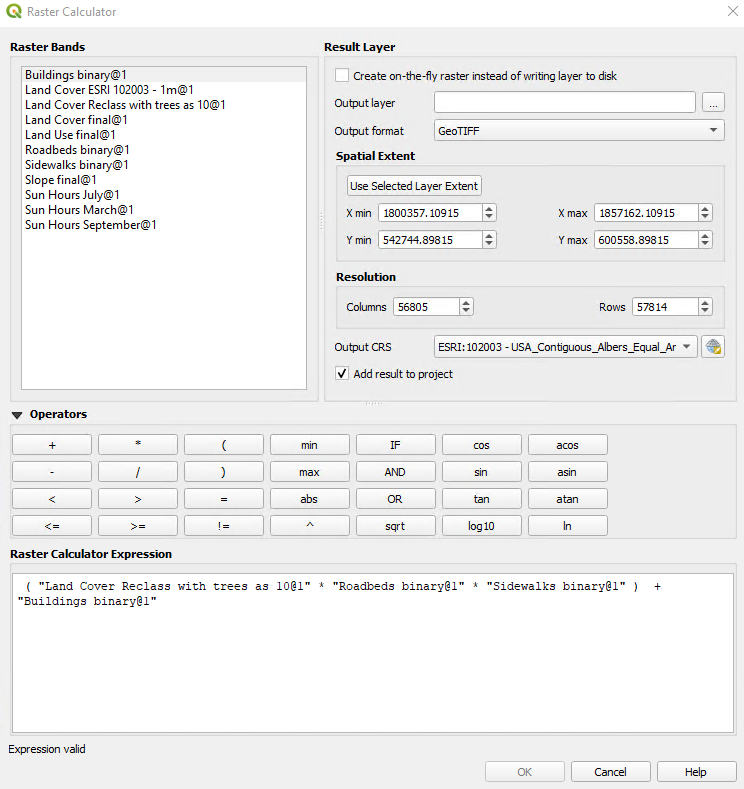
Proceed exactly as above with the Sidewalks and Roadbeds datasets. Reproject them, then rasterize them. Each one of these should be set to a binary layer upon rasterization, where the sidewalks and roads are 0 and everything else is 1. To accomplish this, set the burn in value to 0 and the no data default value to 1. We'll fix this after we run the command.

Once all the rasterization is complete, we'll need to complete one last step before running the raster algebra. We used the No Data value strategically -- we need to go in and cancel this setting under Properties >> Transparency for each layer. Once we do this, we should be able to do our simple algebra without returning lots of No Data. If for some reason this fails, you may find r.null very useful for filling in the no data on one or other files.

## Synthesize Land Cover Layer

Finally, we can bring all four layers together and do the final reclassification. 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 simple algebra is (LandCover \* Roads \* Sidewalks) + Buildings. Land Cover already has unusable spaces as 0, paved spaces and trees as 1, and grass/dirt as 2. We now make sure that any roads and sidewalks are 0 by multiplying the layers where we set those to 0, and we add buildings back in with the layer where we coded them as 3. This will result in a layer with values 0-5, which we will deal with in the final step below.



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 (which is completely unusable space)
* 1=1 (which is impermeable or under a tree so we assume it's impermeable
* 2=2 (which is grass or dirt)
* 3 4 5 = 3 (which is buildings) (if you use trees as 10, it’s actually 3 4 5 13 = 3 (for buildings) and 10 11 12 = 4 (for trees))

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.

# Land Use and Ownership 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 -- 80 (PLUTO Code 11)
* Roads and sidewalks – 90 (From CityMap)
* Existing Ag – 100 (From other data)

The ownership codes (CMOPX) will get turned into 12345, respectively. We’ll do this fairly straightforward recoding in parallel.

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 borough boundaries - [download and metadata](https://data.cityofnewyork.us/City-Government/Borough-Boundaries/tqmj-j8zm)
3. NYC Sidewalks dataset - [download and metadata](https://data.cityofnewyork.us/City-Government/Sidewalk/vfx9-tbb6/data?no_mobile=true)
4. NYC Roadbeds dataset - [download and metadata](https://data.cityofnewyork.us/City-Government/Roadbed/xgwd-7vhd/data?no_mobile=true)
5. NYC Parking lots dataset - [download and metadata](https://data.cityofnewyork.us/City-Government/Parking-Lot/h7zy-iq3d/data?no_mobile=true)
6. NYC Parks dataset - [download and metadata](https://data.cityofnewyork.us/Recreation/Parks-Properties/enfh-gkve) (actually a combination of parks and playgrounds)

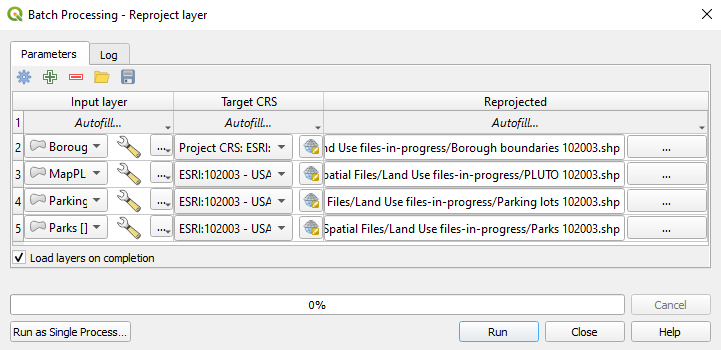
All of these are updated fairly regularly, so if it’s been a while since you ran this analysis (> 6 months), it might be helpful to refresh the data being used.

## 

## 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.

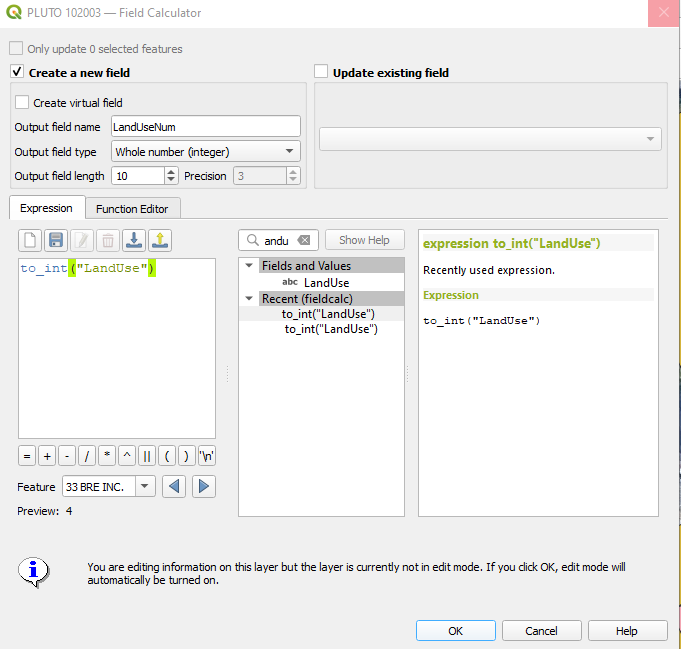
We can do this as a batch process to save a little time. I find it simplest to save them all as shapefiles, since occasionally different formats can interact with different algorithms in weird ways. To start a batch process, click the bottom-left button on the Reproject Layer screen. This will bring 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.



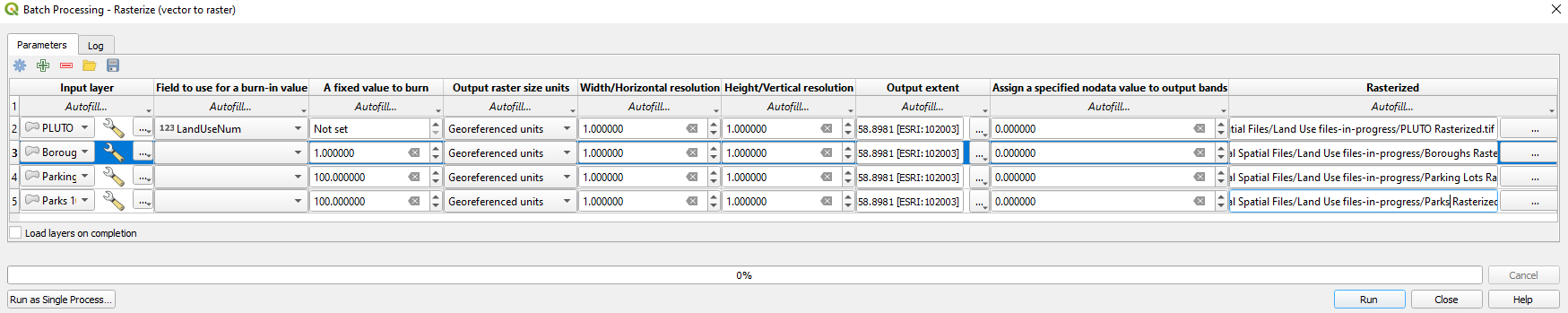
## Rasterize all layers

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 100 with nodata as 0, and Parking lots hardcoded to 100 with nodata as 0.

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. 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. (You may have to zoom in to make out the parameters, or see the original image at this [link](https://drive.google.com/file/d/19C7_LJ62c3YyzcypGL3Mc0emKWtjObI1/view?usp=sharing))



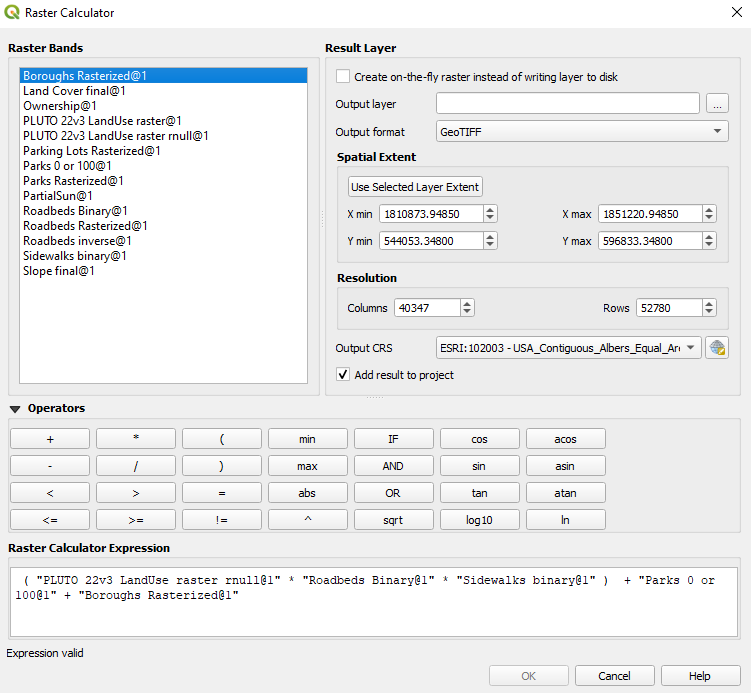
After this runs, check the results in the streets and other areas that PLUTO doesn’t cover. You may need to run r.null and add 0s in in place of No Data.

## 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. There are three quirks here. First, we need to do some fancy raster algebra to make sure our land use layer doesn’t include water as roads. Technically this isn’t a big deal, since we have a land cover layer, but we’re being extra thorough by adding in this step. Second, we’ll add in parking lots and parks one at a time, since we coded them both as 100 and it’s just safer if we do it in stages. Third, since we’ll be recoding anything greater than 100 to a certain value, it’s easier to use “Reclassify by Table” instead of r.reclass. So to start off, we need to add in parks, at which point we’ll also use the boroughs base layer to add 1 to everything and make sure that water is the only true 0. We can also integrate roadbeds and sidewalks in that initial operation. Once everything is ready, run the following calculation:



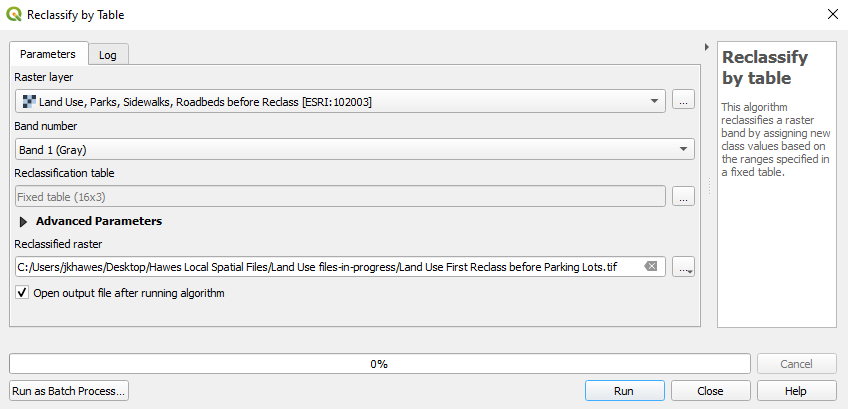
This will take a little while, perhaps 10 minutes. Once this is complete, we should have the following file:

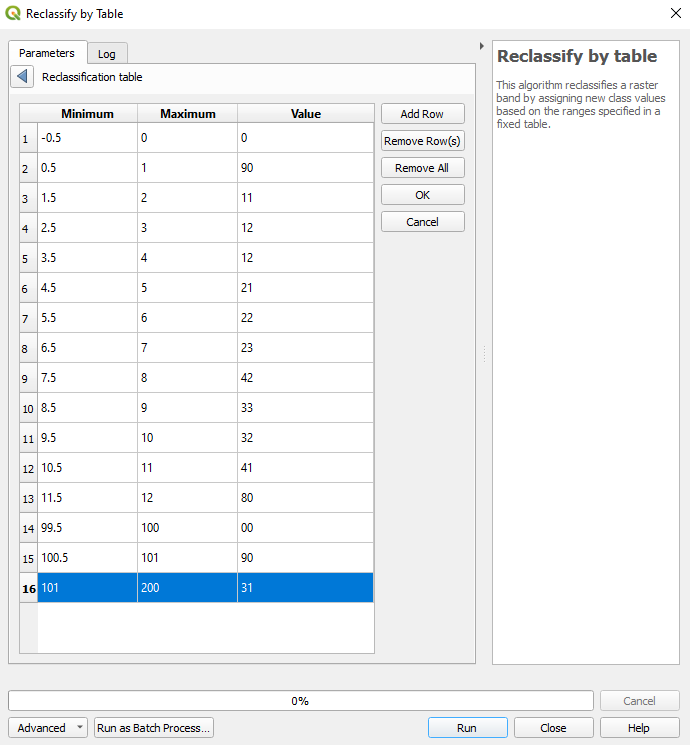
1. 00 = Water
2. 01 = Roadbeds, sidewalks, and edges between files
3. 02 = One & Two Family Buildings
4. 03 = Multi-Family Walk-Up Buildings
5. 04 = Multi-Family Elevator Buildings
6. 05 = Mixed Residential & Commercial Buildings
7. 06 = Commercial & Office Buildings
8. 07 = Industrial & Manufacturing
9. 08 = Transportation & Utility
10. 09 = Public Facilities & Institution
11. 10 = Open Space & Outdoor Recreation
12. 11 = Parking Facilities
13. 12 = Vacant Land
14. 100 = Water in parks (or the edge of land that overlaps with mapped water)
15. 101 = Roadbeds, sidewalks in parks
16. >101 = Parks

This turns into the following reclassify by table:

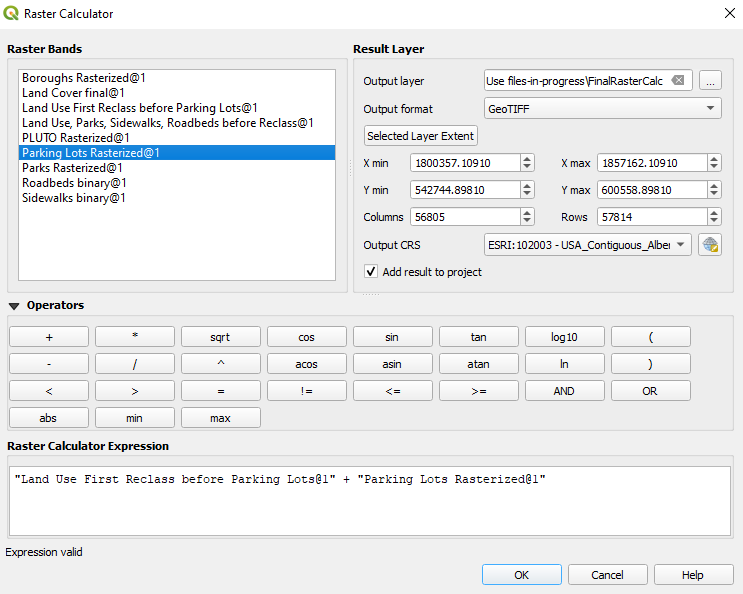
1. 00 = Water => 00
2. 01 = Roadbeds, sidewalks, and edges between files => 90
3. 02 = One & Two Family Buildings => 11
4. 03 = Multi-Family Walk-Up Buildings => 12
5. 04 = Multi-Family Elevator Buildings => 12
6. 05 = Mixed Residential & Commercial Buildings => 21
7. 06 = Commercial & Office Buildings => 22
8. 07 = Industrial & Manufacturing => 23
9. 08 = Transportation & Utility => 42
10. 09 = Public Facilities and Institutions => 33
11. 10 = Open Space & Outdoor Recreation => 32
12. 11 = Parking Facilities => 41
13. 12 = Vacant Land => 80
14. 100 = Water in parks => 00
15. 101 = Roadbeds, sidewalks in parks => 90
16. >101 = Parks => 31

We should end up with very little of 60 outside of cemeteries and we will capture a number of schoolyards included in the more detailed parks file. Note that Reclassify by table operates on a (Min,Max] systems such that you need the min to be lower than your actual min and you max to be equal to your actually max. See below for one possible solution to this.

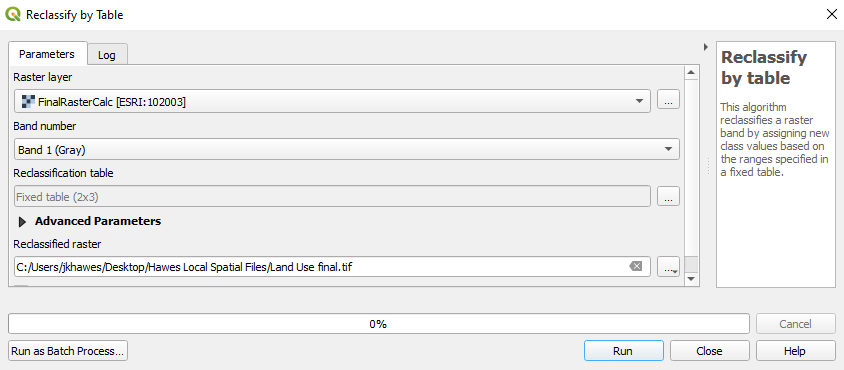


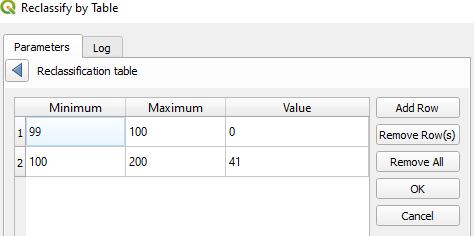


This should run in about 3.5 minutes. Once this first round of calculation is completed, we can do the same thing with the Parking Lots layer to produce our final Land Use layer. Using Raster Calculator, simply add the two files together (again, making sure that the no data box is unchecked under preferences).



Once we add Parking Lots, it should be that everything stays the same except that >100 is coded to 41. We also need 100 to be set back to 0 again, because there are number of parking lots on the edge of the city that get included in the shapefile (e.g., the north side of the Bronx).





Now we have a high-resolution land use layer with the right level of detail. The only quirk is that the borough boundaries layer missed a few inland waterways, so those are coded as roads. Fortunately, the land cover layer should take care of that. Next, we can produce a slope layer to make sure we’re only looking at relatively flat ground.

# 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:

* Ineligible, over 15% grade-- 0
* Flat ground – 1
* Flat roof – 2

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.

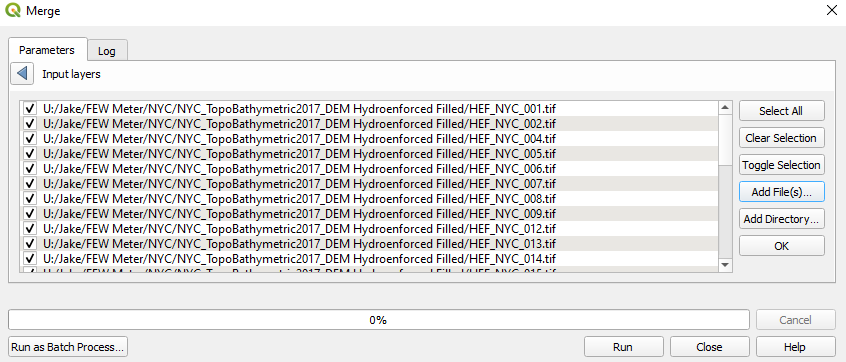
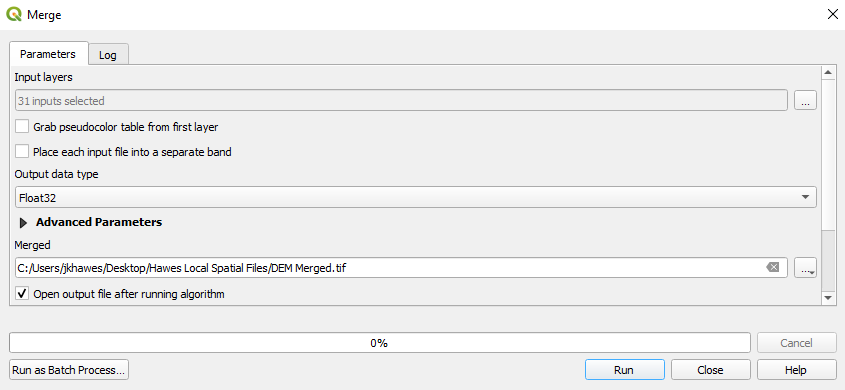
We will use the following data sets:

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.

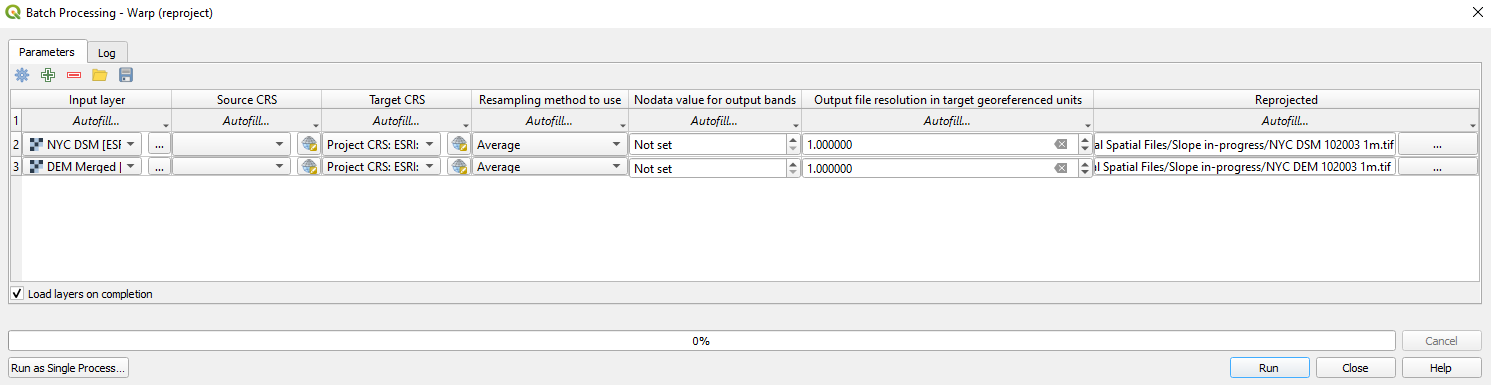
## Create city-wide DEM and DSM layers

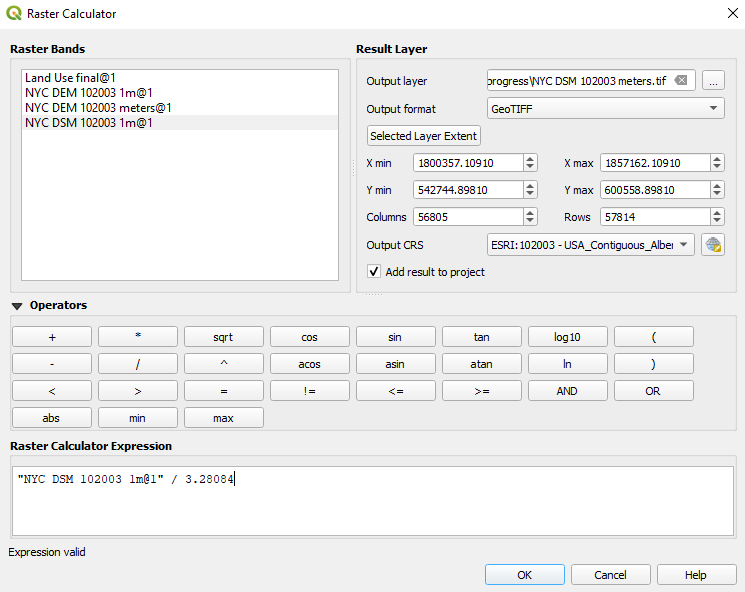
First, as a sort of step zero, we need to stitch together the DEM and DSM layers for NYC. They are delivered in tiles. This is fairly straightforward with the ***merge*** tool. We don’t even need to import the rasters in Q first (and we shouldn’t, because it would take a while) - instead we can select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



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

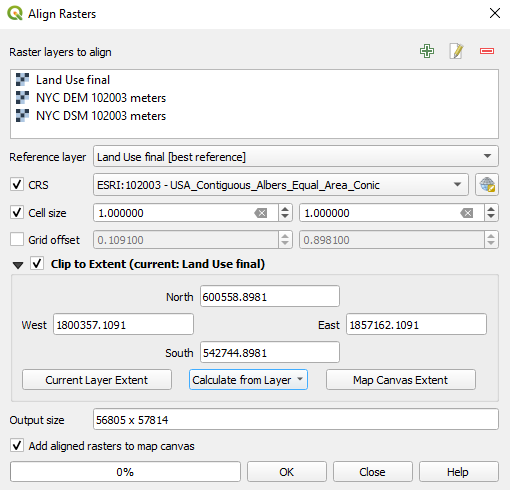
The two layers should be converted to 102003 to work with the rest of the derived data (at which point we can also convert them to 1m\*1m), and we need to make sure that the height values are also in meters. So the simplest solution is to ***Warp*** everything in batch mode, then divide both layers by 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.





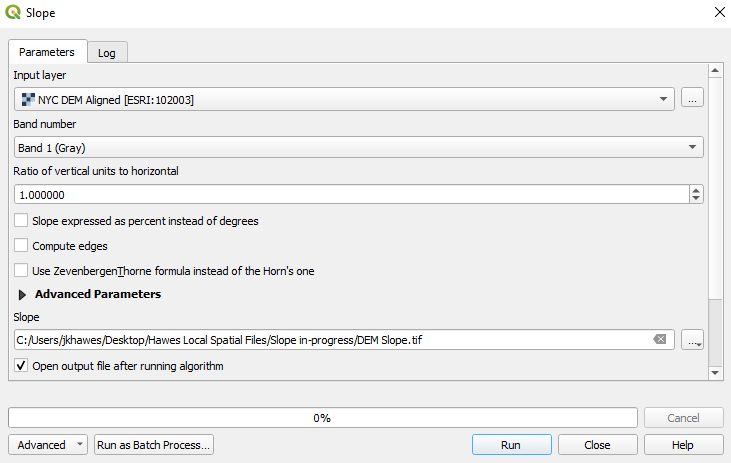
## Align with Land Use final layer

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.

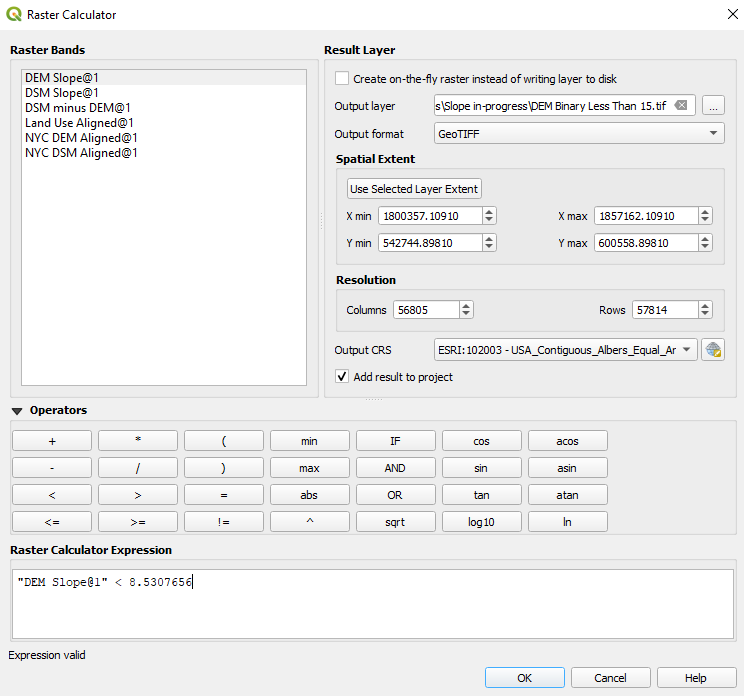


## Calculate Slope from the DEM

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.

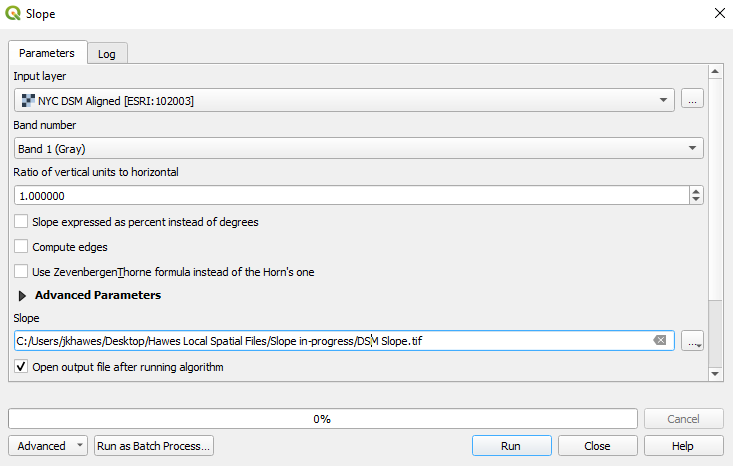


Once we have this Slope file, we can do some simple raster algebra to determine where this is greater than and less than 15%. Expressed as degrees, a 15% slope is arc-tangent of 0.15, which is 8.5307656. So we want to find places where the slope layer is less than that.

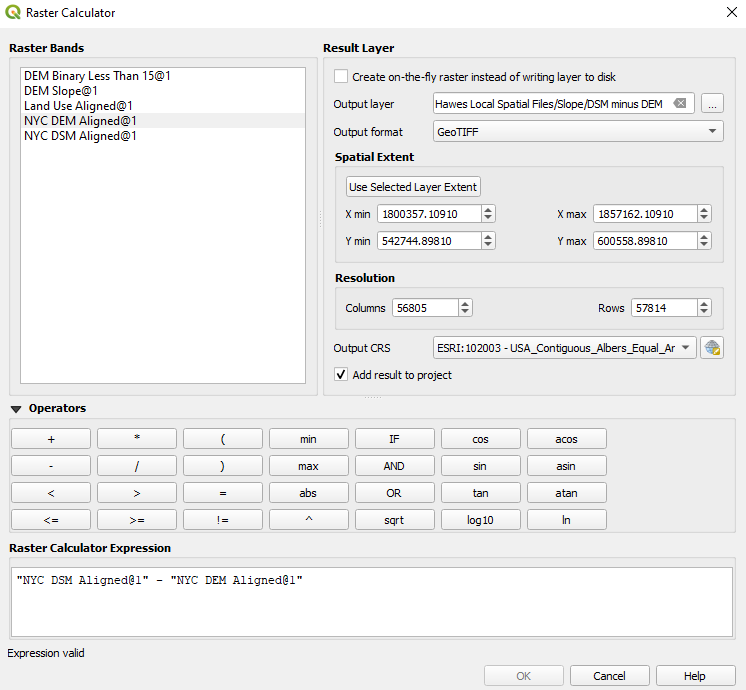


## Identify flat roofs from the DSM

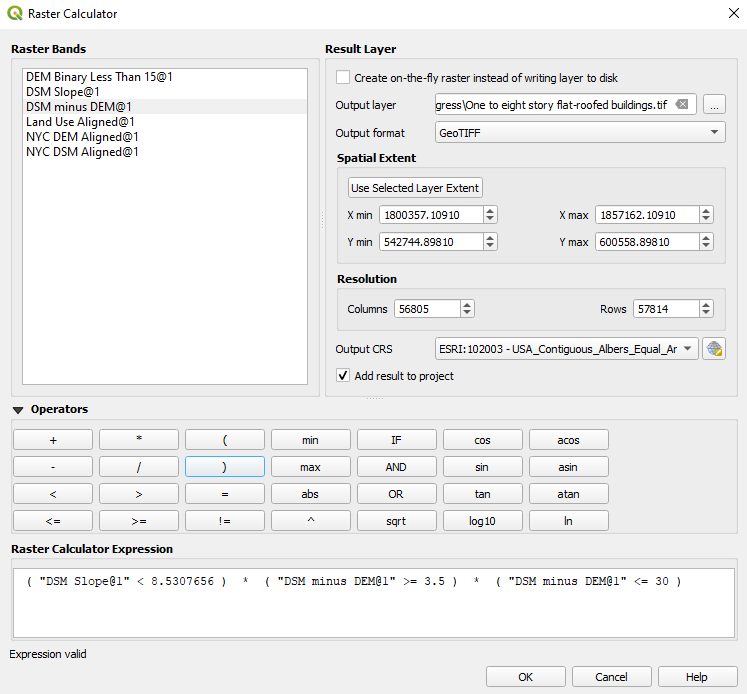
Next, we need to calculate the roof slopes - this is a bit trickier and requires several steps. First, we can run the Slope function under Raster > Analysis on the DSM.



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

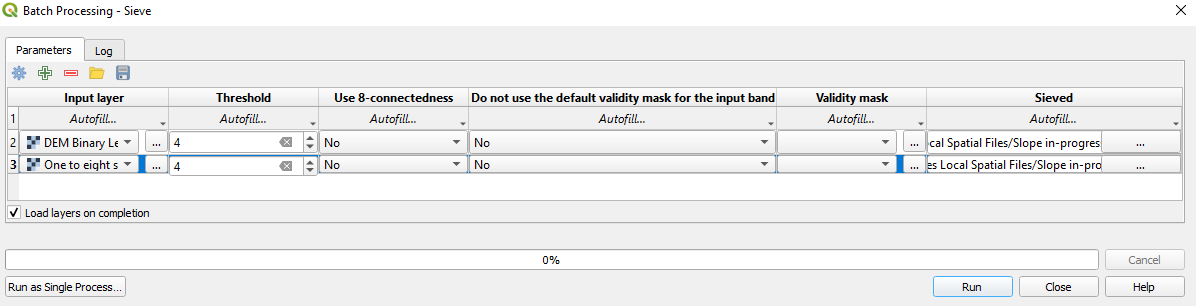


We can then run some simple raster calculations, finding the places where the Slope layer is less than 8.5307656 and the Height of the DSM-DEM layer is more than 3.5m and less than 30m (between 1 and 8 stories) - note that this height is why we need to do the subtraction. If not, we can’t use 30 or something as a roof height, because ground level differs, so some roofs are below ground level in other places in the city.



## Clean up layers with the sieve tool

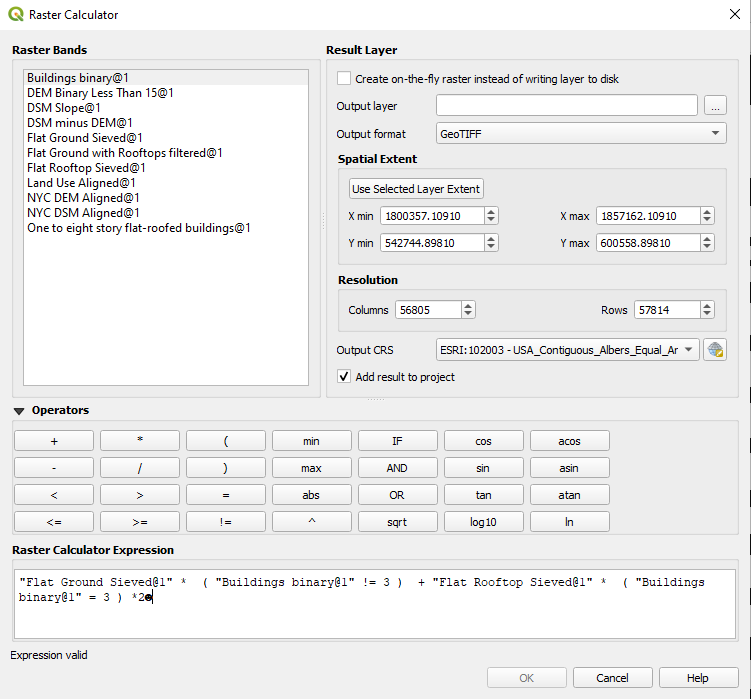
Once we have useful results for both round and rooftops, we can use the sieve tool to clean it up a bit. I used 4 as a threshold because it tends to catch edge cases most effectively. Make sure to fix the symbology to 0-1 after running the function.



## 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% 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

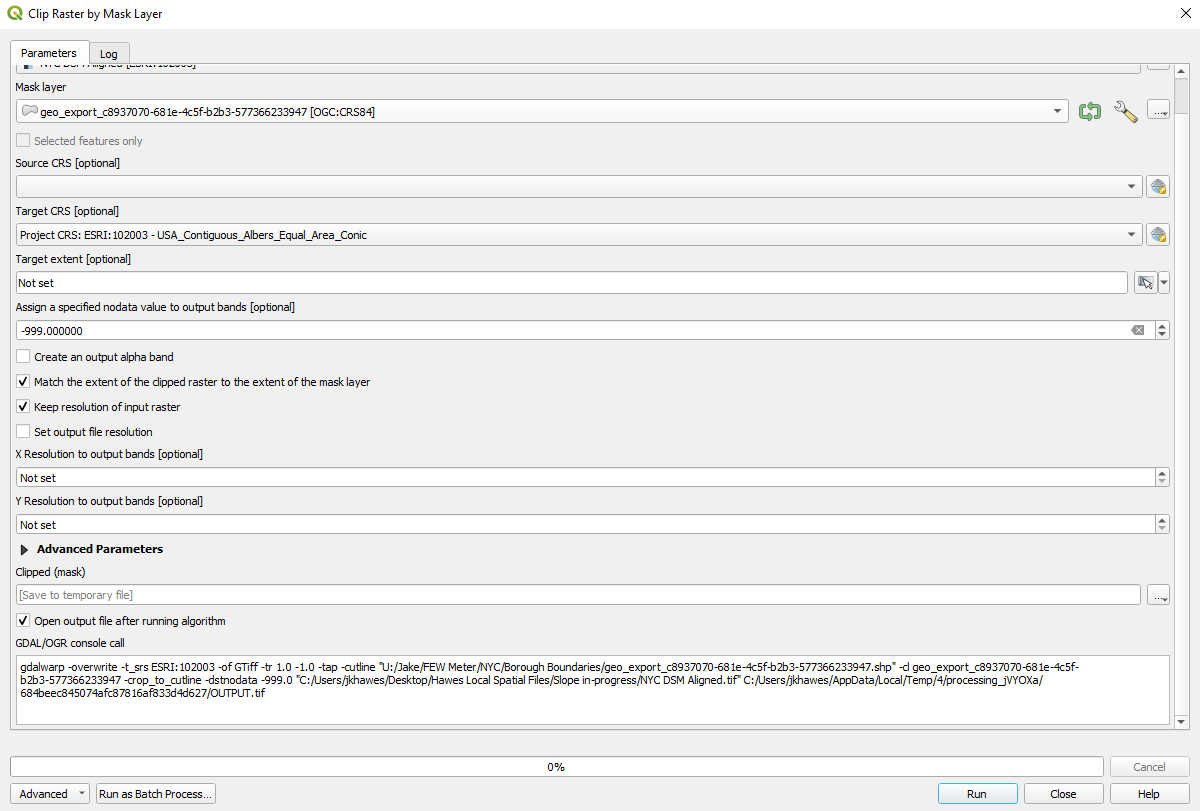
The final layer we will derive is a sunlight availability layer. This will take advantage of the r.sun package in GRASS, which takes the DSM and converts it to solar irradiance. This one is fairly complex, so I derive my process from an [example done in Canada](https://www.sciencedirect.com/science/article/pii/S0038092X10000812?via%3Dihub#fn15). 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. Let’s test it out in NYC.

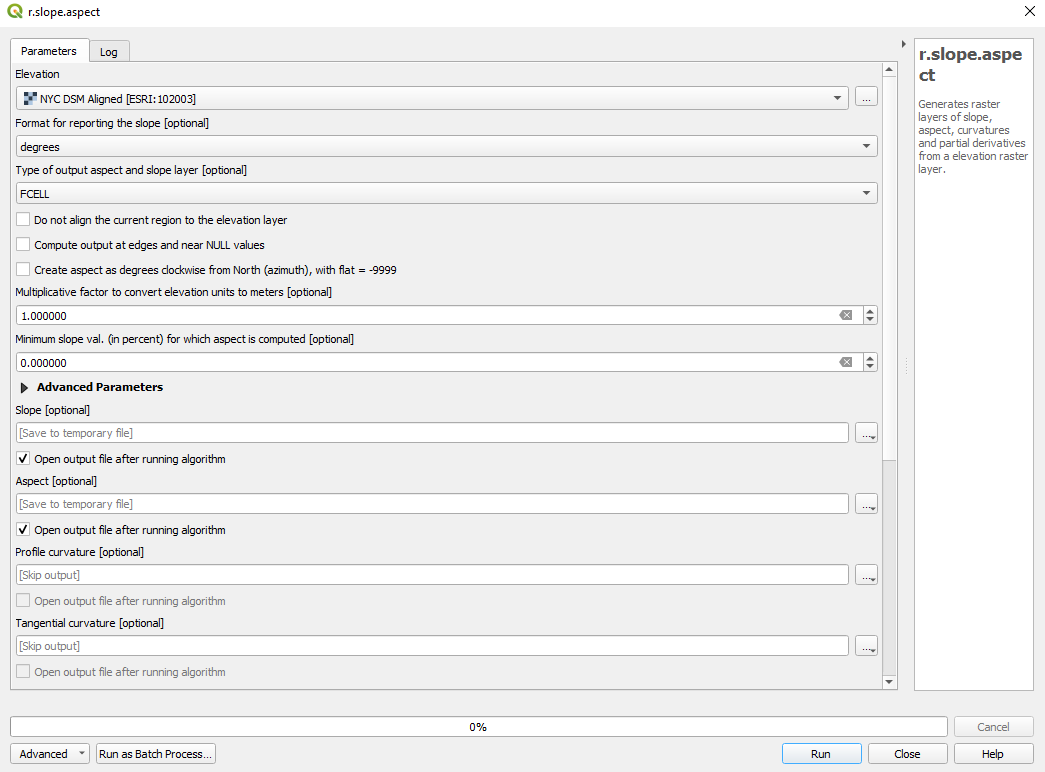
## Clip DSM to work better with GRASS

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. See the settings below.



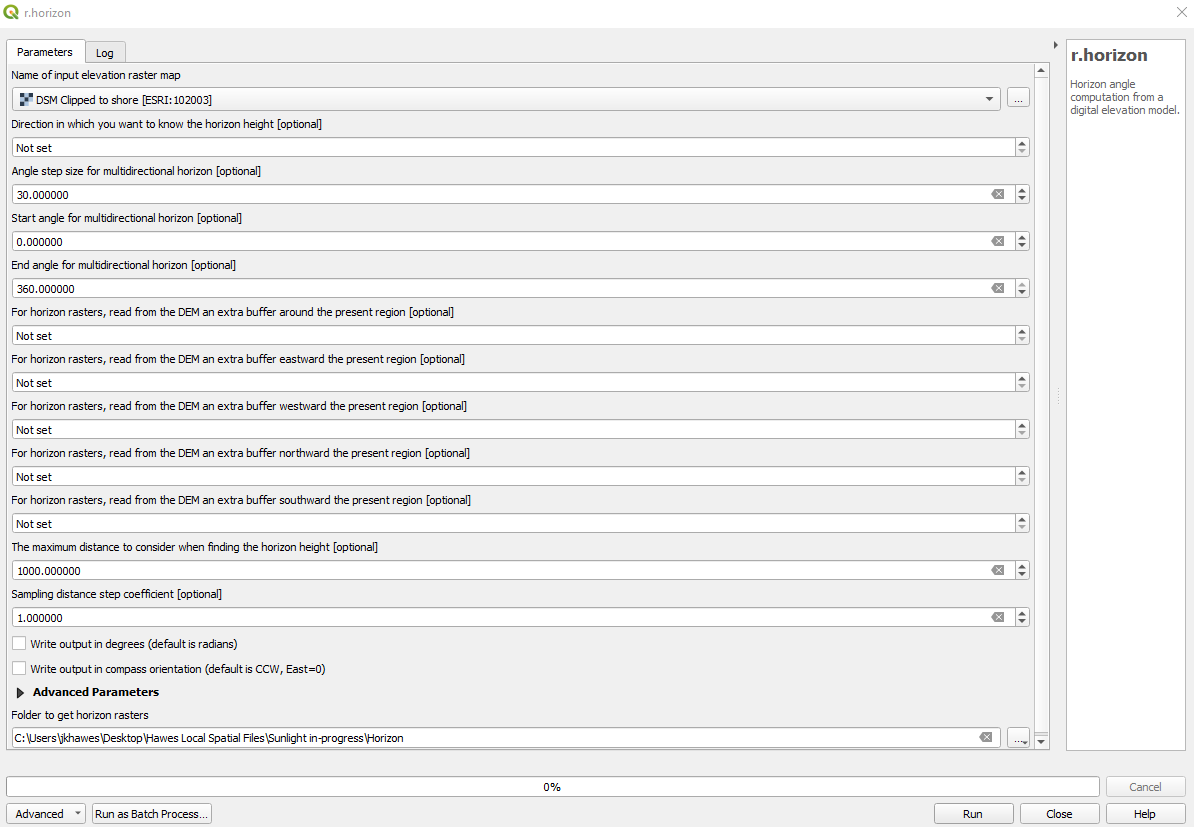
## r.slope.aspect

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.

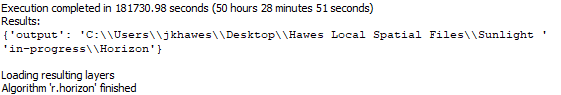


## Optional: r.horizon

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.

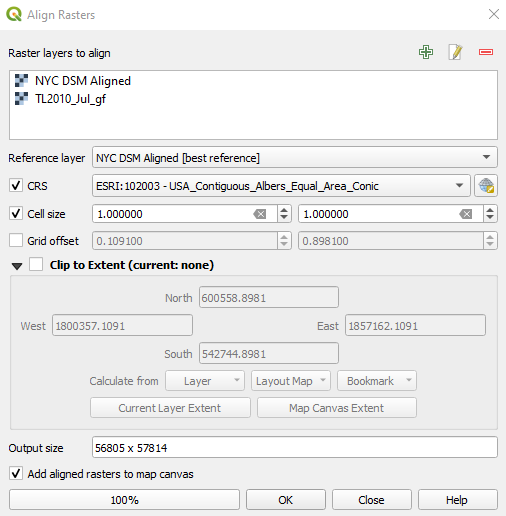


If you do go through GRASS to do this, it will take a very long time to run - on the order of 50 hours on the computer in our lab at 1m resolution. In fact, one of the first times I ran this, it ran for 50 hours and only saved 120-360, so you may have to run it twice to convince it to save everything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



## Aligning Linke

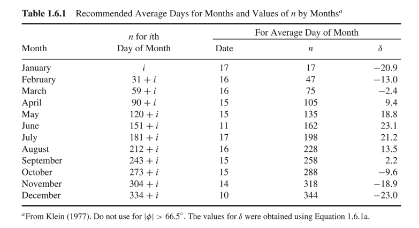
The last step before running the final command is to align the Linke values with the files we’ll be using. This will also clip the raster and will probably take 10-15 minutes.



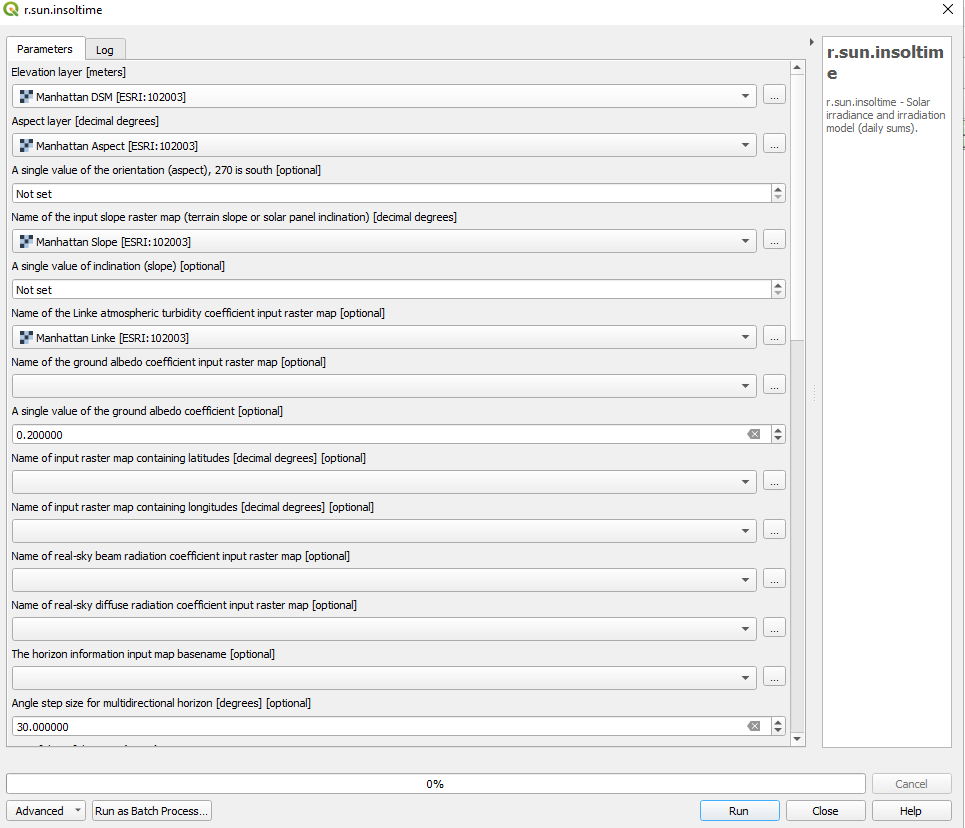
Once the Linke values have been converted to a 1x1m raster, we should be able to clip this to the borough boundaries, as we’ve done with the others.

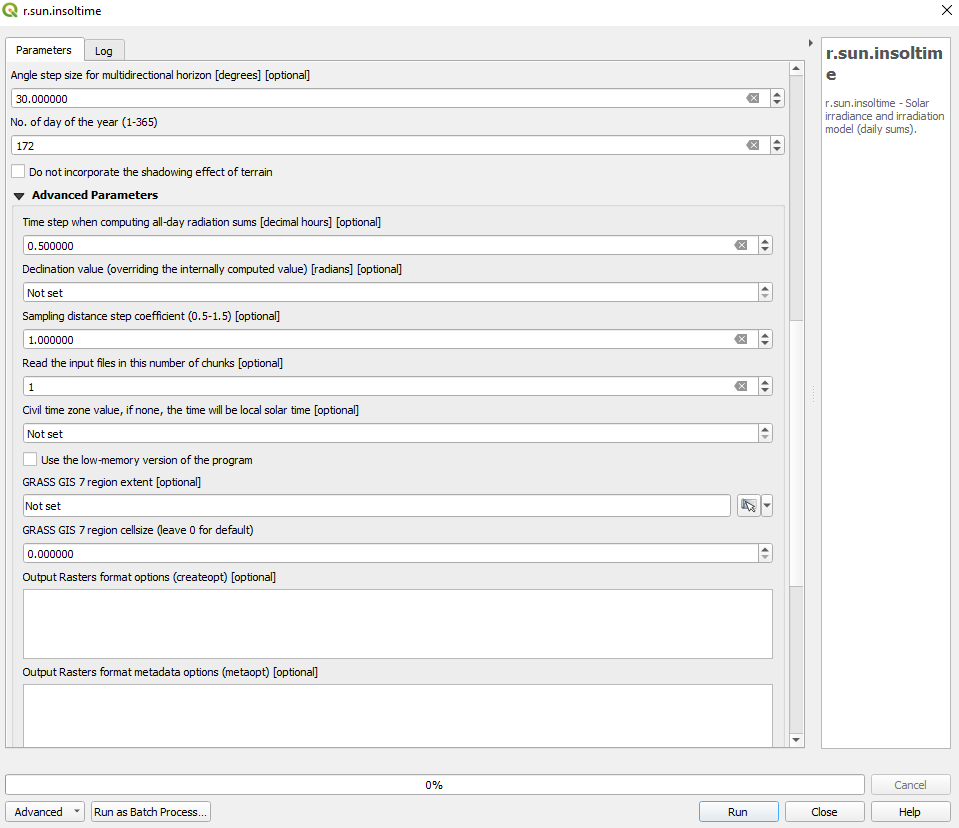
## r.sun.insoltime

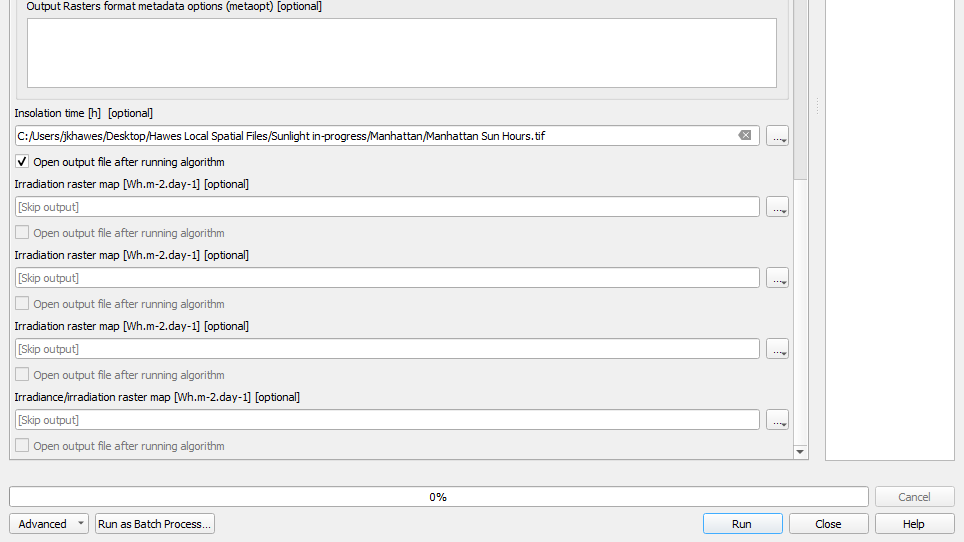
Since we are strictly interested in the number of hours of sunlight, we can simply ignore the more complex aspects like levels of radiation from the NASA data. We can retrieve average days from the book cited in the Canadian paper:



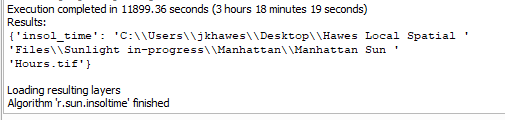
This means that the inputs look like this:



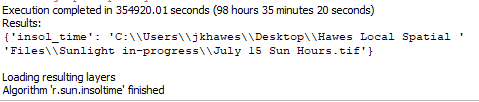




These commands run slowly, but stubbornly. I first ran it on just Manhattan and it took about 4 hours. This may be a useful experiment for you to test your inputs before letting it run for several days on the whole city.



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.



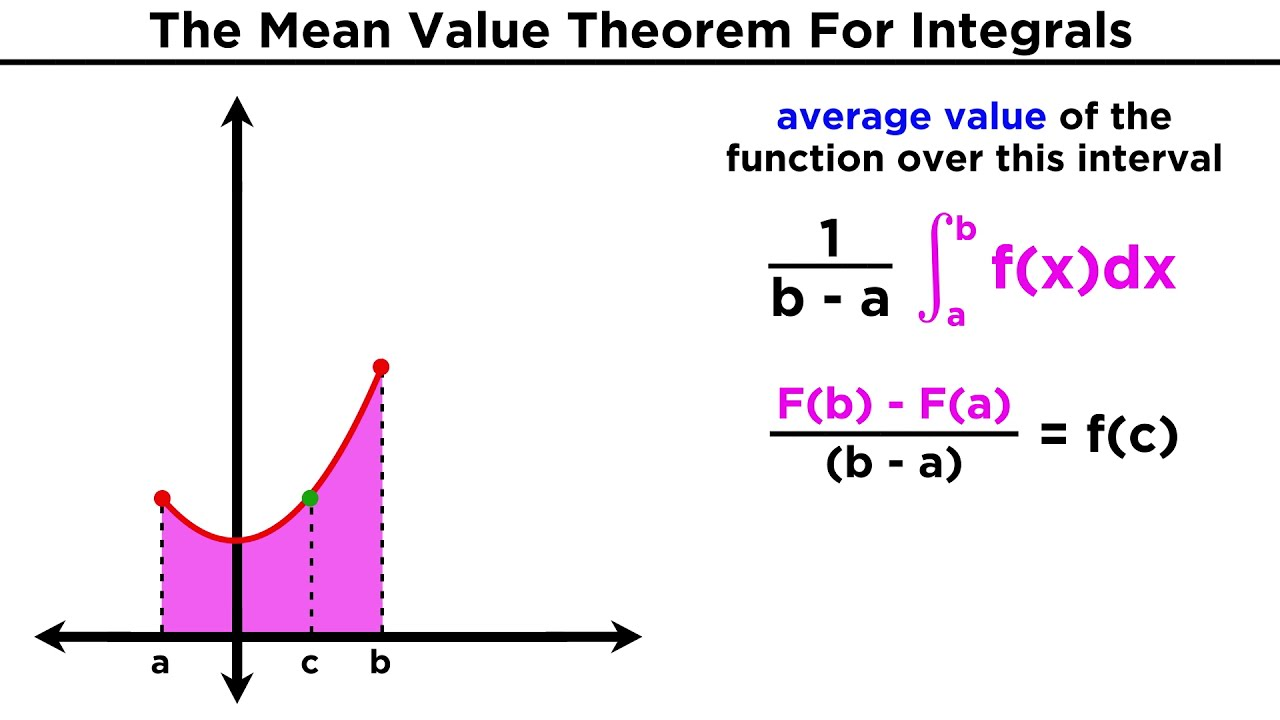


Following this method, I calculate irradiation hours for March, July, and September and take the average across the growing season. From there, I am able to identify places that receive at least four hours of sunlight per day on average during the growing season (at least partial sun).

## Merge results and calculate average

First, we obviously need to recombine these tiles. This can be run as a batch process and should take several hours to run.

Once we have full sunhours layers for each month, we can take the average value of the function over the desired interval (the time between April average and October average). We can either try to fit and upside down parabola or we can assume two piecewise linear functions. The second produces more reproducible math between locations and on each cell, so we live with that simplification. The rest of the calculus in this whole process has been hidden in algorithms in QGIS, but you can actually follow along as we derive this particular equation. The mean value theorem says that we can calculate the average value of a function over any particular interval on which it is continuous, which our piecewise function is. For piecewise functions, we calculate the mean value theorem of the constituent pieces and then take a weighted mean.



We derive our equation with the standard y = mx+b, assuming that slope is linear:

For April to July, this is:

For July to October, this is:

So once we conduct the integration, the y goes away and the x gets filled in, but that still leaves us with b1 and b2. So before we can jump to the mean value theorem, we have to calculate the value of b1 and b2. We can do this by simply plugging in values we have already:

If we plug July into April to July, we have:

Solving, we end up with:

If we plug July into July to October, we have:

Solving, we end up with:

We can run both of these as raster calculations and end up with b1 and b2 as rasters. Now we can go ahead and integrate.

This is obviously a more complicated bit of math. For April to July, we end up with:

Following this same math for July to October, we end up with:

At the end, this turns out to be five raster calculations. We calculate and first with separate raster calculations, then we can directly calculate and .

The final calculation is just one final raster calculation - .

For b1, the raster calculation looks like this: "July Sunhours@1" - ( 198\* ( ( "July Sunhours@1" - "April Sunhours@1" ) / 93 ) )

For b2, the calculation looks like this: "July Sunhours@1" - ( 198 \* ( ( "October Sunhours@1" - "July Sunhours@1" ) / 90 ) )

For MeanValue1, the calculation looks like this: ( 28179 / 17298 ) \* ( "July Sunhours@1" - "April Sunhours@1" ) + "b1@1"

For MeanValue2, the calculation looks like this: ( 43740 / 16200 ) \* ( "October Sunhours@1" - "July Sunhours@1" ) + "b2@1"

For MeanValueOverall, the calculation looks like this: ( ( 93 \* "MV1@1" ) + ( 90 \* "MV2@1" ) ) / 183

# NYC Facilities Layer Derivation

The final facilities layer needs to include a variety of things. First, it needs to aggregate the results at the parcel level, indicating the total area available and whether or not this is enough for consideration as a potential site. Second, it needs to capture what *type* of site is possible at each location - this is a land use question. Finally, it needs to indicate if the site would be rooftop or not. The land use classification for rooftop gardens is relatively similar to the ground sites, but community gardens are more restricted - it is unlikely that a lot of public community gardens will sprout on a lot of rooftops, and in particular we remove any sites where liability is likely to prevent it (e.g., public buildings, hospitals). This means that there should actually be five intermediary rasters - one for community gardens, one for individual gardens on the ground, one for farms on the ground, one for individual gardens on the roof, one for farms on the roof. Each of these is just a binary 1m^2 raster, which can then be summarized with Zonal Statistics to create a column in the PLUTO file, which indicates the total area of that type of garden on a site. So the raster calculations to generate these layers are as follows:

## Raster derivation

### Community gardens on the ground

(LandUse = 12, 21, 41, or 70) \* Partial Sun \* Flat surface \* Grass or concrete land cover (or under tree)

*Raster calculation: ( "LU@1" = 12 OR "LU@1" = 21 OR "LU@1" = 41 OR "LU@1" = 70 ) \* ( "MeanValueGrowingSeason@1" >= 4 ) \* "Slope binary@1" \* ( "LC@1" = 1 OR "LC@1" = 2 OR "LC@1" = 4 )*

### Individual gardens on the ground

(LandUse = 11) \* Partial Sun \* Flat surface \* Grass or concrete land cover

*Raster calculation: ( "LU@1" = 11) \* ( "MeanValueGrowingSeason@1" >= 4 ) \* "Slope binary@1" \* ( "LC@1" = 1 OR "LC@1" = 2 OR "LC@1" = 4 )*

### Urban farms on the ground

(LandUse = 22 or 23) \* Partial Sun \* Flat surface \* Grass or concrete land cover

*Raster calculation: ( "LU@1" = 22 OR "LU@1" = 23) \* ( "MeanValueGrowingSeason@1" >= 4 ) \* "Slope binary@1" \* ( "LC@1" = 1 OR "LC@1" = 2 OR "LC@1" = 4 )*

### Community gardens on rooftops

(LandUse = 12, 41, or 70) \* Partial Sun \* Flat surface \* LC = Buildings

*Raster calculation: ( "LU@1" = 12 OR "LU@1" = 41 OR "LU@1" = 70) \* ( "MeanValueGrowingSeason@1" >= 4 ) \* "Slope binary@1" \* ( "LC@1" = 3)*

### Individual gardens on rooftops

(LandUse = 11) \* Partial Sun \* Flat surface \* LC = Buildings

*Raster calculation: ( "LU@1" = 11) \* ( "MeanValueGrowingSeason@1" >= 4 ) \* "Slope binary@1" \* ( "LC@1" = 3)*

### Urban farms on rooftops

(LandUse = 21, 22, or 23) \* Partial Sun \* Flat surface \* LC = Buildings

*Raster calculation: ( "LU@1" = 21 OR "LU@1" = 22 OR "LU@1" = 23) \* ( "MeanValueGrowingSeason@1" >= 4 ) \* "Slope binary@1" \* ( "LC@1" = 3)*

## Translate rasters to PLUTO

### Set all non-one values to NULL

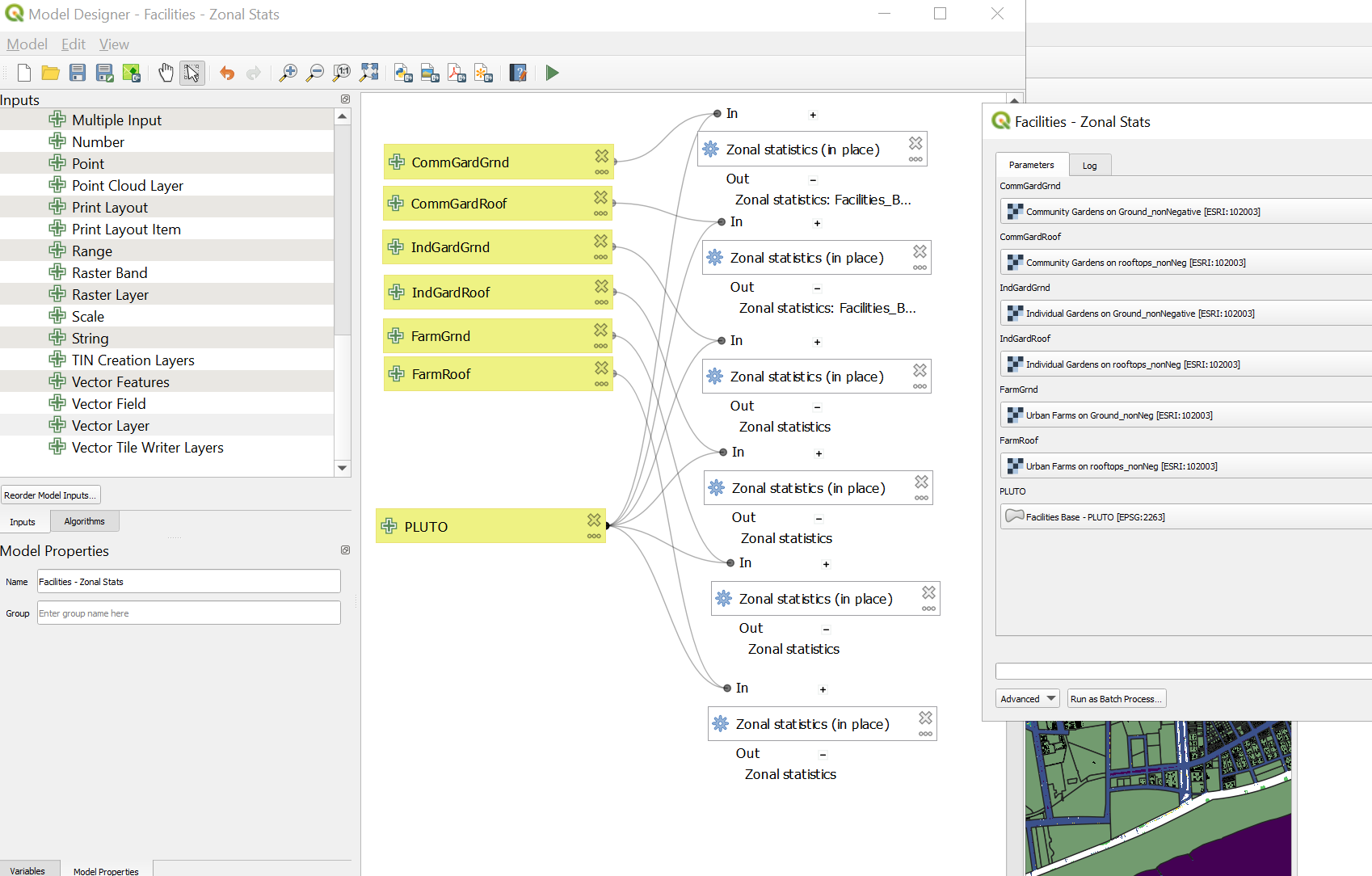
Because raster calculator causes very strange NULL results when things don’t overlap just right, we have to take some time to set any random negative values to NULL. We can go ahead and include zero in this, just to let the ones stand out. The simplest way to do this is [with raster calculator](https://gis.stackexchange.com/questions/81640/setting-all-pixels-with-value-0-to-nodata-in-dem-raster) - the first answer in that question correctly identifies the solution as (("x">0)\*"x") / (("x">0)\*1 + ("x"<=0)\*0). This takes any “x” raster and sets all things less than or equal to zero as NULL, because anything less than or equal to zero results in 0 / 0.

For example: ((“Community Gardens on Ground@1” >0)\* “Community Gardens on Ground@1”) / ((“Community Gardens on Ground@1” >0)\*1 + ( “Community Gardens on Ground@1” <=0)\*0)

An alternative to this would’ve been using the raster calculator to set everything not equalling one to zero before the raster calculations. At the end of the day, it should be the same amount of computation and time either way.

### Zonal statistics

Once we have cleaned rasters with just a simple binary (that actually doesn’t even have zero), we can run zonal statistics to know the total area of garden/farm space available in each PLUTO parcel. You can do this one by one, but I use the model builder for two reasons. First, you can run them all at once, which is obviously great. Second, you can run “Zonal statistics (in place)” which just appends the column to the existing data and for some reason isn’t available in the normal processing toolbox. See below for the configuration I used. I run only “sum” for the zonal statistics, since the other variables get confusing with all the NULL values hanging around.



At this point, you now have all the basic information you need about where facilities are plausible. From here, we move to the Scenario Derivation, since the final facilities layer is going to depend on the weighting schemes associated with each scenario. If you so choose, this could be a good time to eliminate some of the PLUTO categories. You really don’t need most of them except possibly the following:

1. Borough, Block, Lot, Community District - “CD” - and Census GEOID “BCTCB2020”
2. I’m not saving these, but I want to highlight Zoning Districts 1-4 as well as the commercial overlays, special purpose designations, etc. - in future iterations, this might play into more realistic scaling up scenarios.
3. Land Use and OwnerType - Might be useful later for post-hoc analysis
4. LotArea, BldgArea, ComArea, ResArea, OfficeArea, RetailArea, etc. – all might be useful for post-hoc analysis.
5. NumBldgs, NumFloors, UnitsRes, and UnitsTotal - All important variables we might use in deriving facilities info.
6. AssessLand and AssessTot - for post-hoc value/cost analysis
7. Year built and year altered - might be useful for rooftop gardens
8. Lat, Long - Just in case we need to re-geocode this at any point.
9. Shape\_Length and Shape\_Area - just because sometimes GIS systems really want you to have these standard fields for polygons.
10. Obviously the six fields we just added

As long as you’re not running short on storage space, I recommend doing this via the Save As function - this allows you to pick only the variables you care about while saving the old version of the file - unlike the process of deleting columns with the attribute table. It also allows you to reproject PLUTO to 102003.

### Convert wide data to long

At this point, we have three separate types of facilities in one row in QGIS. Technically, I think there is a way to do pivot table work in QGIS with the GroupStats plugin, and moreover you could do this in Python, but both of those would require something I’m less comfortable with, so I’d rather do it quickly in R. I’ve included the code below to convert this new file that we have into something that has all facilities listed separately with the basic data that we need preserved.

library(tidyverse)

library(magrittr)

library(sf)

library(sp)

########### Wide to long ###########

## Early step of converting things from wide to long.

# Load the new facilities data tatthat we just generated into R.

Facilities\_raw <- st\_read("U:/Jake/Urban Gardens/Location allocation/NYC/NYC Location Allocation/Facilities Layers/Facilities - Base - PLUTO format.shp") %>%

mutate(id = row\_number())

# Mutate so that one parcel can only have one type of facility. This is both for the sake of common sense and for the sake of modeling.

## Things get really hairy if we have multiple copies of a parcel once we start dissolving things later.

Facilities\_raw %<>% mutate(CommGrnd\_s = ifelse(((FarmGrnd\_s > 0) | (FarmRoof\_s > 0) | (CommGrnd\_s > 0) | (CommRoof\_s > 0)) & ((IndGrnd\_su > 0) | (IndRoof\_su > 0)),

CommGrnd\_s + IndGrnd\_su, CommGrnd\_s),

IndGrnd\_su = ifelse(((FarmGrnd\_s > 0) | (FarmRoof\_s > 0) | (CommGrnd\_s > 0) | (CommRoof\_s > 0)) & ((IndGrnd\_su > 0) | (IndRoof\_su > 0)),

0, IndGrnd\_su),

CommRoof\_s = ifelse(((FarmGrnd\_s > 0) | (FarmRoof\_s > 0) | (CommGrnd\_s > 0) | (CommRoof\_s > 0)) & ((IndGrnd\_su > 0) | (IndRoof\_su > 0)),

CommRoof\_s + IndRoof\_su, CommRoof\_s),

IndRoof\_su = ifelse(((FarmGrnd\_s > 0) | (FarmRoof\_s > 0) | (CommGrnd\_s > 0) | (CommRoof\_s > 0)) & ((IndGrnd\_su > 0) | (IndRoof\_su > 0)),

0, IndRoof\_su),

FarmGrnd\_s = ifelse(((CommGrnd\_s > 0) | (CommRoof\_s > 0)) & ((FarmGrnd\_s > 0) | (FarmRoof\_s > 0)),

CommGrnd\_s + FarmGrnd\_s, FarmGrnd\_s),

CommGrnd\_s = ifelse(((CommGrnd\_s > 0) | (CommRoof\_s > 0)) & ((FarmGrnd\_s > 0) | (FarmRoof\_s > 0)),

0, CommGrnd\_s),

FarmRoof\_s = ifelse(((CommGrnd\_s > 0) | (CommRoof\_s > 0)) & ((FarmGrnd\_s > 0) | (FarmRoof\_s > 0)),

CommRoof\_s + FarmRoof\_s, FarmRoof\_s),

CommRoof\_s = ifelse(((CommGrnd\_s > 0) | (CommRoof\_s > 0)) & ((FarmGrnd\_s > 0) | (FarmRoof\_s > 0)),

0, CommRoof\_s)

)

# Pivot longer so the three types and two setting appear in one column with labels

Facilities\_long <- Facilities\_raw %>% pivot\_longer(cols = FarmRoof\_s:FarmGrnd\_s, names\_to = "FacType", values\_to = "Area") %>%

filter(Area != 0)

# Create an additional column to capture the setting, then mutate the facility types to better reflect their real names and make sure that RoofArea is stored separately.

## Then summarize the area so that each of the three types only appears in one row per parcel.

Facilities\_summarized <- Facilities\_long %>% mutate(RoofGrnd = ifelse(FacType == "FarmRoof\_s", "Roof",

ifelse(FacType == "IndRoof\_su", "Roof",

ifelse(FacType == "CommRoof\_s", "Roof",

"Ground"))),

FacType = ifelse(FacType == "FarmRoof\_s", "Farm",

ifelse(FacType == "FarmGrnd\_s", "Farm",

ifelse(FacType == "IndRoof\_su", "Individual Garden",

ifelse(FacType == "IndGrnd\_su", "Individual Garden",

"Collective Garden")))),

RoofArea = ifelse(RoofGrnd == "Roof", Area, 0)) %>%

select(-RoofGrnd) %>% group\_by(id, FacType) %>% mutate(Area = sum(Area),RoofArea = sum(RoofArea)) %>% unique()

# Now that we have the facilities sorted into their own polygons, we can filter out any parcels with less than 100 m2 of useful space. Since we plan to merge these parcels, this could technically be smaller, but this is a useful step both for feasibility and for simulation practicality.

Facilities\_summarized %<>% filter(Area >= 100)

st\_write(Facilities\_summarized,"U:/Jake/Urban Gardens/Location allocation/NYC/NYC Location Allocation/Facilities Layers/Only one type per parcel/Facilities - Base - By Garden Type.shp")

## Dissolve and reform facilities by type

Before moving on to create the facilities files for each separate simulation, we need to do one last bit of cleaning. Both for the sake of the simulation running in a reasonable amount of time and in the name of general common sense, we merge neighboring gardens when they have the same garden type. This is a two-part process. First, we use the Dissolve function to pull together the different facility types into one multi-part polygon. Then we re-separate these and pull back in the relevant data from the original facility sites.

The dissolve function is straightforward, just requires you to select FacType as the dissolve field. If we were going to leave everything in the multipart polygons or were able to only group when the polygons were touching, we could do this with Aggregate and keep all the relevant data. But we can’t do that, so we have to dissolve everything, then separate it, then pull the data back in. Dissolve will take less than 5 minutes to run on a fast computer, and we follow this with Multipart to singleparts, which will run very quickly. However, we finish with the one part of this that will take a while - a join attributes by location which will pull in most of the original data from our garden sites to our new garden polygons. Note - there is a quicker alternative to this if you want a quicker simulation and don’t need most of the data stored in the original file.

Long method: First, we clean out the new facilities attribute table, deleting everything. Although it’s quick to highlight all the columns with ctrl-a in the delete function, it will still take a while to actually delete everything. A couple minutes later, once this finishes running, you can use Join Attributes by Location (summary) to create the new data for this file. You do have to do this in two steps - the first captures the text variables, the second captures the quantitative stuff. Do not use “Unique” to capture the qualitative data - it just counts the number of unique observations instead of pulling in only the unique words.

Short method: Accept all the dissolved data and only retrieve the area of gardens. To do this, run the zonal statistics function we developed earlier, which automatically runs the process once for each binary facility map. Then, use the field calculator to create a master column for total area and a second column for roof area. This should be all you need to run the basic simulations, since we still know the master facility type from the dissolved features. Once you’ve run the zonal statistics, the field calculation is fairly straightforward - add everything together for total facility area ("IndRoof\_su" + "IndGrnd\_su" + "CommGrnd\_s" + "CommRoof\_s" + "FarmRoof\_s" + "FarmGrnd\_s"), then add only the three roof columns together for roof area ( "IndRoof\_su" + "CommRoof\_s" + "FarmRoof\_s" ). You can just update the old columns or delete and replace them. It’s worth noting that you can always retrieve the old metadata later once the facilities are selected - so this method is shorter now and doesn’t preclude post-hoc analysis. Just means you can’t incorporate anything else in your location allocation.

### Add existing facilities to file

The last step in the generic facilities file creation is to add the existing gardens to the layer. Basically, you only need to worry about having matching attribute columns for things you need to retain for later. Check the attributes for both layers so the names you care about match (eliminate anything you don’t want in the process), then merge the files. They should merge the matching attributes and create blanks for the rest. If QGIS automatically imported your IDs as integers, it may be useful to switch this to text so you can give the existing gardens a unique id that has an E in it and the new gardens an N. You should also create an “existing” binary column in both files (obviously 0 for the new facilities, 1 for the old). This will be useful for setting the required facilities later. Once you’re ready, just use the Merge Vector Layers command to put everything into one file. Here are some useful commands to make all this work - all in field calculator:

For both:

* id\_num = $id

For existing:

* Area = "GardenArea"
* RoofArea = 0
* Existing = 1
* id = concat('e', to\_string("id\_num"))
* FacType = ‘Existing’

For new:

* Existing = 0
* id = concat('n', to\_string("id\_num"))