**Agisoft Metashape Python Scripts Usage Notes: Joseph Campbell 2/12/2021**

The following listed python scripts were created as a learning tool to develop a skill set in Python as well as a deeper understanding of the photogrammetry software Agisoft Metashape. This document is a basic summary of the usage and actions performed by each script.

**Note\*** each script is written in a way that it can be run as a stand-alone action. So there is some repetition in the code. But I have a separate script -not included- where I have just copied and pasted the necessary parts of each script to create a full workflow from start to finish. Or you can build a workflow via the ‘Batch’ command in Agisoft.

|  |  |
| --- | --- |
| **Script Name** | **Summary** |
| **Metashape\_PythonScripts\_01-CreateProject-SaveAs.py** | * Prompts user to save the project and provide a file name. |
| **Metashape\_PythonScripts\_02-LoadPhotos-AnalyzeImages-CameraGroups.py** | * Prompts the user to select and load the images used for Alignment. * Runs the ‘Analyze Images’ process which populates the ‘Quality’ field found under the ‘Photos’ Pane when in Detail view. * Creates a new Camera Group for each Lens ( ex. 35mm , 50mm…) found in the EXIF data of the loaded images. * Then moves the cameras into their respective Camera Group based on the EXIF data. |
| **Metashape\_PythonScripts\_03-MatchPhotos-AlignCameras.py** | * Runs ‘Match Photos’ process on High Quality. * Runs ‘Align Cameras’ process (uses quality level from Match Photos) |
| **Metashape\_PythonScripts\_04-Part\_01-OptimizeCameras-FilterPointCloud.py** | * Duplicates the active Chunk to preserve the original alignment. * Sets the duplicated Chunk as the active Chunk * Initiates the Point Cloud Filter for the active Chunk * Loops through each of the following Filter Options and   1. Selects…then Removes the selected points returned by the given value:      1. Reconstruction Uncertainty, value 10      2. Projection Accuracy, value 4.0      3. Projection Accuracy, value 3.0   2. Optimize the Cameras between each filter option * Prepares the Reference Pane Settings for CHI scalebar workflow:   1. Sets ‘Scalebar Accuracy’ to 0.0001   2. Sets ‘Tie point Accuracy’ to 0.1   3. Sets ‘Marker Projection Accuracy’ to 0.1 |
| **Metashape\_PythonScripts\_04-Part\_02\_DetectMarkers-OptimizeMarkerError.py** | * Runs ‘Detect Marker’ Process for 12bit Circular Targets * Runs ‘Detect Marker’ Process for Cross (Non Coded) Target * Removes any Marker with *less than* 9 camera projections. * Loop through each Camera …for each remaining Marker, and then remove any Marker that has a Projection Error *greater than* 0.5. * Redo - Remove any Marker with *less than* 9 projections   **Notes\*** The user will then manually create the scalebars and input the expected Distance value for each scalebar. If this script is run successfully then each scalebar’s Error(m) value will immediately be within tolerance and no further cleanup will be needed. |
| **Metashape\_PythonScripts\_04-Part\_03-Optimizecameras-FilterPointCloud-TenPercent.py** | * Initiates the Point Cloud Filter for the active Chunk using the Reprojection Error filter * Returns the ‘Max Reprojection Error’ value for the point cloud. * Copies and Sorts the returned filter values for the point cloud. * Calculates the Reprojection Error slider value (threshold) that is equal to 10% of the current point count in the point cloud. * While the Max Reprojection Error for is *greater than* 0.301, then   1. select 10% of the sorted points… and remove them.   2. Optimize the Cameras for each iteration. * Using a helper script, Disables any Camera with *less than* 200 projections for the active Chunk. * Resets the bounding box region.   **Notes\*** This script iterates the Reprojection Error filter until the Max Reprojection Error found in the ‘Show Info’ pop up for the active chunk is *less than* 0.301. The danger here is that there are too many iterations which will introduce a ripple like effect into the point cloud. |
| **Metashape\_PythonScripts\_05-CreateMasksFromSparseCloud.py** | * Builds a low-resolution model using the Sparse Cloud as source * Runs the ‘Import Masks’ process with low-resolution model as source * Creates the following subfolders, relative to the saved Project PATH:   1. *\_Output*       1. *\_Active Chunk Name*         1. *\_Masks* * Exports the generated masks to the \_Masks subfolder * Removes the low-resolution model   **Notes\*** If the subfolders already exist, then nothing is changed. |
| **Metashape\_PythonScripts\_06-BuildDepthMaps-BuildDenseCloud.py** | * Run the ‘Build Depth Maps’ process on Ultra High settings. Unlike the match photos algorithm, the Ultra High setting here uses the full resolution image data and does not up-sample the data. * Run the ‘Build Dense Cloud’ process. Which uses the same quality settings as the Depth Map process.   **Notes\*** for the ‘Build Dense Cloud’ process, the Max Neighbors variable is at default set to 100. I’ve updated it to 300. A value of ‘-1’ will load all cameras. Essentially, a higher value will equal a higher quality dense cloud, but exponentially lengthens the processing time. |
| **Metashape\_PythonScripts\_07-BuildModel-DecimateModel.py** | * Creates list of targeted Model face counts: * Runs the ‘Build Model’ process with the quality set to High and Dense Cloud as source. * Returns the face count of the High resolution model * Enumerates (loop) through the list of target face counts and creates a decimated model for each face count. This process incorporates an index value that sets the source model for the decimation to the previously created model. * Loops through each created model, and renames the mesh to match a scheme that is relevant to the models given face count.   **Notes\*** When creating the decimated models, the enumerated loop will skip a target face count if the source model for the decimation has a lower face count than the target face count. The renaming loop also takes this into count and can determine which target face counts were skipped. |
| **Metashape\_PythonScripts\_08-BuildTextures.py** | * Enumerates all models and runs ‘Build Texture’ process with Diffuse Map texture type. The ‘Texture Size’ is 8192 pixels for the higher resolution models, and the switches to 4096 pixels for medium and low-resolution models. * For each model excluding the highest resolution model, run the ‘Build Texture’ process with the Normal Map and the Occlusion Map as texture types. This uses the next higher resolution model as the source for these texture types. |
| **Metashape\_PythonScripts\_09-ExportModels-withTextures.py** | * Creates the following subfolders, relative to the saved Project PATH:   1. *\_Output*       1. *\_Active Chunk Name*         1. *\_Models* * Loops through the existing models and exports both the model and the available texture maps for each. * For Higher resolution models the texture maps are exported as TIFF images. Lower resolution texture maps are exported as JPEG format.   **Notes\*** If the subfolders already exist, then nothing is changed. This is repeated in case the script is run in an isolated action. |
| **Metashape\_PythonScripts\_10-ExportDenseCloud.py** | * Creates the following subfolders, relative to the saved Project PATH:   1. *\_Output*       1. *\_Active Chunk Name*         1. *\_Models* * Exports the Dense Cloud for the active chunk.   **Notes\*** If the subfolders already exist, then nothing is changed. This is repeated in case the script is run in an isolated action. |