Instructions for scripts

1. Environment

Scripts were developed in a Windows 10 environment, but should be usable on other operating systems as well, possibly with minor changes required. The exception to this is the iPSR algorithm, which is compiled from source code for Windows only. To get it to run on other platforms, it needs to be recompiled, code available at: https://github.com/houfei0801/ipsr.

To use the scripts, LOD2 data for the target buildings is required. This was created by splitting the up-to-date LOD2 model of the entire Sendai test area into separate .obj files, each of which contains the information of a single building. This collection of building models needs to be available alongside the LiDAR dataset.

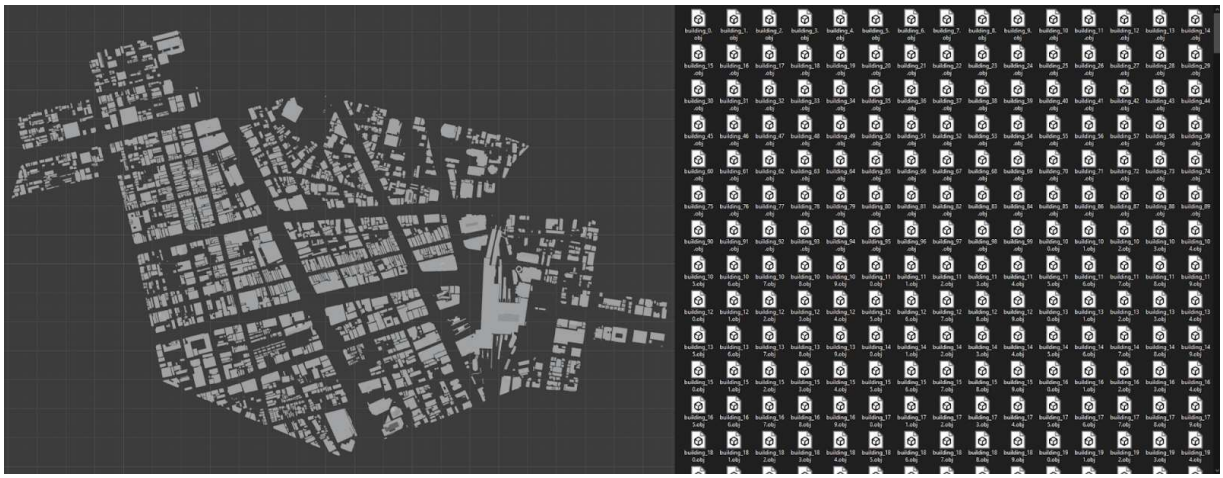

Image 1. LOD2 model of the city center, split into 2406 files.

2. Create building footprints

Horizontal 2D footprint polygons are computed for every LOD2 model in the input path, and the resulting polygon nodes are stored in the output text file, one building definition per line.

Script name	“01_create_footprint_polygons.py”	
Usage	python 01_create_footprint_polygons.py	
Dependencies	pyproj	
Configuration		
Variable name	Value	Explanation
input_path	“C:/Data/LOD2/Buildings/”	Source path that contains target building LOD2 models in Wavefront .OBJ format.
output_filename	“C:/Data/LOD2/footprint_polygons.txt”	The text file into which footprint polygon definitions are written to.
dilation_amount	0.0	How much is the polygon expanded (in meters). Script does not consider nearby buildings, so this should be kept at 0.0.
visualize_results	True	If set to ‘True’, results are rendered into an image.

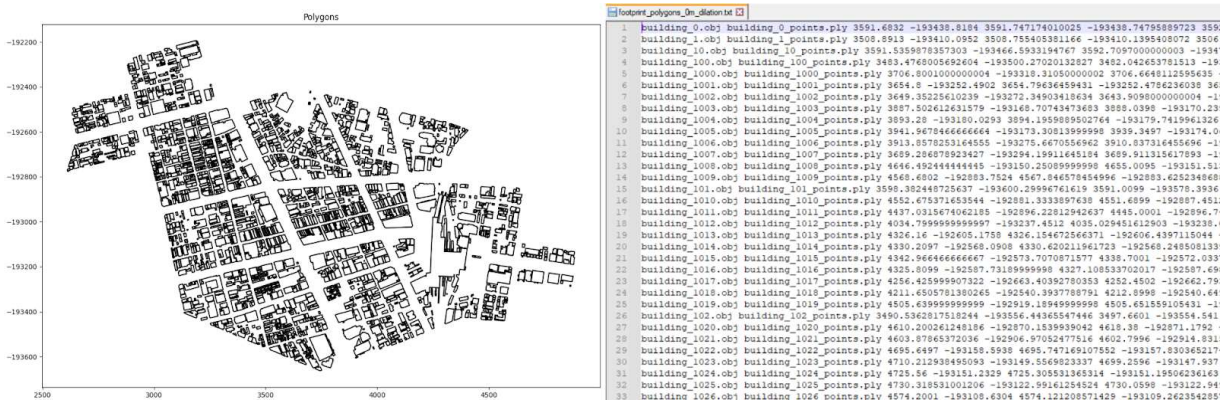

Image 2. 01_create_footprint_polygons.py output.

3. Expand footprint polygons

Building footprint polygons are expanded to cover a larger area in the buildings' surroundings, while taking care not to intersect nearby buildings' (non-dilated) footprint polygons.

Script name	"02_polygon_expansion.py"	
Usage	python 02_polygon_expansion.py	
Dependencies	Matplotlib Numpy Shapely	
Configuration		
Variable name	Value	Explanation
process_all_buildings	True	If set to 'True', all buildings in 'input_filename' are processed.
target_building	"building_349.obj"	If 'process_all_buildings' is set to 'False', a single building is defined.
input_filename	"C:/Data/LOD2/footprint_polygons.txt"	The original footprint definition file.
output_filename	"C:/Data/LOD2/dilated_polygons.txt"	The file to write expanded footprint polygon definitions to.
dilation_amount	1.5	Polygon expansion in meters.
neighbor_distance_threshold	1.5	Same as 'dilation_amount'.
visualize_results	True	Render results into an image?
output_results_to_file	True	Write results to 'output_filename'?

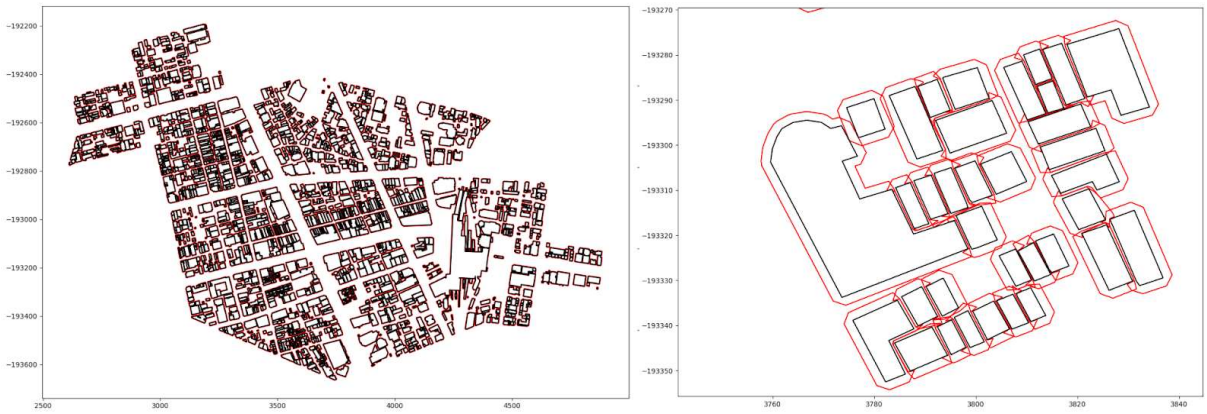

Image 3. 02_polygon_expansion.py output.

4. Defining a target area

If multiple buildings are processed simultaneously, scripts to define the target area are included (LiDAR and LOD2 data for all areas/buildings is required). If only a single building is being processed, skip the two scripts in this chapter and continue to chapter 5.

Use any available online coordinate picker tool (for example: https://www.latlong.net/) to define a polygon on the map, the buildings in which should be processed. Input the latitude and longitude coordinates to the script *“epsg_converter.py”* to output matching positions in the coordinate system used by the LiDAR dataset.

Script name	epsg_converter.py	
Usage	python epsg_converter.py	
Dependencies	Trimesh Matplotlib	
Configuration		
Variable name	Value	Explanation
input_type	“latlon”	Input format.
output_type	“epsg”	Output format.
input_epsg_code	4326	Source coordinate system.
output_epsg_code	6678	Target coordinate system.
input_lat/lon	XX.XXXXXX	In case ‘input_type’ is ‘latlon’.
input_x/y	XXXX.XX	In case ‘input_type’ is ‘epsg’.

Image 4. Use a coordinate picking tool to get latitude/longitude points of the target area.

The converted positions are then used in another script, “03_get_building_list_in_polygon.py”, which will compare the LOD2 footprint polygons to the defined target polygon, and extract all buildings contained in it. The building list is stored in a separate text file.

Script name	03_get_building_list_in_polygon.py	
Usage	python 03_get_building_list_in_polygon.py	
Dependencies	Matplotlib	
Configuration		
Variable name	Value	Explanation
footprints_file	“C:/Data/LOD2/footprint_polygons.txt”	Undilated LOD2 footprint polygons.
output_file	“C:/Data/buildings.txt”	Text file that contains a list of all LOD2 .OBJ files contained in the target polygon.
polygon_points	[[x,y],[x,y],[x,y],[x,y],[x,y]]	A list of 2D points that define the target polygon in the same coordinate system as the LiDAR dataset.

5. Extract buildings and align data

A script called “create_dataset_bounding_box_list.py” can be used to create a bounds file, which contains bounding box definitions for each large point cloud in the LiDAR dataset being used. This will speed up the process of searching for buildings in large datasets. For this project the bounds file has already been generated, so this script can be ignored.

The script “04_split_dataset_into_buildings_with_realignment.py” goes through all the point clouds in the dataset, extracting points that fall within the building’s expanded footprint polygon. The point clouds are aligned against each other, and combined into a single LAS file. Each point in the LAS file has a UserData field, which is used to store the IDs of the points’ originating point clouds in case further adjustments are needed.

In the script, in case of processing a single building, set the name of the target building object file in the list ‘input_building_list’ (e.g. *["building_1316.obj"]*). Multiple object names can be inserted, separated by a comma (*e.g. [“building_1316.obj”, “building_1317.obj”]*). A list of these building object names can be produced by the scripts defined in the previous section.

Script name	04_split_dataset_into_buildings_with_realignment.py	
Usage	python 04_split_dataset_into_buildings_with_realignment.py	
Dependencies	Open3d Laspy Numpy Matplotlib	
Configuration		
Variable name	Value	Explanation
building_polygons_file	“C:/Data/LOD2/dilated_polygons.txt”	Expanded LOD2 footprint polygons.
bounds_file	“C:/Data/las_bounds.txt”	Text file that contains bounding boxes for point clouds in the LiDAR dataset.
input_dataset_path	“C:/Data/Test_segmented_data/”	Path that contains the segmented and filtered LiDAR dataset.
aligned_output_path	“C:/Data/Aligned_point_clouds/”	Path to output the combined and aligned point clouds.
unaligned_output_path	“C:/Data/Aligned_point_clouds/”	Path to output the combined point clouds without alignment

input_building_list	[“building_349.obj”]	Comma-separated list of buildings to be processed, listed according to their LOD2 model names.
visualize_alignment_steps	False	If ‘True’, all steps of the alignment process are rendered into images.
radius_normal	0.4	Normal vector calculation parameter.
matching_resolution	0.02	Alignment resolution (in meters).
force_las_output	True	If set to ‘False’, the output will be in PLY format.

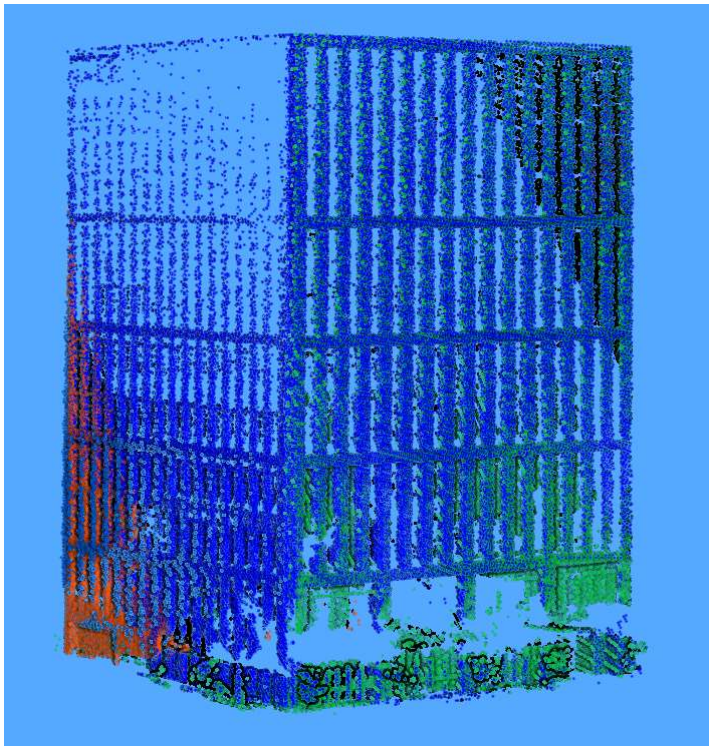

Image 5. Output of “04_split_dataset_into_buildings_with_realignment.py”, with all partial point clouds color coded based on their ID.

6. Combine LiDAR data with LOD2 data

LiDAR data never covers 100% of a building, and these holes can create problems with mesh generation. They are therefore patched using data sampled from the LOD2 models. The script “05_combined_LOD2_with_point_cloud.py” attempts to do this by measuring distances to nearest LiDAR data points from each LOD2 data point, and inserting LOD2 points to where LiDAR data cannot be found.

It needs to be noted that this script does not work well, when the LOD2 model dimensions are not accurate. It is, for example, quite common that a wall in LOD2 data is located even several meters away from the LiDAR data positions, in which case the script output can be very bad.

Script name	05_combine_LOD2_and_point_cloud.py	
Usage	python 05_combine_LOD2_and_point_cloud.py	
Dependencies	Open3d Matplotlib Numpy Trimesh Laspy	
Configuration		
Variable name	Value	Explanation
process_entire_directory	True	If ‘False’, the target building needs to be defined.
building_number	349	If processing only a single building, the building number is defined here.
The highlighted parameters are for development, or for features not fully developed, so they should be left disabled.		
use_preprocessed_data	False	If ‘True’, skip LiDAR/LOD2 data calculations and continue directly for mesh generation and window detection.
create_meshes	False	Standard Poisson Surface Reconstruction method. Included only for window detection purposes.
filter_low_density_from_mesh	False	If ‘True’, cleans up the Poisson mesh by removing areas with low point density before output.

skip_lod2_on_mesh_creation	False	For testing how ignoring LOD2 data will affect mesh generation negatively.
perform_window_detection	False	To perform window detection, ‘create_meshes’ needs to be set to ‘True’.
input_point_cloud_path	“C:/Data/Aligned_point_clouds/”	Path to segmented and aligned LiDAR point clouds.
input_lod2_obj_path	“C:/Data/LOD2/Buildings/”	Path to LOD2 models.
output_point_cloud_path	“C:/Data/Combined_point_clouds/”	Path to output the combined point clouds to.
output_mesh_path	“C:/Data/Meshes/”	Path to output generated meshes to.

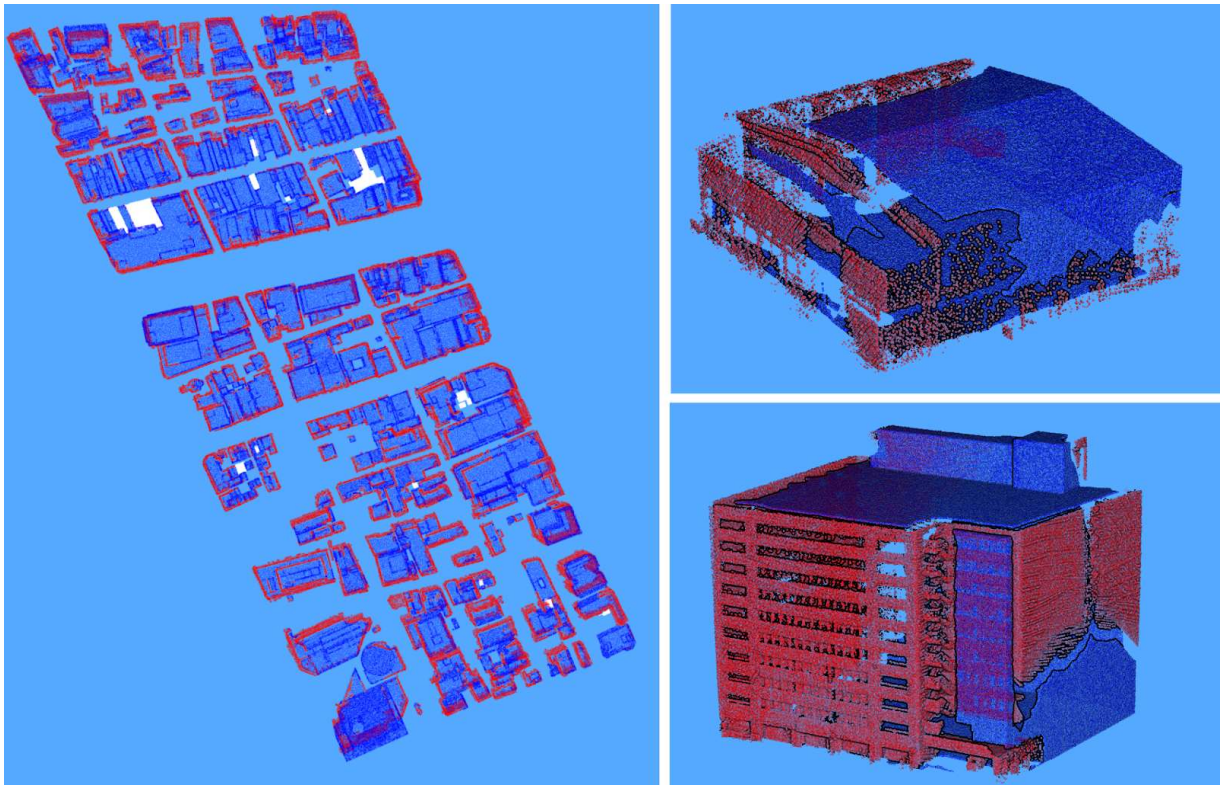

Image 6. Examples of combined LiDAR and LOD2 data. Red points originate from LiDAR data, blue points from LOD2 data.

7. Performing window detection

Window detection is performed using the same script from the previous chapter, “05_combined_LOD2_with_point_cloud.py”. Both ‘perform_window_detection’ and ‘create_meshes’ flags need to be set to ‘True’ to enable this functionality. Additional parameters that also need to be set can be found in the Python script under section “Window detection parameters”. Because the feature is still under development, they need to be adjusted heavily from building to building. The default configuration is for the included sample building “MMS scan of Hotel Central Sendai” (=‘building_349’).

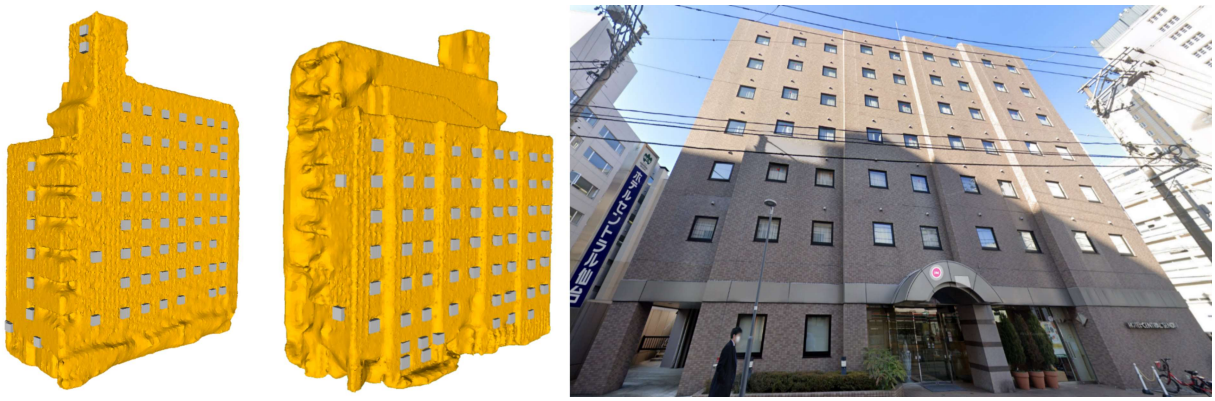

Image 7. Results of window detection algorithm on building_349.

8. Create a mesh using iPSR algorithm

Mesh generation is performed using an external program implementing an algorithm called iPSR, or iterative Poisson Surface Reconstruction. This algorithm is superior to the standard Poisson Surface Reconstruction algorithm due to not requiring any prior knowledge of surface normals, which is generally the case with point clouds.

Script name	06_run_iPSR.py	
Usage	python 06_run_iPSR.py	
Dependencies	Laspy Open3d Numpy	
Configuration		
Variable name	Value	Explanation
process_entire_folder	False	If 'True', process all point clouds.
building_number	349	If 'process_entire_folder' is 'False', the building number needs to be defined.
input_path	"C:/Data/Combined_point_clouds/"	Path to input point clouds.
output_path	"C:/Data/Meshes/"	Path to write meshes to.

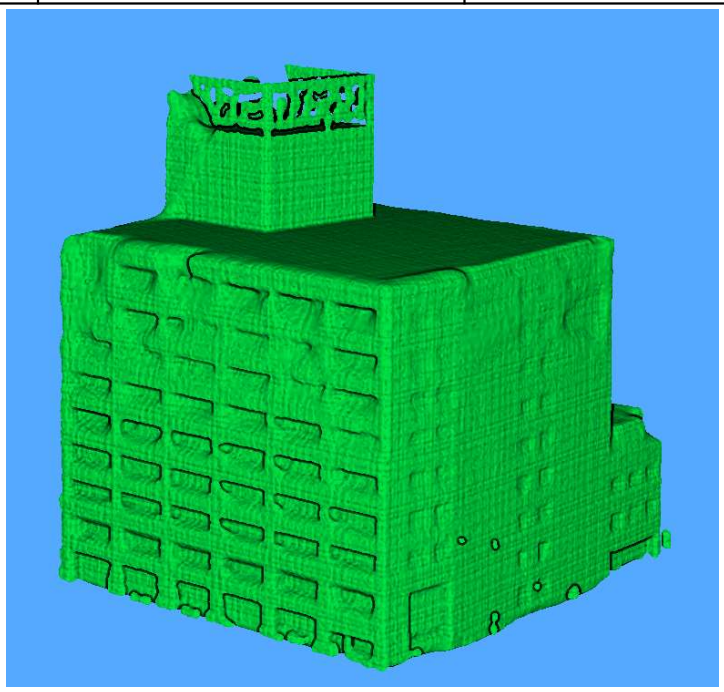

Image 14. Output of the iPSR algorithm, when LiDAR and LOD2 data complement each other well.