03/23/2024 BC. Langley

Creation of 3D models from a LiDAR point clouds file using Python and Open 3D

RESEARCH ADVISOR PROFESSOR
Andrew J. Park
RESEARCHER
César Antonio Hernández Hernández
Trinity Western University

Technologies such as LiDAR in recent years have brought new ways of digitally interpreting the environment that surrounds us, to be able to use it in multiple software and in this way generate technological advances in multiple fields of research.

The type of data used for this research is point clouds, A point cloud is a discrete set of data points in space. The points may represent a 3D shape or object. Each point position has its set of Cartesian coordinates (X, Y, Z). Point clouds are generally produced by 3D scanners or by photogrammetry software, which measure many points on the external surfaces of objects around them. As the output of 3D scanning processes, point clouds are used for many purposes, including to create 3D computer-aided design (CAD) or geographic information systems (GIS) models for manufactured parts, for metrology and quality inspection, and for a multitude of visualizing, animating, rendering, and mass customization applications.

What was sought throughout this research is the creation of meshes for 3D models from a point cloud file that is as similar as possible. For this, a very detailed mesh would need to be generated that could show all the information that is provided. by the cloud points. Likewise, to achieve this, it was concluded that to generate such a reliable mesh, many cloud points are required, the more cloud points, the more definition in the 3D model.

With LiAR data, we seek to create 3D models automatically. By simply entering the data, we will obtain exportable models for use in different programs such as game engines for different purposes.

The document "AlgoritmoPropio" is developed in spyder with Python 3.11.7, it generates a 3D model from the introduction of a laz file, or las and and exports an fbx document that can be used in 3D editing software.

The "lidar_vectorization" document is developed to be executed in Jupyter with Python 3.10.13, from a las format document, which covers a large surface of some environment (for example, a segment of a city scanned with LiDAR) generates a map with basic 3D models, which can be used in 3D editing software for multiple purposes

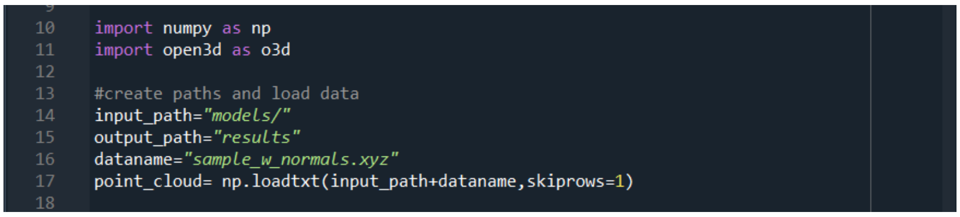
Both documents use Open3D as the main library, which helps us generate 3D shapes for different purposes.

The results of this investigation will be shown below, supported by screenshots to be able to explain in a more graphic way what could be obtained as a result, followed by a conclusion and future for the improvement of this system and for the continuation of this research

3D MODEL GENERATION FROM POINT CLOUDS

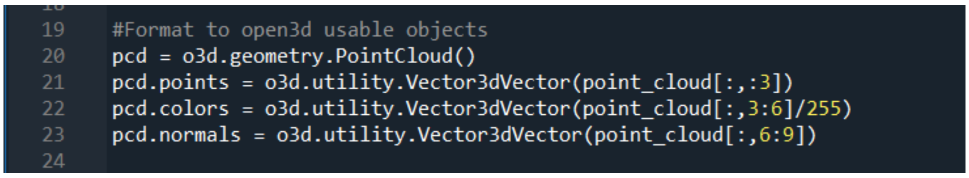
The document "AlgoritmoPropio.py" was developed in the Spyder editor and the libraries that were installed for said project were NumPy and open 3d, it is developed in Python 3.11.7.

1. We install NumPy and Open3D, so that all the code works, it is assigns them specific names to be able to reference them within the code, create paths and load data. The libraries that are going to be used throughout the code are imported.
2. To simplify steps, variables were created that were assigned the source and destination folders in which the data will be taken.



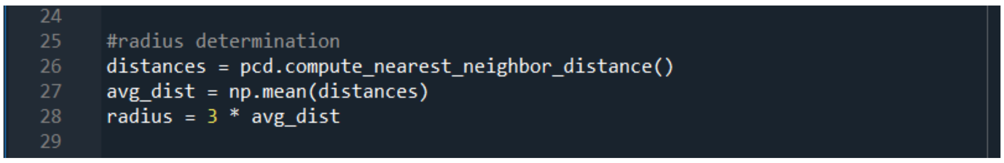
```python
import numpy as np
import open3d as o3d

#create paths and load data
input_path="models/"
output_path="results"
dataname="sample_w_normals.xyz"
point_cloud= np.loadtxt(input_path+dataname,skiprows=1)

```

3. Format to open3d usable objects.



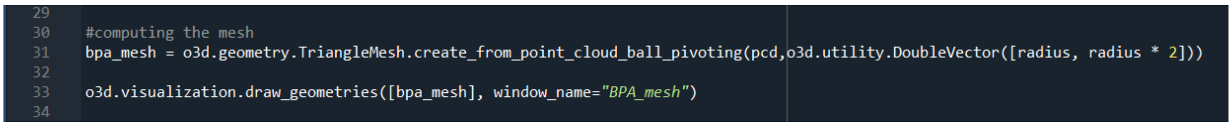
```python
#Format to open3d usable objects
pcd = o3d.geometry.PointCloud()
pcd.points = o3d.utility.Vector3dVector(point_cloud[:,:3])
pcd.colors = o3d.utility.Vector3dVector(point_cloud[:,3:6]/255)
pcd.normals = o3d.utility.Vector3dVector(point_cloud[:,6:9])

```

4. Radius determination.

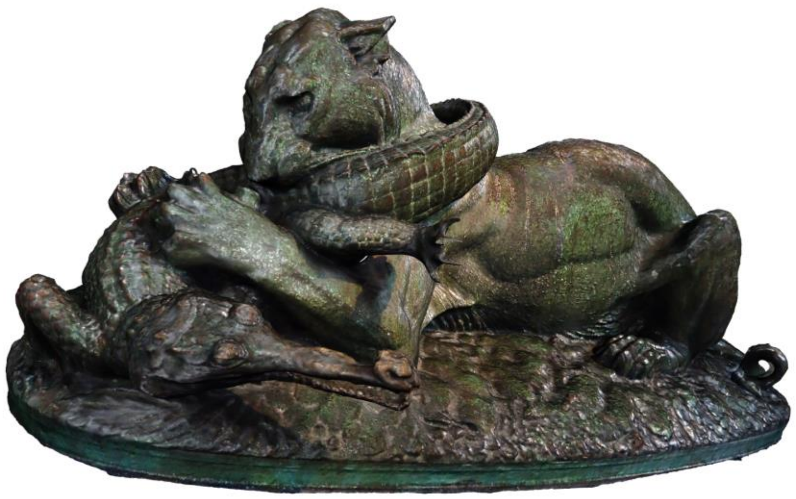


```python

#radius determination
distances = pcd.compute_nearest_neighbor_distance()
avg_dist = np.mean(distances)
radius = 3 * avg_dist

```

5. Computing the mesh and the file constructed by the cloud points is displayed

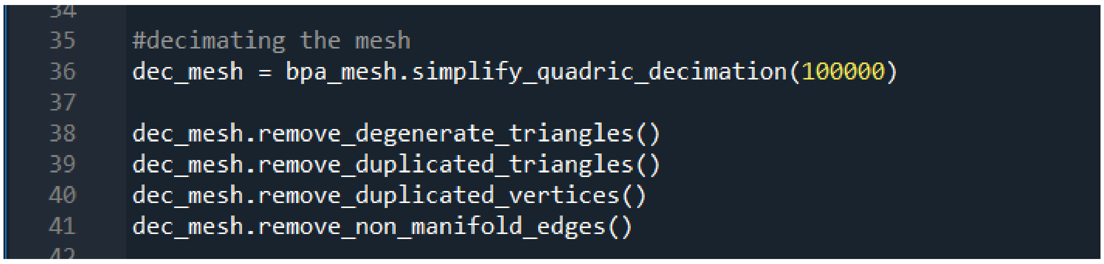


```python

#computing the mesh
bpa_mesh = o3d.geometry.TriangleMesh.create_from_point_cloud_ball_pivoting(pcd,o3d.utility.DoubleVector([radius, radius * 2]))

o3d.visualization.draw_geometries([bpa_mesh], window_name="BPA_mesh")

```



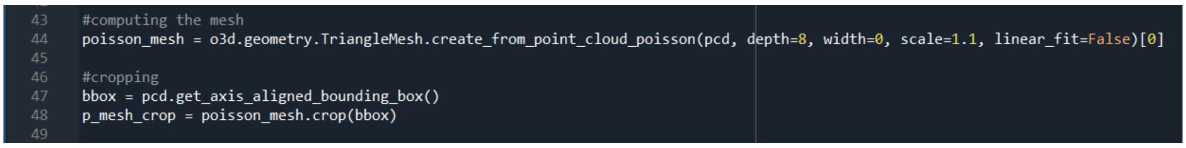
6. Decimating the mesh

03/23/2024 BC. Langley



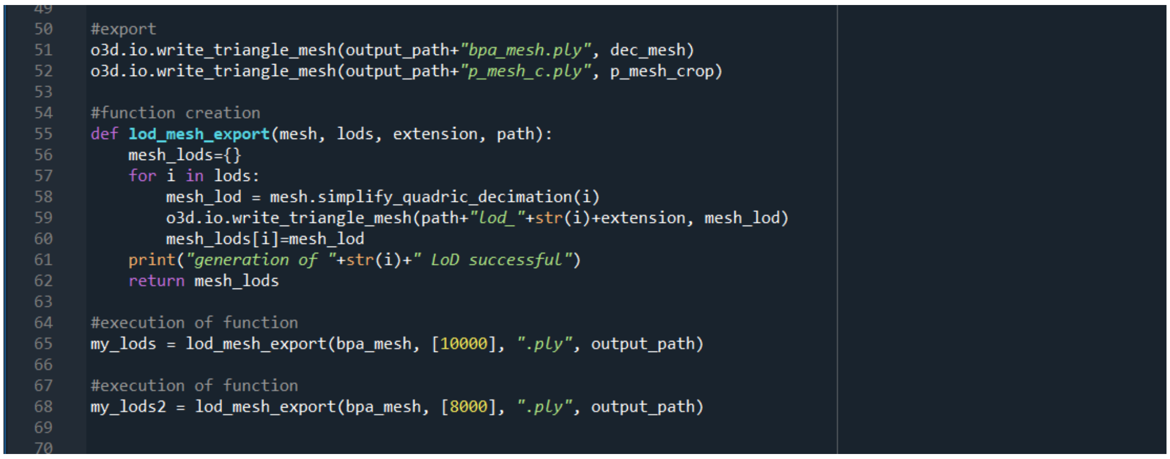
```python
#decimating the mesh
dec_mesh = bpa_mesh.simplify_quadric_decimation(100000)

dec_mesh.remove_degenerate_triangles()
dec_mesh.remove_duplicated_triangles()
dec_mesh.remove_duplicated_vertices()
dec_mesh.remove_non_manifold_edges()
```

7. Computing the mesh and cropping

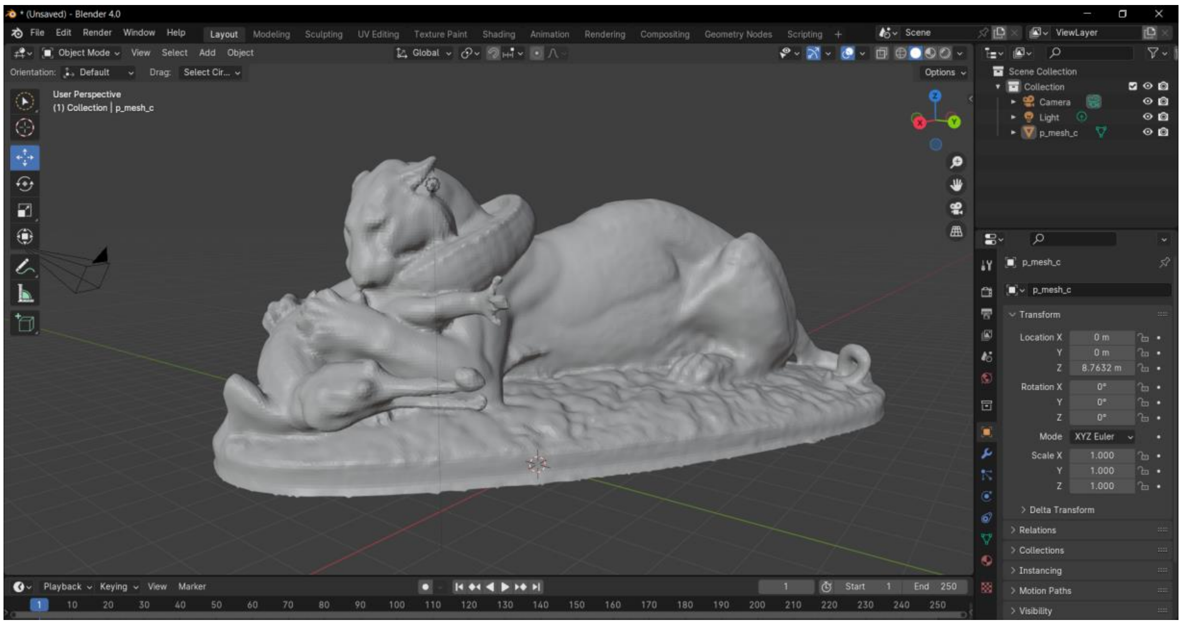


```python
#computing the mesh
poisson_mesh = o3d.geometry.TriangleMesh.create_from_point_cloud_poisson(pcd, depth=8, width=0, scale=1.1, linear_fit=False)[0]

#cropping
bbox = pcd.get_axis_aligned_bounding_box()
p_mesh_crop = poisson_mesh.crop(bbox)
```

8. Export to use in 3d editing software



```python
#export
o3d.io.write_triangle_mesh(output_path+"bpa_mesh.ply", dec_mesh)
o3d.io.write_triangle_mesh(output_path+"p_mesh_c.ply", p_mesh_crop)

#function creation
def lod_mesh_export(mesh, lods, extension, path):
    mesh_lods={}
    for i in lods:
        mesh_lod = mesh.simplify_quadric_decimation(i)
        o3d.io.write_triangle_mesh(path+"lod_"+str(i)+extension, mesh_lod)
        mesh_lods[i]=mesh_lod
    print("generation of "+str(i)+" LoD successful")
    return mesh_lods

#execution of function
my_lods = lod_mesh_export(bpa_mesh, [10000], ".ply", output_path)

#execution of function
my_lods2 = lod_mesh_export(bpa_mesh, [8000], ".ply", output_path)
```


* (Unsaved) - Blender 4.0
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Geometry Nodes Scripting
Scene
ViewLayer
Object Mode View Select Add Object
Global
Options
User Perspective
(1) Collection | p_mesh_c
Scene Collection
Collection
Camera
Light
p_mesh_c
Transform
Delta Transform
Relations
Collections
Instancing
Motion Paths
Visibility
Playback Keying View Marker
Start 1 End 250

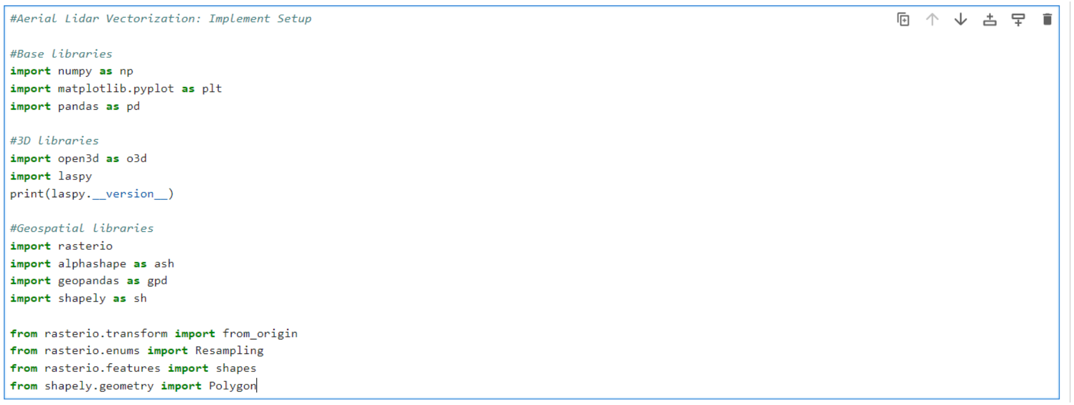
Orientation: Default Drag: Select Cir...

Location X 0 m
Y 0 m
Z 8.7632 m
Rotation X 0°
Y 0°
Z 0°
Mode XYZ Euler
Scale X 1.000
Y 1.000
Z 1.000

03/23/2024 BC. Langley

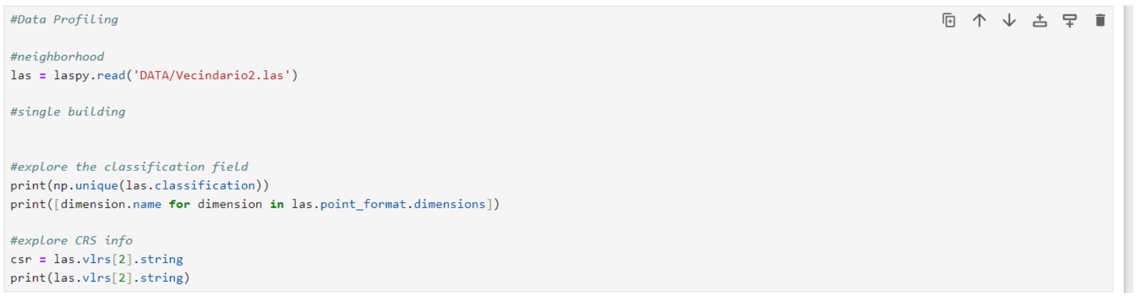
LiDAR VECTORIZATION FOR NEIGHBORHOODS

1. Aerial Lidar Vectorization: Implement Setup



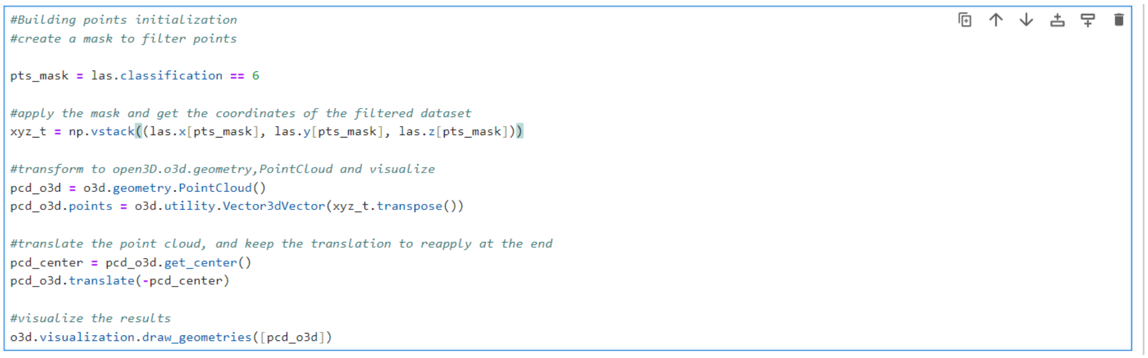
```
#Aerial Lidar Vectorization: Implement Setup

#Base libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#3D libraries
import open3d as o3d
import laspy
print(laspy.__version__)

#Geospatial libraries
import rasterio
import alphashape as ash
import geopandas as gpd
import shapely as sh

from rasterio.transform import from_origin
from rasterio.enums import Resampling
from rasterio.features import shapes
from shapely.geometry import Polygon
```

2. Data Profiling



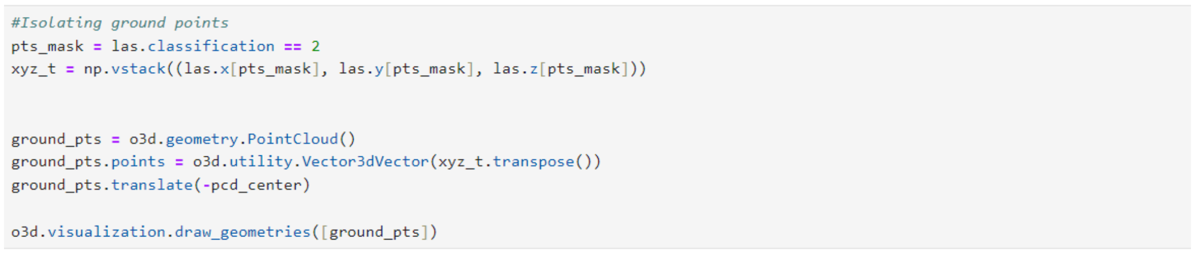
```
#Data Profiling

#neighborhood
las = laspy.read('DATA/Vecindario2.las')

#single building


#explore the classification field
print(np.unique(las.classification))
print([dimension.name for dimension in las.point_format.dimensions])

#explore CRS info
csr = las.vlrs[2].string
print(las.vlrs[2].string)
```

3. Building points initialization and create a mask to filter points



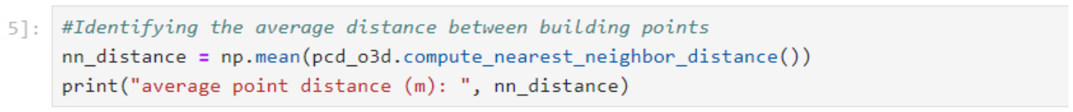
```
#Building points initialization
#create a mask to filter points

pts_mask = las.classification == 6

#apply the mask and get the coordinates of the filtered dataset
xyz_t = np.vstack((las.x[pts_mask], las.y[pts_mask], las.z[pts_mask]))

#transform to open3D.o3d.geometry,PointCloud and visualize
pcd_o3d = o3d.geometry.PointCloud()
pcd_o3d.points = o3d.utility.Vector3dVector(xyz_t.transpose())

#translate the point cloud, and keep the translation to reapply at the end
pcd_center = pcd_o3d.get_center()
pcd_o3d.translate(-pcd_center)

#visualize the results
o3d.visualization.draw_geometries([pcd_o3d])
```

4. Isolating ground points



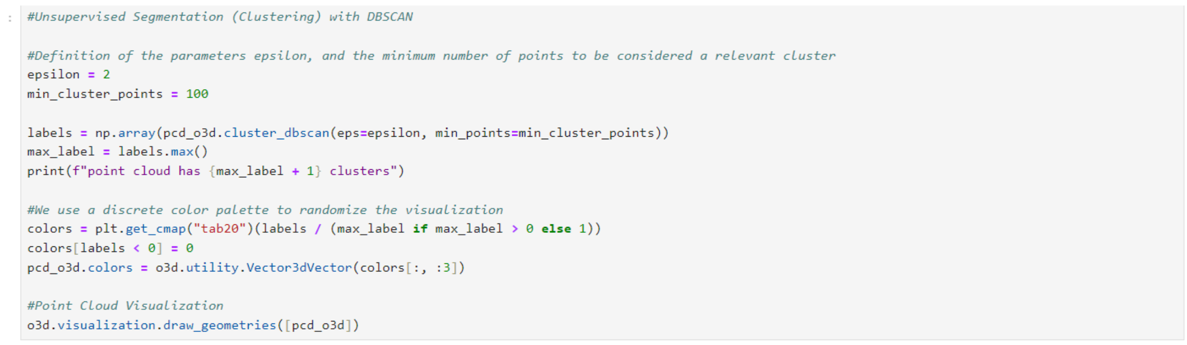
```
#Isolating ground points
pts_mask = las.classification == 2
xyz_t = np.vstack((las.x[pts_mask], las.y[pts_mask], las.z[pts_mask]))


ground_pts = o3d.geometry.PointCloud()
ground_pts.points = o3d.utility.Vector3dVector(xyz_t.transpose())
ground_pts.translate(-pcd_center)

o3d.visualization.draw_geometries([ground_pts])
```

5. Identifying the average distance between building points



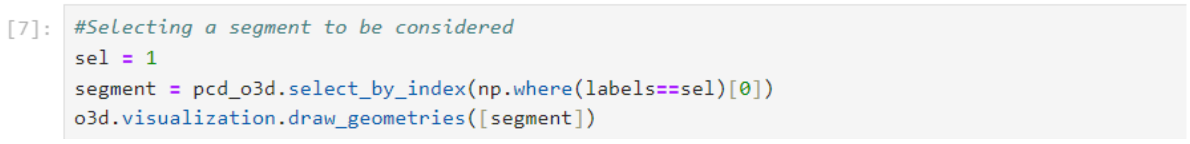
```
5]:  #Identifying the average distance between building points
     nn_distance = np.mean(pcd_o3d.compute_nearest_neighbor_distance())
     print("average point distance (m): ", nn_distance)
```

6. Unsupervised Segmentation (Clustering) with DBSCAN



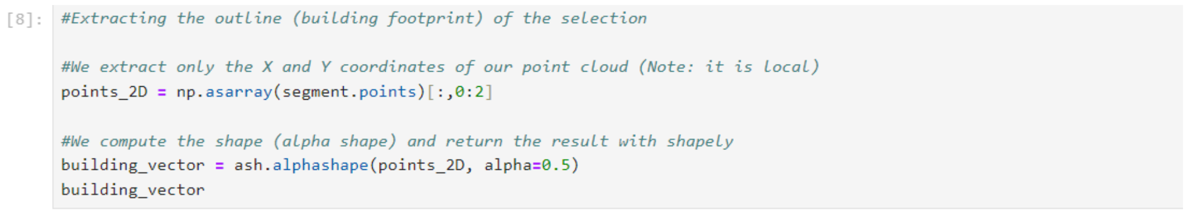
```
:  #Unsupervised Segmentation (Clustering) with DBSCAN

   #Definition of the parameters epsilon, and the minimum number of points to be considered a relevant cluster
   epsilon = 2
   min_cluster_points = 100

   labels = np.array(pcd_o3d.cluster_dbscan(eps=epsilon, min_points=min_cluster_points))
   max_label = labels.max()
   print(f"point cloud has {max_label + 1} clusters")

   #We use a discrete color palette to randomize the visualization
   colors = plt.get_cmap("tab20")(labels / (max_label if max_label > 0 else 1))
   colors[labels < 0] = 0
   pcd_o3d.colors = o3d.utility.Vector3dVector(colors[:, :3])

   #Point Cloud Visualization
   o3d.visualization.draw_geometries([pcd_o3d])
```

7. Selecting a segment to be considered



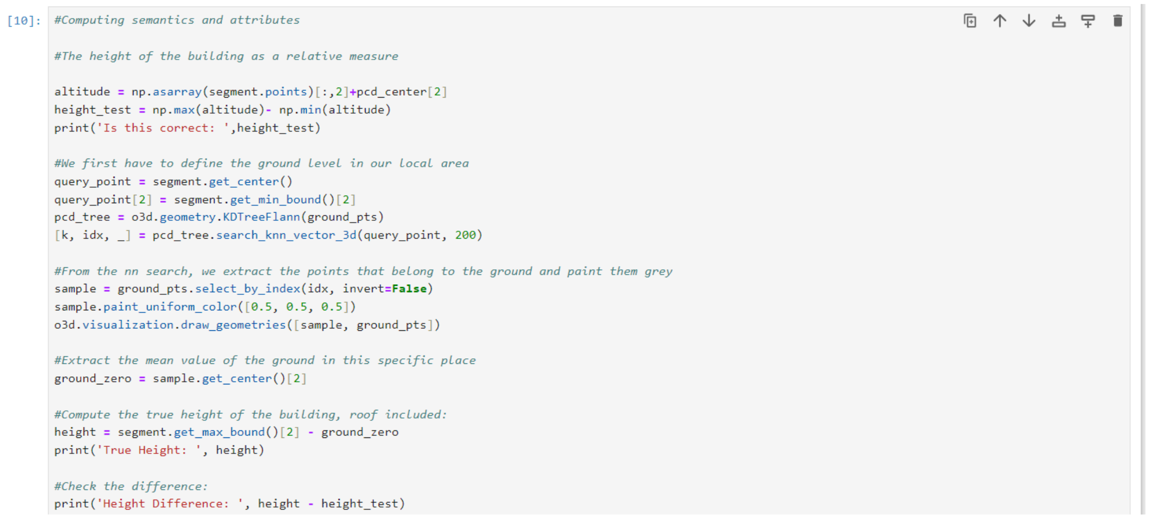
```
[7]:  #Selecting a segment to be considered
      sel = 1
      segment = pcd_o3d.select_by_index(np.where(labels==sel)[0])
      o3d.visualization.draw_geometries([segment])
```

8. Extracting the outline (building footprint) of the selection



```
[8]:  #Extracting the outline (building footprint) of the selection

      #We extract only the X and Y coordinates of our point cloud (Note: it is local)
      points_2D = np.asarray(segment.points)[:,0:2]

      #We compute the shape (alpha shape) and return the result with shapely
      building_vector = ash.alphashape(points_2D, alpha=0.5)
      building_vector
```

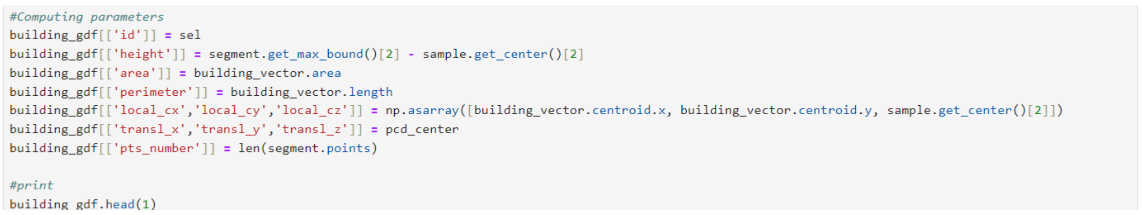
9. Computing semantics and attributes



```
[10]:  #Computing semantics and attributes

       #The height of the building as a relative measure

       altitude = np.asarray(segment.points)[:,2]+pcd_center[2]
       height_test = np.max(altitude)- np.min(altitude)
       print('Is this correct: ',height_test)

       #We first have to define the ground level in our local area
       query_point = segment.get_center()
       query_point[2] = segment.get_min_bound()[2]
       pcd_tree = o3d.geometry.KDTreeFlann(ground_pts)
       [k, idx, _] = pcd_tree.search_knn_vector_3d(query_point, 200)

       #From the nn search, we extract the points that belong to the ground and paint them grey
       sample = ground_pts.select_by_index(idx, invert=False)
       sample.paint_uniform_color([0.5, 0.5, 0.5])
       o3d.visualization.draw_geometries([sample, ground_pts])

       #Extract the mean value of the ground in this specific place
       ground_zero = sample.get_center()[2]

       #Compute the true height of the building, roof included:
       height = segment.get_max_bound()[2] - ground_zero
       print('True Height: ', height)

       #Check the difference:
       print('Height Difference: ', height - height_test)
```

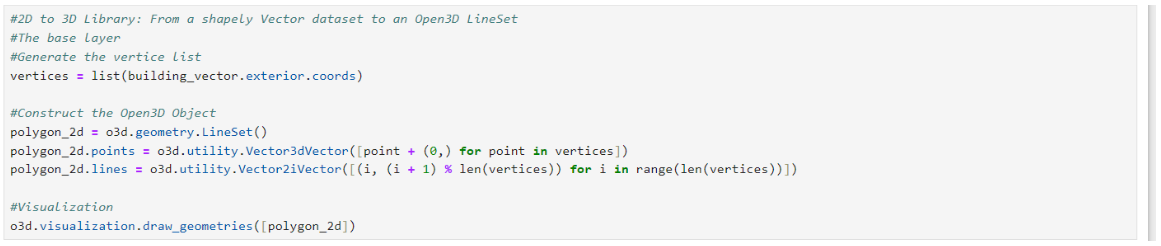
10. Computing parameters

03/23/2024 BC. Langley



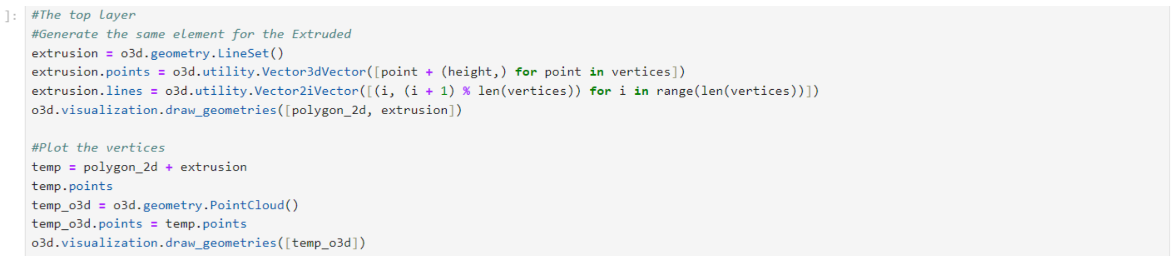
```python
#Computing parameters
building_gdf[['id']] = sel
building_gdf[['height']] = segment.get_max_bound()[2] - sample.get_center()[2]
building_gdf[['area']] = building_vector.area
building_gdf[['perimeter']] = building_vector.length
building_gdf[['local_cx','local_cy','local_cz']] = np.asarray([building_vector.centroid.x, building_vector.centroid.y, sample.get_center()[2]])
building_gdf[['transl_x','transl_y','transl_z']] = pcd_center
building_gdf[['pts_number']] = len(segment.points)

#print
building_gdf.head(1)
```

11. 2D to 3D Library: From a shapely Vector dataset to an Open3D LineSet



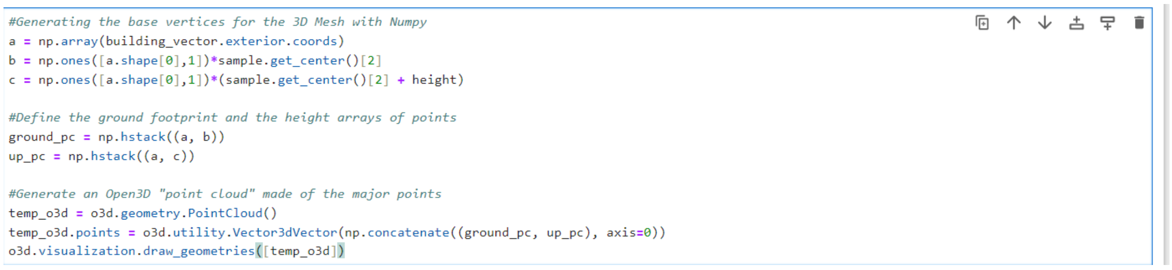
```python
#2D to 3D Library: From a shapely Vector dataset to an Open3D LineSet
#The base layer
#Generate the vertice list
vertices = list(building_vector.exterior.coords)

#Construct the Open3D Object
polygon_2d = o3d.geometry.LineSet()
polygon_2d.points = o3d.utility.Vector3dVector([point + (0,) for point in vertices])
polygon_2d.lines = o3d.utility.Vector2iVector([(i, (i + 1) % len(vertices)) for i in range(len(vertices))])

#Visualization
o3d.visualization.draw_geometries([polygon_2d])
```

12. The top layer and plot the vertices



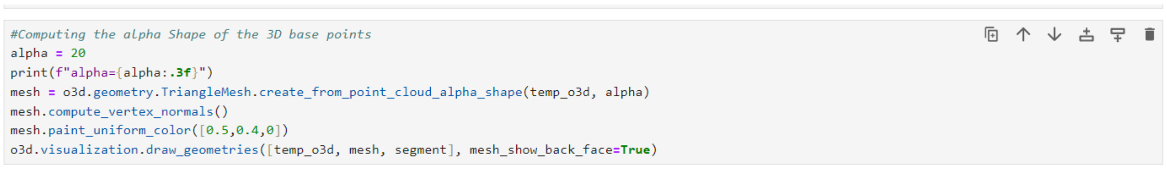
```python
#The top layer
#Generate the same element for the Extruded
extrusion = o3d.geometry.LineSet()
extrusion.points = o3d.utility.Vector3dVector([point + (height,) for point in vertices])
extrusion.lines = o3d.utility.Vector2iVector([(i, (i + 1) % len(vertices)) for i in range(len(vertices))])
o3d.visualization.draw_geometries([polygon_2d, extrusion])

#Plot the vertices
temp = polygon_2d + extrusion
temp.points
temp_o3d = o3d.geometry.PointCloud()
temp_o3d.points = temp.points
o3d.visualization.draw_geometries([temp_o3d])
```

13. Generating the base vertices for the 3D Mesh with Numpy



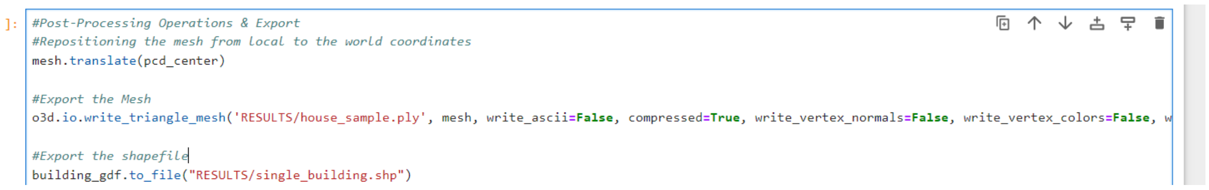
```python
#Generating the base vertices for the 3D Mesh with Numpy
a = np.array(building_vector.exterior.coords)
b = np.ones([a.shape[0],1])*sample.get_center()[2]
c = np.ones([a.shape[0],1])*(sample.get_center()[2] + height)

#Define the ground footprint and the height arrays of points
ground_pc = np.hstack((a, b))
up_pc = np.hstack((a, c))

#Generate an Open3D "point cloud" made of the major points
temp_o3d = o3d.geometry.PointCloud()
temp_o3d.points = o3d.utility.Vector3dVector(np.concatenate((ground_pc, up_pc), axis=0))
o3d.visualization.draw_geometries([temp_o3d])
```

14. Computing the alpha Shape of the 3D base points



```python
#Computing the alpha Shape of the 3D base points
alpha = 20
print(f"alpha={alpha:.3f}")
mesh = o3d.geometry.TriangleMesh.create_from_point_cloud_alpha_shape(temp_o3d, alpha)
mesh.compute_vertex_normals()
mesh.paint_uniform_color([0.5,0.4,0])
o3d.visualization.draw_geometries([temp_o3d, mesh, segment], mesh_show_back_face=True)
```

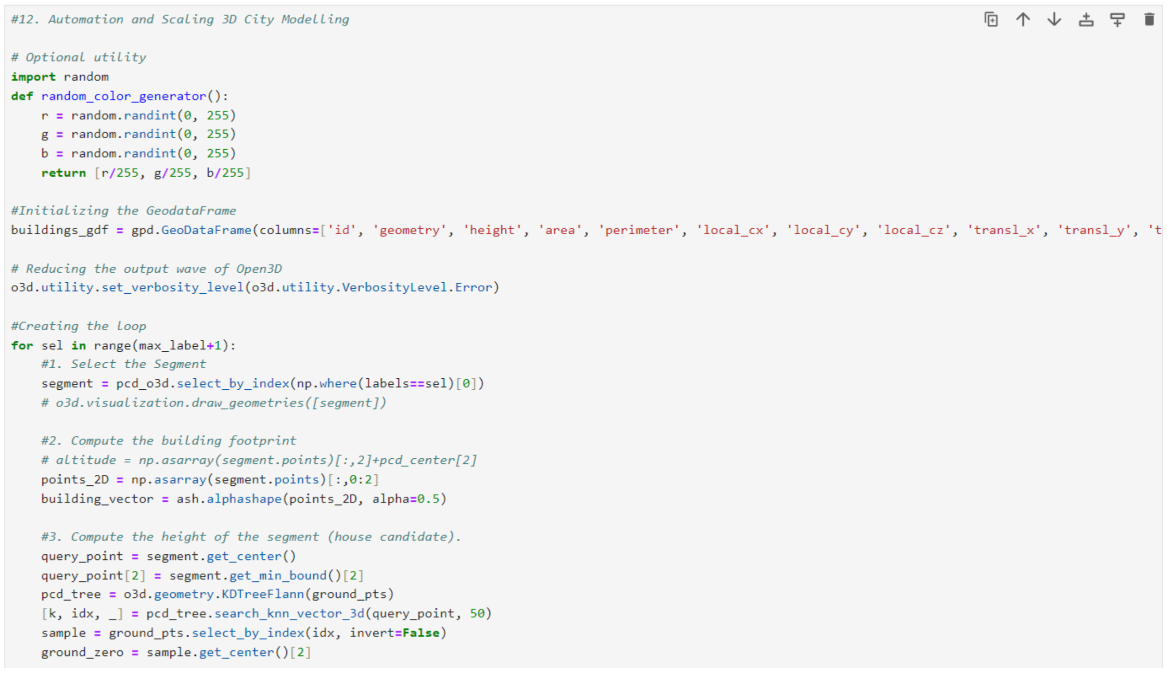
15. Post-Processing Operations & Export



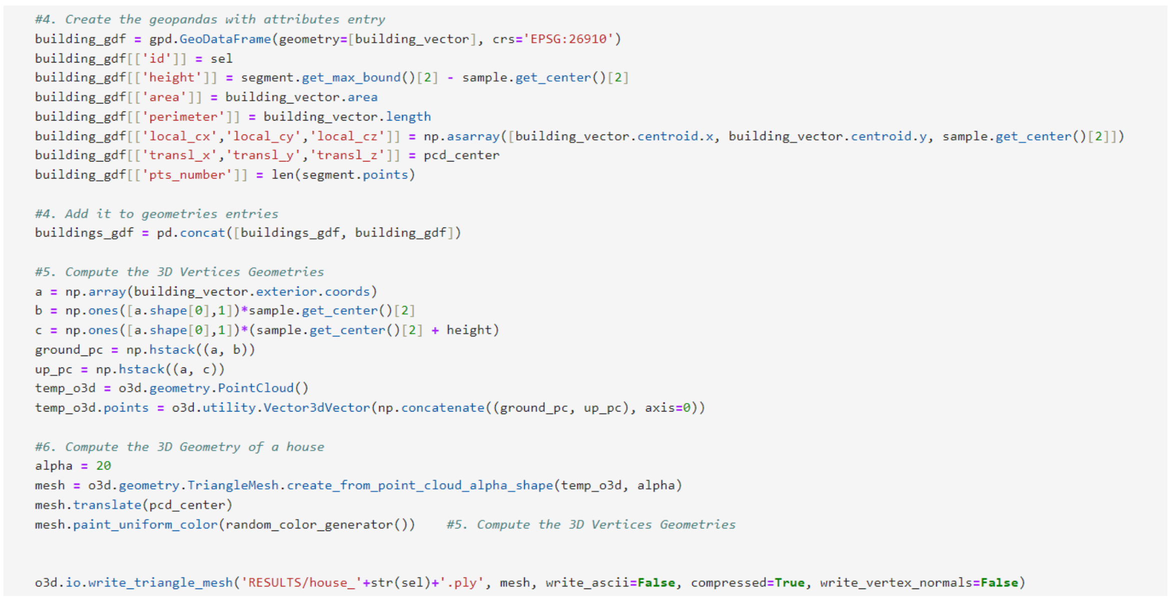
```python
#Post-Processing Operations & Export
#Repositioning the mesh from local to the world coordinates
mesh.translate(pcd_center)

#Export the Mesh
o3d.io.write_triangle_mesh('RESULTS/house_sample.ply', mesh, write_ascii=False, compressed=True, write_vertex_normals=False, write_vertex_colors=False, w

#Export the shapefile
building_gdf.to_file("RESULTS/single_building.shp")
```

16. Automation and Scaling 3D City Modelling

03/23/2024 BC. Langley

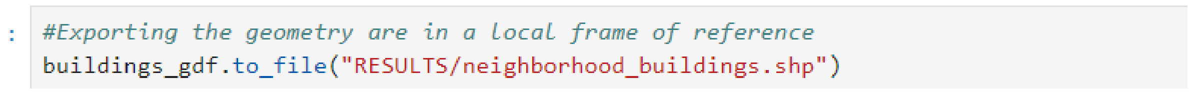


```python
#12. Automation and Scaling 3D City Modelling

# Optional utility
import random
def random_color_generator():
    r = random.randint(0, 255)
    g = random.randint(0, 255)
    b = random.randint(0, 255)
    return [r/255, g/255, b/255]

#Initializing the GeodataFrame
buildings_gdf = gpd.GeoDataFrame(columns=['id', 'geometry', 'height', 'area', 'perimeter', 'local_cx', 'local_cy', 'local_cz', 'transl_x', 'transl_y', 't

# Reducing the output wave of Open3D
o3d.utility.set_verbosity_level(o3d.utility.VerbosityLevel.Error)

#Creating the loop
for sel in range(max_label+1):
    #1. Select the Segment
    segment = pcd_o3d.select_by_index(np.where(labels==sel)[0])
    # o3d.visualization.draw_geometries([segment])

    #2. Compute the building footprint
    # altitude = np.asarray(segment.points)[:,2]+pcd_center[2]
    points_2D = np.asarray(segment.points)[:,0:2]
    building_vector = ash.alphashape(points_2D, alpha=0.5)

    #3. Compute the height of the segment (house candidate).
    query_point = segment.get_center()
    query_point[2] = segment.get_min_bound()[2]
    pcd_tree = o3d.geometry.KDTreeFlann(ground_pts)
    [k, idx, _] = pcd_tree.search_knn_vector_3d(query_point, 50)
    sample = ground_pts.select_by_index(idx, invert=False)
    ground_zero = sample.get_center()[2]
```



```python
    #4. Create the geopandas with attributes entry
    building_gdf = gpd.GeoDataFrame(geometry=[building_vector], crs='EPSG:26910')
    building_gdf[['id']] = sel
    building_gdf[['height']] = segment.get_max_bound()[2] - sample.get_center()[2]
    building_gdf[['area']] = building_vector.area
    building_gdf[['perimeter']] = building_vector.length
    building_gdf[['local_cx','local_cy','local_cz']] = np.asarray([building_vector.centroid.x, building_vector.centroid.y, sample.get_center()[2]])
    building_gdf[['transl_x','transl_y','transl_z']] = pcd_center
    building_gdf[['pts_number']] = len(segment.points)

    #4. Add it to geometries entries
    buildings_gdf = pd.concat([buildings_gdf, building_gdf])

    #5. Compute the 3D Vertices Geometries
    a = np.array(building_vector.exterior.coords)
    b = np.ones([a.shape[0],1])*sample.get_center()[2]
    c = np.ones([a.shape[0],1])*(sample.get_center()[2] + height)
    ground_pc = np.hstack((a, b))
    up_pc = np.hstack((a, c))
    temp_o3d = o3d.geometry.PointCloud()
    temp_o3d.points = o3d.utility.Vector3dVector(np.concatenate((ground_pc, up_pc), axis=0))

    #6. Compute the 3D Geometry of a house
    alpha = 20
    mesh = o3d.geometry.TriangleMesh.create_from_point_cloud_alpha_shape(temp_o3d, alpha)
    mesh.translate(pcd_center)
    mesh.paint_uniform_color(random_color_generator())    #5. Compute the 3D Vertices Geometries


    o3d.io.write_triangle_mesh('RESULTS/house_'+str(sel)+'.ply', mesh, write_ascii=False, compressed=True, write_vertex_normals=False)
```

17. Exporting the geometry are in a local frame of reference



```python
#Exporting the geometry are in a local frame of reference
buildings_gdf.to_file("RESULTS/neighborhood_buildings.shp")
```

CONCLUSION

As we can see, it is possible to generate 3D models from cloud point files, but to do so the file must have many cloud points if the 3D model is to be as similar as possible.

It was also possible to demonstrate that a large amount of space, such as a neighborhood, can be generated and efficiently obtain simple 3D models of the structures found near the area.

The final objective of this research is to be able to generate a 3D model of an environment that covers a large area, such as an entire neighborhood, a metropolitan area or a city as faithfully as possible, including detailed buildings and other objects that are found. in the environment

However, in this project the objective could not be reached but a path was started to achieve it in the future.

One of the ideas that exists to achieve this is the use of AI to be able to interpret the cloud point files and thus generate 3D models in a more efficient way that do not require many cloud points. For example, having a tree, when the AI reads a file of cloud points and interprets that it is a tree, it can generate a predefined tree model, in this way if the file does not have a large number of cloud points, This is not an impediment to having a well-defined tree model with a realistic appearance. Using this logic, we can apply it to an entire city, since the use of an AI would reduce the number of resources that would be needed to create a 3D model of an entire city.

As such, there is no software that applies the use of AI in the interpretation of cloud points in this way, but if there are research articles that have made progress in this, links to said articles are left below for analysis and use. as a reference to follow this project

- "Automated 3D Point Cloud Data Processing Using AI" https://discovery.ucl.ac.uk/id/eprint/10095489/
- "Learning Efficient Point Cloud Generation for Dense 3D Object Reconstruction" https://ojs.aaai.org/index.php/AAAI/article/view/12278
- "Point cloud dataset creation for machine learning on CAD models" https://pureadmin.qub.ac.uk/ws/files/205558238/PointCloud.pdf
- "From point cloud to surface: the modeling and visualization problem" https://www.research-collection.ethz.ch/handle/20.500.11850/369698
- "Deep Learning on 3D Point Clouds" https://www.mdpi.com/2072-4292/12/11/1729