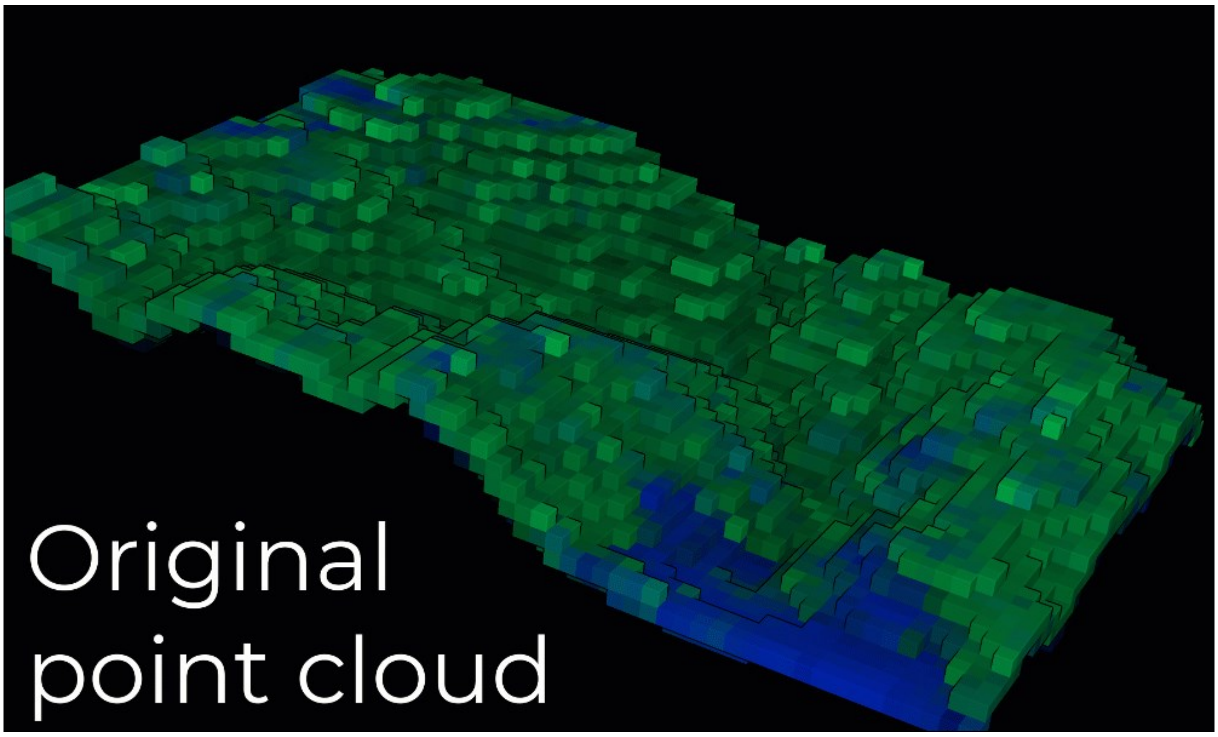
How to automate LiDAR point cloud sub-sampling with Python

The ultimate guide to subsample 3D point clouds from scratch, with Python. Two efficient methods are shown to import, process, structure as a voxel grid, and visualise LiDAR data.



Point cloud sampling results by following the strategies explained in this guide. © F. Poux

In this article, I will give you my two favourite 3D processes for quickly structuring and sub-sampling point cloud data with python. You will also be able to automate, export, visualize and integrate results into your favourite 3D software, without any coding experience. I will focus on code optimization while using a minimum number of libraries (mainly NumPy) so that you can extend what you learnt with very high flexibility! Ready 😁?

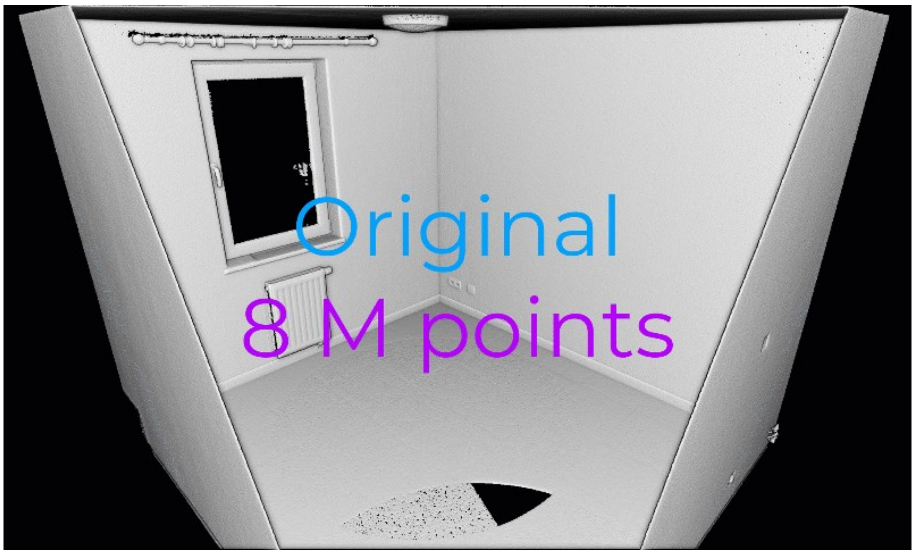
Why do we need to sub-sample point clouds?

Published in Towards Data Science

Point cloud datasets are marvellous! You can get a geometric description of world entities by discretizing them through a bunch of points, which, aggregated together, resemble the shape—the environment—of interest.

This is a point cloud of an abandoned wool factory. It was obtained by combining 3D Laser scanning technology with photogrammetry. We created it with my friend Roman Robroek. While super interesting, learning underlying 3D capture techniques extends the scope of the article.

But a major problem with 3D point clouds is that the data density may be more than necessary for a given application. This often leads to higher computational cost in subsequent data processing or visualisation. To make the dense point clouds more manageable, their data density can be reduced. This article provides you with the knowledge and actual scripts to implement sub-sampling methods for reducing point cloud data density.



Adapting the number of points in the point cloud is often a savant use of domain knowledge to balance representativity & information redundancy. © Florent Poux

Let us dive in 🤿!

Published in Towards Data Science

Some light 3D theory, don't you think?

Ha, I tricked you 🙃. Before directly diving to the implementation of sampling strategies, let us first review the typical sub-sampling methods for point cloud data thinning. These include the random, the minimal distance and the grid (often tagged as uniform) methods. The random method is the simplest for reducing data density, in which a specified number of data points is selected randomly.

Original
8 M points

The point cloud of this indoor room is sampled randomly. © Florent Poux

In the minimal distance method, the data point selection is constrained by a minimum distance so that no data point in the selected subset is closer to another data point than the minimum distance specified.

The point cloud is sampled spatially, by making sure each point is at least 2 cm (scenario 1) or 5 cm (scenario 2) from any point. © Florent Poux

In the grid method (which can be uniform), a grid structure—the handier being a voxel grid structure—is created and a representative data point is selected.

The point cloud is sampled using a voxel grid, with different voxel sizes. For each voxel, one representative point is retained. © Florent Poux

The latter two methods can achieve a more homogeneous spatial distribution of data points in the reduced point cloud. In such cases, the average data spacing is determined by the minimal distance or the voxel edge length specified.

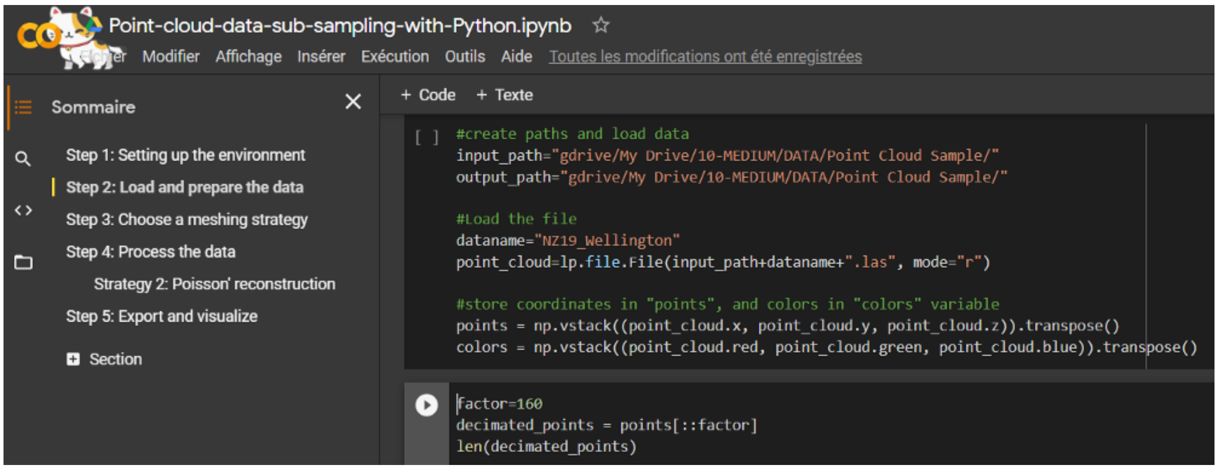
Okay for the theory, let us put it into action 🤠!

Step 1: Launch your Python environment.

In the previous article below, we saw how to set-up an environment easily with Anaconda and how to use the IDE Spyder for managing your code. I recommend continuing in this fashion if you set yourself up to becoming a fully-fledge python app developer 😆.

Discover 3D Point Cloud Processing with Python
*Tutorial to simply set up your python environment, start processing and visualize 3D point cloud data.*towardsdatascience.com

But hey, if you prefer to do everything from scratch in the next 5 minutes, I also give you access to a Google Colab notebook that you will find at the end of the article. There is nothing to install; you can just save it to your google drive and start working with it, also using the free datasets from Step 2 👇.


Point-cloud-data-sub-sampling-with-Python.ipynb
Fichier Modifier Affichage Insérer Exécution Outils Aide Toutes les modifications ont été enregistrées
Sommaire
Step 1: Setting up the environment
Step 2: Load and prepare the data
Step 3: Choose a meshing strategy
Step 4: Process the data
Strategy 2: Poisson' reconstruction
Step 5: Export and visualize
Section
+ Code + Texte

```
#create paths and load data
input_path="gdrive/My Drive/10-MEDIUM/DATA/Point Cloud Sample/"
output_path="gdrive/My Drive/10-MEDIUM/DATA/Point Cloud Sample/"

#Load the file
dataname="NZ19_Wellington"
point_cloud=lp.file.File(input_path+dataname+".las", mode="r")

#store coordinates in "points", and colors in "colors" variable
points = np.vstack((point_cloud.x, point_cloud.y, point_cloud.z)).transpose()
colors = np.vstack((point_cloud.red, point_cloud.green, point_cloud.blue)).transpose()
```

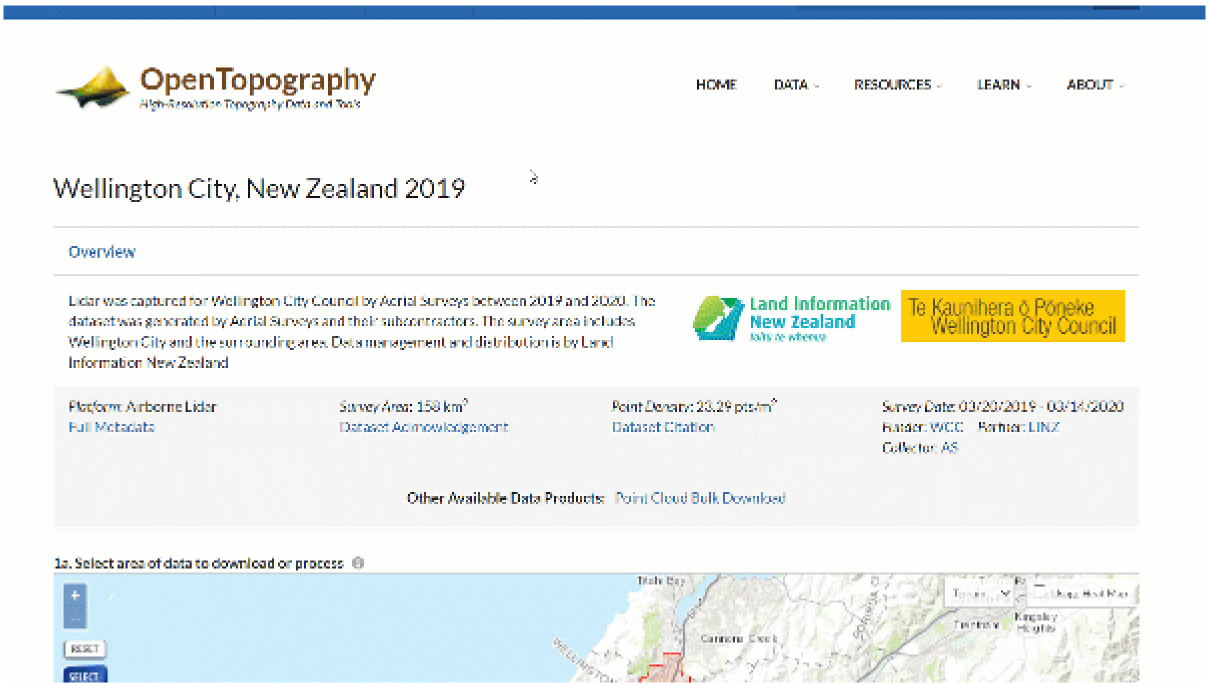
```
factor=160
decimated_points = points[::factor]
len(decimated_points)
```

In the Google Colab file, you can just run the script cell by cell and benefit from a direct coding experience, on the web. A great way to start experimenting with Python. (Yes, cats are walking in the window 😸). © F. Poux

Step 2: Download a point cloud dataset

In previous tutorials, I illustrated point cloud processing and meshing over a 3D dataset obtained by using photogrammetry: the jaguar, that you can freely download from this repository.

In this tutorial, we will extend the scope, and test on a point cloud obtained through an aerial LiDAR survey. This is an excellent opportunity to introduce you to the great Open Data platform: Open Topography. It is a collaborative data repository for LiDAR users. Through a web map, you can select a region of interest, and download the related point cloud dataset with its metadata in different file formats (.laz, .las or as an ASCII file).


OpenTopography
High-Resolution Topography Data and Tools
HOME
DATA
RESOURCES
LEARN
ABOUT
Land Information New Zealand
Te Kaunihera o Pōneke Wellington City Council
RESET
SELECT
Titahi Bay
Canmore Creek
Kingsley Heights

Wellington City, New Zealand 2019

Overview

Lidar was captured for Wellington City Council by Aerial Surveys between 2019 and 2020. The dataset was generated by Aerial Surveys and their subcontractors. The survey area includes Wellington City and the surrounding area. Data management and distribution is by Land Information New Zealand

Platform: Airborne Lidar
Full Metadata

Survey Area: 158 km^2
Dataset Acknowledgement

Point Density: 23.29 pts/m^2
Dataset Citation

Survey Date: 03/20/2019 - 03/14/2020
Funder: WCC Partner: LINZ
Collector: AS

Other Available Data Products: Point Cloud Bulk Download

1a. Select area of data to download or process

Download a point cloud of interest from the OpenTopography Open Data Platform. This will be the dataset that we will use for this tutorial. © Florent Poux

At this phase, what is important to know is that you can easily process both the ASCII file and the .las file with python (the .laz is more tricky). The .las file is far more compressed than the ASCII file (355 Mo vs 1026 Mo for the example in this guide), but it will necessitate that you use a library called LasPy. So now, if you need 3D point cloud datasets over a large region, you know where you can find great datasets easily 🗺.

🤓 ***Note**:* For this how-to guide, you can use the point cloud in this repository, that I already filtered, colourized and translated so that you are in the optimal conditions. If you want to visualize and play with it beforehand without installing anything, you can check out the webGL version.

Okay, now that we are set-up, let us write some code 💻. First, we install the library package that is missing to read .las files. If you are with anaconda, I suggest you run the following command by looking up the conda-forge channel:

```
conda install -c conda-forge
```

Else, in general, you can use the pip package installer for Python by typing:

```
pip install laspy
```

Then, let us import necessary libraries within the script (NumPy and LasPy), and load the .las file in a variable called point_cloud.

```
import numpy as np
import laspy as lp
```

```
input_path="gdrive/My Drive/10-MEDIUM/DATA/Point Cloud
Sample/"
dataname="NZ19_Wellington.las"
```

```
point_cloud=lp.file.File(input_path+dataname+".las", mode="r")
```

Nice, we are almost ready! What is great, is that the LasPy library also give a structure to the point_cloud variable, and we can use straightforward methods to get, for example, X, Y, Z, Red, Blue and Green fields. Let us do this to separate coordinates from colours, and put them in NumPy arrays:

```
points = np.vstack((point_cloud.x, point_cloud.y,
point_cloud.z)).transpose()
```

```
colors = np.vstack((point_cloud.red, point_cloud.green,
point_cloud.blue)).transpose()
```

🤓 ***Note**:* We use a vertical stack method from NumPy, and we have to transpose it to get from (n x 3) to a (3 x n) matrix of the point cloud.

And we are set up! Moving on to step 3 👇.

Published in Towards Data Science

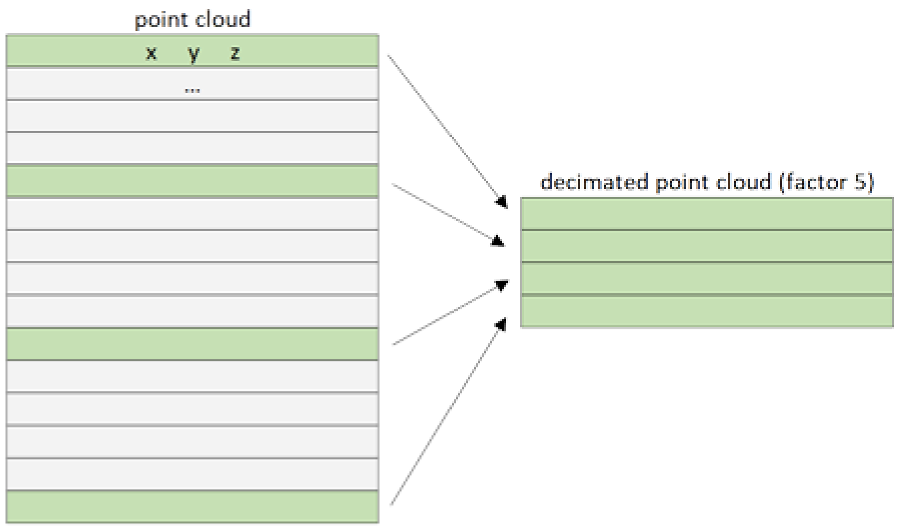
Step 3: Choose a sampling strategy.

We will focus on decimation and voxel grid sampling. Now is the time to pick a side ☺

💡 ***Hint:*** *I will give you code scripts that actually maximize the use of NumPy, but know that you can achieve similar results with widely different implementations (or through importing other packages). The main difference is often the execution time. The goal is to have the best execution runtime while having a readable script.*

Strategy 1: Point Cloud Random subsampling

If we define a point cloud as a matrix (m x n), then the **decimated cloud** is obtained by keeping one row out of n of this matrix :



At the matrix level, the decimation simply acts by keeping points every nth row depending on the n factor. Of course, this is made based on how are stored the points in the file. © F. Poux

Slicing a list in python is pretty simple with the command l[start:end:step]. To shorten and parametrize the expression, you can just write the lines:

```
factor=160
decimated_points_random = points[::factor]
```

🤓 ***Note****:* Running this will keep 1 row every 160 rows, thus diving the size of the original point cloud by 160. It goes from 13 993 118 points to 87 457 points.

Top-view of the point cloud and its decimated counterpart. © F. Poux

Strategie 2: Point Cloud Grid Sampling

The grid subsampling strategy will be based on the division of the 3D space in regular cubic cells called voxels. For each cell of this grid, we will only keep one representative point. This point, the representant of the cell, can be chosen in different ways. For example, it can be the barycenter of the points in that cell, or the closest point to it.

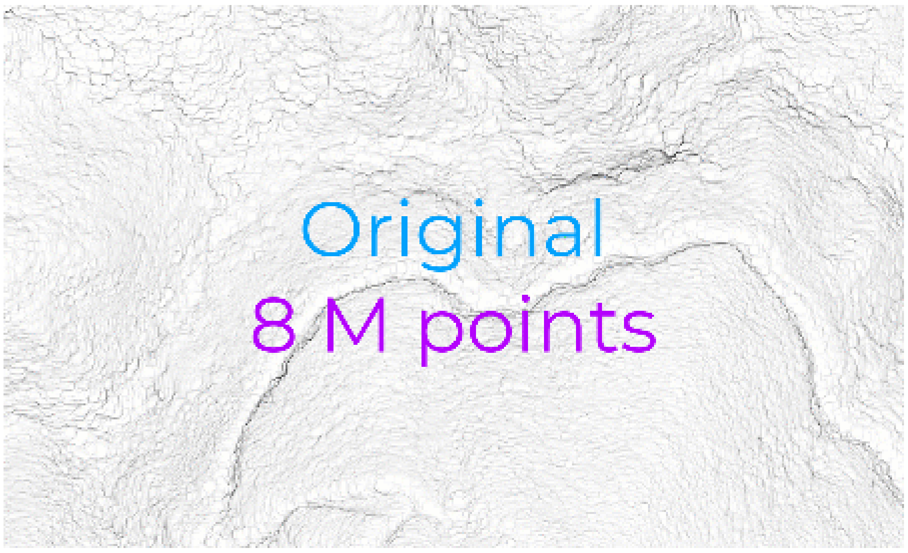


Illustration of the voxel grid sampling methodology. © F. Poux

We will work in two sub-steps.

1. First, we create a grid structure over the points. For this, we actually want to initially compute the bounding box of the point cloud (i.e. the box dimensions that englobe all the points). Then, we can discretize the bounding box into small cubic grids: the voxels. These are obtained by setting the length, width and height of the voxel (which is equal), but it could also be set by giving the number of desired voxels in the three directions of the bounding box.

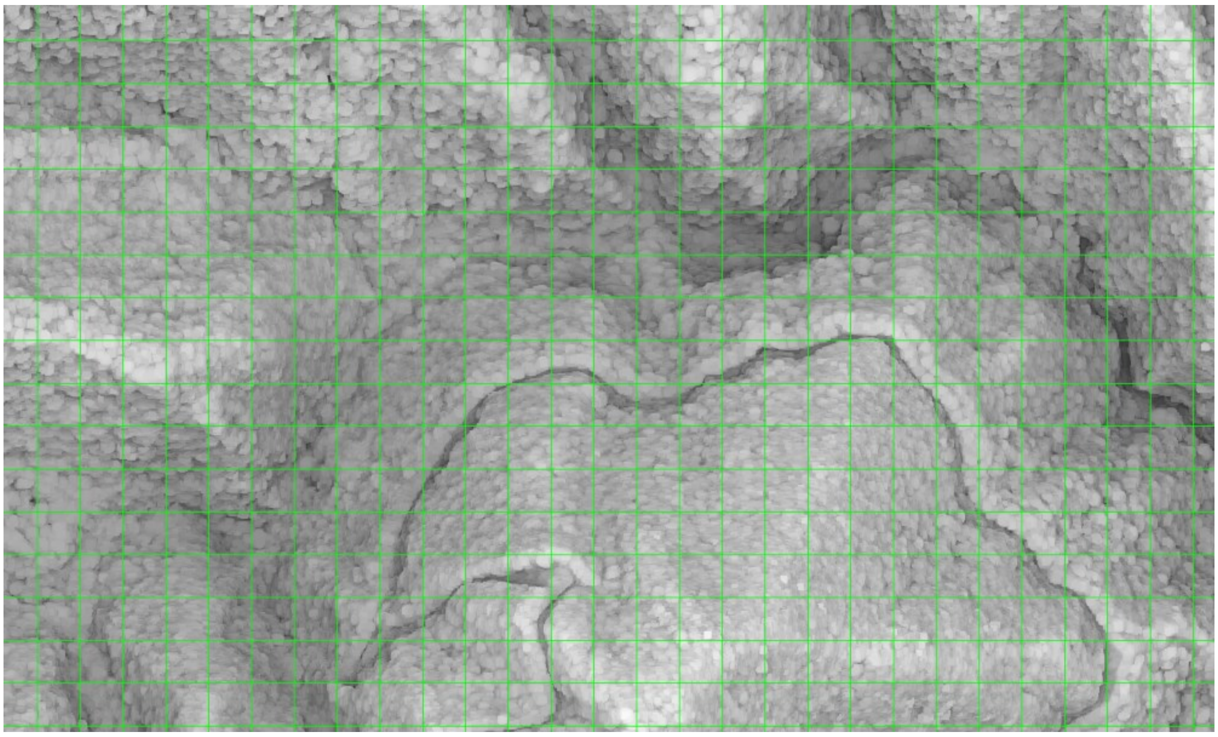
```
voxel_size=6
nb_vox=np.ceil((np.max(points, axis=0) - np.min(points,
axis=0))/voxel_size)
```

🤓 ***Note**:* You can see the little axis=0 that is actually fundamental if you want to be sure you apply the max method "per column". The ceil then will make sure to keep the ceiling of the difference (element-wise), and thus, when divided by the voxel_size, it returns the number of empty voxels in each direction. With a cubic size of 6 m, we get 254 voxels along X, 154 voxels along Y and 51 along Z: 1 994 916 empty voxels.

2. For each small voxel, we test if it contains one or more points. If it does, we keep it, and we take note of the points indexes that we will have to link to each voxel.

```
non_empty_voxel_keys, inverse, nb_pts_per_voxel = np.unique(((points
- np.min(points, axis=0)) // voxel_size).astype(int), axis=0,
return_inverse=True, return_counts=True)
idx_pts_vox_sorted=np.argsort(inverse)
```

🤓 ***Note**:* We want to work with indices rather than coordinates for simplicity and efficiency. The little script above is a super-compact way to return the "designation" of each non-empty voxel. On top, we want to access the points that are linked to each non_empty_voxel through idx_pts_vox_sorted, and how many there are (nb_pts_per_voxel). This is done by first looking out unique values based on the integer "indices" gathered for each point. The argsort method is actually returning the index of the points that we can later link to the voxel index.


The grid obtained over the point cloud data. © F. Poux

3. Finally, we compute the representant of the voxel. I will illustrate this for both the barycenter (grid_barycenter) and the closest point to the barycenter (grid_candidate_center).

💡 ***Hint:*** *The use of python dictionaries to keep the points in each voxel is my recommendation. This sparse structure is more adapted than full arrays which will use all your memory on bigger point clouds. A dictionary cannot take a [i, j, k] vector of coordinates as key if it is a list, but converting it to a tuple (i, j, k) will make it work.*

- We initialise self-explanatory variables of which a counter last_seen:

```
voxel_grid={}
grid_barycenter,grid_candidate_center=[],[]
last_seen=0
```

- We create a loop that will iterate over each non-empty voxel, while allowing to work with both the index idx of the array, and the value vox, which is actually the *[i, j, k]* of the voxel.

```
for idx,vox in enumerate(non_empty_voxel_keys):
```

- Then (don't forget to indent) we feed the loop with a way to complete the voxel_grid dictionary with contained points.

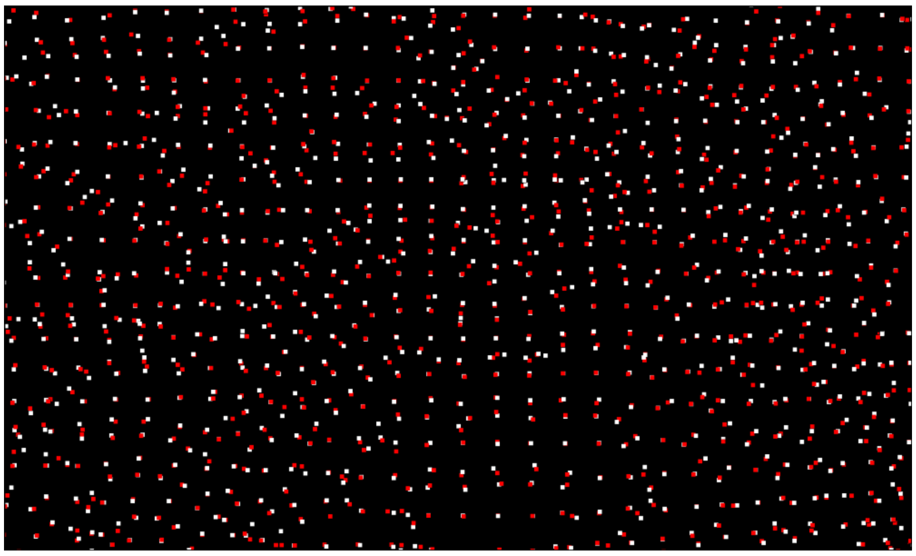
```
voxel_grid[tuple(vox)]= points[idx_pts_vox_sorted[
last_seen:last_seen+nb_pts_per_voxel[idx]]]
```

- Still in the loop, you can now pick/compute the representative of the voxel. It can be the barycenter that you append to the list of all barycenters:

```
grid_barycenter.append(np.mean(voxel_grid[tuple(vox)],axis=0))
```

- Or it can be the closest point to the barycenter (uses Euclidean distances):

```
grid_candidate_center.append(
voxel_grid[tuple(vox)][np.linalg.norm(voxel_grid[tuple(vox)] -
np.mean(voxel_grid[tuple(vox)],axis=0),axis=1).argmin()])
```



Notice the difference of results between the voxel subsampling keeping the barycenter (white points) vs the closest point to it (red points). © F. Poux

- Finally, don't forget to update your counter, to make sure the selection in the array of points is correct:

```
last_seen+=nb_pts_per_voxel[idx]
```

🤓 ***Note****:* Most of my M.Sc. students will accomplish the task with a bunch of imbricated "for" or "while" loop. It does work, but it is not the most efficient. You have to know Python is not very optimized with loops. Thus, when processing point clouds (which are often massive), you should aim at a minimal amount of loops, and a maximum amount of "vectorization". With NumPy, this is by "broadcasting", a mean of vectorizing array operations so that looping occurs in C instead of Python (more efficient). Take the time to digest what I do in this third step (especially the details of playing with indexes and voxels), or check out the Google Colab script for more in-depth information.

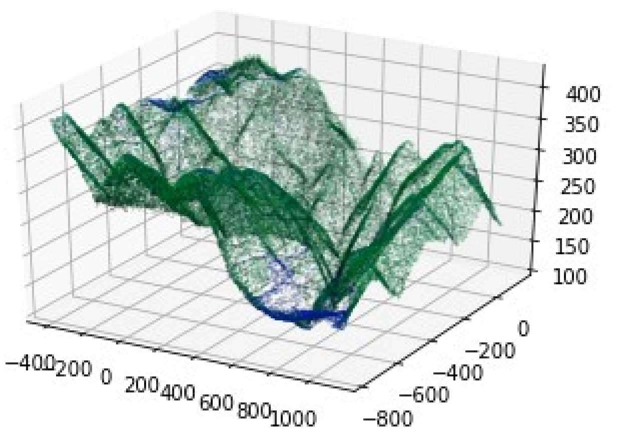
This voxel sampling strategy is usually very efficient, relatively uniform, and useful for downward processes (but this extend the scope of the current tutorial). However, you should know that while the point spacing can be controlled by the size of the grid, we cannot "accurately" control the number of sampling points.

Step 4: Visualize your results

To simply visualize in-line your results (or within Python), you can use the matplotlib library, with its 3D toolkit (see the previous article for understanding what happens under the hood). Run the following command, illustrated over the decimated point cloud :

```
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

decimated_colors = colors[::factor]

ax = plt.axes(projection='3d')

ax.scatter(decimated_points[:,0], decimated_points[:,1],
decimated_points[:,2], c = decimated_colors/65535, s=0.01)

plt.show()
```

🤓 ***Note:*** Looking at the number of possible points, **I would not recommend** in-line visualisation with classical libraries such as matplotlib if your subsampled results exceed the million mark.



The decimated point cloud visualized in MatplotLib within Python. © F. Poux

Published in Towards Data Science

In the very likely event your point cloud is too heavy for visualizing this way, you can export the data in an eatable file format for your software of choice. To get an ASCII file, you can use the command:

```
np.savetxt(output_path+dataname+"_voxel-best_point_%s.poux" % (voxel_size), grid_candidate_center, delimiter=";", fmt="%s")
```

🤓 ***Note**:* A “;” delimited ASCII file is created, ending with .poux 🤭. The “fmt” command is to make sure the writing is most standard, for example as a string.

💡 ***Hint:** If you also want to make operations to retrieve the colour of voxels representatives, be careful with the NumPy dtype of the sum of colours. The colour type “uint16” can take values from 0 to* 65535. *Change the type when summing and go to uint16 (or uint8) after the final division.*

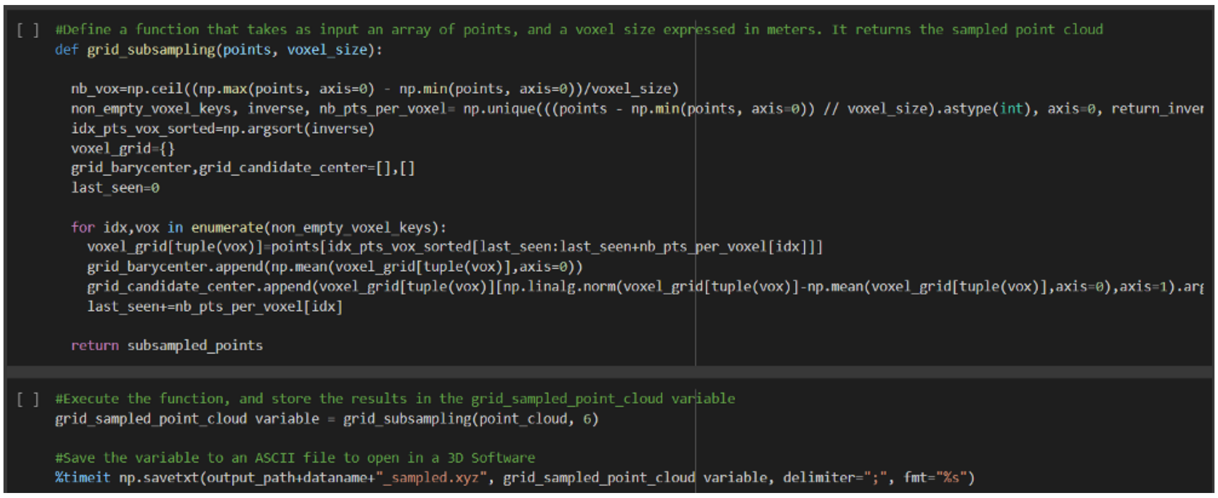
Step 5: Automation in 3D processing

Now that you addressed steps 1 to 4, it is time to create functions and put them together in an automated fashion 🤘. Basically, we want (a) to load the data and libraries, (b) set parameters value, (c) declare functions, (d) call them when needed, (e) return some kind of results. This can be to show in-line the results and/or to export a sampled point cloud file to be used in your 3D software, outside of Python.

You already know how to do a, b and e, so let us focus on b and c 🎯. To create a function, you can just follow the provided template below:

```
def cloud_decimation(points, factor):
  # YOUR CODE TO EXECUTE
  return decimated_points
```

🤓 ***Note**:* The function created is called cloud_decimation, and eats two arguments which are points and factor. It will execute the desired code written inside and return the variable decimated_points when it is called. and to call a function, nothing more straightforward: simply write cloud_decimation(point_cloud, 6) (the same way you would use the function print(), but here you have two arguments to fill by the values/variables that you want to pass to the function).



```python
[ ] #Define a function that takes as input an array of points, and a voxel size expressed in meters. It returns the sampled point cloud
    def grid_subsampling(points, voxel_size):

      nb_vox=np.ceil((np.max(points, axis=0) - np.min(points, axis=0))/voxel_size)
      non_empty_voxel_keys, inverse, nb_pts_per_voxel= np.unique(((points - np.min(points, axis=0)) // voxel_size).astype(int), axis=0, return_inver
      idx_pts_vox_sorted=np.argsort(inverse)
      voxel_grid={}
      grid_barycenter,grid_candidate_center=[],[]
      last_seen=0

      for idx,vox in enumerate(non_empty_voxel_keys):
        voxel_grid[tuple(vox)]=points[idx_pts_vox_sorted[last_seen:last_seen+nb_pts_per_voxel[idx]]]
        grid_barycenter.append(np.mean(voxel_grid[tuple(vox)],axis=0))
        grid_candidate_center.append(voxel_grid[tuple(vox)][np.linalg.norm(voxel_grid[tuple(vox)]-np.mean(voxel_grid[tuple(vox)],axis=0),axis=1).arg
        last_seen+=nb_pts_per_voxel[idx]

      return subsampled_points
```

```python
[ ] #Execute the function, and store the results in the grid_sampled_point_cloud variable
    grid_sampled_point_cloud variable = grid_subsampling(point_cloud, 6)

    #Save the variable to an ASCII file to open in a 3D Software
    %timeit np.savetxt(output_path+dataname+"_sampled.xyz", grid_sampled_point_cloud variable, delimiter=";", fmt="%s")
```

A function for voxel grid sampling of a point cloud. Get the code from the Google Colab script. © F. Poux

By creating a suite of functions, you can then use your script directly, just changing the set parameters at the beginning.

The full code is accessible here: [Google Colab notebook](#).

Additionally, you can check out the follow-up article if you want to extend your capabilities using the library Open3D, and learn specific commands related to 3D point clouds and 3D mesh processing.

Conclusion

You just learned how to import, sub-sample, export and visualize a point cloud composed of millions of points, with different strategies! Well done! But the path does not end here, and future posts will dive deeper in point cloud spatial analysis, file formats, data structures, visualization, animation and meshing. We will especially look into how to manage big point cloud data as defined in the article below.

Other advanced sampling methods for point cloud exist. For example, you could follow a uniform sampling method such as the Farthest Point method, a more advanced geometric sampling [1] or even semantic sampling. Also, the voxelisation algorithm given here can be used for advanced processing such as 3D semantic modelling [2] or semantic segmentation, as shown in [3].

Published in Towards Data Science

References

1. Poux, F. The Smart Point Cloud: Structuring 3D intelligent point data, Liège, 2019.
2. Poux, F.; Valembois, Q.; Mattes, C.; Kobbelt, L.; Billen, R. Initial User-Centered Design of a Virtual Reality Heritage System: Applications for Digital Tourism. *Remote Sens.* **2020**, *12*, 2583, doi:10.3390/rs12162583.
3. Poux, F.; Neuville, R.; Nys, G.-A.; Billen, R. 3D Point Cloud Semantic Modelling: Integrated Framework for Indoor Spaces and Furniture. *Remote Sens.* **2018**, *10*, 1412, doi:10.3390/rs10091412.
4. Billen, R.; Jonlet, B.; Luczfalvy Jancsó, A.; Neuville, R.; Nys, G.-A.; Poux, F.; Van Ruymbeke, M.; Piavaux, M.; Hallot, P. La transition numérique dans le domaine du patrimoine bâti: un retour d'expériences. *Bull. la Comm. R. des Monum. sites Fouill. 30* **2018**, 119–148.
5. Poux, F.; Billen, R. Voxel-based 3D Point Cloud Semantic Segmentation: Unsupervised geometric and relationship featuring vs deep learning methods. *ISPRS Int. J. Geo-Information* **2019**, *8*, doi:10.3390/ijgi8050213.
6. Kharroubi, A.; Hajji, R.; Billen, R.; Poux, F. Classification And Integration Of Massive 3d Points Clouds In A Virtual Reality (VR) Environment. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2019**, *42*, 165–171, doi:10.5194/isprs-archives-XLII-2-W17-165-2019.
7. Bassier, M.; Vergauwen, M.; Poux, F. Point Cloud vs. Mesh Features for Building Interior Classification. *Remote Sens.* **2020**, *12*, 2224, doi:10.3390/rs12142224.
8. Poux, F.; Ponciano, J. J. Self-Learning Ontology For Instance Segmentation Of 3d Indoor Point Cloud. In *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*; ISPRS, Ed.; Copernicus Publications: Nice, 2020; Vol. XLIII, pp. 309–316.
9. Poux, F.; Mattes, C.; Kobbelt, L. Unsupervised segmentation of indoor 3D point cloud: application to object-based classification. In *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*; 2020; Vol. XLIV–4, pp. 111–118.
10. Poux, F.; Billen, R.; Kaspryzk, J.-P.; Lefebvre, P.-H.; Hallot, P. A

Built Heritage Information System Based on Point Cloud Data: HIS-PC. *ISPRS Int. J. Geo-Information* **2020**, *9*, 588, doi:10.3390/ijgi9100588.

11. Poux, F.; Billen, R. A Smart Point Cloud Infrastructure for intelligent environments. In *Laser scanning: an emerging technology in structural engineering*; Lindenbergh, R., Belen, R., Eds.; ISPRS Book Series; Taylor & Francis Group/CRC Press: London, United States, 2019; pp. 127–149 ISBN in generation.
12. Tabkha, A.; Hajji, R.; Billen, R.; Poux, F. Semantic Enrichment Of Point Cloud By Automatic Extraction And Enhancement Of 360° Panoramas. *ISPRS - Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2019**, *XLII-2/W17*, 355–362, doi:10.5194/isprs-archives-XLII-2-W17-355-2019.
13. Poux, F.; Neuville, R.; Hallot, P.; Van Wersch, L.; Jancsó, A. L.; Billen, R. Digital investigations of an archaeological smart point cloud: A real time web-based platform to manage the visualisation of semantical queries. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. - ISPRS Arch.* **2017**, *XLII-5/W1*, 581–588, doi:10.5194/isprs-Archives-XLII-5-W1-581-2017.
14. Poux, F.; Hallot, P.; Jonlet, B.; Carre, C.; Billen, R. Segmentation semi-automatique pour le traitement de données 3D denses: application au patrimoine architectural. *XYZ* **2014**, *141*, 69–75.
15. Novel, C.; Keriven, R.; Poux, F.; Graindorge, P. Comparing Aerial Photogrammetry and 3D Laser Scanning Methods for Creating 3D Models of Complex Objects. In *Capturing Reality Forum*; Bentley Systems: Salzburg, 2015; p. 15.