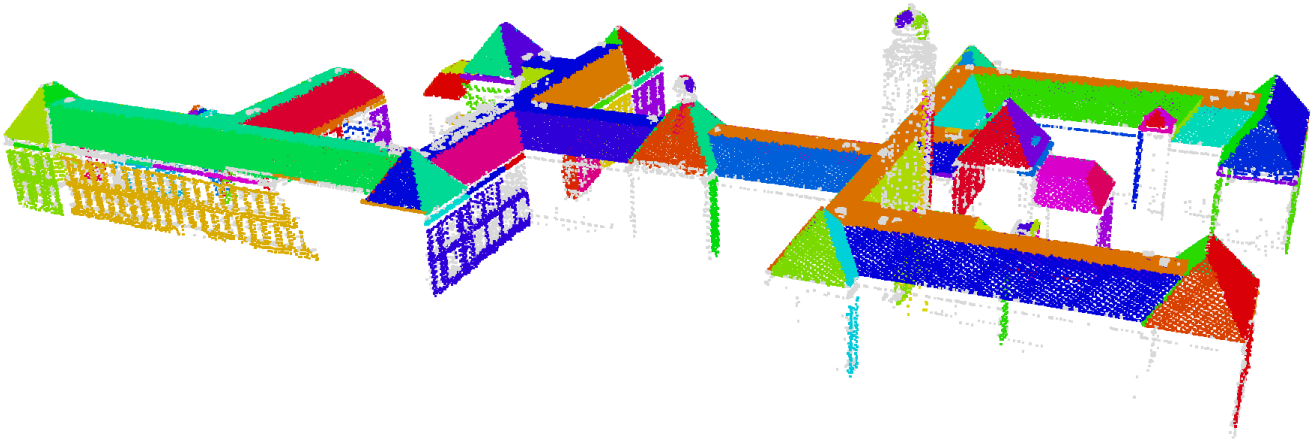
Homework 04: Plane detection



> *Deadline is **2019-02-04 13:00**.*

> *Late submission? 10% will be removed for each day that you are late.*

> *You're allowed for this assignment to work in a group of 2 (and thus submit only one report for both of you). If you prefer to work alone it's also fine (but we don't recommend it).*

Overview

In this assignment you will implement a region growing segmentation algorithm for plane detection in a point cloud. In short, you will have to

1. read the non-ground points from a LAS file, optionally with thinning,
2. perform region-growing plane segmentation, and
3. store the result in a PLY file.

In addition to the code you'll have to write down and submit the answers to a few questions about your results.

What you are given to start

code_hw04.zip

We give you the skeleton of the Python program:

1. `geo1015_hw04.py` is the `main()`. You are *not* allowed to modify this file!
2. `mycode_hw04.py` is where *all* your code should go. You will need to implement the function `detect_planes()`. You are of course allowed to add any other functions you want.
3. `params.json`: a JSON file with the parameters as described below.
4. `urban.las`: input point cloud.
5. `example_segmentation.ply`: an example output file.
6. `view_segmentation.glsl`: a custom shader file for use with the Displaz point cloud viewer as explained below. It allows you to easily visualise the segments of the PLY output.

These are the parameters given in `params.json`:

- `input-file`: the input LAS file
- `output-file`: the output PLY file
- `thinning-factor`: thinning factor used to thin the input points prior to giving them to the segmentation algorithm. A factor of 1x means no thinning, 2x means remove 1/2 of the points, 10x means remove 9/10 of the points, etc.
- `minimal-segment-count`: the minimal number of points in a segment.
- `epsilon`: distance threshold to be used during the region growing. It is the distance between a candidate point and the plane of the growing region.
- `neighbourhood-type`: specifies the type of neighbourhood search to be used, valid values are `knn` and `radius`.
- `k`: size of the knn neighbourhood searches.
- `r`: radius for he fixed-radius neighbourhood searches.

You may assume that `params.json` doesn't need to be validated, and that the parameters are also valid.

The region growing algorithm

The aim of this assignment is to implement the region growing algorithm (also called surface growing) that was introduced in Lesson 13. In summary, to grow a new region we need to:

```
Select a seed point s
Label s with a new region id i
Add s to a stack S

while S is not empty:
  Pop a point p from S
```

```
for each q in neighbours(p):
  if q is unsegmented:
    if valid_candidate(q):
      Label q to be part of region i
      Add q to S
```

This needs to be repeated until no more suitable seed points can be found.

You will have to use the algorithm to do plane detection. This means you need to define a suitable `valid_candidate()` function that checks if the candidate point `q` fits with the plane `P` of the current region, i.e. if the distance from `q` to `P` is smaller than the parameter `epsilon`.

The `neighbours()` functions is either a knn search or a fixed-radius search depending on the `neighbourhood-type` parameter.

Also notice that if the number of points in a fully grown region is smaller than `minimal-segment-count`, it needs to be discarded. This means that all its point need to be set to unsegmented again.

Output specification

You will have to assign each segment a unique identifier. This is an integer with a value of `1` or higher and it should be outputted for each point. In the PLY output this field should be labelled `segment_id`. The header of the PLY file should thus look like this:

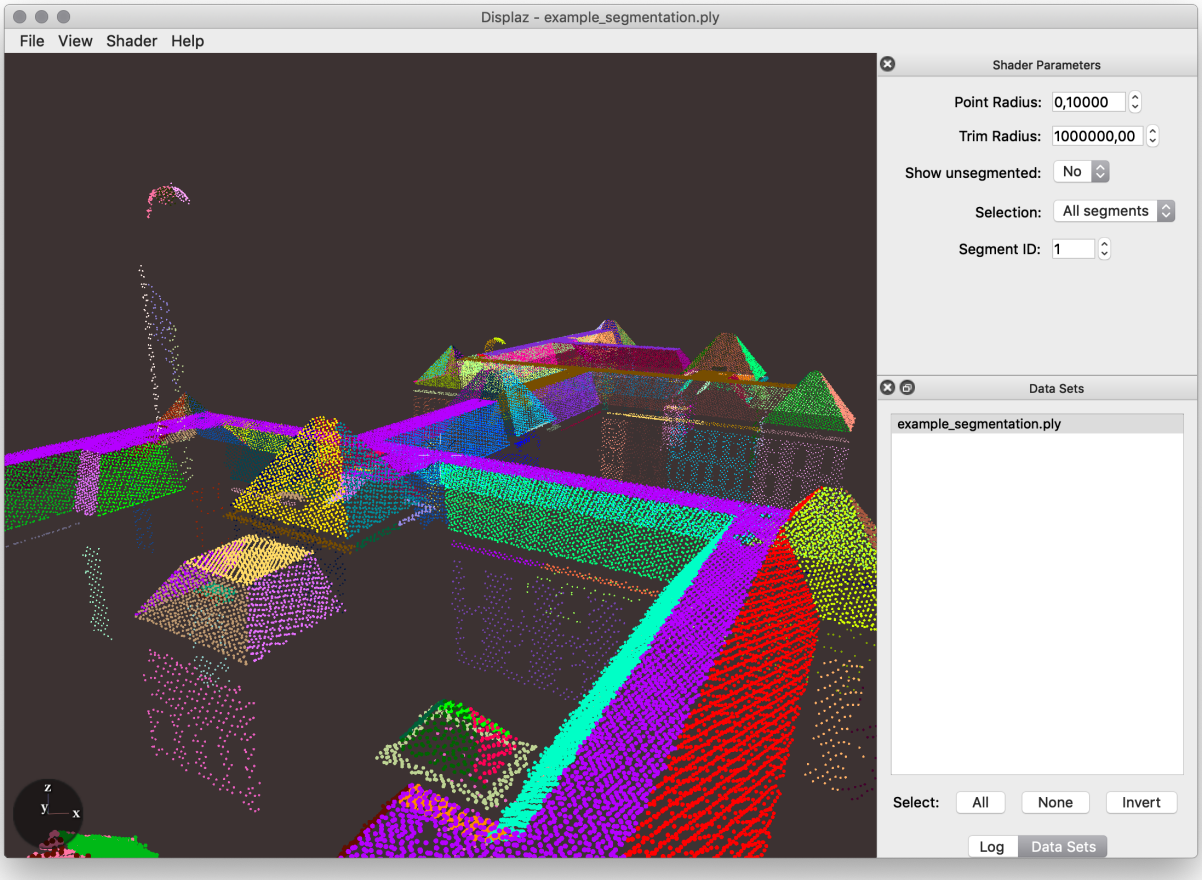
```
ply
format ascii 1.0
element vertex <total number of points>
property float x
property float y
property float z
property uint segment_id
end_header
```

Points that could not be assigned to a segment should get a value of `0` for the `segment_id`.

See the file `example_segmentation.ply` for a complete example.

Visualising your results

We recommend to use the Displaz with the view_segmentation.ply shader viewer together with the `view_segmentation.glsl` shader that we give you. This shader gives each segment a different color and allows you to show only the points that belong to a particular segment. You can load the shader through the menu `Shader > Open`.



Notice also that you can pick points with `shift` + `left mousebutton`. If you pick a point its attributes (ie. the ones that you provide in the PLY file) are printed in the `log` panel.

Good to know

- You are only allowed to import modules from the Python standard library (anything you installed through `pip` is thus not allowed, except *laspy*, *scipy* and *numpy*).
- A function that fits a plane to a set of points is given in the file `mycode_hw04.py`.
- It is a good idea to use a kd-tree to perform the nearest neighbour queries. Notice that the `cKDTree` in `scipy.spatial` is a faster variant of the regular `KDTree` in scipy.
- Use the laspy documentation.
- Use only the non-ground points of the input point cloud.
- If you want to speed up testing, but you don't want to increase the thinning factor, you can also create a small subset of the `urban.las` point cloud (eg. using the `lasclip` tool from LAStools).
- The CRS of the input will always be in meters.

What to submit and how to submit it

You have to submit a **zip-archive** with these 4 files:

1. The Python file `my_code_hw04.py`, where your names are clearly identified at the top of the file.

2. The file `params.json` fine tuned by you for the best possible result for the `urban.las` dataset (set the thinning factor to 2). The best possible result has the highest quality plane fit for each segment and as few unsegmented points as possible.
3. An output PLY file `segments.ply` generated with the parameters from your `params.json`.
4. A PDF file (no Word file please) with your answers to the questions below.

Questions for you to answer

1. Does it matter how you pick your seed points, ie. is there a better way than to randomly pick the seeds?
2. During region growing you can determine the neighbours of a point using 1) a fixed radius search and 2) its k nearest neighbours. What works best?
3. How does the thinning factor affect the segmentation result?
4. Do you get better results if you re-estimate the plane to all the points in the region as it is growing?
5. Do you think you can improve your results if different/additional criteria are used for the `valid_candidate()` function?
6. Why are the parameters that you give in `params.json` the best?
7. Did you make any other significant observations that helped you to improve your segmentation result?

Clearly explain each answer and perform experiments and show examples to prove and illustrate your answers. Don't forget to label and reference your figures/tables.

Do *not* submit your assignment by email, but upload the requested files zipped to this Dropbox file request page. Make sure that you put the full name of one of the members of the team (only one is sufficient).

You'll get a confirmation email when everything has been successfully uploaded, keep it in case things go wrong.

[last updated: 2019-01-13]
