CONFIDENTIAL

3dproc v2.1

Custom Unit Separation

V1.00

Guide

The following chapter provides a couple of useful guidelines, examples and explanations about the mechanics of using 3dproc v2.1 as a standalone tool.

Manual optimisation of the processing

The example JSON files (that appear in the following section) can be found in the repo: hg/monorepo/phena/3dproc-v2/config_exmaples/*.json

Optimising preprocessing

Preprocessing is one of the most vital parts for the processing chain, therefore it is recommended to optimise the configuration parameters to achieve best performance. Any noise / undesired material that is left in the point cloud after preprocessing will have an impact on the resulting plant parameters and will significantly reduce data quality.

Recommended steps:

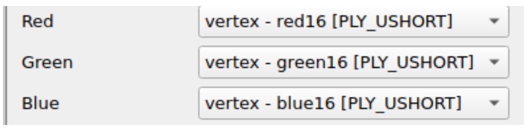
1. Disable cropping and colour filter
2. Define preprocessing-only chain for quick processing times
3. Find optimal transform angles
4. Identify scan volume and define crop boundaries
5. Enable cropping
6. Setup colour filter and enable if needed

The following config covers steps 1 & 2:

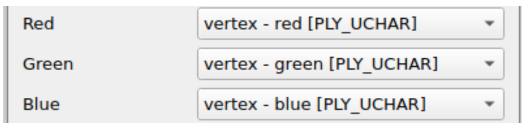
```
{"chain":
   {
      "preprocessing":
      {
         "enabled":true, "scanner_file_output":true,
         "steps": {"transform":true, "crop":false, "color_filter":false}
      },
      "pooling":{"enabled":false},
      "unit_separation":{"enabled":false},
      "parameter_extraction":{"enabled":false}
   }
}
```

The config above defines a chain that will generate an output ply file called <filename>_scannerfile_<index>.ply for every input ply file supplied, where <filename> is the basename provided with the -o option and <index> denotes the order of the input files starting from 0.

Note that Cloudcompare will try to load the RGB scalers by regex matching “red”, “green” and “blue” and since the 16-bit values are preceding the 8-bit ones, the former will be loaded by default. Cloudcompare visualisations do not work with 16-bit values. If you load them anyway you will end up with rainbow colours. If you need RGB visuals, switch the scalars to the 8-bit values.

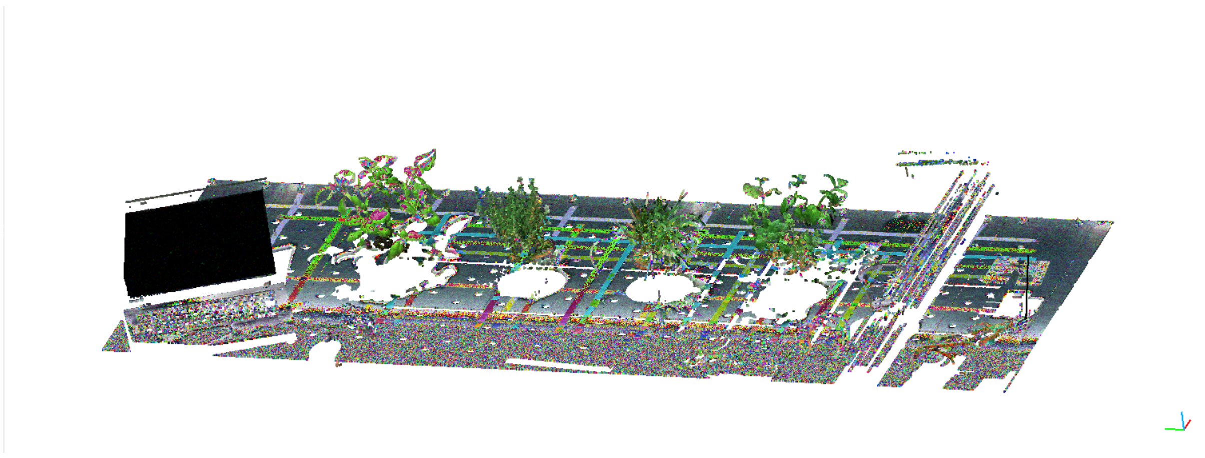


Red	vertex - red16 [PLY_USHORT]
Green	vertex - green16 [PLY_USHORT]
Blue	vertex - blue16 [PLY_USHORT]



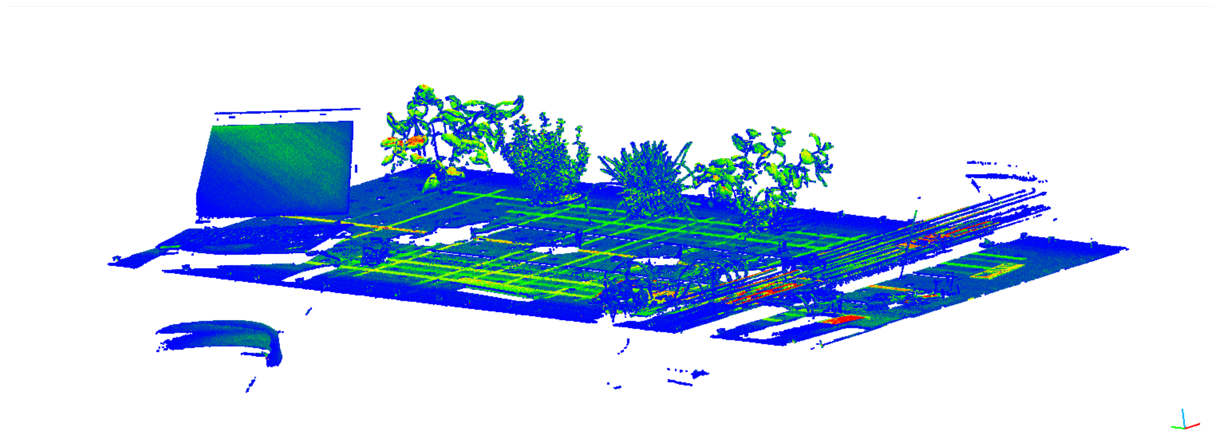
Red	vertex - red [PLY_UCHAR]
Green	vertex - green [PLY_UCHAR]
Blue	vertex - blue [PLY_UCHAR]

Example of (wrong) 16-bit value colours:

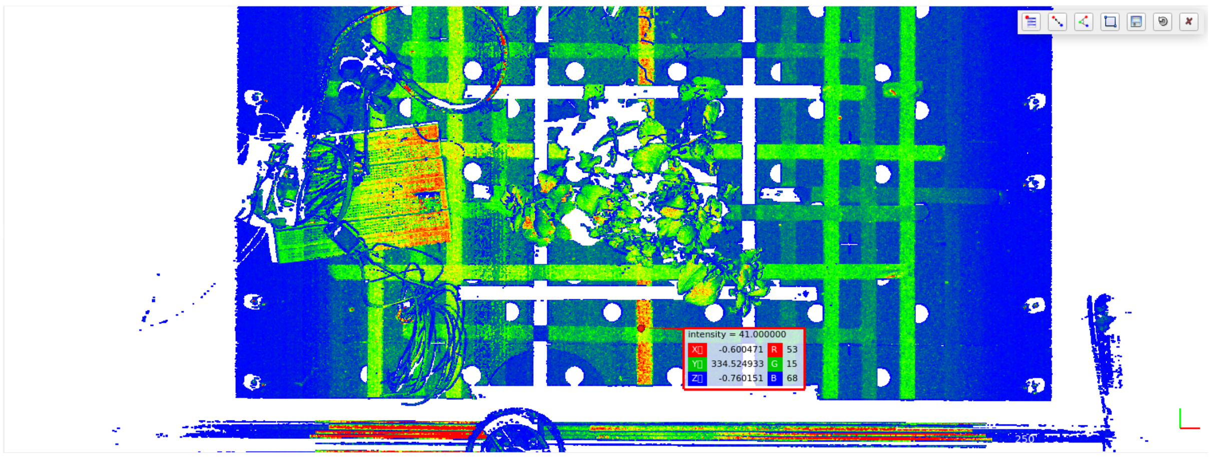


.

Using a dual scan setup it is convenient to use the config given above. The output scanner files can be used as references to check the displacement / rotation between the two scans. Once the correct parameters are identified it is required to replace the transform configuration (step 3) in the header of the input files. Since transformation is scan specific it can NOT be modified using the input json. Given correct transform settings the scanner files should be overlapping and aligned as well as humanly possible:



Before enabling the cropping, use Cloudcompare’s point selection tool to identify the boundaries of the scan volume. To define the boundary the min and max values are needed for each of the dimensions (x,y,z). These crop settings can already be defined in the json without having an effect before enabling the crop.



```
intensity = 41.000000
X   -0.600471   R 53
Y  334.524933   G 15
Z   -0.760151   B 68
```

Note that at this point only the preprocessing part of the pipeline is used so the field configurations in the input ply headers will have no effect. Also the cropping never affects pot_height or the height related plant parameters, the sole purpose of the cropping is to get rid of large parts of the point cloud that are of no interest (e.g. table, wall). In any case this is recommended, because the processing times will be much lower with the data reduction properly set up.

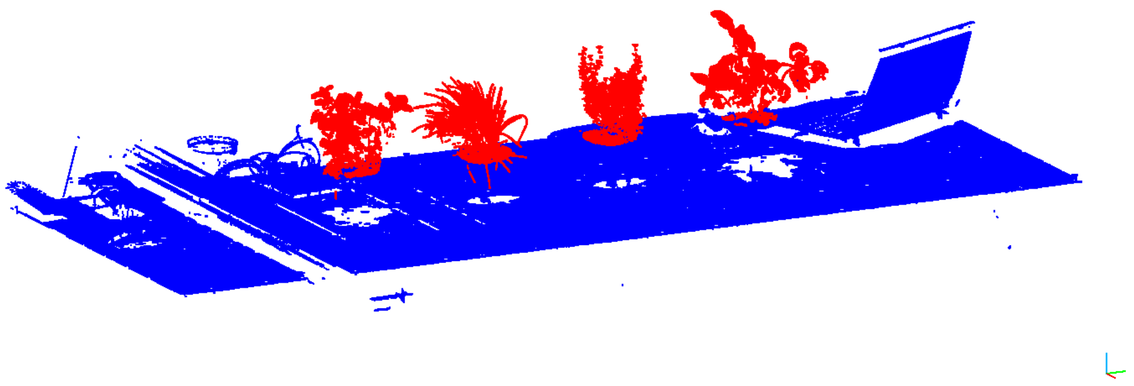
Let’s add the the crop settings to our json:

```
{   "chain":
    {
        "preprocessing":
        {
            "enabled":true, "scanner_file_output":true,
            "steps": {"transform":true, "crop":true, "color_filter":false}
        },
        "pooling":{"enabled":false},
        "unit_separation":{"enabled":false},
        "parameter_extraction":{"enabled":false}
    },

    "crop":{"min_x": -168,"max_x": 163,
            "min_y": 334,"max_y": 1505,
            "min_z": 45,"max_z": 500
    }
}
```

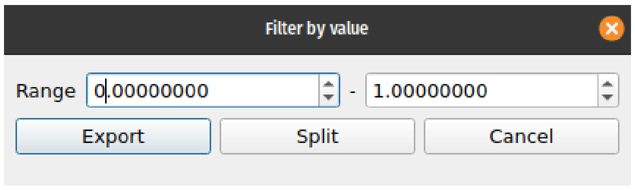
Once cropping is enabled, the result can be visualised. Upon loading the resulting scanner file(s), add “proc_tag” as a scalar value. Note that vertices are never deleted from the scanner file, even if the vertices are filtered.

Processing tag (proc_tag) identifiers:

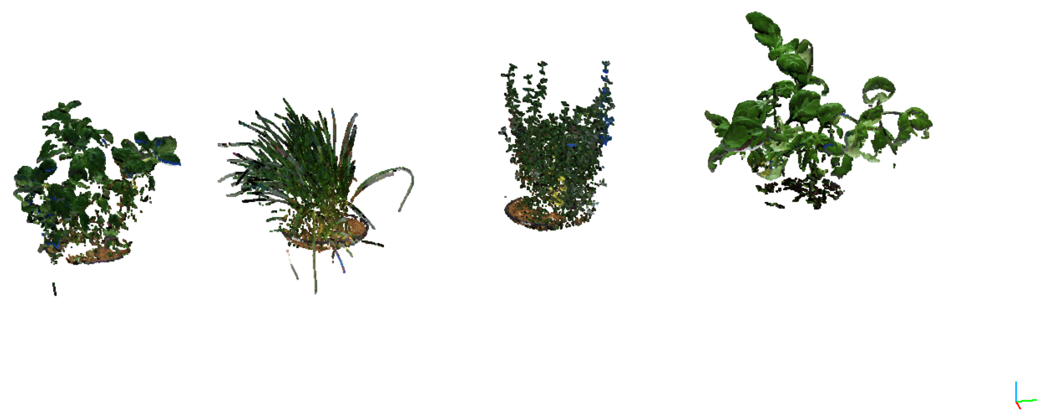
- Unprocessed (0): Indicates errors, for some reason the vertex was not properly processed.
- Preprocessed (1): Indicates that the vertex is considered valid after preprocessing and will be used in further steps of the processing pipeline.
- Cropped (-1): The vertex lies outside the scanning volume.
- Filtered (-2): The vertex does not satisfy the colour filter conditions.



If you need to (for whatever strange reason) visualise the cropped part separately in RGB, use Cloudcompare, select the ply files and make sure that "proc_tag" is the active scalar field. click edit → scalar fields → filter by value and apply a split filter from 0:

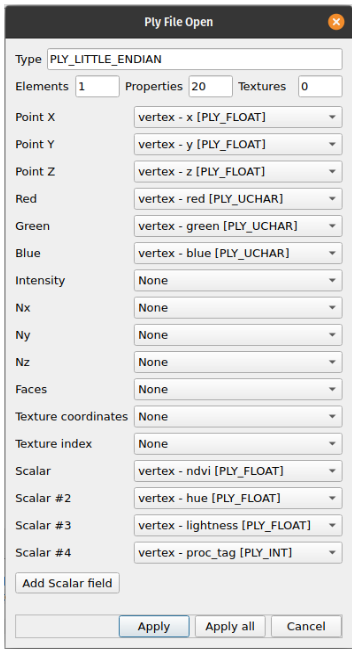


Range 0.00000000 - 1.00000000



Note, that the cropping is not perfect, the soil and the pot are still visible. That is not a problem and in most cases it is recommended to set up crop settings so that no plant parts are removed from the scans. The parts belonging to the pot may be removed by setting a higher field origin (while keeping the same cropping settings) or by using the colour filter. NDVI, Hue and Lightness generally make good soil filters. Let’s add those indices as vertex properties to our ply file:

```
{
   "chain":
   {
      "preprocessing":
      {
         "enabled":true, "scanner_file_output":true,
         "steps": {"transform":true, "crop":true, "color_filter":false}
      },
      "pooling":{"enabled":false},
      "unit_separation":{"enabled":false},
      "parameter_extraction":{"enabled":false}
   },

   "crop":{"min_x": -168,"max_x": 163,
         "min_y": 334,"max_y": 1505,
         "min_z": 45,"max_z": 500
   },
   "custom_vertex_properties": ["ndvi", “hue”, "lightness"]
}
```



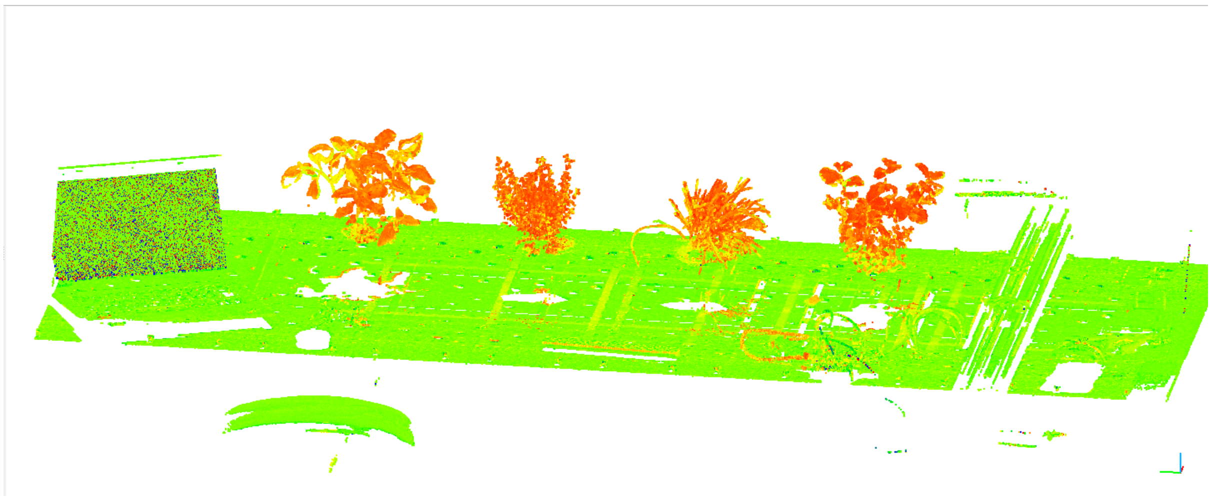
Ply File Open

Type PLY_LITTLE_ENDIAN

Elements 1 Properties 20 Textures 0

Point X	vertex - x [PLY_FLOAT]
Point Y	vertex - y [PLY_FLOAT]
Point Z	vertex - z [PLY_FLOAT]
Red	vertex - red [PLY_UCHAR]
Green	vertex - green [PLY_UCHAR]
Blue	vertex - blue [PLY_UCHAR]
Intensity	None
Nx	None
Ny	None
Nz	None
Faces	None
Texture coordinates	None
Texture index	None
Scalar	vertex - ndvi [PLY_FLOAT]
Scalar #2	vertex - hue [PLY_FLOAT]
Scalar #3	vertex - lightness [PLY_FLOAT]
Scalar #4	vertex - proc_tag [PLY_INT]

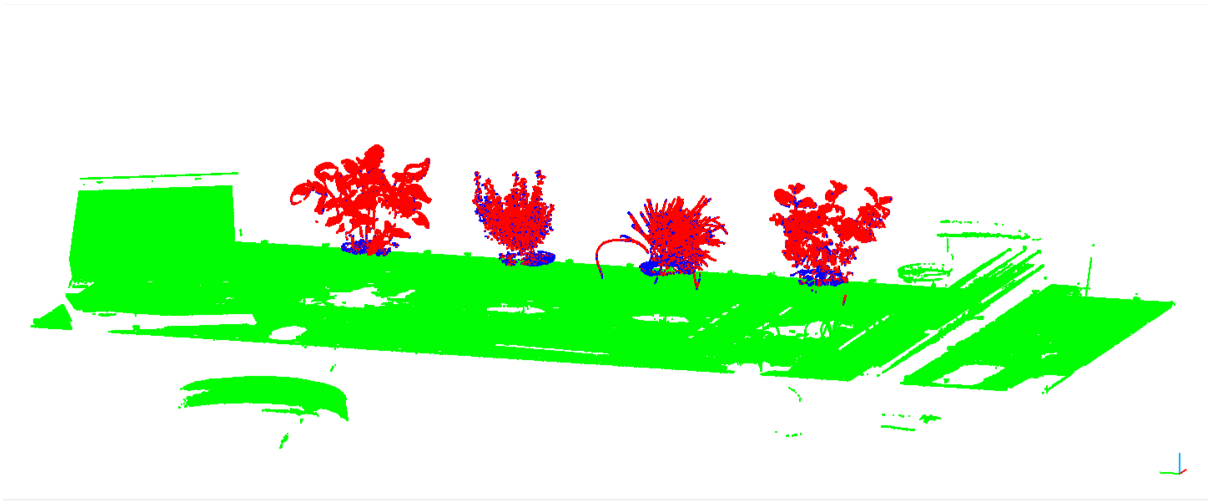
If the properties are added to the json then the index can be visualised and it can be easily determined if a certain index is suitable to be used as a filter.



Let’s set an NDVI filter from 0 to remove dead objects and a Hue filter around green (60,180):

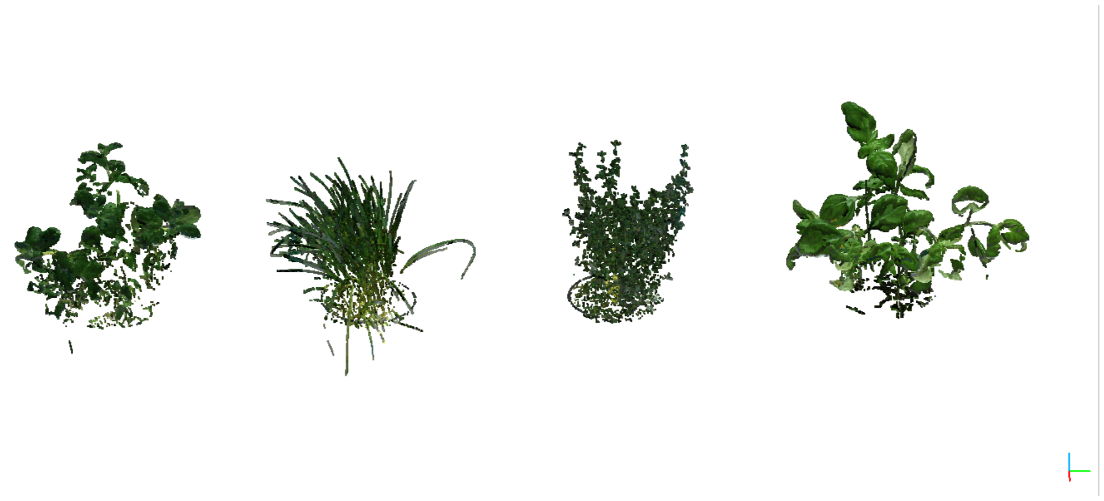
```
{
   "chain":
   {
      "preprocessing":
      {
         "enabled":true, "scanner_file_output":true,
         "steps": {"transform":true, "crop":true, "color_filter":true}
      },
      "pooling":{"enabled":false},
      "unit_separation":{"enabled":false},
      "parameter_extraction":{"enabled":false}
   },

   "crop":{"min_x": -168,"max_x": 163,
         "min_y": 334,"max_y": 1505,
         "min_z": 45,"max_z": 500
   },
   "custom_vertex_properties": ["ndvi", "lightness", "hue"],
   "color_filter":{"ndvi":[[0,1]], “hue”:[[60,180]]}
}
```

Visualising the processing tags shows the annotation of the vertices:



Green : cropped
Blue: colour filtered
Red: processed

As you can see most of the soil and pot is removed and most of the plant material is preserved in the scan.



With more specific colour filtering it is possible to achieve even better results. On the other hand it is NOT recommended to set an NDVI filter above 0 or set other very specific colour ranges because it might remove otherwise desirable variance from the data. The remaining noise may be removed by the segment filter later on or can be disregarded in the calculations by setting a higher field origin. Note that at this point even the overhang leaves are preserved in the scan.

The visualisation above is a good example of how the scans are expected to look like after preprocessing.

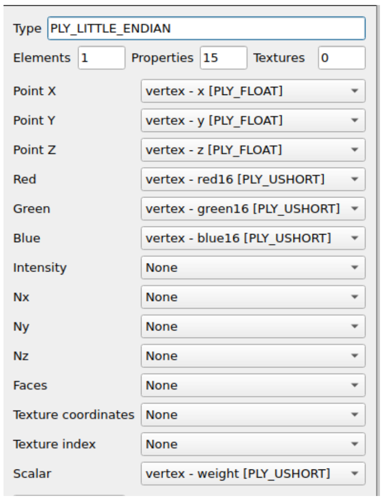
Optimising scan fusion and selecting optimal resolution

There are two ways of running the data pooling module. Either, the same raw input files can be used as in the previous section with the pooling configs appended to the JSON or the output scanner files from the previous step can be used as input with everything disabled in the JSON apart from the pooling configs. The latter is recommended if the preprocessing results are acceptable and it is unlikely that its configurations need to be changed.

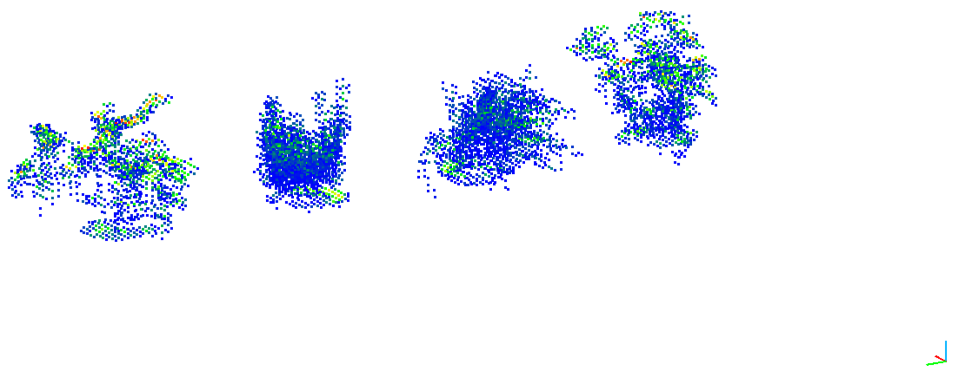
Currently, the only tunable pooling parameter is the voxel resolution. The vertices (voxels) of the resulting voxelized file will contain a weight property that denotes the number of vertices inside each voxel. The resolution of the voxels is defined in millimetres and can be set to an arbitrary number. The default is 0.5mm (F600) or 0.8mm (F500). It is possible to obtain very low resolution ply files, if the processing times are extremely critical or it might be needed for external processing. Data reduction using the resolution parameter is statistically sound. If the resolution is set very high then almost no merging of vertices will take place.

The resolution should be set to the highest value possible while it is low enough, so that the minor alignment errors are corrected (double leaves) and the processing times are not too long. The resolution also has an impact on the unit separation and its value should be kept in mind while tuning the segmentation and unit separation.

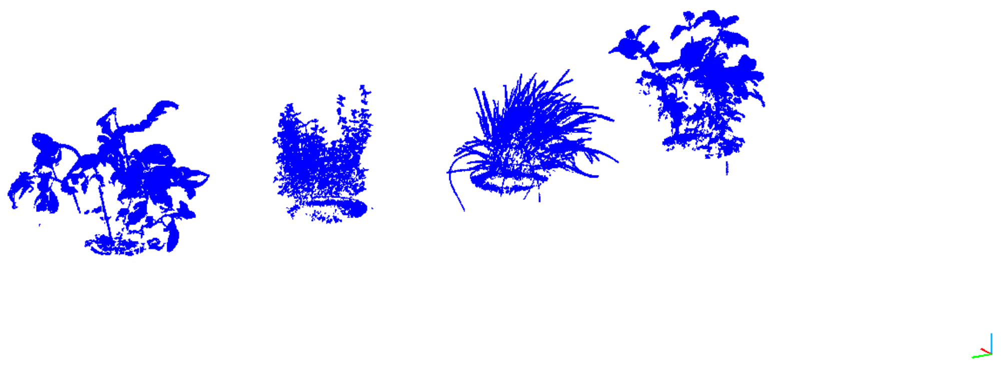
```
{
   "chain":
   {
      "preprocessing": {"enabled":false},
      "pooling":{"enabled":true, "merged_file_output":true},
      "unit_separation":{"enabled":false},
      "parameter_extraction":{"enabled":false}
   },
   "image3d":{"resolution":0.5}
}
```



Type	PLY_LITTLE_ENDIAN
Elements	1
Properties	15
Textures	0
Point X	vertex - x [PLY_FLOAT]
Point Y	vertex - y [PLY_FLOAT]
Point Z	vertex - z [PLY_FLOAT]
Red	vertex - red16 [PLY_USHORT]
Green	vertex - green16 [PLY_USHORT]
Blue	vertex - blue16 [PLY_USHORT]
Intensity	None
Nx	None
Ny	None
Nz	None
Faces	None
Texture coordinates	None
Texture index	None
Scalar	vertex - weight [PLY_USHORT]



resolution (too low) at 10mm, number of vertices in voxel: 1-1350



resolution (too high) at 0.05mm, number of vertices in voxel: 1-2

Optimising segmentation and unit separation

Similarly, like in the previous chapter the unit separation can be run using the voxel file output from the previous step. However, there is a significant difference between using the scanner file (preprocessing output) and using the voxel file (pooling output) as input. If the voxel file is used, even though the weight of the vertices is baked into the ply file, the underlying vertex data is lost. The parameter extraction module runs the calculations on the underlying vertices in the voxels whenever is possible and feasible.

There are a few specific use-cases in which running the unit separation and the pooling separately make sense but it is recommended that these modules are run in conjunction since the tunable parameters also interact and the pooling step will be run implicitly as part of the unit separation anyway. Therefore, even though it is possible, running solely the unit separation is disabled and will result in an error: [1026] Voxelgrid from PLY: functionality unsupported.

Let's set up a valid chain that will run the two modules and use the scanner files as input. Apart from the chain, the relevant config groups are "segmentation" and "unit_separation". It is recommended to start the tuning process by using hard border cut first.

```
{
   "chain":
   {
      "preprocessing": {"enabled":false},
      "pooling":{"enabled":true, "merged_file_output":false},
      "unit_separation":{"enabled":true, "full_file_output":true, "unitbox_file_output":true},
      "parameter_extraction":{"enabled":false}
   },
   "image3d":{"resolution":0.5},
   "segmentation":{"voxel_distance":1,"min_segment_size":10,"max_segment_size":1000000},
   "unit_separation":{"split_method": "hard_border"}
}
```

The JSON above is an example, if you use it as it is it will throw a missing parameter error in most cases.

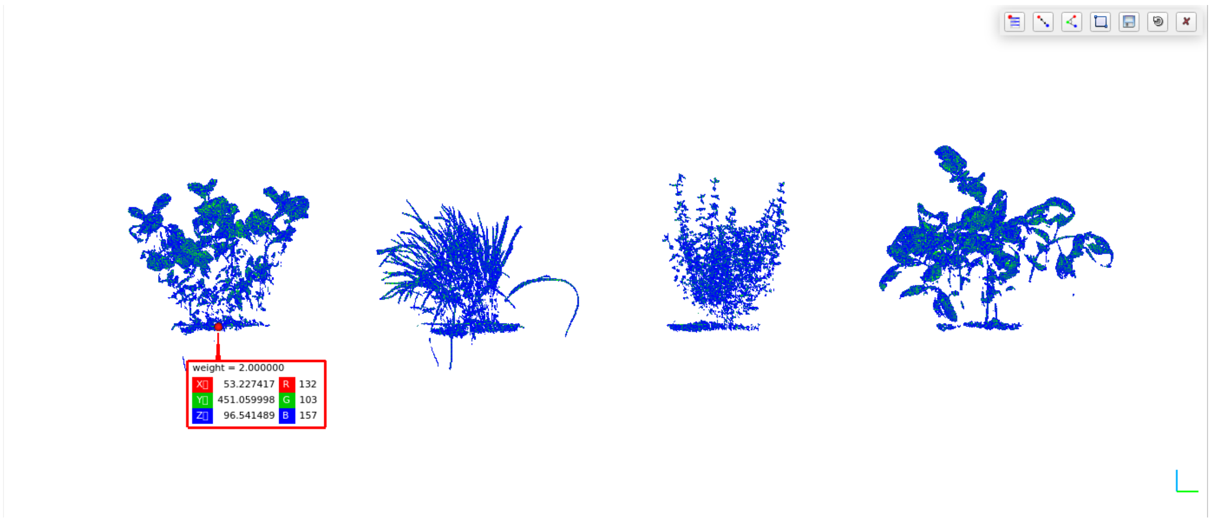
The resolution above is set to 0.5mm and the voxel distance is 1, which means that the neighbourhood of every voxel is defined by a 0.5mm x 0.5mm x 0.5mm cube. The neighbouring relationships are relevant for the segmentation. The lower the resolution, the higher the probability that a voxel has a more saturated neighbourhood.

It is recommended to keep voxel distance as low as possible, because the search in the neighbourhood is cubic, meaning that the search with voxel distance 1 will require 3^3 - 1 (26) operations per voxel, with voxel distance 2 it quickly grows to 5^3 -1 (124) operations, significantly increasing processing times.

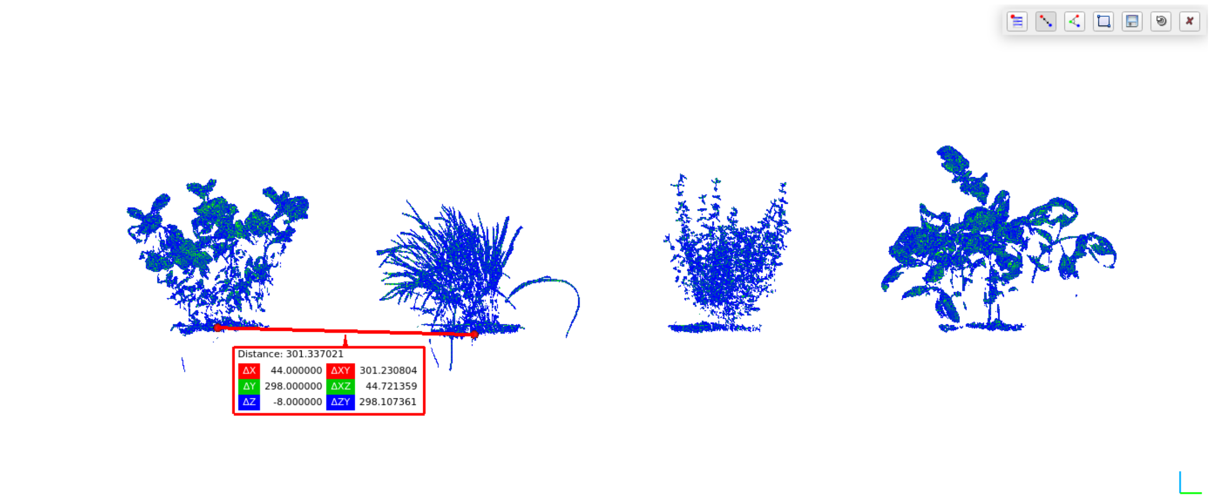
Since the scanner files are used as input for the chain defined above some additional global configurations need to be added to the JSON. The "field" settings, which are usually defined in a V1 header are required by the unit separation module but not by the pooling, that is the reason why the pooling JSON worked but the JSON above would not.

Let's setup the plant field configuration first using the "field" config group:

Let’s load in our voxel file from the previous chapter and determine the position of the first plant and the difference between the first two plants over the y-axis:



```
weight = 2.000000
X  53.227417   R 132
Y  451.059998  G 103
Z  96.541489   B 157
```

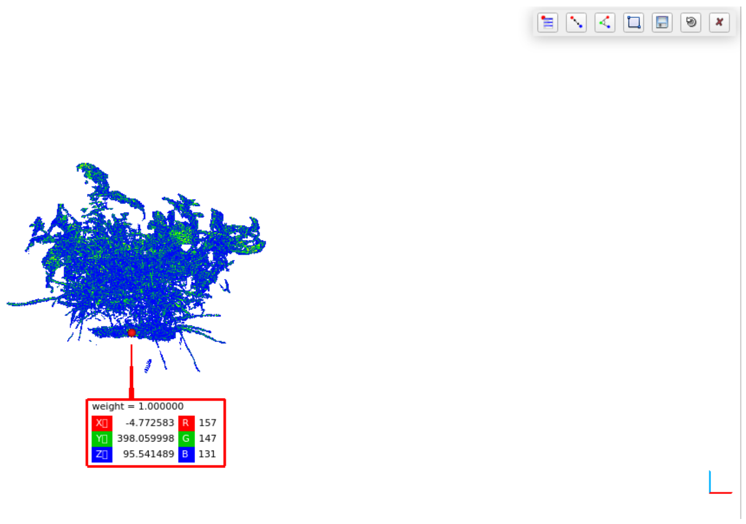


```
Distance: 301.337021
ΔX  44.000000   ΔXY 301.230804
ΔY  298.000000  ΔXZ  44.721359
ΔZ  -8.000000   ΔZY 298.107361
```

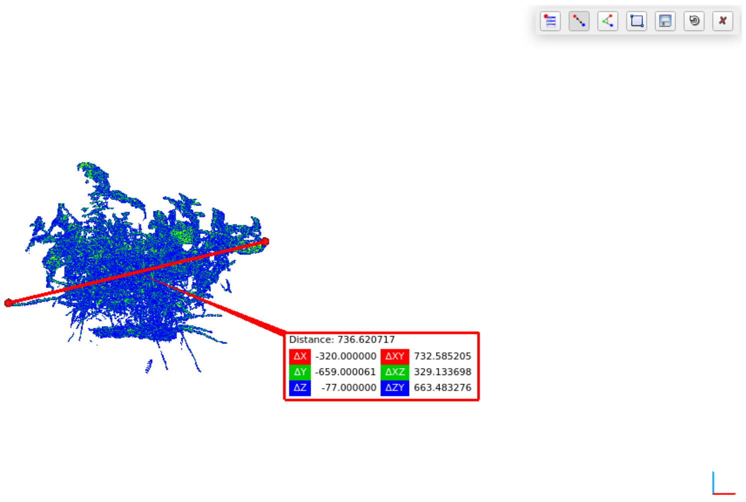
From the values above we note down the following values:

“x_units”:1,
“y_units”:4,
“origin_y”:450,
“period_y”:300

Now let’s determine a reasonable height of origin and the centre of the plant row over the x axis. NOTE: the definitions of the plant field are needed for the unit separation, changing the origin values will not have an effect on the values of the vertex coordinates. If you want to change the values of the height measurements use the pot_height parameter.



```
weight = 1.000000
X  -4.772583 R 157
Y 398.059998 G 147
Z  95.541489 B 131
```



```
Distance: 736.620717
ΔX -320.000000 ΔXY 732.585205
ΔY -659.000061 ΔXZ 329.133698
ΔZ  -77.000000 ΔZY 663.483276
```

Enrich our notes with the new observations:

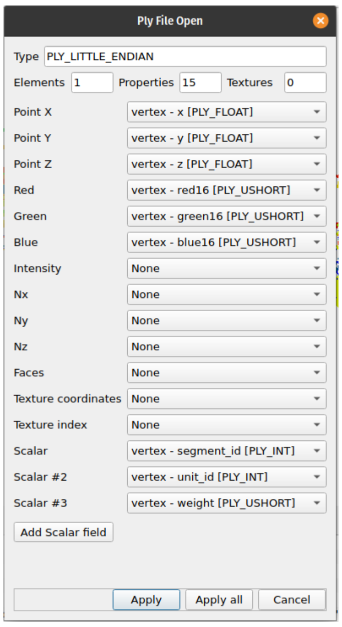
"x_units":1,
"y_units":4,
"origin_x":0,
"origin_y":450,
"origin_z":95,
"period_x":350,
"period_y":300,
"range_z":400

Since there is only 1 column of plants, the period_x value does not really matter, we just need to ensure that the period is larger than the x span of the plant column. If there are more columns, the configs can be obtained similarly to the period_y. The range_z is only tunable for completeness, but is deprecated.

The values we have noted down are from the perspective of the first plant, we need to translate the coordinates to the origin of the first unit box by subtracting the half of the period from the coordinates:
"field":{"x_units":1, "y_units":4, "origin_x":-175, "origin_y":300, "origin_z":95, "period_x":350, "period_y":300, "range_z":400}

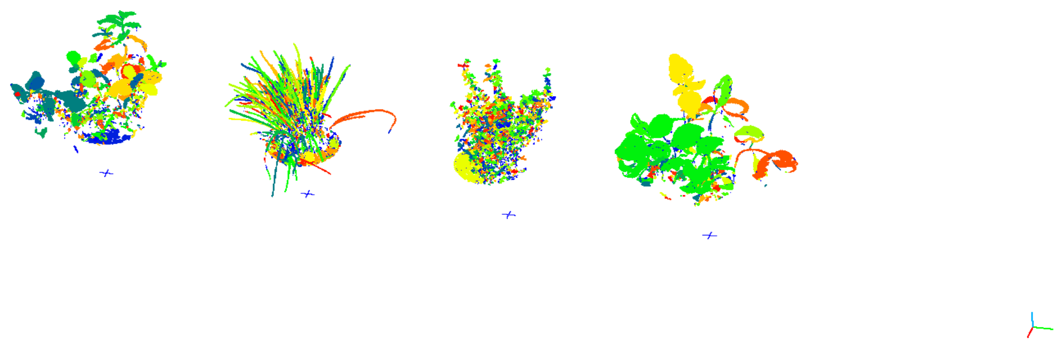
After identifying the correct field settings, we can build our valid JSON, let's also enable the unitbox file, which will show the centre of the units. The full file will then contain the segment and unit IDs.

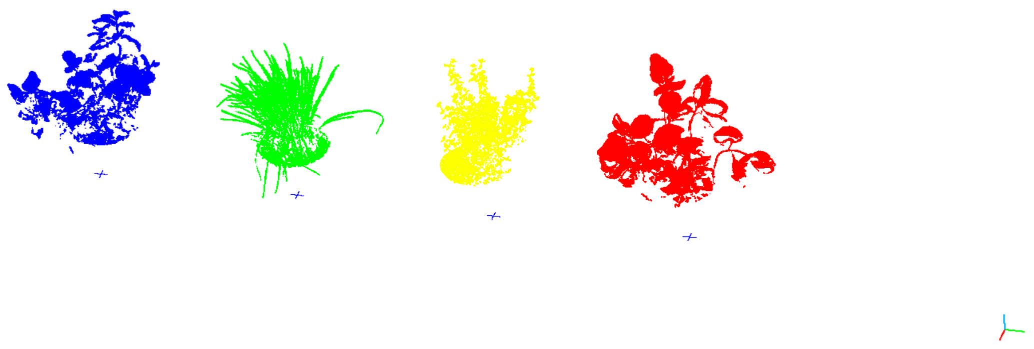
```
{
    "chain":
    {
        "preprocessing": {"enabled":false},
        "pooling":{"enabled":true, "merged_file_output":false},
        "unit_separation":{"enabled":true, "full_file_output":true, "unitbox_file_output":true},
        "parameter_extraction":{"enabled":false}
    },
    "image3d":{"resolution":0.5},
    "segmentation":{"voxel_distance":1,"min_segment_size":10,"max_segment_size":1000000},
    "unit_separation":{"split_method": "hard_border"},
    "field":{"x_units":1, "y_units":4, "origin_x":-175, "origin_y":300, "origin_z":95,
    "period_x":350, "period_y":300, "range_z":400}
}
```


Ply File Open

Type	PLY_LITTLE_ENDIAN
Elements	1
Properties	15
Textures	0
Point X	vertex - x [PLY_FLOAT]
Point Y	vertex - y [PLY_FLOAT]
Point Z	vertex - z [PLY_FLOAT]
Red	vertex - red16 [PLY_USHORT]
Green	vertex - green16 [PLY_USHORT]
Blue	vertex - blue16 [PLY_USHORT]
Intensity	None
Nx	None
Ny	None
Nz	None
Faces	None
Texture coordinates	None
Texture index	None
Scalar	vertex - segment_id [PLY_INT]
Scalar #2	vertex - unit_id [PLY_INT]
Scalar #3	vertex - weight [PLY_USHORT]

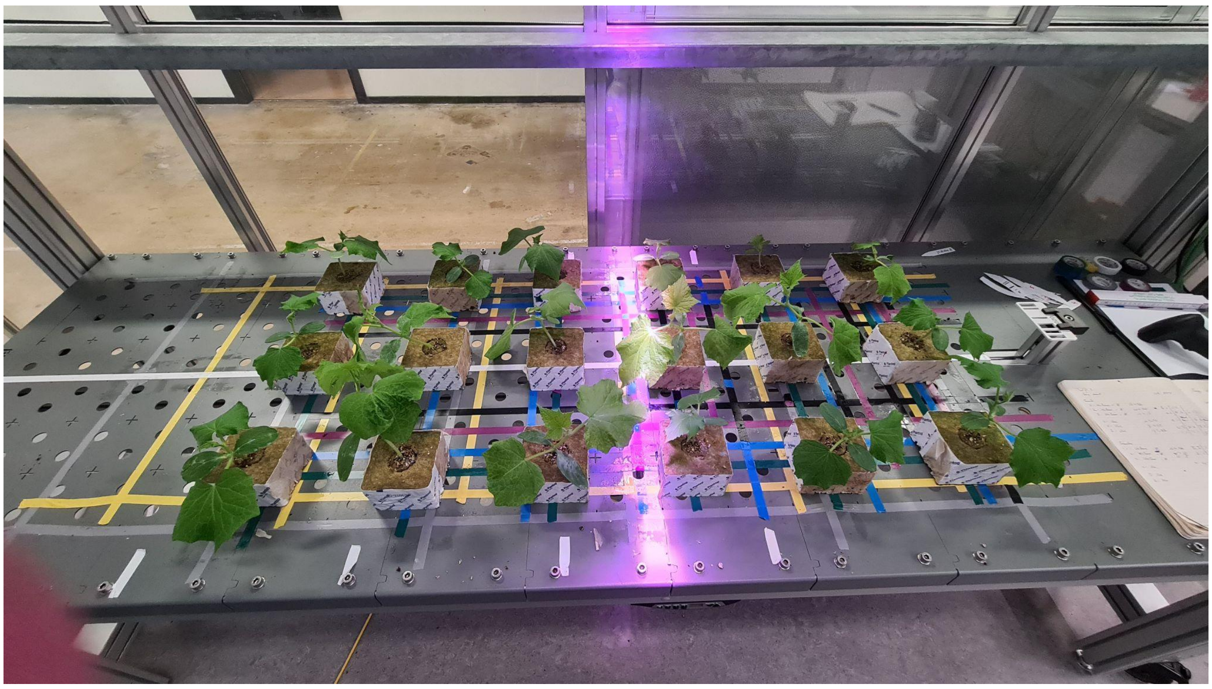
We can visualise both the units and segments that are generated by the unit separation module and the unitbox file may be overlaid on the full file to see if the definition of the unit centres is of acceptable accuracy.





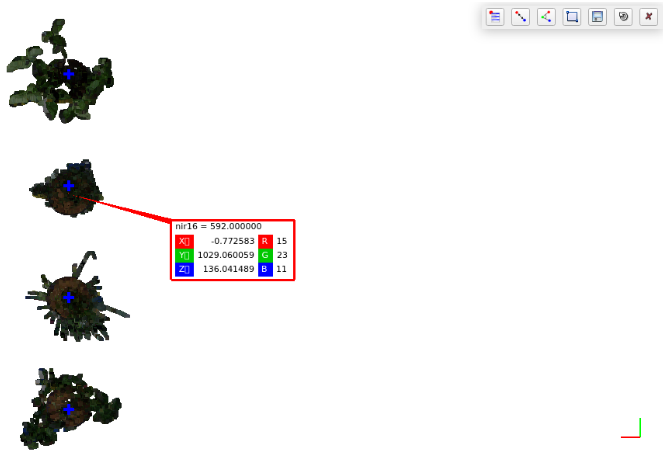
There are specific cases in which the pot centres are relatively far from the centre of mass of the plant material itself. In those cases the current splitting mechanism will deliver suboptimal results. However, since the unit centres can be manually defined. The splitting process can be aided by the manual definition of unit centres instead of the rigid grid definition.

Example of pot - unit centre mismatch:

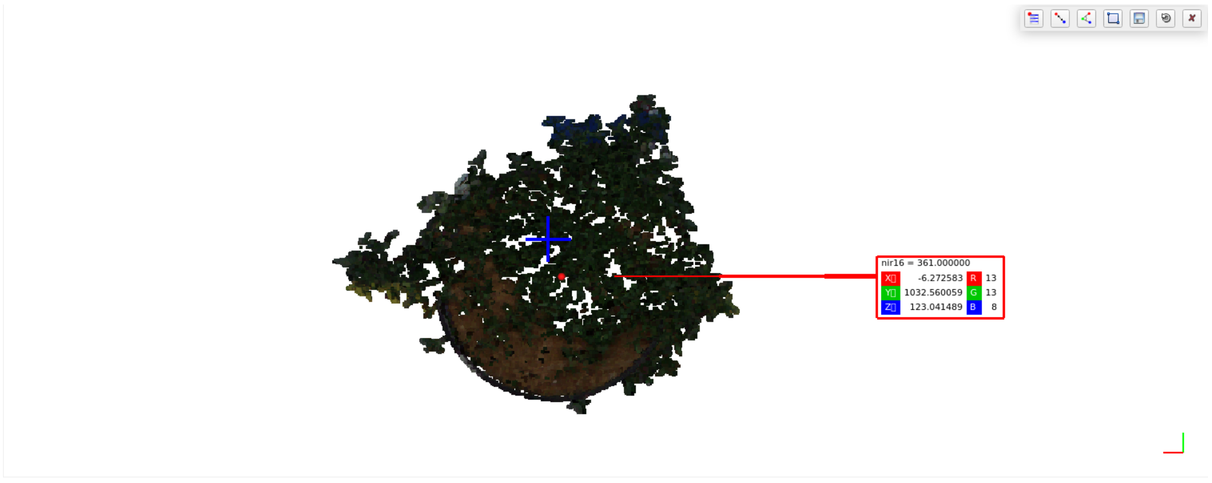


Additionally, even if the distribution of the plant material around the pot centre is stable and the pot distance is stable. The grid definition rarely translates to perfect unit centres. Even if visually all the units fall in their unit boxes, from the perspective of the splitting mechanism (CoM) that is irrelevant.

Let’s set up the splitting by manually entering the unit centres and thus correcting the inaccuracies in its definition.



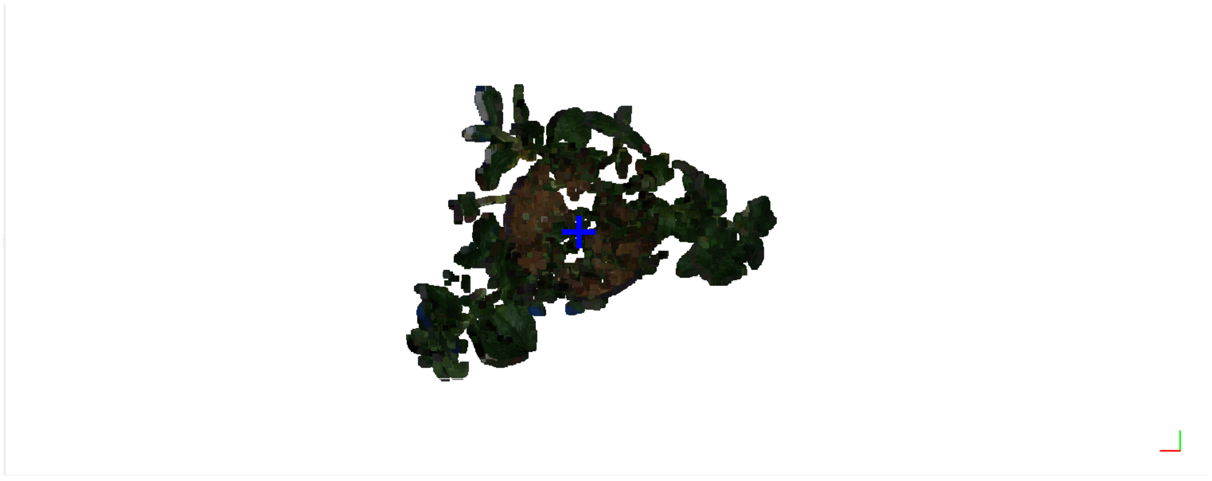
```
nir16 = 592.000000
X       -0.772583  R  15
Y     1029.060059  G  23
Z      136.041489  B  11
```



```
nir16 = 361.000000
X   -6.272583  R 13
Y 1032.560059  G 13
Z  123.041489  B  8
```

```
{
   "chain":
   {
      "preprocessing": {"enabled":false},
      "pooling":{"enabled":true, "merged_file_output":false},
      "unit_separation":{"enabled":true, "full_file_output":true, "unitbox_file_output":true},
      "parameter_extraction":{"enabled":false}
   },
   "image3d":{"resolution":0.5},
   "segmentation":{"voxel_distance":1,"min_segment_size":10,"max_segment_size":1000000},
   "unit_separation":{
      "split_method": "center_of_mass",
      "unit_centers":[  {"x":1, "y":1, "pos":[3, 452]},
                        {"x":1, "y":2, "pos":[-2,744]},
                        {"x":1, "y":3, "pos":[-5,1031]},
                        {"x":1, "y":4, "pos":[0, 1334]}]
   }
}
```

After the unit centres have been accurately defined. The center of mass splitting mechanism is expected to deliver decent results.



Custom segmentation & unit separation with 3dproc-v2

Requirements:
- Linux machine to execute the 3dproc-v2.1 binary
- Python 3
- Numpy

Files in the custom segmentation package:

- 3dproc-v2.1: The (linux) binary executable
- preprocessing.json: 3dproc-v2.1 config skeleton for preprocessing (stage 1)
- parameter_extraction.json: 3dproc-v2.1 config skeleton for parameter extraction (stage 2)
- custom_unit_separation_example.py: Demonstrates how to work with the intermediary PLY files of 3dproc-v2.1 to provide the segmentation masks externally
- ply_tools.py: small python class containing helper methods to work with ply files

To use 3dproc-v2 parameter extraction function alongside with a custom segmentation or unit separation method (e.g. machine learning), follow the procedure:

1, Follow the “guide” for optimising the preprocessing and pooling parameters for the setup in which the raw PLY files are generated.

2, Execute the preprocessing part of 3dproc-v2.1 providing the “raw” scanner files to acquire the “full file”:

```python
3dproc-v2.1 -i raw_file_1.ply raw_file_2.ply -o output_file.ply -c
preprocessing.json
```

Replace the raw_file_1.ply and raw_file_2.ply with the path of the raw input files that you want to process. Replace the output_file.ply with your output path and filename. Provide the path to the preprocessing.json after the -c option.

3, Annotate the PLY file “output_file_full.ply” (from the previous example CMD) using the unit_id and optionally the segment_id vertex properties. Check the python example “custom_unit_separation_example.py” for more information.

4, Execute the parameter extraction part of 3dproc-v2.1, providing the annotated PLY file and the stage 2 JSON.

Python

```
3dproc-v2.1 -i your_annotated_file.ply -o parameters.ply -c
parameter_extraction.json
```

3dproc-v2.1 will run all the standard extraction functions and a CSV is produced with the plant parameters. A mesh file is also generated which can be Inspected to detect processing errors.

IMPORTANT!

The parameter extraction will only work on the “output full ply file” that is acquired from running the raw files through 3dproc-v2.1 with the preprocessing configuration. PLY files with different structures (vertex properties, header) will **not** work.