GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Practical Report

submitted in partial fulfillment of the requirements for the course
„Extended Advanced Research Training - Applied Computer Science“

Implementation of point cloud tools in GroIMP

Maurice Müller

Department Ecoinformatics, Biometrics and Forest Growth, Büsgeninstitut
Georg-August University of Göttingen

1. October 2022

Supervisor:
Prof. Dr. Winfried Kurth

Contents

1 Introduction

GroIMP is a software to simulate the growth behavior of plants. 3D objects can either be generated with replacement rules, or manually with a large set of object transformation tools. Plants are normally represented with groups of fundamental 3D models, like spheres, cylinders or frustums. Even with replacement rules it is possible to build complex plants that consist only of these objects.

One of the most important goals of the GroIMP project is to find the rules that describe the growth behavior of specific plants as natural as possible. From a scientific point of view, this has the advantage that replacement rules are much simpler to handle and to understand than large amounts of independent components. Furthermore, differences between rules of multiple kinds of plants can be compared to get a more detailed understanding of how plants grow.

An other possibility to handle plants are point clouds. Point clouds are lists of 3D points. In most cases, point clouds are generated by laser scanners and exported to normal text files. Point clouds can be displayed in GroIMP, but until now there are no efficient tools to edit them.

Goal of this practical course is to extend the scope of functions in the GroIMP core to edit point clouds. This practical report describes the planning and implementation of following features:

- **Importing point clouds from files**
- **Clustering a point cloud to multiple point clouds**
- **Splitting a point cloud at a plane into two point clouds**
- **Merging multiple point clouds to one point cloud**
- **Selecting all displayed point clouds**
- **Exporting point clouds to files**

All functions will be available as XL functions and as graphical tools in the menu. The XL functions will be public java methods that can be accessed with an import statement in the XL code later. The graphical tools will be implemented in an own submenu and will always be available when the 3D view is also present in GroIMP.

This practical report has multiple target groups. The report itself is focused on the used algorithms, technical details of the implementation and the integration into GroIMP. The appendix contains a brief overview about the usable XL functions and the available tools in the objects menu. It also contains some example XL programs.

2 Preparation

To be able to implement new features in GroIMP, the software code must be downloaded, edited, and uploaded to the main repository again.

Because GroIMP is a very large project with multiple plugins and a sensitive project structure, the preparation is a bit complex. A detailed installation guide can be found in the wiki of the official repository of GroIMP.

The link to the current version of the developer guide can be found in the references of this document.

Summarized, these are all relevant steps during the GroIMP development cycle:

1. **Get a Gitlab account**
   The GroIMP project is located on the official Gitlab - not on the GWDG Gitlab. This is the reason why the student account does not work and a new account must be created there. After creating an account, an SSH key must be added.
2. **Be developer in the grogra/groimp repository**
   To be able to do some fundamental things in the official GroIMP repository, a membership with at least the level „Developer“ is required.
3. **Fork the repository**
   Because GroIMP itself and the belonging plugins are kept in one repository, there is only one main branch for the GroIMP core and one main branch for each plugin. The main branches are protected branches and contain the current official version of GroIMP. To be able to push the changed code to a remote repository and merge it later, an own repository is required. This repository is a clone (also known as „fork“) of the official repository, but with the difference that it can be accessed with owner permissions. Code changes must be pushed to this repository. When the new features or bugfixes have been finished and tested, a merge request can be created in the official repository. The source branch of the merge process can then be chosen from the own (forked) repository.
4. **Install Eclipse**
   In principle, it does not matter which kind of development environment is used. GroIMP can also be developed in the terminal with manual built scripts. But the GroIMP developer teams proven IDE is Eclipse.
5. **Install EGit in Eclipse**
   EGit is a Eclipse extension that provides all important features of Git in the graphical user interface. The most important features are `git clone`, `git pull`, `git add`, `git commit` and `git push`.
6. **Clone the repository**
   The next step is to clone the **own** repository to Eclipse. If the official one is cloned here, the `git push` interaction will fail later.
7. **Setup the project**
   After the repository (and plugins) have been cloned, the project settings must be adapted. GroIMP is only compatible to a compliance level of 1.7 and a JRE version between 1.5 and 1.8.
8. **Create a starter**
   To start GroIMP with the green start button in Eclipse, a starter must be created. The main method is `de.grogra.pf.boot.Main`. Make sure to add the program parameter `--project-tree`, because GroIMP is executed in the development project structure and not in the installed project structure. Otherwise, it cannot find any plugins and will show an error „No application found.“
9. **Implement the feature**
   Keep in mind that lots of beginners work on this project. So write clean code and make use of useful comments (Javadoc will be the best option).
10. **Create a commit**
    First, the changed code must be staged for the commit. Then, the commit can be done. The commit should have a descriptive message and should be on a separate branch.
11. **Push to the remote repository**
    The changed code is now packed into a commit and can be pushed to a remote repository. This is the cloning repository for now.
12. **Create a merge request**
    After the code has been pushed and the feature finished, a merge request can be created in the official repository. The source branch must be selected from the own (cloning) repository. The target branch is the belonging main branch in the official repository.

3 Design

3.1 New functions

The most common way of using point clouds in GroIMP is to use existing point clouds that have been exported to text files by 3D laser scanners. These files must be read to get the stored point coordinates. Until now, there is no such function available, so it must be programmed and copied to each XL code file separately. So the first new point cloud relating feature will be an **import** function. The point cloud files can then be selected with a file dialog or imported from file objects in XL code.

The **clustering** algorithm is the most complicated part of all new point cloud features. It analyzes the distance between all points to detect whether points belong to the same cluster or not. For this, multiple parameters are required. The first and most important parameter is the maximum distance that must be fulfilled by two points to be categorized into the same cluster.

By analyzing the point distances, each point gets a certain number of neighbor points. In real scan results, there may be many points that do not belong to any cluster or have only one neighbor point. These points are called „noise“ and can optionally be removed during the clustering. The second parameter defines how many neighbor points are required to keep the point as non-noise point.

The third parameter is responsible for the algorithm's efficiency. The first intuitive attempt to analyze all neighborships between the points would be to compare each point with each other point. But because this method has a quadratic running time, it is inapplicable for large point clouds. To avoid this, the point cloud is converted to an octree. An octree is a structure that divides a cuboid, in this case the bounding box of the point cloud, into eight parts. It splits the cuboid in the center in x-direction, in the center in y-direction and in the center in z-direction. This is repeated until the size of each resulting box is slightly larger than the minimum neighbor distance. Octree child elements without points are ignored and do not consume computational effort. The advantage of this tree structure is that all points are sorted by their position in three dimensions and can directly be compared with their potential neighbors. With an optimal third parameter for the octree depth, the computing time can be reduced from n^2 to $n \cdot log(n)$.

In XL, the clustering algorithm will be available as a java function that takes exactly these parameters. By using the function in the menu, a dialog will be used to ask the user for the parameters. In both cases, the original point clouds will be replaced by a list of new ones. Technically, the clusters are also point clouds. So they are fully compatible for further edition.

It can happen that the clustering algorithm does not detect clusters, if they are connected with single points that have a very small distance. To have a possibility to correct this, a **split** function will be available. This function needs a point cloud and a 3D plane. The plane should have a position and an orientation so that some of the points of the point cloud are on the one side of the plane and all other points are on the other side. If all points of the point cloud are on one side of the plane, the second point cloud will be empty after executing this operation. This function will be equivalent in XL and in the menu and replaces the currently handled point cloud with two new ones.

An other useful tool will be the **merge** function. It can either be used to merge a subset of point clouds or all existing point clouds. In both cases, a new point cloud is created and will contain all points of all used initial point clouds. The old point clouds are then removed. In the XL code, there will be a function that takes a list of point clouds. For the function in the menu, multiple point clouds must be selected in the 3D view.

After editing point clouds, it can be useful to **export** them. Until now, there is no function to do this automatically. In XL code, a function will be available that takes a file object to write the points to. The function in the menu will use a file dialog so that the user can select the file that should be written.

3.2 Code structure

Like all other implemented 3D objects, point clouds are a part of the module `IMP-3D` of the GroIMP core. Because all new features are designed to work on point clouds, they will be implemented in a way so that they are fully compatible to the class `de.grogra.imp3d.objects.PointCloud`.

Because the implemented methods are very long and need lots of code lines, they are outsourced to an own class `PointCloudTools`. This class contains all methods that are meant to be used in the XL code and the methods that are automatically called by the buttons in the menu.

A difficulty is that the clusterization algorithm needs some classes like `PointCloud`, `Octree`, `Point` and `Cluster`. These classes do already exist in GroIMP, but with other usages. A possible solution would be to add the own functionality to them, but by doing this the structure of all other classes would become destroyed. An other possible solution would be to change the implementation of the existing classes. But because all of them can also be used directly in XL code, it is widely unknown what can happen if the classes do not work as before anymore and projects of other people do not work anymore too. The third possible solution would be to add the required functions to that classes, so that they work with the existing data. The problem of this way is that the already existing data does not fit to requirements of the clusterization algorithm. The data would have to be imported and exported between different data formats each time and this would cause a very high computing effort. Especially for the clusterization of large point clouds, this causes long computing times and is not acceptable for the user.

To add a new package with redundant classes would normally be a very irresponsible solution. But due to the lots of difficulties, this is the only way to ensure that the existing functions and the new functions can reliably work after adding the new features. The new package is called `de.grogra.imp3d.pointcloud` and is located next to the existing package `de.grogra.imp3d.objects`. This makes it possible to outsource all the code and not overload the original `de.grogra.imp3d.objects.PointCloud` class.

Because all new functions are located in an own class (`PointCloudTools`), the target point clouds must be given as parameters and the functions have a static context (a class context). In XL code, the required objects can be created with belonging constructors and then used as parameters. The returned objects are newly created and do not have any dependency to the old ones. In the XL code context, the new point clouds are not added to the RGG graph automatically. To use the functions in the objects menu, the target objects must be selected in the 3D view first. If the objects in the selection do not fit to the clicked option, an information window will appear. The changes are directly made in the RGG graph and are visible in the 3D view afterwards.

The graphical tools to do the operations by hand will be added in the GroIMP main menu. When the 3D view is loaded, the main menu contains an item „Objects“. This menu will get a new submenu „Edit Point Clouds“ containing all new features. The implementation of the menu entries is done in the plugin descriptor file `IMP-3D/src/plugin.xml` and in the resource file `IMP-3D/src/plugin.properties`.

The keyboard shortcuts and the access via the `ALT` key are partially automatically added by the graphics library.

4 Implementation

In general, the point cloud features are implemented on a lower level and on a higher level. Functions on the lower level can be used in the XL code and are called with the required objects as parameters. Their return value contains the expected object or array of objects. Functions on the higher level are automatically used by the point cloud buttons in the objects menu. They internally use the low level functions and are made to interact with the user and the current 3D scene graph.

This implementation chapter is also structured into two parts. The first part describes the internally used (and XL compatible) functions. This part is focused on how the functions work and which algorithms are used. The second part describes the graphical functions. Because they are based on the internal functions, the second part is focused on how the interaction with the user and the current 3D scene graph is implemented.

4.1 XL Functions

All XL compatible functions are designed to be usable with prepared java objects. The objects are expected to have the correct object type and because the internal java methods can only be called with the correct parameters, a type check is widely unnecessary. All implemented functions work with point clouds and optionally some other objects.

Because there are two types of point clouds (reasons: see introduction), all point clouds have to be converted internally. Due to the fact that XL code is only able to use the default point clouds, all point clouds that are used as parameters must be of the default point cloud type. Also the returned point clouds or arrays of point clouds are of the default point cloud type and fully compatible to XL and the RGG graph. This makes it possible to use multiple functions in combination to do more complex operations on point clouds.

The internally used point clouds are of an other point cloud type because the newly added point cloud java class is more efficient and more suitable for most of the used operations. All point cloud types are automatically converted and not relevant for a GroIMP user. Because the functions are implemented in the PointCloudTools class, they are used in static context and do not require a reference object. Code examples for XL use cases can be found in the appendix.

4.1.1 Importing point clouds

Importing point clouds is one of the most important features. Point clouds are usually created by laser scanners and are written to large TXT files or CSV files. Each line of a file contains exactly one point, considering all three dimensions. In general, it is **not** required that all numbers have the same number of digits. Integer numbers and floating point numbers can also be mixed.

Example CSV file:

```
...
0.1, 0.2, 0.3
0.3, 0.4, 0.4
0.4, 0.6, 0.5
0.5, 0.8, 0.6
...
```

There are also TXT files without the comma notation. In many cases, TXT files have the file ending „xyz“ to indicate that they only contain coordinate triples.

Example XYZ file:

```
...
0.1 0.2 0.3
0.3 0.4 0.4
0.4 0.6 0.5
0.5 0.8 0.6
...
```

For the import function, it does not matter which type the file has and whether the commas are present or not. Because all floating point numbers are notated with a dot, commas are internally detected and replaced by spaces. Summarized, it does not matter how the lines are formatted. The only important thing is that numbers are separated by at least one space or one comma (or both).

The file name ending is ignored too. This makes it possible to use file name endings of external software or other naming conventions.

Importing multiple point clouds from one file

It can (and will) happen that a project contains multiple point clouds. While it would be acceptable to import two or three point clouds from different files separately, it becomes more and more impractical the more they become. To avoid this, point cloud files support a point cloud id for each point. This is an integer number that can be added to the end of each line in the point cloud file. During the import process, all points from the file are collected in a list of point clouds and are added to the point cloud with the index that is equal to the point cloud id of the currently imported point. If no id is available, a point is added to the point cloud with the id/index 0. The examples (see above) would be imported as a list with exactly one point cloud.

Example CSV file with point cloud ids:

```
...
0.1, 0.2, 0.3, 0
0.3, 0.4, 0.4, 0
0.4, 0.6, 0.5, 1
0.5, 0.8, 0.6, 2
...
```

The example would be imported as a list that contains 3 point clouds (with the indices 0, 1 and 2). The first both points are added to point cloud 0 while the others are added to point cloud 1 and 2. This also works for XYZ or TXT files with customized separators (commas or spaces):

```
...
0.1 0.2 0.3 0
0.3 0.4 0.4 0
0.4 0.6 0.5 1
0.5 0.8 0.6 2
...
```

4.1.2 Clustering point clouds

The cluster function is the most important feature of this practical report. It makes it possible to split large objects, like 3D–scanned trees or other plants. A condition for the successful clustering is that all future „clusters“ in the plant are separable by their distance. Points inside one cluster and points inside another cluster must be very close to each other, while the nearest points between one point cloud and the other one must have a distance that is much larger than the internal distances.

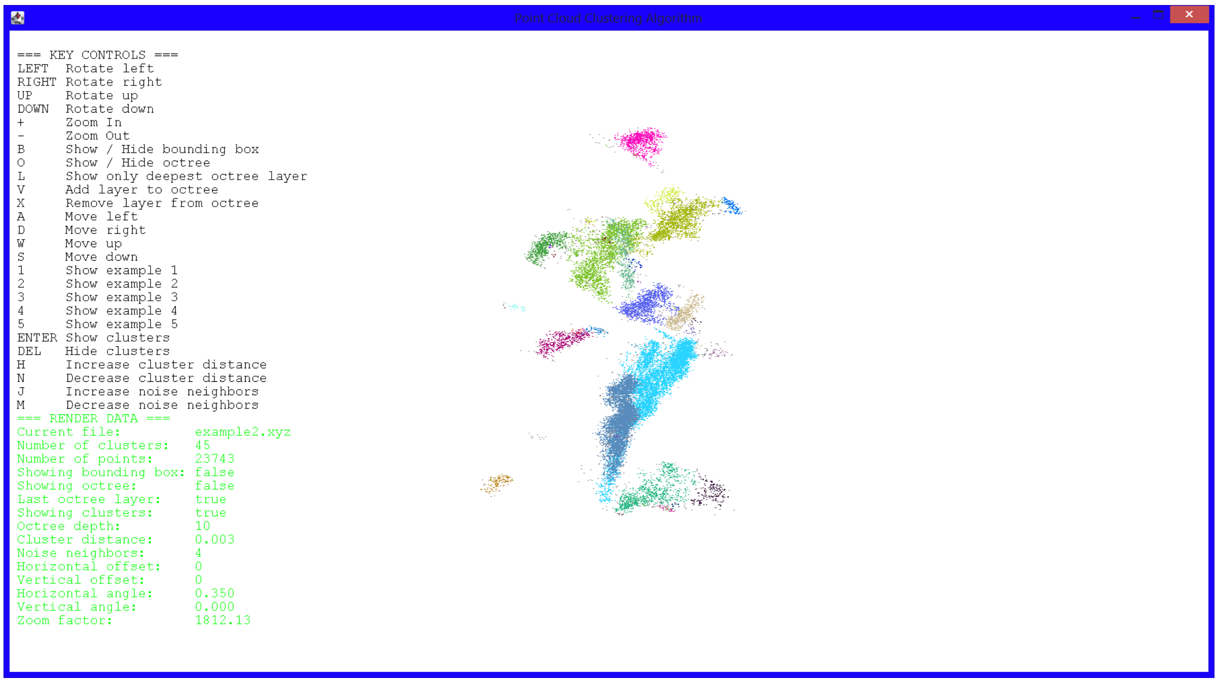
The clustering algorithm was described very detailed in the Bachelor's Thesis „Object recognition in point clouds using a density–based clustering algorithm“ by Jörg Krause, written in 2007. The Bachelor's

Thesis was provided with an executable software, called ORIPUCA (**O**bject **R**ecognition **I**n **P**oint Clouds **U**sing **C**luster **A**nalysis). This software is written in C++ and contains the algorithm, as described in the belonging document.

Goal of this practical report was to reuse the work of J. Krause and reimplement the algorithm in Java to make it usable in GroIMP.

Simulation software

To test the own Java implementation of the clustering algorithm, an external software was necessary. Because it seemed impossible to implement lots of debug output, custom key bindings, an efficient 3D navigation, direct effects on the shown output and automatic display of the bounding box as well as the octree, the software shown in the following screenshot was developed.



```
=== KEY CONTROLS ===
LEFT  Rotate left
RIGHT Rotate right
UP    Rotate up
DOWN  Rotate down
+     Zoom In
-     Zoom Out
B     Show / Hide bounding box
O     Show / Hide octree
L     Show only deepest octree layer
V     Add layer to octree
X     Remove layer from octree
A     Move left
D     Move right
W     Move up
S     Move down
1     Show example 1
2     Show example 2
3     Show example 3
4     Show example 4
5     Show example 5
ENTER Show clusters
DEL   Hide clusters
H     Increase cluster distance
N     Decrease cluster distance
J     Increase noise neighbors
M     Decrease noise neighbors
=== RENDER DATA ===
Current file:          example2.xyz
Number of clusters:    45
Number of points:      23743
Showing bounding box:  false
Showing octree:        false
Last octree layer:     true
Showing clusters:      true
Octree depth:          10
Cluster distance:      0.003
Noise neighbors:       4
Horizontal offset:     0
Vertical offset:       0
Horizontal angle:      0.350
Vertical angle:        0.000
Zoom factor:           1812.13
```

Figure 1: The simulation software allows lots of custom key bindings, render information output, a direct effect on the rendering and debugging features that are not easily implementable in GroIMP.

The view can be moved with the keys W, A, S and D (like in the most computer games), rotated with the arrow keys, zoomed with + and −, and so on. Most of the interesting debug output is directly shown in the left of the view. The bounding box can be enabled with B, and the octree with O. Layers can be added with V and removed with X.

Summarized, the simulation software was a very useful and important tool to implement, test and debug the implementation of octrees and the clustering algorithm.

After the implementation of the clustering algorithm was finished, all relevant code has been migrated to GroIMP and extended as described in the following chapters.

The density–based clustering algorithm

A first idea for the density–based clustering algorithm would be to compare each point with all other points to find the distance between them. After the comparison of all points is finished, the point pairs with a distance lower than the given maximum distance can be put into one cluster. This can be repeated recursively, because neighbors of neighbors are expected to be in the same cluster.

The expected runtime of a comparison of each point with all other points is $O(n^2)$. Especially in large point clouds, the computing time ends in minutes or hours and is not reasonable.

If the inefficiency of the algorithm is analyzed in more detail, it is noticeable that lots of points are compared that obviously can not be neighbored. For example, if a plant is 1 meter high and the neighbor distance is 5 millimeter, a point in the root of the plant should not be compared to all points that are more than 5 millimeter above the ground.

An efficient approach is to divide the 3D volume of the point cloud into subvolumes. Then, each point of the point cloud gets sorted into the concerning subvolume, depending on its individual position in the point cloud. The subvolumes should be as small as the given maximum neighbor distance (also called `epsilon`).

For the „3D volume of the point cloud“, a bounding box is used. A bounding box is a cuboid with the minimum and maximum x, y and z coordinate of the contained object. This ensures that all points of the point cloud are contained in the cuboid. The most simple way to divide the cuboid into subvolumes is to create an octree. An octree is a tree structure where each node has 8 child elements. In case of the cuboid, the 3D volume is divided in the center position in all three dimensions. The result are eight cuboid octants (front top left, front top right, front bottom left, front bottom right, back top left, back top right, back bottom left and back bottom right).

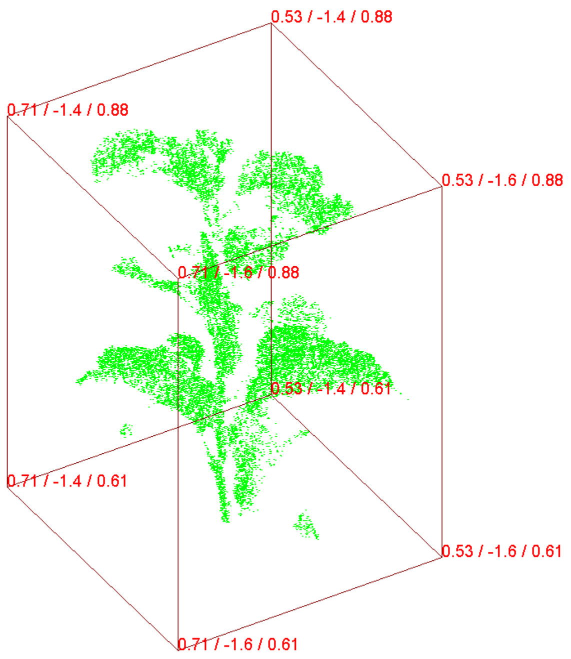


Figure 2: The bounding box is a cuboid, defined by the minimum and maximum x, y and z coordinate of the point cloud.

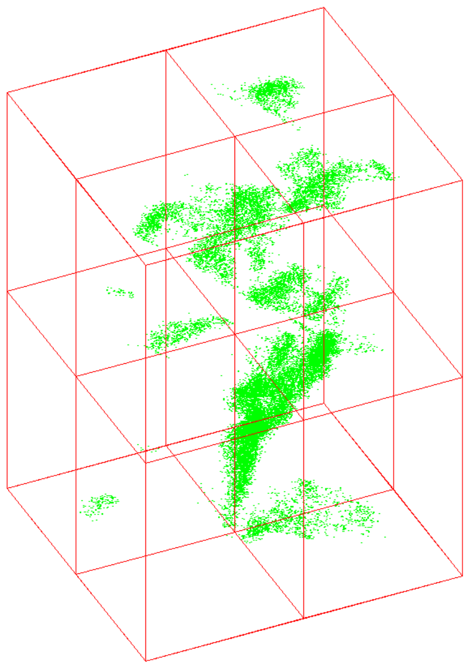


Figure 3: The octree with two layers is shown here. The root layer is equivalent to the bounding box (layer 0) and the second layer (layer 1) is the first subdivision into eight octants.

The division into eight octants can be repeated for each octant, until the required size is reached. Octants that do not contain any points are not further processed and ignored. This saves lots of subbranches in the tree and avoids an exponential growth of computing time and required memory.

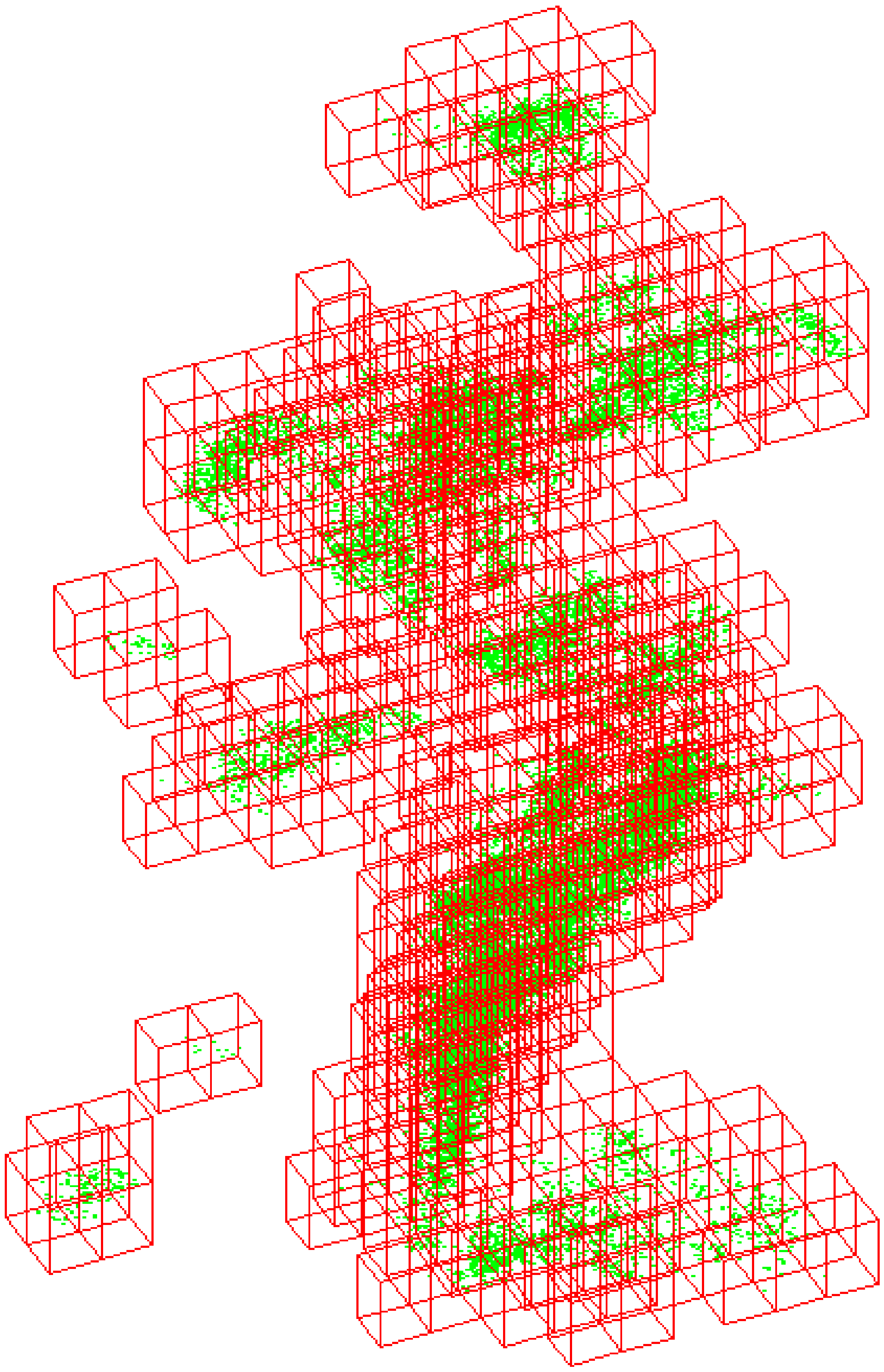


Figure 4: This image shows only octree subnodes on the deepest layer. All subnodes have the same size. The points are sorted into the nodes. All non–used subnodes do not exist to not increase the computing time or the required memory.

To place the points in the octree structure, all points are sorted into the path of octants with the position fitting to the concerning point. This step is repeated for each point. In the end of the iteration, all points are sorted into one of the subnodes on the deepest existing layer in the octree. Then, all points can directly be compared with the points from the same subnode and with the points from the direct neighbor subnodes in all 26 directions in the surrounding 3x3x3 cube (6 sides, 12 diagonals and 8 corners). By only comparing the points with all points from the neighbored subnodes, the number of point comparisons can be reduced from n^2 to an average number between n and $n \cdot log(n)$. The real efficiency depends on the concrete point cloud and the optimization of the input parameters of the algorithm.

With the sorted points, the main algorithm can be started. Goal of the algorithm is now to cluster the point cloud by creating the clusters as required and assign the fitting one to each of the points. An additional cluster is used as noise cluster and is filled with the points that have no neighbor points.

Independently from the sorted points in the octree structure, all points are also stored in an array. To ensure that each point gets a cluster assigned, a loop iterates over all points in the array and jumps to the next unassigned point. If a point without an assigned cluster was found, a new cluster is created and assigned to that point. After that, all neighbor points are requested from the octree and are also assigned to this cluster. It can not happen that neighbor points are found with an already assigned cluster. If this case would exist, the currently analyzed point would have been forgotten when the neighbored points of the other cluster were collected. Because the points in the octree and in the array are the same and the structures have only references to the points stored, the assignment of the cluster is also requestable in the loop that jumps to the next point in the array that has no cluster assigned.

If the loop reaches the end of the array, all points have a cluster assigned. The clusters are collected in a dynamic list. If requested, the noise cluster is added to that list afterwards. At last, the algorithm returns the list of clusters. During the assignment of the fitting cluster to a point, also the point was added to the belonging cluster. Technically, the result of the algorithm is a list of lists now. The main list is the list of clusters and each list (cluster) contains the belonging list of points.

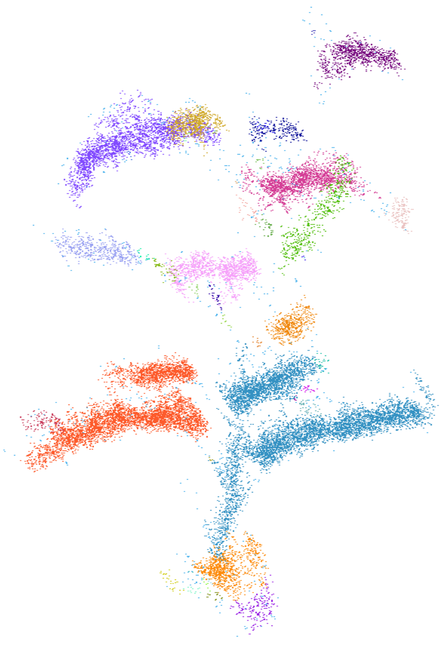


Figure 5: The different clusters of the plant are shown in randomly chosen colors.

Parameters of the clustering algorithm

To use the cluster function, a point cloud and some further parameters must be provided. Because the algorithm is generalized to any kind of point clouds, it can not assume what should be clustered and in which degree of detail. To let the algorithm know how to cluster the point cloud, some parameters are required. They are described here in detail:

`PointCloud pointCloud`
The given point cloud is used as reference object to get all points from. The point cloud itself is not manipulated by the algorithm and can be used after the clusterization without any changes. The clusters (also point clouds) are returned in an array with newly created point cloud objects.

`double epsilon`
This parameter specifies the maximum distance for points (P1 and P2) that should be classified as one cluster. All points that have a larger distance are assumed to be in different clusters, except if there is an other point P3 (or multiple points) between P1 and P2 and both neighborhoods (from P1 to P3 and from P3 to P2) are small enough. Points with a lower distance than `epsilon` are called „neighbors“.

`int minimumNeighbors`
The minimum number of neighbors is important to detect the noise points. If there are points with a lower number of neighbored points than the value of this parameter, the point is classified as noise point.

`int octreeDepth`
The octree depth is the number of layers in the used octree. It should always be larger than 1, for the most use cases a value between 5 and 15 may be practical. If this value is too low (< 2), the clustering algorithm is still working correctly, but with a runtime of $O(n^2)$. Especially in large point clouds, this will result in a very time–consuming and inefficient execution of the algorithm and can lock the interactivity of GroIMP for several minutes.

`boolean removeNoiseCluster`
This parameter can be set to `true` to automatically delete noise points and not add an own „noise cluster“ to the returned array of normal clusters. If this parameter is set to `false`, the last point cloud in the returned array is the additional „noise cluster“ and contains all points that can not be classified to any other cluster. If there is no noise in the 3D scan result and the noise cluster should be used, the cluster remains empty and a point cloud without any points is added.

4.1.3 Splitting a point cloud

After a plant has been scanned and imported to GroIMP as point cloud, the plant is represented as one large point cloud. With the cluster function, most of the plant parts can be separated. But sometimes there are parts in the scan result that can not be clustered because there are connecting points between two clusters and the clusterization algorithm decided to let the two clusters be a large one. In that case, it can be useful to do this rework manually.

A point cloud consists of a list of points in a 3D coordinate system. To split a point cloud into two smaller ones, a logical decision rule is required to distinguish between points that are put to the first new point cloud and points that are put to the other one. The most simple rule to split a 3D room into two halves is to add a plane with a given position and a given rotation. This ensures a very simple dividing strategy, but makes a high adaptability possible.

The given plane provides a matrix that contains all relevant information, like a vector from the coordinate systems origin to a point on the plane and a normal vector of the plane. The normal vector can be used to describe the direction of the plane. The relative position between the plane and a point in the point cloud can be calculated by subtracting the position vector of the point from the position vector of the plane. The result of this subtraction is a new vector and can be compared to the normal vector of the plane. For this, the scalar product can be calculated between both vectors. If the scalar product is equal to 0, the vectors are ortogonal and the point is located on the plane. Otherwise, the scalar product is greater or less than 0 and the position (of the point) is before or behind the plane. By repeating this for all points of the point cloud, two subsets of points can be created and returned.

4.1.4 Merging point clouds

If point clouds are too detailed after they have been clustered or a union set of all points of multiple point clouds is required for other purposes, the merge function can be used. The function requires a list of point clouds and returns a large point cloud that contains all points from the given point clouds.

The merging function also works with one point cloud. This ensures that an unknown number of point clouds can be merged during runtime, even if it is unknown whether there are multiple point clouds or only one. If the merging function is explicitly called with only one point cloud, the function clones the old point cloud and returns the clone. The newly created point cloud does not contain information about where the points came from.

4.1.5 Exporting point clouds

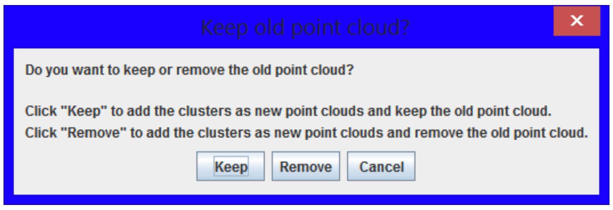
After point clouds have been edited, they can be exported to TXT/XYZ or CSV files. This becomes important to share the project results with other users or store the results of a very time consuming clustering process of large point clouds. The export function is implemented as the opposite of the import function, so the exported files are fully compatible to the import function.

All point clouds are exported in a CSV compatible format. That means that numbers are always separated by a comma and a space. If there is only one point cloud, the point cloud id (index) is removed during the export process and the file only contains the three dimension values for each point. Otherwise, the point cloud id (index) is added to each exported point. This makes it even possible to export a large number of point clouds to only one file without losing information about the different point clouds.

4.2 Graphical Functions

The graphical features are designed to be used by the belonging menu items in the objects/pointclouds menu. The graphical features to import, cluster, split, merge and export point clouds are based on the XL functions described above. They are made to be used interactively with the currently displayed RGG graph as working context. While the XL functions use java objects that have been created in the XL code, these functions check the currently selected objects in the 3D view and request some parameters with input dialog windows.

Additionally, all graphical functions have the possibility to show error message dialogs if objects are not selected as expected. Because the 3D view is a bit more sensible than the objects that exist in the XL code, there is a small feature that helps removing old objects. Each graphical function that generates new objects from old ones is able to optionally delete the old objects. This can be used to either add the new objects or replace the old ones by new ones.


Keep old point cloud?

Do you want to keep or remove the old point cloud?

Click "Keep" to add the clusters as new point clouds and keep the old point cloud.
Click "Remove" to add the clusters as new point clouds and remove the old point cloud.

Figure 6: With this dialog window, the old object(s) can be removed or kept in the RGG graph. This window can also be used to cancel the operation if it was clicked accidentally. The text in the window (plural, singular) is adapted to the different usecases automatically.

The menu entries are added in the file `IMP-3D/src/plugin.xml` at the registry path `/objects/3d/geometry/`. They are grouped in an own submenu that is called „pointcloud“. The inserted code in the `plugin.xml` file looks as follows:

```
<group name="pointcloud" itemGroup="true">
   <command name="import" run="de.grogra.imp3d.pointcloud.PointCloudTools.guiImportFromFile" />
   <command name="cluster" run="de.grogra.imp3d.pointcloud.PointCloudTools.guiCluster" />
   <command name="split" run="de.grogra.imp3d.pointcloud.PointCloudTools.guiSplit" />
   <command name="merge" run="de.grogra.imp3d.pointcloud.PointCloudTools.guiMerge" />
   <command name="export" run="de.grogra.imp3d.pointcloud.PointCloudTools.guiExportToFile" />
   <command name="select" run="de.grogra.imp3d.pointcloud.PointCloudTools.guiSelectAllPointClouds" />
</group>
```

The added code lines add a new submenu in GroIMP if the 3D view is loaded (or a 3D project is currently edited). The submenu can be found in „Objects“ → „Edit Point Clouds“ in the main menu. Each menu item calls the belonging static java method in the class `de.grogra.imp3d.pointcloud.PointCloudTools` with the automatically added parameters „Item item“, „Object information“ and „Context context“. The java methods then check the current selection, request some parameters with input dialogs and handle the resulting point cloud manipulation.

The names for the menu items are defined in the file IMP-3D/src/plugin.properties. The ampersand characters are used to mark the keyboard shortcut letter. This letter is later underscored when the text is rendered in the menu. The inserted code in this file looks as follows:

```
/objects/3d/geometry/pointcloud.Name = Edit Point &Clouds
/objects/3d/geometry/pointcloud/import.Name = &Import Point Clouds From File
/objects/3d/geometry/pointcloud/cluster.Name = &Cluster Point Cloud
/objects/3d/geometry/pointcloud/split.Name = &Split Point Cloud
/objects/3d/geometry/pointcloud/merge.Name = &Merge Multiple Point Clouds
/objects/3d/geometry/pointcloud/export.Name = &Export Point Clouds To File
/objects/3d/geometry/pointcloud/select.Name = Select &All Point Clouds
```

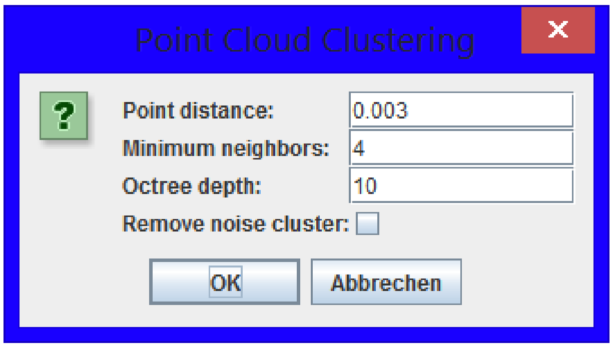
4.2.1 Importing point clouds

One or more point clouds can be imported from a file. The import function opens a file dialog to choose a point cloud file. Due to the fact that point cloud files do not have a common file name ending, the file name is not checked. If a directory, symlink or other file system object is selected, an error message is shown. After confirming the selected file, the file is read and all point clouds are imported as described in the chapter about the XL point cloud import function. Multiple point clouds are also supported here. If the points do not have a point cloud id, all points are added to one point cloud.

All imported point clouds are inserted into the scene graph and are selected automatically. Point clouds are always inserted as direct child elements of the scene graph root node (next to the RGG root node) to avoid dependencies between single point clouds. The imported point clouds have the newest (and highest) object IDs in the graph. This is useful if the objects are searched in the hierarchical object inspector window.

4.2.2 Clustering point clouds

To cluster a point cloud, exactly one point cloud must be selected first. Otherwise, an error message is shown and asks the user to select the targeted point cloud, including the deselection of all other objects. The clustering algorithm requires some parameters. They are requested with an own input dialog and validated afterwards. Additionally to the numeric parameters for the clustering algorithm, a boolean flag can be set to enable or disable the noise cluster.


Point Cloud Clustering

Point distance: 0.003
Minimum neighbors: 4
Octree depth: 10
Remove noise cluster: ☐

Figure 7: This is the input dialog window that requests the cluster parameters and provides the shown default values. Unfortunately, the text „Abbrechen“ is set automatically and does not fit to the GroIMP language in most cases. If the language on the operating system is English, „Cancel“ appears here instead.

The Java graphics library does not provide a numeric input field. This makes it possible to enter invalid texts in the numeric fields. They are validated when one of the buttons is clicked. If an invalid field was found and the button was not the cancel button, an error dialog window is shown and the parameter window appears again. The old values are stored and can be edited directly. If all values are valid and the clusterization should be started, the next window asks whether the old point cloud should be kept or removed. If it should be kept, the points of the new point clouds are at the same position as the points of the old point cloud. This may be a bit complicated to handle later. If the old point cloud should be removed, it is deleted from the RGG graph.

The created clusters are added to the scene graph as direct child elements of the scene graph root node (next to the RGG root node). This makes it possible to use the point clouds without dependencies between other ones. The clusters are of the same type „PointCloud“ as the old one and fully compatible to the other operations introduced in this document. The new point clouds have the highest (and newest) object IDs and can be found at the end of the RGG graph in the „Hierarchical Object Inspector“window.

In the parameter window, there was an option to automatically remove the noise cluster. The noise cluster is an additional cluster that contains all points without direct neighbor points. Normally, the bounding box of the noise cluster is larger than the bounding box of the union set of all normal clusters, because it contains the noise around all normal clusters. Due to this, it may be that the noise cluster is the only selectable object and all other point clouds are „hidden“ in it, because the noise points appear in nearly all directions and are nearly always „the most camera near“ points.

If the noise cluster is still there, but not needed anymore, it can be selected and removed with the belonging option in the window menu.

4.2.3 Splitting a point cloud

To split a point cloud, exactly one point cloud and one plane must be selected in the 3D selection. If the selection contains other objects or not the required ones, an error dialog window appears. If the selection is valid, a further dialog window appears and asks whether the old point cloud and the plane should be kept or removed. If this function was clicked accidentally, it can also be canceled here.

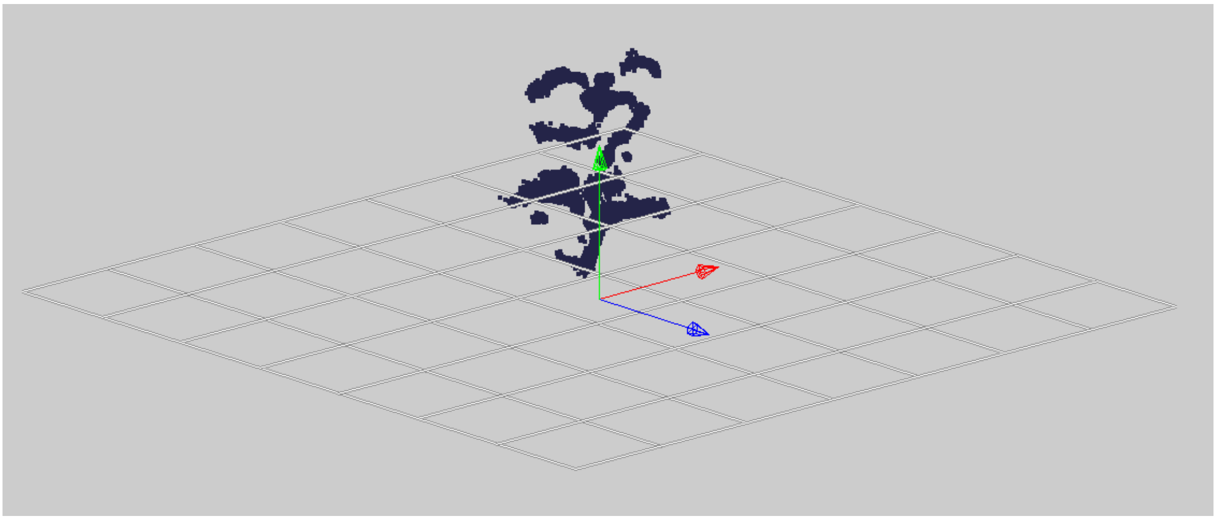


Figure 8: To split a point cloud, a plane is required. Here, the plane does not have a rotation. In general, the plane can be placed anywhere and have any required angle.

Now, the point cloud is split into two parts. Normally, the plane should be placed so that it cuts the point cloud into two subsets of points. For this, the plane must have been placed and rotated before by hand, with the attribute editor, or with an XL script. After starting the splitting process, two new point clouds are added as direct child elements of the root node in the scene graph. They have the newest (and highest) object IDs. In case that the plane does not cut the point cloud geometrically, one of the point clouds (front side or back side) remains empty.

If the old objects should be kept, it can be a bit confusing, because all points of both point clouds have the same positions as the points of the old point cloud. Additionally, the new point cloud that is displayed „behind“ the plane may be unselectable. This is due to the fact that the plane is displayed as bounded rectangle, but infinitely large in theory.

If the old objects should be removed, the old point cloud and the plane are removed from the RGG graph.

4.2.4 Merging point clouds

To use the merge function graphically, a sub set of point clouds must be selected. If no point clouds are in the current 3D selection or there are wrong objects contained, an error dialog window is shown. If the selection is a valid point cloud subset, a further request window appears and asks whether the old point clouds should be kept or removed.

The new point cloud is created so that cloned points of all points of all selected point clouds are added to the new point cloud. The points are copied points and have no dependency to the old ones and no reference to the source point cloud. The new point cloud is created so that all points have the same position as the points of the old points clouds. This can be a bit complicated, especially if the old point cloud is not removed. In that case, all points are on the same position and an external selection tool should be used to select the old or the new point cloud. If the old point clouds should be removed, they are deleted from the RGG graph. The „keep or remove“ window can also be used to cancel the merge function.

The new point cloud is added to the RGG graph and gets the next iterated object id. This makes it possible to easily find it in the hierarchical object inspector window. The chosen color is a random color, but can be changed in the attribute editor.

4.2.5 Exporting point clouds

The export function is the opposite of the import function. To export one or multiple point clouds, the targeted subset of point clouds must be selected in the 3D view. If there is no point cloud in the current selection or it contains wrong objects, an error dialog window is shown. If the selection only consists of point clouds, a file save dialog is opened.

In the file save dialog, a valid file name must be entered or an already existing file must be selected. If the file does not exist and there is no other file system object with that name, the file is created. Otherwise, if the file does already exist, the file is overwritten. It is recommended, but not required, to give the file a descriptive name and an obvious file ending, like „.xyz“. Because there is no default file name ending or standardized file format, the save dialog does not provide a standard file ending. This makes it possible to use a customized file name ending, depending on the own project or preferences.

4.2.6 Selecting all point clouds

The „Select All Point Clouds“ function is an additional feature. It does not change any geometrical or structural properties of point clouds. It is only used to select all point clouds that can be found by recursive search in the current RGG graph. For example, this function can be used to automatically select all clusters of a plant when the clusters should be exported to a file. The current number of selected objects, in this case the total number of existing point clouds, can be found in the attribute editor.

4.3 Example project

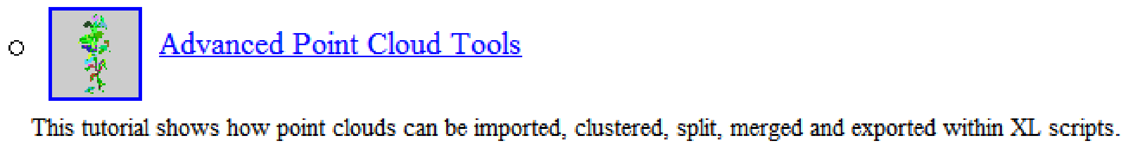
The added point cloud tools in the 3D object menu are relatively intuitive to use. In contrast to them, the available XL functions can not be found in GroIMP directly. To make it possible for GroIMP users to see how the functions work, a tutorial has been added that describes all available functions in detail. It can be accessed via the „Examples“ page. It is located in the category „Technical Models“ and is called „Advanced Point Cloud Tools“.

To add a project example is pretty easy and is described here in a few steps:

1. Create a new RGG project. This project will later be exported and used as example project.
2. Create a new `Model.rgg` file. This file contains the code that should appear in the integrated text editor.
3. Write the code and comments into the text editor and save the file.
4. Export the project as `.gsz` file. This is a project file and already contains the `Model.rgg`.
5. Create a screenshot that looks like the results of what the example project introduces and save it as square `.png` image with the width and height of 500 pixels.
6. Add the exported project file and the image file to the directory „Examples“ inside the project „plugins“. The files can then be accessed via the `examples.html` page.
7. Add the following HTML code at the correct position to the `examples.html` file in the same directory and adapt all file names to the concerning files:

```
<li>
   <a href="command:opendemo?pfs:PointCloudTools.gsz">
      <img
         src="PointCloudTools.png"
         alt"figure"
         width="50"
         height="50"
         hspace="10"
         vspace="5"
         align="middle">
   </a>
   <font size="+1">
      <a href="command:opendemo?pfs:PointCloudTools.gsz">
         Advanced Point Cloud Tools
      </a>
   </font>
   <br>
   This tutorial shows how point clouds can be imported,
   clustered, split, merged and exported within XL scripts.
</li>
```

The result can be found on the examples page:



Advanced Point Cloud Tools

This tutorial shows how point clouds can be imported, clustered, split, merged and exported within XL scripts.

Figure 9: This is the link to the point cloud tools tutorial. Here, it is called „Advanced“ because there is also a basic point cloud example.

5 Conclusion

5.1 Achieved goals

- The most important goal has been reached. All planned features could successfully be implemented and work as expected. This contains the possibility to import and export point clouds from and to files as well as clustering, splitting and merging them. All features are now available as XL functions and in the objects menu of the 3D view in GroIMP.
- The point cloud import function is fully compatible to old files. The accepted separator between numbers may be any combination of spaces and commas. This ensures that simple TXT files with spaces can be imported and CSV files with commas as separator characters can also be handled.
- The most important property of the import and export functions is the compatibility between both functions. Single point clouds are exported without point cloud ID information while lists of point clouds can be exported with point cloud ID information. This makes it possible to store a large number of point clouds in only one file. During the import, both file types are accepted. If the ID information is available, multiple point clouds are created. Otherwise, only one is returned.
- The algorithms to cluster, split and merge point clouds work reliably and are very efficient. This makes it possible to manipulate large point clouds without long waiting times.
- The transfer of the density–based clustering algorithm from ORIPUCA (C++) to GroIMP (Java) has worked well. The java algorithm seems to be a bit faster than the original one in C++. It could be that some details in the Java algorithm are more runtime–optimized than in the original one.
- The existing functions are not affected. This ensures that old XL scripts can still be used without any changes in the used functions.

5.2 Difficulties and solutions

There are many difficulties while developing new features for GroIMP. They are listed here and possible solutions are explained for most of them. This list can also be used as „list of experiences“ for future developers.

- The installation of GroIMP in Eclipse is a bit difficult. Depending on the used operating system and installed drivers, there are several errors and GroIMP is not able to start. An installation guide is now available in the wiki pages of the Gitlab repository of GroIMP to make the installation easier for new developers.
- Not all code can be used everywhere. The software is divided into multiple modules that depend on eachother, but without cycles. New features should be placed in a package that has all other required packages and classes available without dependency cycles.
- Lots of java methods are not implemented in the classes where they could be expected. It is only possible to find the targeted functions with a full text search tool (for example in Eclipse). Several other methods are missing or do not work reliably. In that case, it is only possible to add these functions on a place where they can globally be used in the future.
- Some functions are reliable when they are called once, but become unreliable when they are called multiple times. This effect occurs because some methods terminate before the expected behavior is executed. The idea behind this behavior is to save time on the main thread and execute time consuming operations on a parallel thread. The problem with this is that most of the affected objects will result in an uncoordinated end state and have incorrect properties after the concerning method is executed too many times in a certain time interval. The best solution here is to manually

write the own code with thread save algorithms or to implement a waiting loop to only execute the next method call after the expected behavior of the previous one can be detected.

- Lots of the already existing classes exist multiple times with the same or similar features. The most clear possibility to use classes with the same name is to not import any of them and add the full class descriptor in each occurrence of the class name instead.
- Some of the already existing classes do not provide functions with the expected behavior. Most of them are concepted for other usages and when new methods are added, the old ones and the new ones get into conflict because it is not clear which of them are made for the old concept and which of them are added for the new concept. In this case, it is easier to add a new class with new features to not disturb the existing one with its differing concept.
- It is difficult to keep the old methods working while adding new features. Because lots of java methods can be imported into XL scripts, their behavior may not be changed. Otherwise, it can be that XL scripts of other users do not work anymore.
- If multiple objects are added to the 3D view too fast, the multithreading algorithm fails and produces incorrect objects. This has the effect that some of the objects are not selectable anymore and are not added to the selection in the attribute editor, even if „all“ objects in the RGG graph are selected automatically.
- Point clouds and other objects that are added to the RGG graph with XL rules, are added as sub elements of other tree nodes. This leads to the side effect that objects depend on others. If some of them are selected and deleted for example, the sub elements are also affected. This leads to unexpected behavior, even if the user does not know the real RGG graph. Due to this problem, all new point cloud functions add implemented so that the new point clouds are added as direct child elements of the scene graph root node.
- Most of the existing code is not implemented in an intuitive way. Many java functions use too complicated operator combinations, variables without names and very confusing code with side effects that are really difficult to understand. This is the reason why new developers have to spend many hours or days to understand functions that could normally be understood in a few seconds.
- Large components in the code base are implemented with java objects without specific types. There are many abstract concepts, interfaces, not directly implemented features and many layers of reflection. The combination of all these properties makes it very hard to follow stack traces or java call hierarchies.
- There is not much documentation. The usage of most of the java methods can only be analyzed by reading the code directly. Because many of the functions are implemented a bit unintuitively, it becomes very time consuming to find usable methods for the development of new features.

5.3 Available documentation

As described in the preparation chapter, there is a tutorial in the belonging Gitlab repository available that describes in detail how GroIMP can be obtained from Git and installed in Eclipse. It can be found in the Wiki pages under „GroIMP Developer Guide“.

The most important available documentation for GroIMP developers (and maybe XL developers) is the java documentation. The whole code in the package `de.grogra.imp3d.pointcloud` is completely documented with JavaDoc. Even parameters, return values and throwable exceptions are documented to give future developers a better understanding about what the code does.

As an extra feature, an example project was added to the existing example projects inside GroIMP. It can be found directly under „PointCloud primitive“ and is called „Advanced Point Cloud Tools“. The example project is not directly executable, but contains lots of code snippets and comments about how to use the java functions in XL code.

At last, there is this practical report. It contains the planning, the implementation and lots of further experiences about the development of the new features in GroIMP.

5.4 Future possibilities

There are several possibilities what the results of this practical report can be used for in the future:

- Combination with other projects: Currently, there is a project with the goal to add more selection tools to GroIMP. This can be useful to select very complex subsets of points or point clouds.
- An interesting feature could be to group point clouds. Until now, they are only added as direct child elements to the scene graph root node to avoid dependencies between them. To group point clouds with common parent elements could be useful to structure plants or other objects.
- Until now, the import and export functionalities only work with the file format described in this document (*.xyz). More file formats could be introduced to expand the range of possibilities of storing different objects.
- In GroIMP, point clouds can not be converted to geometrical objects like cylinders or spheres. A useful feature would be a point cloud fitting algorithm with the possibility to detect which 3D object the point cloud looks like. This feature could also perform a parameter optimization to create fitting 3D objects for imported point clouds.

6 References

First version of the installation guide for GroIMP in Eclipse:
Title: GroIMP Eclipse Install Notes
Author: Michael Henke

Current version of the installation guide for GroIMP in Eclipse:
Author: Maurice Müller
Url: `https://gitlab.com/grogra/groimp/-/wikis/Developer%20Guide`

Original repository of GroIMP:
SourceForge: `https://sourceforge.net/projects/groimp/files/groimp/1.5/`
Gitlab: `https://gitlab.com/grogra/groimp/-/tree/master`

Java documentation of the GroIMP source code:
Url: `https://wwwuser.gwdg.de/~groimp/api/overview-summary.html`

Doctoral thesis as documentation of GroIMP:
Title: Design and Implementation of a Graph Grammar Based Language for Functional–Structural Plant Modelling
Author: Ole Kniemeyer
Published: Brandenburgische Technische Universität Cottbus-Senftenberg, 2008
Url: `https://opus4.kobv.de/opus4-btu/frontdoor/deliver/index/docId/462/file/thesis.pdf`

Clustering software ORIPUCA and example point clouds:
Title: Object recognition in point clouds using a density–based clustering algorithm
Author: Jörg Krause
Published: Brandenburgische Technische Universität Cottbus-Senftenberg, 2007

Density-Based Clustering Algorithm:
Title: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise
Authors: Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu
Published: Ludwig-Maximilians-Universität München, 1996
Url: `https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf`

Appendix A: Available java functions in XL

This appendix contains the list of available java functions. They can be used in the java code (by future developers) and in XL code (by GroIMP users). All functions can be used when the class `de.grogra.imp3d.pointcloud.PointCloudTools` is imported with the following java/XL command:

```
import de.grogra.imp3d.pointcloud.PointCloudTools;
```

All functions are **static** functions in the `PointCloudTools` class and **not** available as object based functions like `pointcloud.function()`.

The given point clouds and returned point clouds are of the type `de.grogra.imp3d.objects.PointCloud` and fully compatible to the other 3D objects.

All objects that are created by functions in the `PointCloudTools` class are **not** added to the RGG graph. They only exist as objects in the code context and can be added to the graph manually if required.

Importing point clouds

Point clouds can be imported from text files. The file path is expected as a string object and can be declared as a relative or absolute path. This function returns an array with all point clouds from the file. If the file does not contain point cloud ID information (and only declares one point cloud), the array will contain exactly one point cloud.

Declaration in PointCloudTools.java:

```
public static PointCloud[] importFromFile(String file) {
    ...
    return importedPointClouds;
}
```

XL code example for multiple point clouds:

```
String file = "tomato_plants.xyz";
PointCloud[] tomatoPlants = PointCloudTools.importFromFile(file);
```

XL code example for one point cloud:

If only one point cloud is stored in the file, the point cloud has the index `0`.

```
String file = "tomato_plant.xyz";
PointCloud[] imported = PointCloudTools.importFromFile(file);
PointCloud tomatoPlant = imported[0];
```

Clustering point clouds

The clusterization of point clouds requires some parameters. More information about the parameters `epsilon`, `minimumNeighbors` and `octreeDepth` can be found in the implementation part of this document. This function returns an array of newly created point clouds. The given point cloud is not affected.

If the `removeNoiseCluster` parameter is set to `true`, points with a too low number of neighbor points (= noise) are removed during the clusterization. If the parameter is set to `false`, an additional cluster (point cloud) is added to the end of the point cloud array. The additional point cloud contains all noise points. If the noise cluster is added and there are no noise points, the noise cluster remains in an empty point cloud.

Declaration in PointCloudTools.java:

```java
public static PointCloud[] cluster(PointCloud pointCloud, double epsilon, int minimumNeighbors, int
     octreeDepth, boolean removeNoiseCluster) {
    ...
    return pointCloudClusters;
}
```

XL code example:

```java
PointCloud tomatoPlant = ...;
double epsilon = 0.003;
int minimumNeighbors = 4;
int octreeDepth = 10;
boolean removeNoiseCluster = false;
PointCloud[] clusters = PointCloudTools.cluster(tomatoPlant, epsilon, minimumNeighbors, octreeDepth,
     removeNoiseCluster);
```

Splitting a point cloud

A point cloud can be split by a plane. The plane should have a position and a rotation so that the concerning point cloud can be split. This function returns an array with two point clouds. If the plane does not cut the point cloud, one of the resulting point clouds will be empty. This function does not affect the given objects.

Declaration in PointCloudTools.java:

```java
public static PointCloud[] split(PointCloud pointCloud, Plane plane) {
    ...
    return new PointCloud[]{frontPointCloud, backPointCloud};
}
```

XL code example:

```java
PointCloud twoPlants = ...;
Plane plane = new Plane();
plane.setRotate(0, Math.PI/2, 0);// vertical
PointCloud[] frontAndBackPointClouds = PointCloudTools.split(twoPlants, plane);
PointCloud firstPlant = frontAndBackPointClouds[0];
PointCloud secondPlant = frontAndBackPointClouds[1];
```

Merging point clouds

Multiple point clouds can be merged to a large one. This function returns a new point cloud with cloned points and does not affect the given objects. The source point cloud of each point is not stored.

Declaration in PointCloudTools.java:

```java
public static PointCloud merge(PointCloud[] pointClouds) {
    ...
    return mergedPointCloud;
}
```

XL code example:

```java
PointCloud[] lotsOfTooDetailedPointClouds = ...;
PointCloud unionSet = PointCloudTools.merge(lotsOfTooDetailedPointClouds);
```

Exporting point clouds

A certain selection of point clouds can be exported into a text file. The targeted point clouds must be provided as array and are not affected during the export. The file path is expected as a string object and can be declared as a relative or absolute path. If there is only one point cloud in the given array, the point cloud ID information is not stored in the exported file. If there are multiple point clouds in the array, the index of the point cloud in the array is used as point cloud ID and added to each point in the file.

Declaration in PointCloudTools.java:

```
public static void exportToFile(PointCloud[] pointClouds, String file) {
   PointCloudTools.writePointCloudFile(pointClouds, file);
}
```

XL code example for multiple point clouds:

```
PointCloud[] clustersOfPlant = ...;
String file = "plant_clusters.xyz";
PointCloudTools.exportToFile(clustersOfPlant, file);
```

XL code example for one point cloud:

To export one point cloud, an array with one element must be created.

```
PointCloud[] export = new PointCloud[1];
export[0] = pointCloud;
String file = "plant.xyz";
PointCloudTools.exportToFile(export, file);
```

Appendix B: Point cloud tools in the objects menu

All features in this appendix are available when a 3D project or an RGG project is opened in GroIMP. The functions work independently from XL scripts and are only based on the objects that are currently available in the 3D view.

The last used function is also listed in the tool bar below the menu bar. In this screenshot, the import function was recently used:

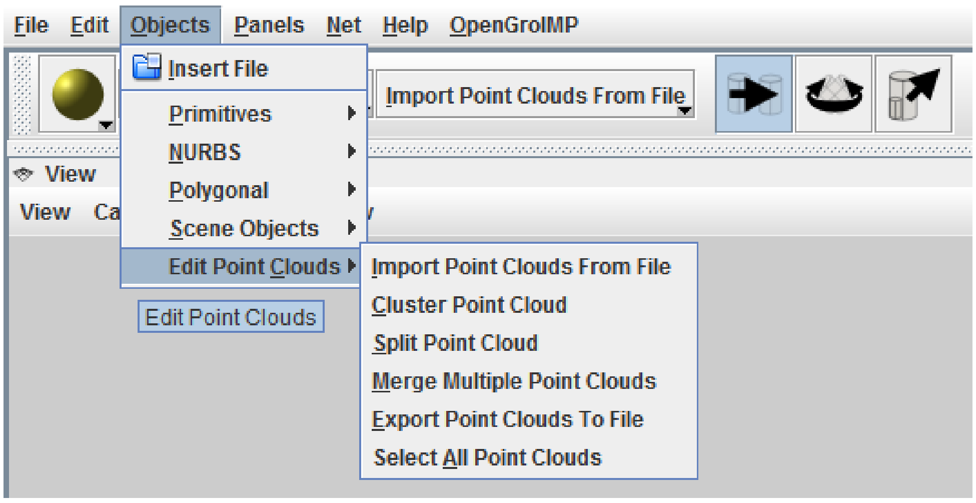


Figure 10: The point cloud menu is located inside the objects menu. The floating blue box „Edit Point Clouds“ is the tooltip text. Unfortunately, the mouse pointer was not captured by the screenshot software.

Warning: In the current version of GroIMP, the shortcut Alt+O does not work. This is due to the fact that 'O' is used twice and the wrong menu entry is focused by default. If the objects menu is already open, the other shortcuts (→ C → [custom key]) work as described.

Overview

Feature	Shortcut	Selection before	Selection after
Import file	Alt+O → C → I	-	Multiple point clouds
Cluster	Alt+O → C → C	One point cloud	Multiple point clouds
Split on plane	Alt+O → C → S	One point cloud, one plane	Two point clouds
Merge selected	Alt+O → C → M	Multiple point clouds	One point cloud
Export file	Alt+O → C → E	Multiple point clouds	-
Select all	Alt+O → C → A	-	Multiple point clouds

Detailed description

The **import** function shows a file open dialog. A compatible file must be selected. The file is read line by line and the point cloud(s) is/are imported. If an error occurs while reading the file, an error dialog window is shown. In case of success, all imported point clouds become selected afterwards. All imported point clouds are inserted as direct child elements of the scene graph root node.

The **cluster** function requests the required parameters with an own input dialog. The input fields are prefilled with example values that fit to the examples used in this practical report. The input dialog can only be confirmed if all fields have been filled correctly. Otherwise, an error dialog shows the cause of the error and returns to the input dialog. To use the clustering algorithm successfully, exactly one point cloud must be selected manually before. After confirming the input dialog a further dialog requests whether the existing point cloud should be kept or removed. Both dialogs have the possibility to cancel the operation. If the clustering was successful, all clusters become selected afterwards. The clusters are inserted as direct child elements of the scene graph root node.

The **split** function requires exactly one point cloud and one plane in the current 3D selection. If both are selected correctly, a dialog window requests whether the used objects should be kept or removed. After confirming one of the options, two new point clouds are added as direct child elements of the scene graph root node. If the splitting was successful, both point clouds are selected afterwards. If the plane did not cut the point cloud, one of the new point clouds remains empty.

The **merge** function requires multiple point clouds in the current 3D selection. Otherwise, an error dialog is shown. Another dialog requests whether the old point clouds should be removed or kept. The merged point cloud is a union set of all points of all selected point clouds. The union set point cloud does not have any dependencies to the old point clouds. The new point cloud is added as direct child element of the scene graph root node and selected afterwards.

The **export** function requires at least one point cloud in the current 3D selection. If only one point cloud is selected, the point cloud can be exported without point cloud ID information. Otherwise, the index in the array of the current selection is also stored in the exported file and can be used to distinguish between the exported point clouds. The export function opens a file save dialog. The file save dialog is able to overwrite existing files and to create new files. During the export, nothing happens to the selected point clouds.

The **select** function is an extra feature that makes it easier to handle a lot of point clouds. It does not change any geometrical or structural properties of any point clouds. Its only task is to select all point clouds that exist in the current scene graph. This function can be used if all currently visible point clouds should be merged or exported.