PlantScan3d Documentation

Release 1.5

Frederic Boudon, Julien Benoit

Jul 24, 2019

Contents

Version 1.5.0

Release 1.5

Date Jul 24, 2019

This manual details functions, modules, and objects included in PlantScan3D, describing what they are and what they do. For a complete reference guide, see refmanual.

> **Warning:** This "User Guide" is still very much in progress. Many aspects of VPlants.treeeditor3D are not covered.
>
> More documentation can be found on the openalea wiki.

CHAPTER 1

User guide

This is the documentation of how to use Plantscan3d.

1.1 Point cloud Treatment

This part explain how to manipulate a point cloud with Plantscan3d's tools.

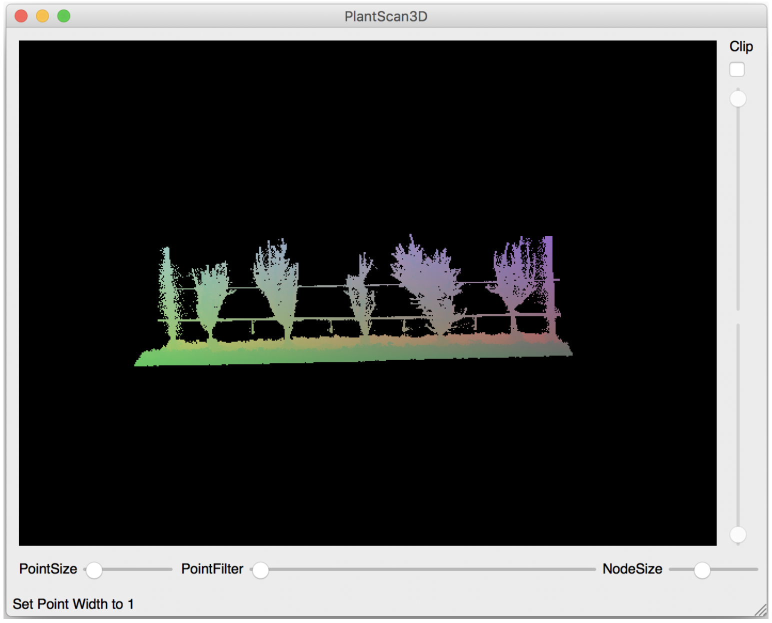
1.1.1 Base

This part explain how to use Plantscan3d.

Selection

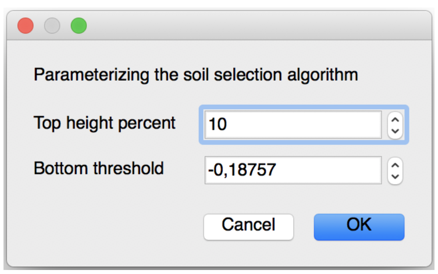
1.1.2 Cleaning Point cloud

This section explain how to use the tools to clean a point cloud and segment it. All algorithms not delete directly the points but select its with the selection system of PlanScan3d (see also: *Selection*).



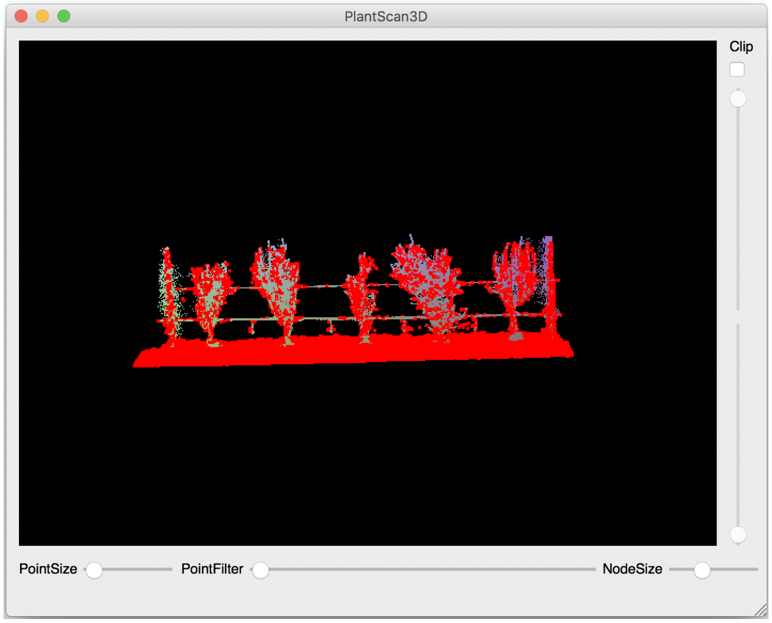
Soil Selection

The goal of this algorithm is to select the points that belongs to the soil and not keep the weed that do up to the soil (**Points -> Selection -> Soil**). To do that, we select a percent of points that are on the top of the scan (see below: the first parameter) and we go down through the point's neighborhood. For each point, we check its height and test if it is below a threshold (see below: the second parameter).

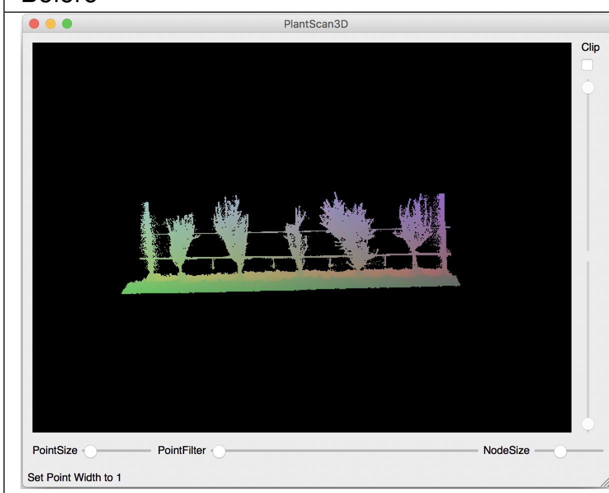
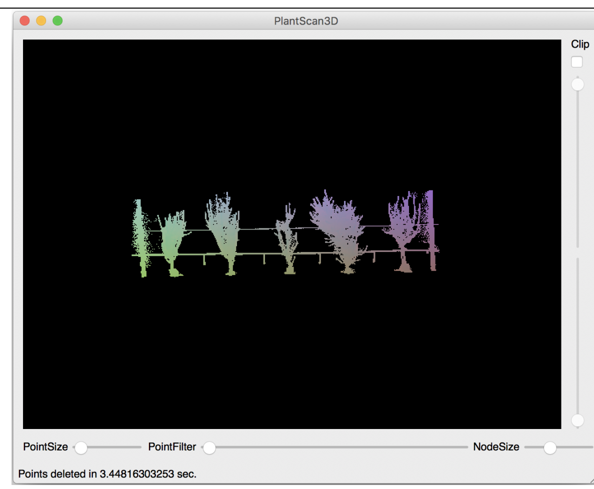


The parameters of this algorithm are :

- Top height percent : this parameter is the percent of the height of the scan that will be taken as start points. The default value is 10% but you can change it if you want but not exceed 70% because it's possible that the algorithm will keep the weed as start.
- Bottom threshold : this parameter control the height were the algorithm will stop go down, this parameter is estimated with the barycenter of the scan. We recommend to keep the value that is calculated.



You can notice that a few points above the soil are select, this is normal because there is isolated points.

Before	After
	

You can also test this algorithm in a python script (See: `point cloud used of this example`)

```
from openalea.plantgl.all import *

scene = Scene('winter.ply')
points = scene[0].geometry.pointList
topPercent = 10
minZ = points[points.getZMinIndex()].z
bottomThreshold = minZ + (points.getCenter().z - minZ) * 0.5

soil_indexes = select_soil(points, IndexArray(0), topPercent, bottomThreshold)

soil_points, other_points = points.split_subset(soil_indexes)
shape1 = Shape(PointSet(soil_points), Material('red', Color3.RED))
shape2 = Shape(PointSet(other_points), Material('blue', Color3.BLUE))
```

(continues on next page)

(continued from previous page)

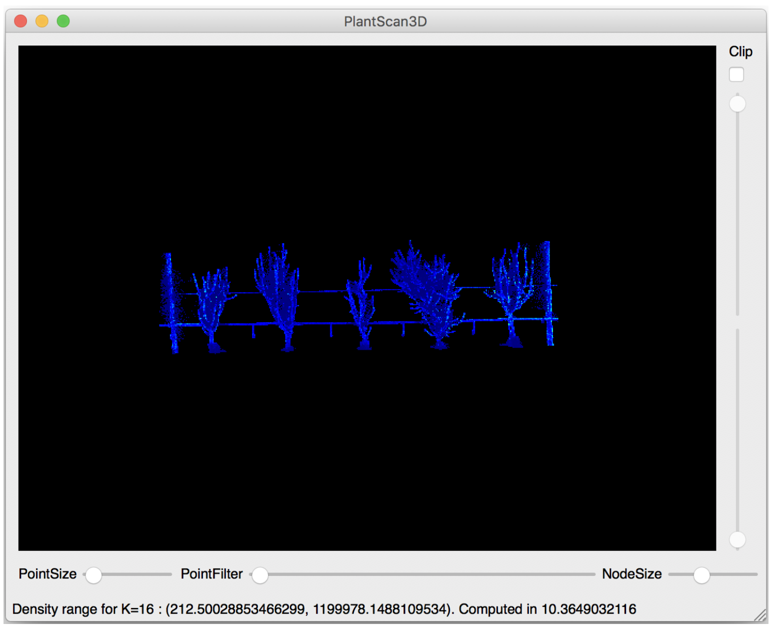
```
Viewer.display(Scene([shape1, shape2]))
```

Wires Selection

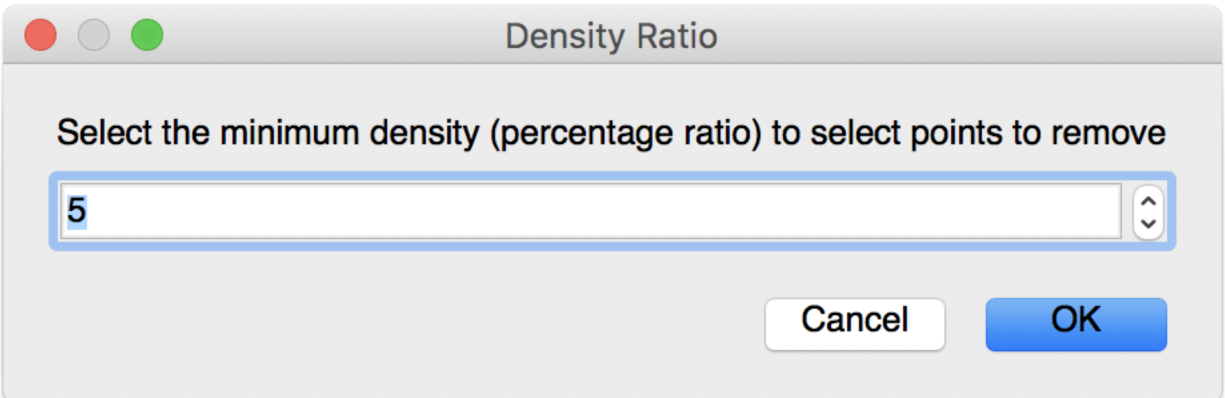
The second step of the treatment pipeline is the clean of the wires. The first things to do is to remove noise of the LIDAR, we calculate the densities of all points according of there neighborhood (**Points -> Density -> K-Density**).

The parameter of the k density is:

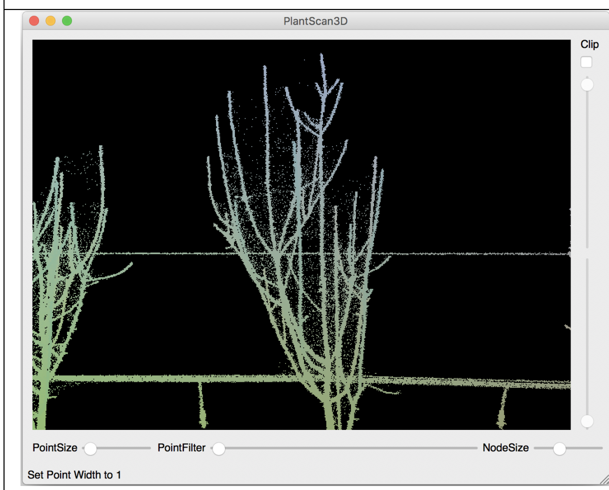
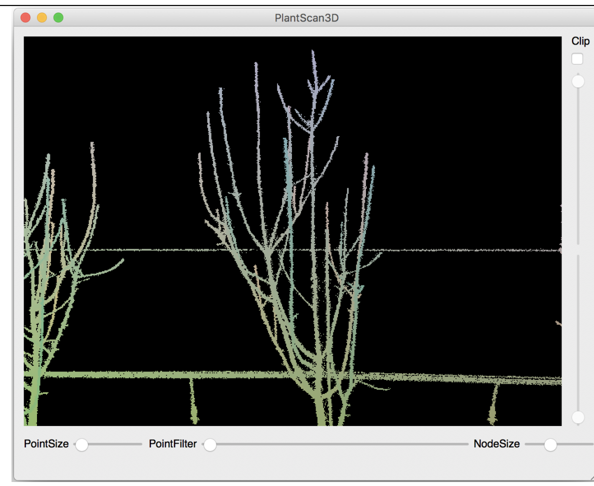
- k is the number of neighborhood that is calculate for one point. We recommend to use the default number (16 neighborhood).



The next step is to remove the points that have a low density, so we use the filter min algorithm (**Points -> Filter -> Filter Min Density**).



The single parameter of this algorithm is the percent of the low densities that will be deleted. The default value is 5 percent but I recommend to set 3 or 2 percent because 5 percent could delete too much points and the algorithm to select the wire could fail.

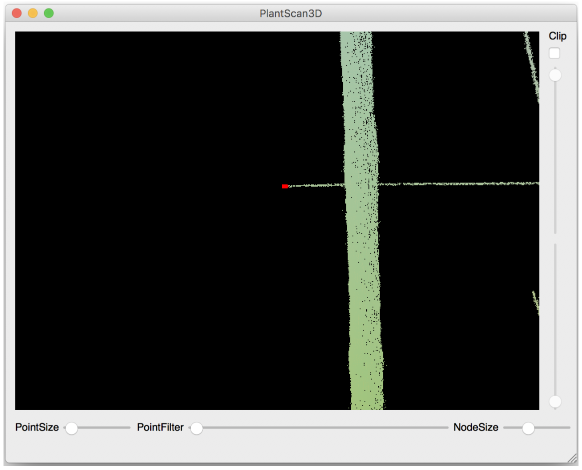
Before	After
	

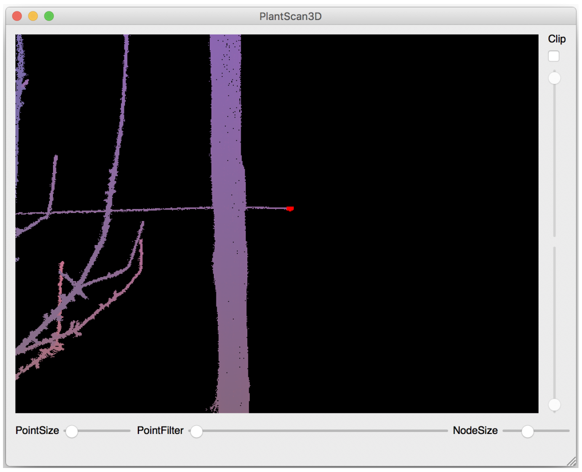
You can also test this algorithm in a python script (See: `point cloud used of this example`)

```
from openalea.plantgl.all import *

scene = Scene('winter_step_01.ply')
points = scene[0].geometry.pointList
k = 16
densityratio = 3

kclosests = k_closest_points_from_ann(points, k)
densities = densities_from_k_neighborhood(points, kclosests, k)

filter_indexes = filter_min_densities(densities, densityratio)

isolate_points, other_points = points.split_subset(filter_indexes)
shape1 = Shape(PointSet(isolate_points), Material('red', Color3.RED))
shape2 = Shape(PointSet(other_points), Material('blue', Color3.BLUE))

Viewer.display(Scene([shape1, shape2]))
```

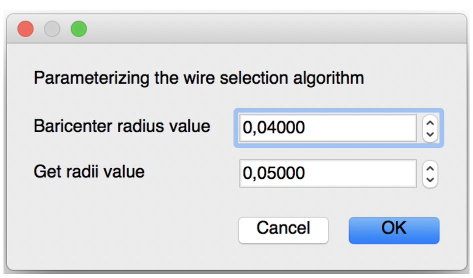
The next step is to select two extremities of the wire with the selection tool and validate the selection with the action

(**Points -> Selection -> Use selection for wire algorithm**).



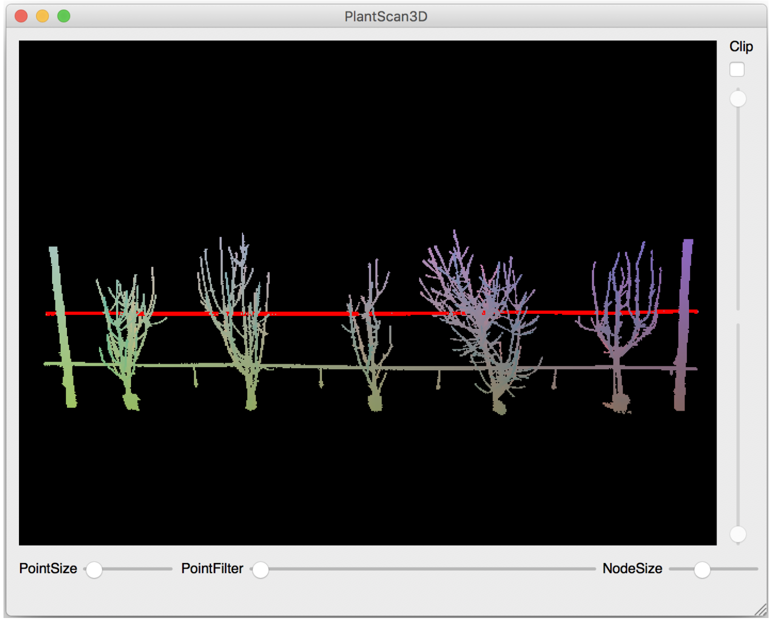


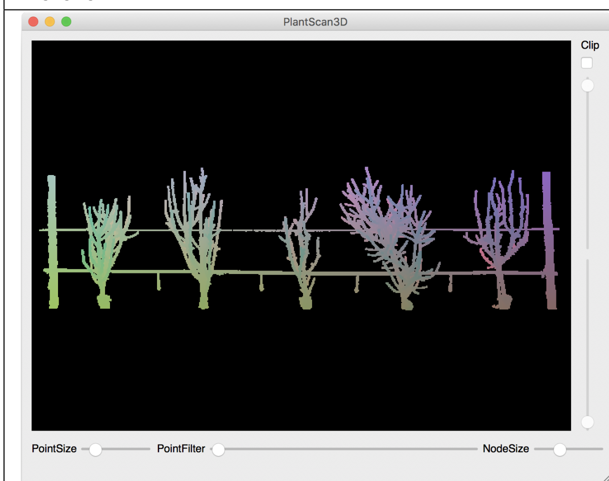
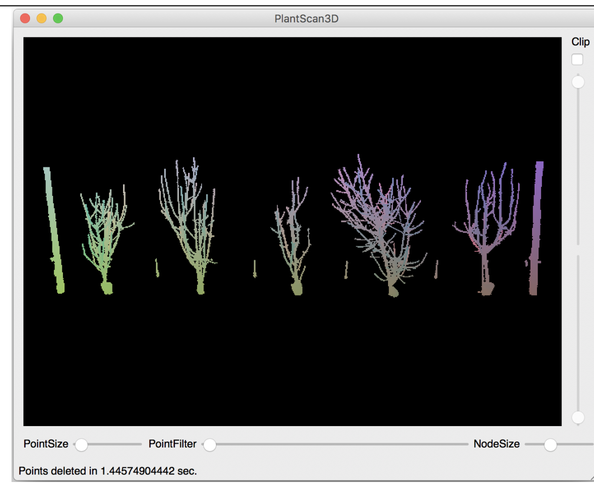
Next start the wire algorithm (Points -> Selection -> Wire). The algorithm calculate the shortest path between the extremities passing to the neighborhood and add a barycenter for each points of this path according to their neighborhood (see below: the first parameter). Next the algorithm estimate radii of the wire for each barycenters (see below: the second parameter) and take the points that is inside a cylinder between two barycenters with the radius.



The parameters of this algorithm are:

- Barycenter radius value : this value is the radius of the neighborhood for each point on the path.
- Get radii value : this is the radius of the neighborhood for each barycenters.



Before	After
	

You can also test this algorithm in a python script (See: `point cloud used of this example`)

```
from openalea.plantgl.all import *
import numpy

scene = Scene('winter_step_02.ply')
points = scene[0].geometry.pointList
Ymin, Ymax = points.getYMinAndMaxIndex()
bariRadius = 0.04
radiiValue = 0.05

kclosest_wire = IndexArray(0)
wire_path = get_shortest_path(points, kclosest_wire, Ymin, Ymax)
newpoint, baricenters = add_baricenter_points_of_path(points, kclosest_wire, wire_
↪path, bariRadius)
```

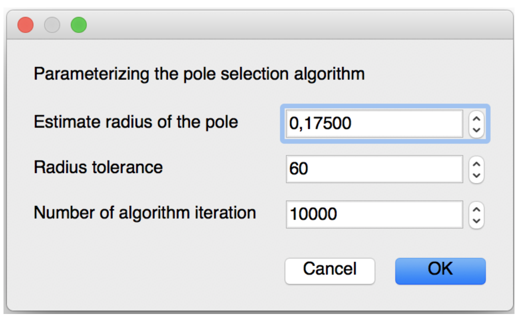
(continues on next page)

(continued from previous page)

```
kclosest = k_closest_points_from_ann(newpoint, 20, True)

radii = get_radii_of_path(newpoint, kclosest, baricenters, radiiValue)
radius = numpy.average(radii)

indexes = select_wire_from_path(newpoint, baricenters, radius, radii)
wire_indexes = Index([])

for i in indexes:
    if i not in baricenters:
        wire_indexes.append(i)

wire_points, other_points = points.split_subset(wire_indexes)
shape1 = Shape(PointSet(wire_points), Material('red', Color3.RED))
shape2 = Shape(PointSet(other_points), Material('blue', Color3.BLUE))

Viewer.display(Scene([shape1, shape2]))
```

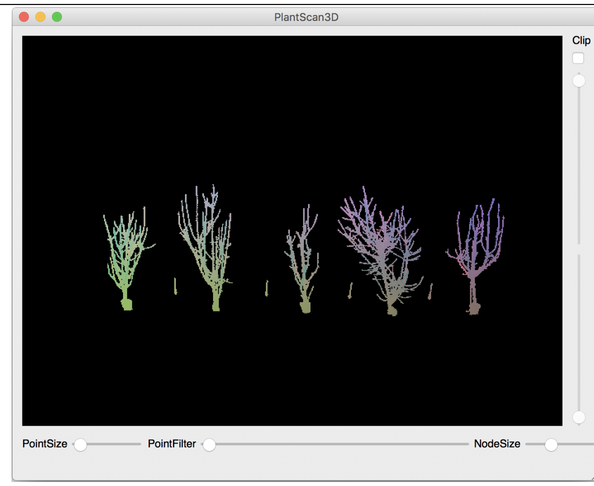
Poles Selection

The poles selection (**Points -> Selection -> Pole**) is base on a Ransac algorithm, it takes two points randomly in the point cloud, construct a cylinder with this two points (direction between the points and a constant radius (see below: the first parameter)), check the number of points inside it and calculate a score of it. This step is repeat x times (see below: the second parameter). Finally the algorithm take the cylinder with the best score.



The parameters are:

- Estimate radius of the pole : the radius of the cylinder created by the Ransac.
- Radius tolerance : the percent of tolerance inside and outside the cylinder to take the points.
- Number of algorithm iteration : this parameter change the number of cylinders generated before take the best one. This parameters impact the processing time.

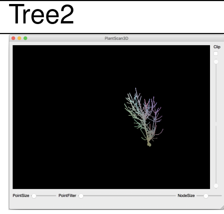
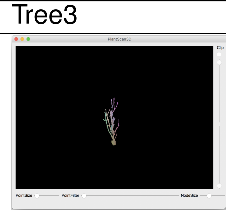
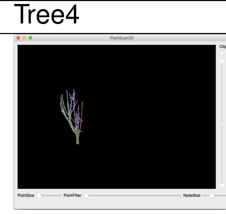
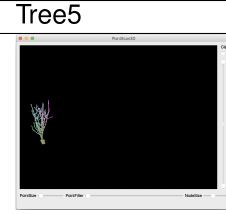
Before	After
	

You can also test this algorithm in a python script (See: `point cloud used of this example`)

```
from openalea.plantgl.all import *

scene = Scene('winter_step_03.ply')
points = scene[0].geometry.pointList
poleRadius = 0.175
tolerance = float(60) / 100
iteration = 10000

pole_indexes, score = select_pole_points_mt(self.points.pointList, poleRadius,
    iteration, tolerance)
print 'score = ' + str(score)

pole_points, other_points = points.split_subset(pole_indexes)
shape1 = Shape(PointSet(pole_points), Material('red', Color3.RED))
shape2 = Shape(PointSet(other_points), Material('blue', Color3.BLUE))

Viewer.display(Scene([shape1, shape2]))
```

Point cloud Segmentation

Here we segment the point cloud to get the trees separately (**Points -> Segment**). You can change to the next tree the next action (**Points -> Next Segmented Tree**). This algorithm is only base on the connex components so it not efficient.

Tree1	Tree2	Tree3	Tree4	Tree5
				

You can also test this algorithm in a python script (See: `point cloud used of this example`)

```
from openalea.plantgl.all import *
import random

scene = Scene('winter_step_04.ply')
points = scene[0].geometry.pointList

kclosest = k_closest_points_from_ann(points, 10, True)
connexsIndex = get_all_connex_components(points, kclosest)

connexPoints = []  # type: List[Point3Array]
for c in connexsIndex:
    if len(c) < 10000:
        continue
    connexPoints.append(points.split_subset(c)[0])

mats = []  # type: List[pglsg.Material]
while len(mats) != len(connexPoints):
    r = random.randrange(0, 256)
    g = random.randrange(0, 256)
    b = random.randrange(0, 256)
    color = Color3(r, g, b)
    inmats = False
    for m in mats:
        if m.ambient == color:
            inmats = True
            break
    if not inmats:
        mats.append(Material("mat" + str(len(mats)), color))

shapes = []  # type: List[pglsg.Shape]
for i in range(len(connexPoints)):
    shapes.append(Shape(PointSet(connexPoints[i]), mats[i]))

Viewer.display(Scene(shapes))
```

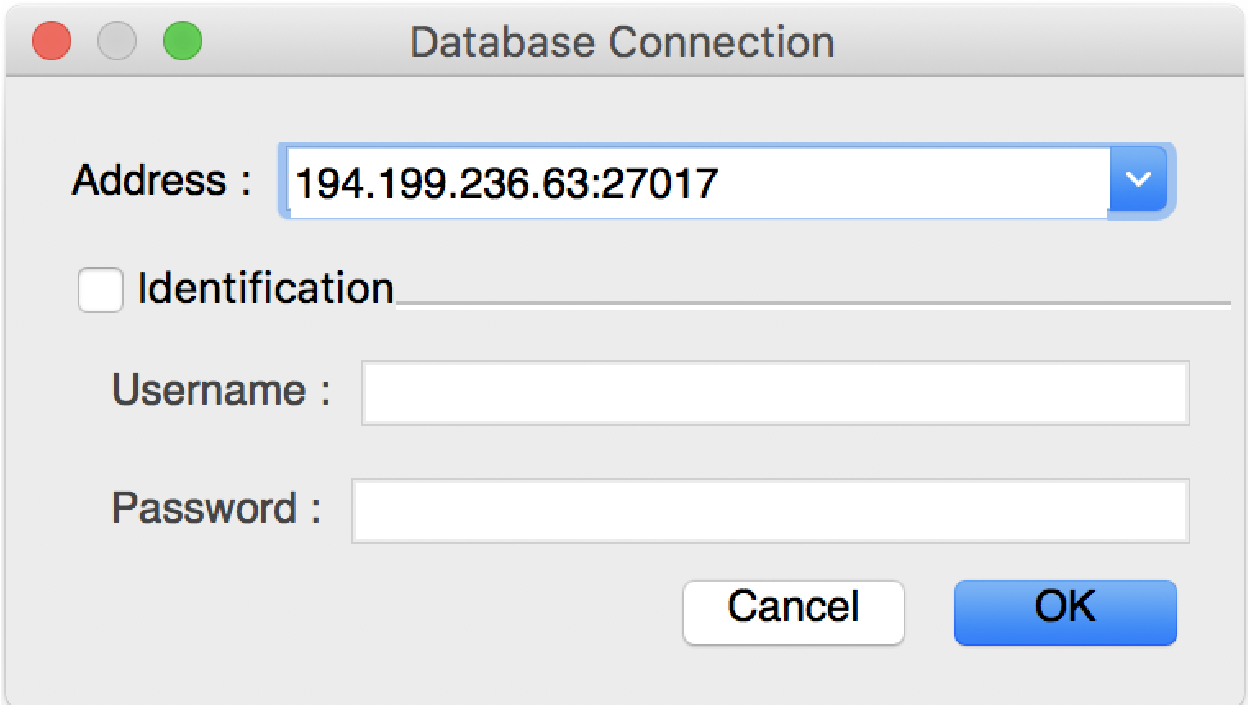
1.1.3 Reconstruction

This section talk about the reconstruction process.

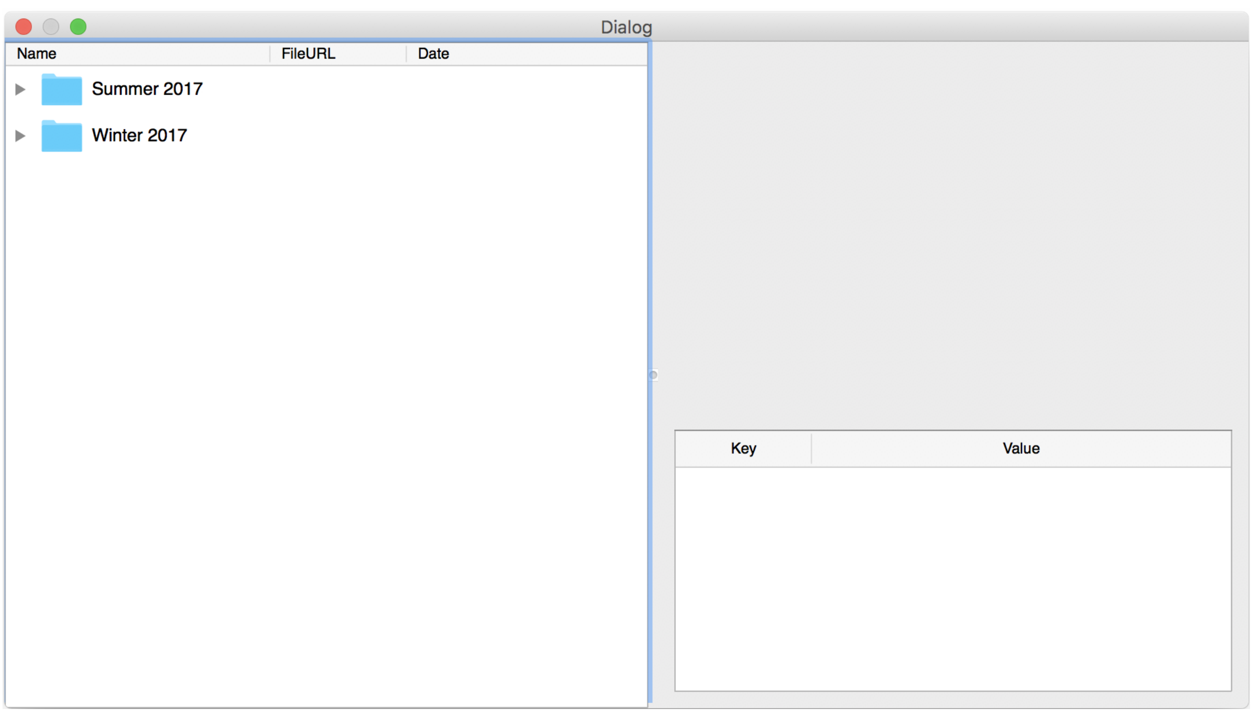
1.2 Database Browser

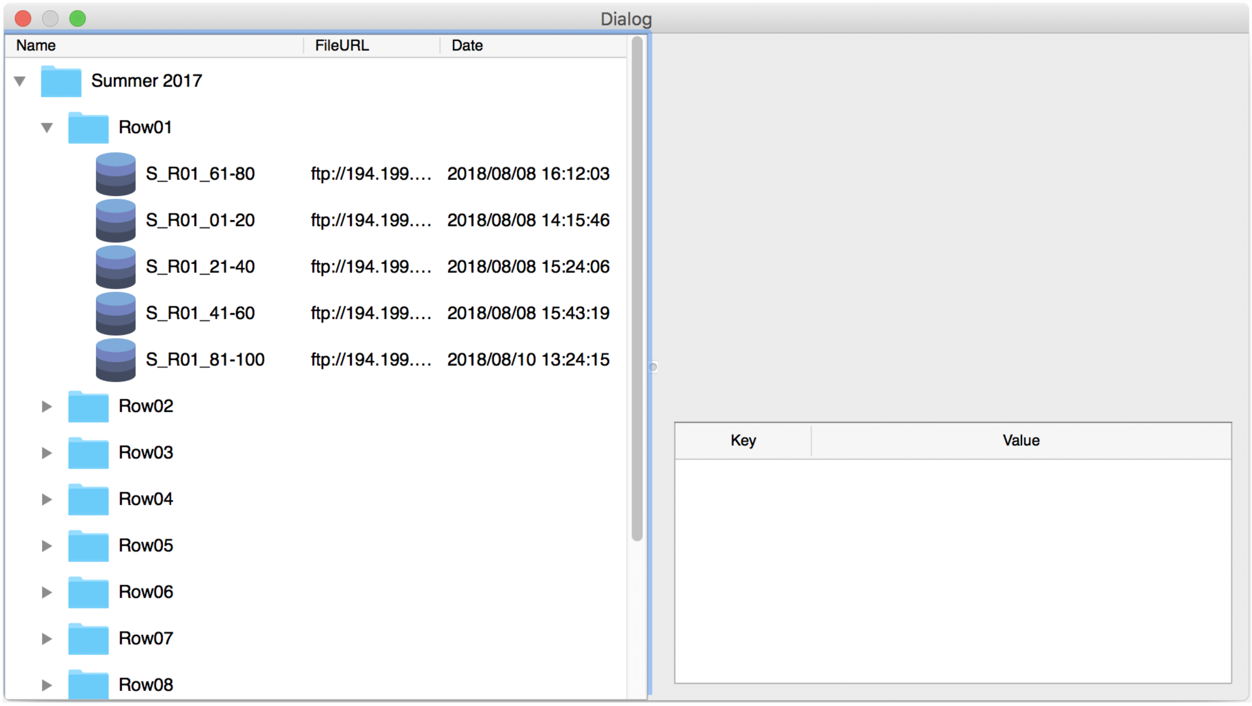
This part explain to use the database browser.

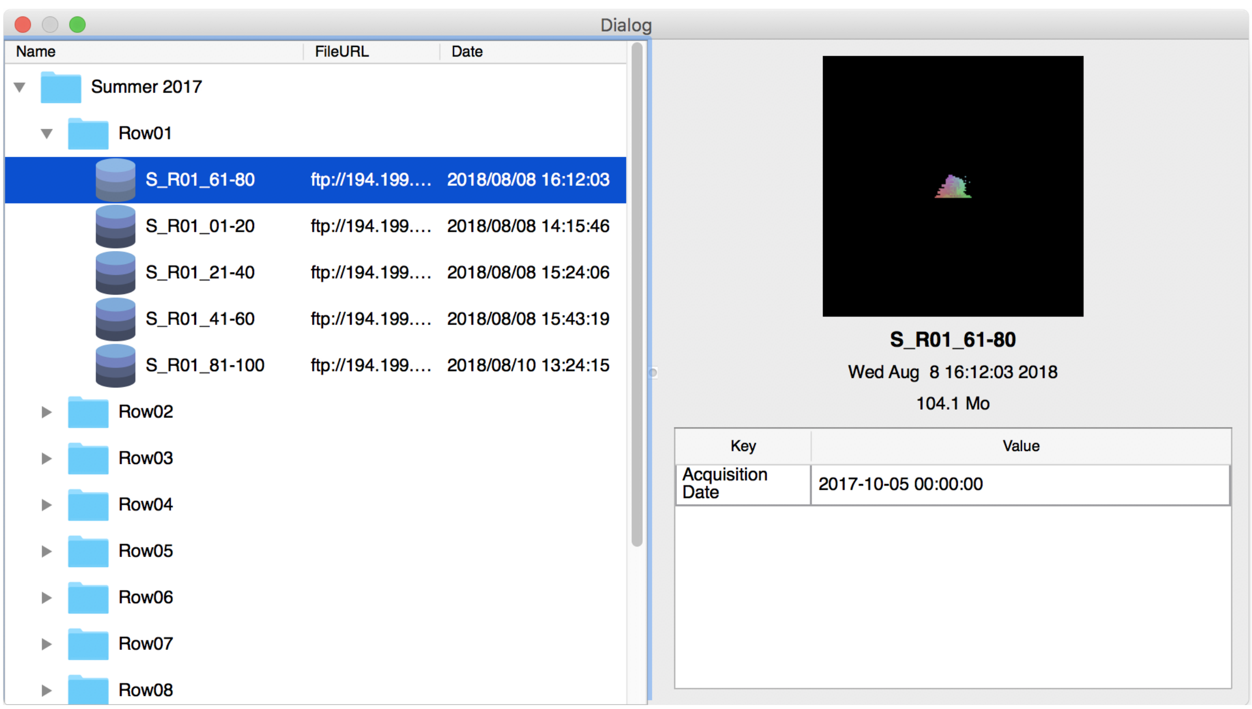
1.2.1 Database Connection



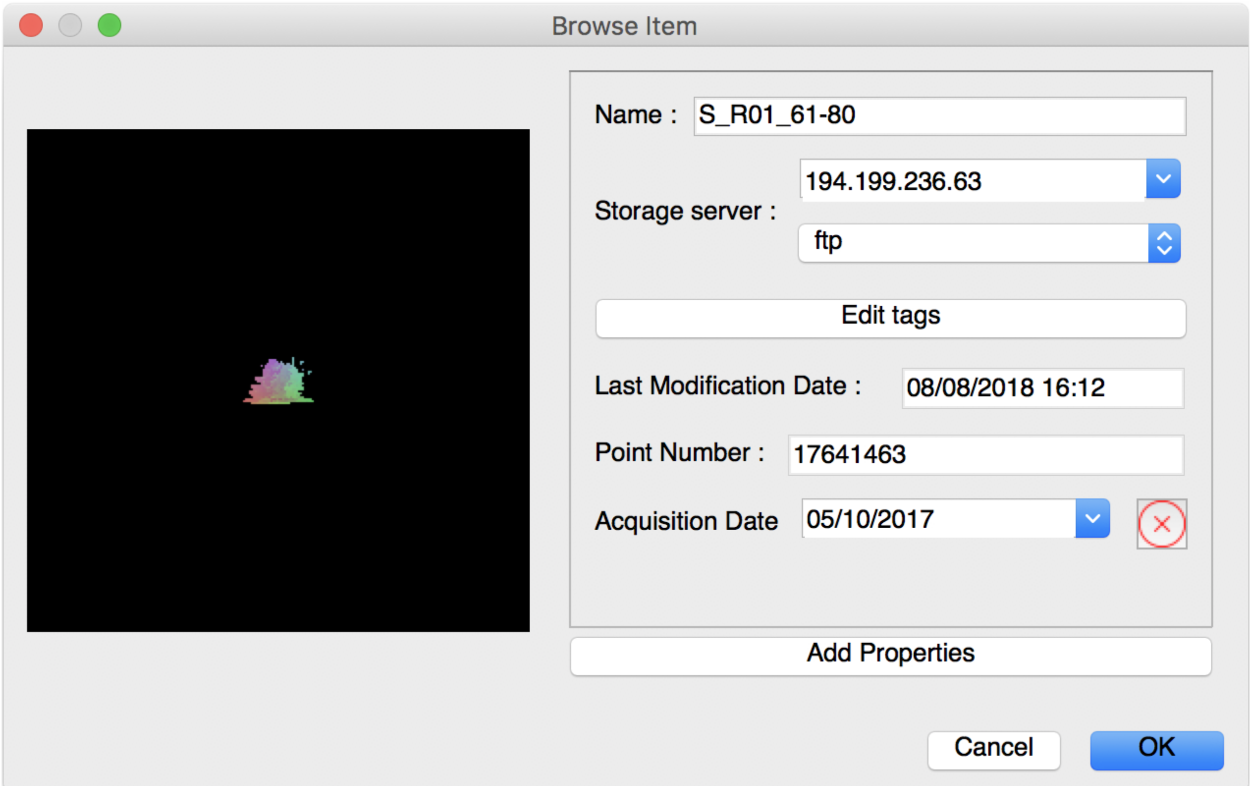
1.2.2 Browse Database

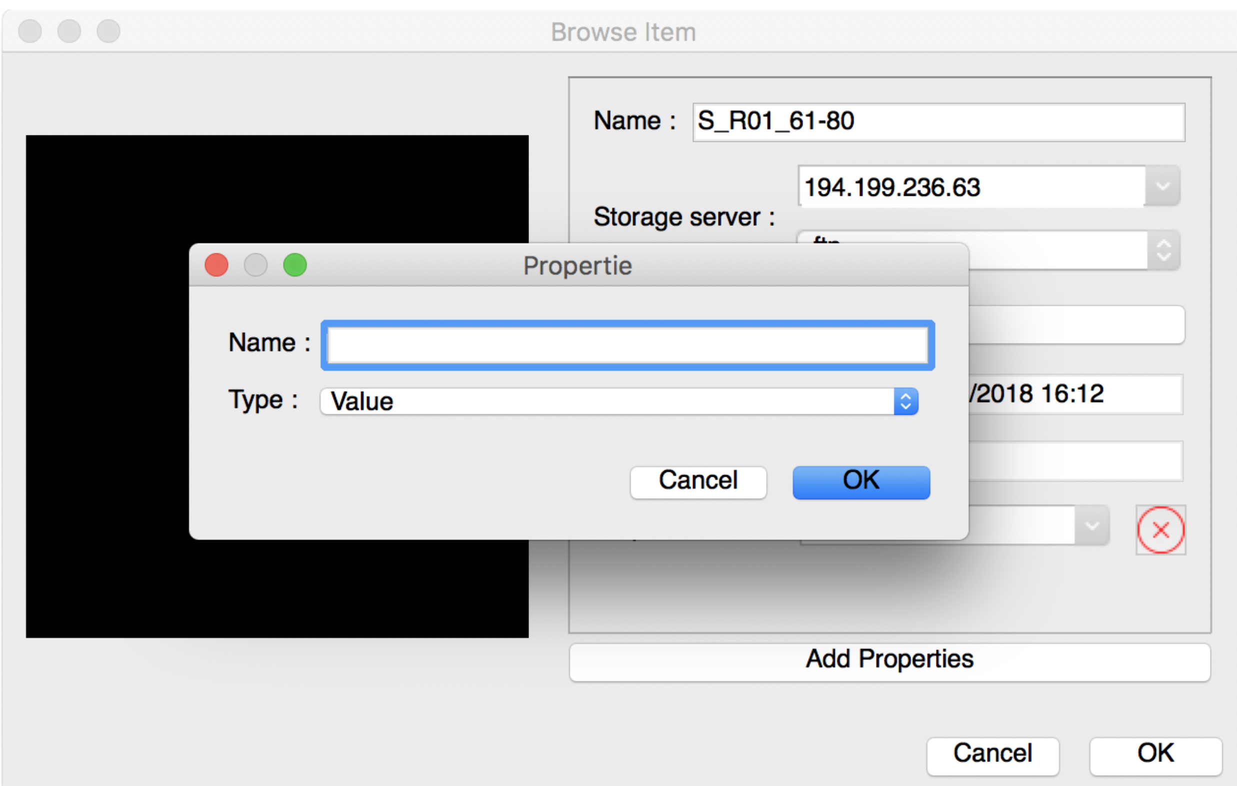


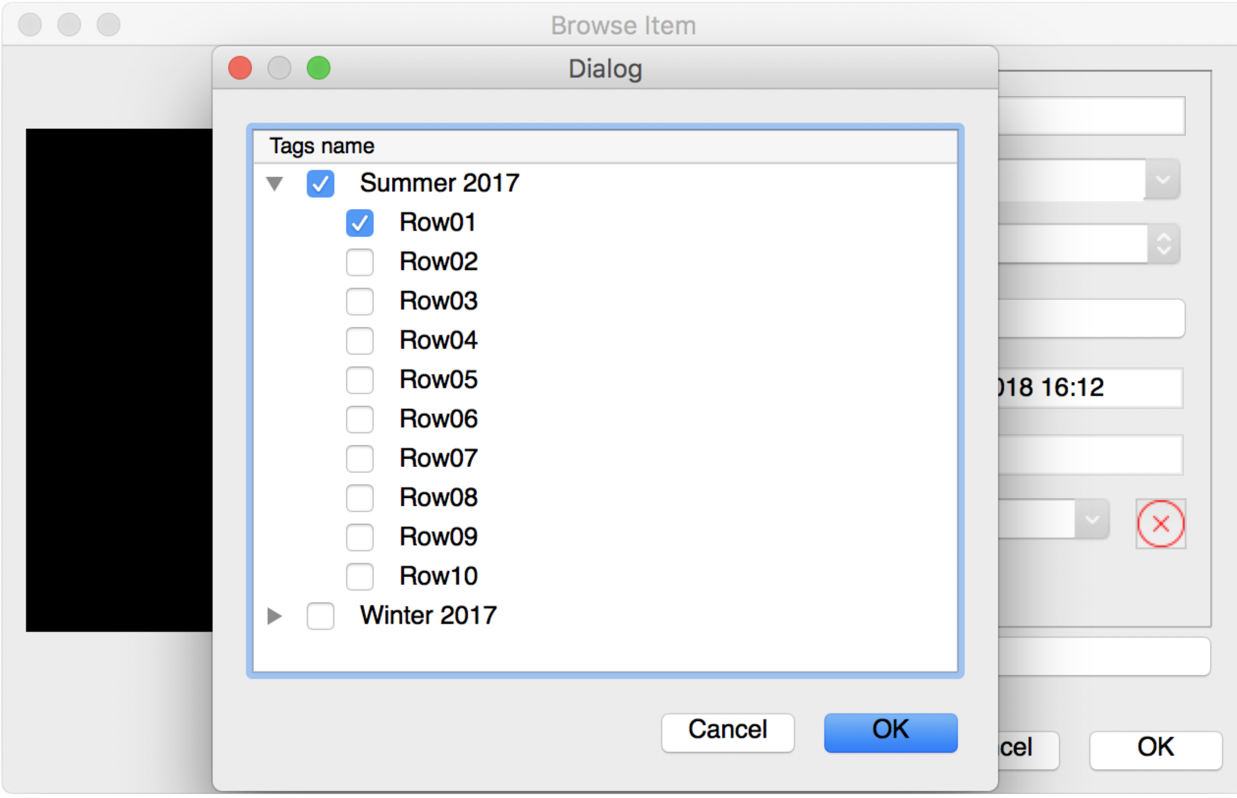

Dialog
Name
FileURL
Date
Summer 2017
Row01
S_R01_61-80 ftp://194.199.... 2018/08/08 16:12:03
S_R01_01-20 ftp://194.199.... 2018/08/08 14:15:46
S_R01_21-40 ftp://194.199.... 2018/08/08 15:24:06
S_R01_41-60 ftp://194.199.... 2018/08/08 15:43:19
S_R01_81-100 ftp://194.199.... 2018/08/10 13:24:15
Row02
Row03
Row04
Row05
Row06
Row07
Row08
Key
Value


Dialog
Name
FileURL
Date
Summer 2017
Row01
S_R01_61-80 ftp://194.199.... 2018/08/08 16:12:03
S_R01_01-20 ftp://194.199.... 2018/08/08 14:15:46
S_R01_21-40 ftp://194.199.... 2018/08/08 15:24:06
S_R01_41-60 ftp://194.199.... 2018/08/08 15:43:19
S_R01_81-100 ftp://194.199.... 2018/08/10 13:24:15
Row02
Row03
Row04
Row05
Row06
Row07
Row08
S_R01_61-80
Wed Aug 8 16:12:03 2018
104.1 Mo
Key
Value
Acquisition Date
2017-10-05 00:00:00

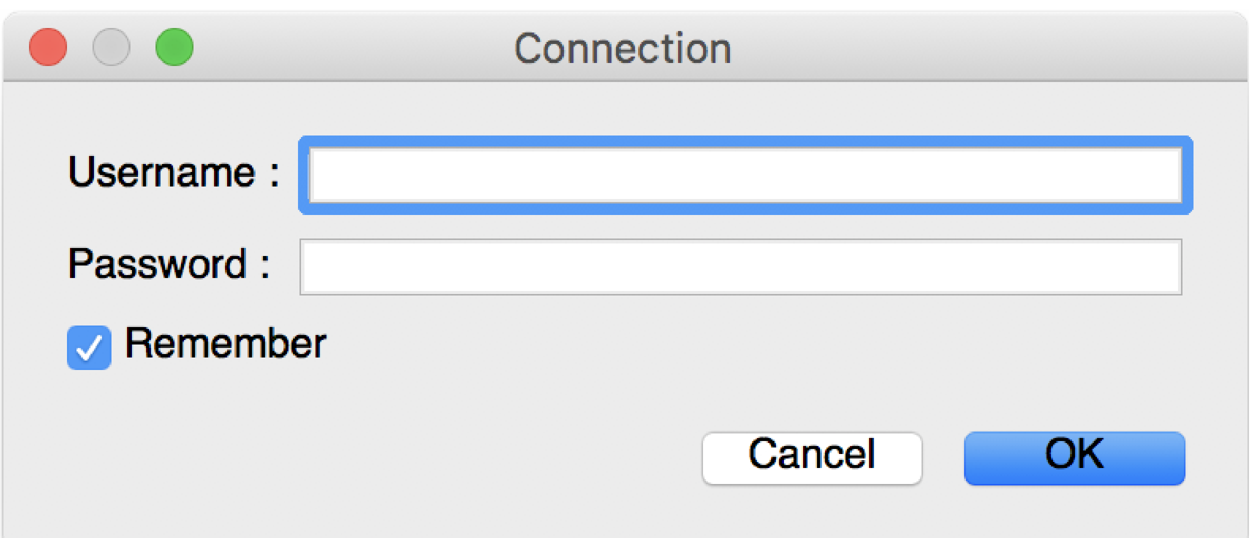
1.2.3 Database Item







1.2.4 Import from the Database



1.2.5 Export to the Database

1.2.6 Edit a Database Item

1.2.7 Delete a Database Item