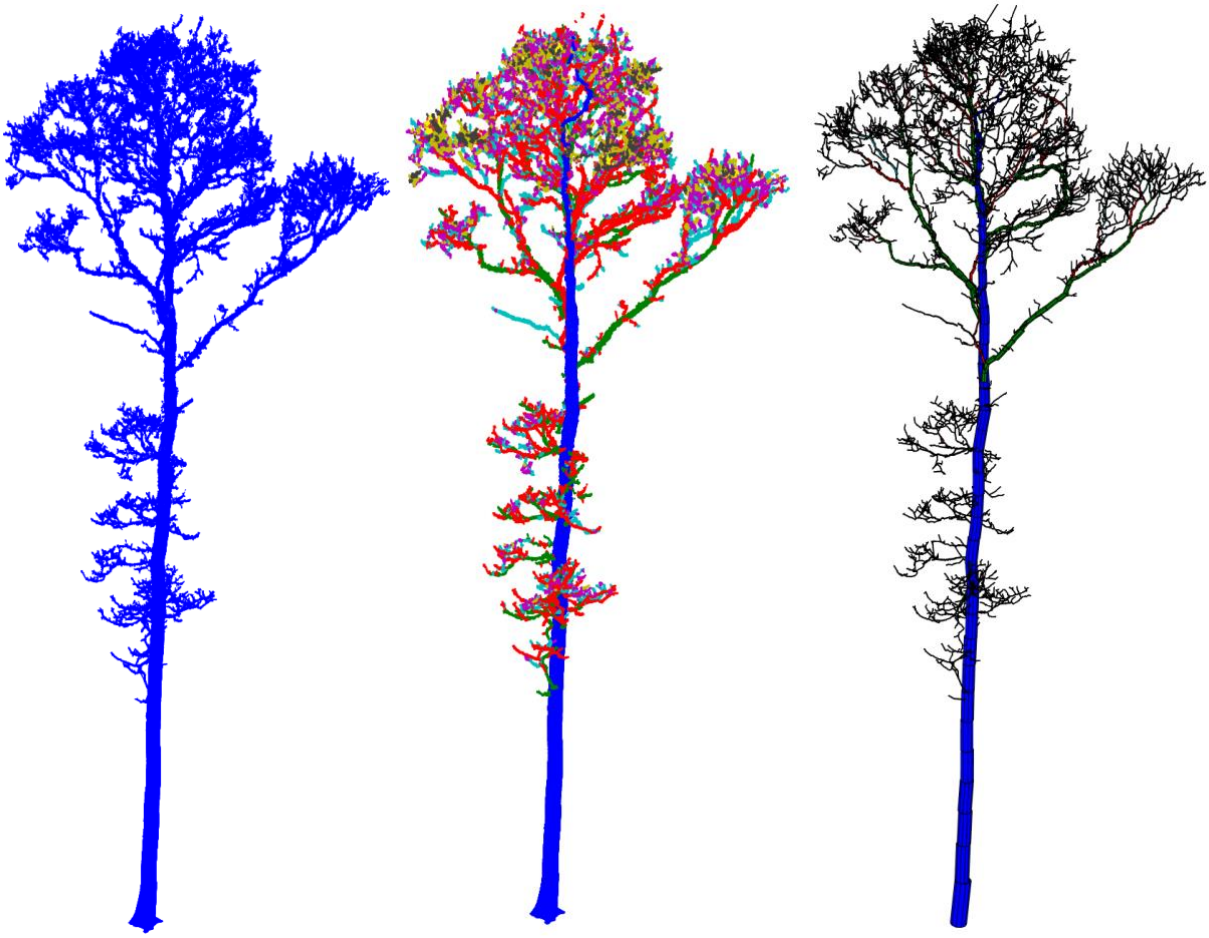
TreeQSM

Quantitative Structure Models of Single Trees from Laser Scanner Data



Instructions for MATLAB-software TreeQSM, *version 2.3.2*.
Document updated 3rd December 2019.

Author: Pasi Raumonen
Mathematics, Tampere University
Email: pasi.raumonen@tuni.fi
Web: http://math.tut.fi/inversegroup/
https://github.com/InverseTampere
Published papers: Raumonen et al. 2013, Remote Sensing
Calders et al. 2015, Methods in Ecology and Evolution
Raumonen et al. 2015, ISPRS Annals
Åkerblom et al. 2015, Remote Sensing

Point cloud data shown in many of the pictures in this document are from Eric Casella, Forest Research Agency, UK.

The software works at least with MATLAB version R2017b 64-bit (Mac OS X). Should work with newer versions and probably also with some of the older versions and also with Windows and Linux. The software may require some toolboxes that are not part of the basic MATLAB installation such as “stats”. If you encounter errors, please send notification of them to the above email, so that they can be corrected for new versions.

License

Copyright (C) 2013-2019 Pasi Raumonen

TREEQSM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

TREEQSM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with **TREEQSM**. If not, see http://www.gnu.org/licenses/

Table of Contents

1. Basics

1.1 What is quantitate structure model or QSM?

A QSM of a tree is a model of the *woody* structure of the tree that describes *quantitatively* its basic *topological* (branching structure), *geometric* and *volumetric* properties. These include properties such as number of branches in total and in any branching order, the parent-child relations of the branches and lengths, volumes, and angles of individual branches and branch size distributions. And there are countless other attributes and distributions that can be easily computed from a QSM.

A QSM consist of building blocks, which usually are some geometric primitives such as cylinders and cones. The circular cylinder is used here and it is the most robust choice and (in most cases) a very accurate choice for estimating diameters, lengths, directions, angles and volumes. A QSM consisting of cylinders (or other simple geometric primitives) offers a compact representation of the tree and as was described above it can store countless number of information about the tree.

1.2 MATLAB setup

1. Start MATLAB and set the main path to the root folder, where *treeqsm.m* is located.
2. Use *Set Path* → *Add with Subfolders* → *Open* → *Save* → *Close* to add the subfolders, where all the codes of the software are, to the paths of MATLAB.
3. Import a point cloud from a tree into the workspace. Let us name it P.

1.3 The basic command

First the basic command that produces QSMs is explained and then more details how the algorithm works is given. The basic command is:

```
QSM = treeqsm(P, inputs);

% P               (Filtered) point cloud, (m_points x 3)-matrix, the rows
%                       give the coordinates of the points.
% inputs          Structure array containing the input parameters
```

"inputs" structure is created by first modifying and then running the script `create_input`. However, if you are not already familiar with the **TREEQSM** (or its older versions), then it might be challenging to understand them and so it is advised that you first skip the next section and return to it later when you have a better understanding of the method.

1.4 Inputs structure

Next the `create_input` script is shown and the parameters briefly explained:

```
%% QSM reconstruction parameters
% The following parameters can be varied and should be optimised (they
% can have multiple values given as vectors, e.g. [4 6]):
inputs.PatchDiam1 = 0.1; % Patch size of the first uniform-size cover
inputs.PatchDiam2Min = [0.02 0.03]; % Minimum patch size of the cover sets
                                        in the second cover
```

```
inputs.PatchDiam2Max = 0.06; % Maximum cover set size in the stem's base in
                             the second cover
inputs.lcyl = [3 6]; % Relative (length/radius) length of the cylinders
inputs.FilRad = 3; % Relative radius for outlier point filtering

% The following parameters can be varied and but usually can be kept as
% shown (i.e. little bigger than PatchDiam parameters):
inputs.BallRad1 = inputs.PatchDiam1+0.01; % Ball radius in the first
                                          uniform-size cover generation
inputs.BallRad2 = inputs.PatchDiam2Max+0.01; % Maximum ball radius in the
                                          second cover generation

% The following parameters can be usually kept fixed as shown:
inputs.nmin1 = 3; % Minimum number of points in BallRad1-balls, generally
                    good value is 3
inputs.nmin2 = 1; % Minimum number of points in BallRad2-balls, generally
                    good value is 1
inputs.OnlyTree = 1; % If 1, point cloud contains points only from the tree
inputs.Tria = 1; % If 1, produces a triangulation
inputs.Dist = 1; % If 1, computes the point-model distances

% Different cylinder radius correction options for modifying too large and
% too small cylinders:
% Traditional TreeQSM choices:
inputs.MinCylRad = 0.0025; % Minimum cylinder radius, used particularly in
                             the taper corrections
inputs.ParentCor = 1; % Radii in a child branch are always smaller than the
                        radii of the parent cylinder in the parent branch
inputs.TaperCor = 1; % Use partially linear (stem) and parabola (branches)
                       taper corrections
% Growth volume correction approach introduced by Jan Hackenberg,
% allometry: GrowthVol = a*Radius^b+c
inputs.GrowthVolCor = 0; % Use growth volume (GV) correction
inputs.GrowthVolFac = 2.5; % fac-parameter of the GV-approach, defines upper
            and lower bound. When using GV-approach, consider setting:
            TaperCorr = 0, ParentCorr = 0, MinCylinderRadius = 0.

%% Other inputs
% These parameters don't affect the QSM-reconstruction but define what is
% saved, plotted, and displayed and how the models are named/indexed
inputs.name = 'pine'; % Name string for saving output files and naming
                        models
inputs.tree = 1; % Tree index. If modelling multiple trees, then they can be
                   indexed uniquely
inputs.model = 1; % Model index, can separate models if multiple models with
                    the same inputs
inputs.savemat = 1; % If 1, saves the output struct QSM as a matlab-file
                      into \result folder
                    % If name = 'pine', tree = 2, model = 5,
                    % the name of the saved file is 'QSM_pine_t2_m5.mat'
inputs.savetxt = 1; % If 1, saves the models in .txt-files
inputs.plot = 1; % If 1, plots the model, the segmentation of the point
                   cloud and distributions
inputs.disp = 2; % Defines what is displayed during the reconstruction:
                 2 = display all; 1 = display name, parameters and
                 distances; 0 = display only the name
```

1.5 Filtering point cloud

The method assumes that most of the points are part of the stem or branches and thus it tries

to reconstruct QSMs using them. If, however, there are lot of points from leaves or noise or "phantom points" that are not part of the woody structure of the tree, then to ensure a good reconstruction most of these points should be removed before QSM reconstruction. Sometimes it is thus better first filter the point cloud. Simple filtering based on local volume point density and separate cluster size can be tried with the following command:

```
Pass = filtering(P0, r1, n1, d2, r2, n2, Scaling, AllPoints);
P = P0(Pass,:);
```

P0	Unfiltered point cloud
r1	Radius of the balls used in the first filtering, defines the volume
n1	Minimum number of points in the accepted balls of the first filtering, defines the point density together with r1
d2	Minimum distance between the centers of the balls in the second filtering
r2	Radius of the balls used in the second filtering
n2	Minimum number of balls in the components passing the second filtering
Optional inputs, default value false:	
Scaling	If true, the first filtering threshold "n1" is scaled along the height with average point density
AllPoints	If true, does the first filtering process for every point

The radii and distances (r1, r2, d2) in the same units as the point cloud. Here the output of the `filtering` is a logical vector describing which points pass the filtering. Notice that the filtering uses covers, which are randomly generated, meaning that the results are little different for every run, even with the same input parameters. Notice also that the point cloud *can* contain some points from the ground and understory without that causing a problem in the QSM reconstruction. Finally, the `filtering` function may not work well in many cases, particularly it usually cannot properly remove leaves, and the user can use some other filtering method.

1.6 Test run

```
>> QSM = treeqsm(P,inputs);
---------------
test
PatchDiam1 = 0.05, BallRad1 = 0.06, nmin1 = 3
PatchDiam2Min = 0.01, PatchDiam2Max = 0.03, BallRad2 = 0.04, nmin2 = 1
lcyl = 6, FilRad = 3.5, Tria = 0, OnlyTree = 1
Progress:
Cover sets       3.5 sec.   Total: 3.5 sec
Tree sets        0.1 sec.   Total: 3.6 sec
Initial segments 1.3 sec.   Total: 4.9 sec
Final segments   0.7 sec.   Total: 5.6 sec
Cover sets       26.4 sec.  Total: 32 sec
Tree sets        17 sec.    Total: 48.9 sec
Initial segments 34.4 sec.  Total: 1 min 23.4 sec
Final segments   9.3 sec.   Total: 1 min 32.7 sec
Cylinders        19.5 sec.  Total: 1 min 52.2 sec
------------
  Tree attributes:
  TotalVolume = 925.6 L
```

TrunkVolume = 601 L
BranchVolume = 324.6 L
TreeHeight = 19.26 m
TrunkLength = 19.49 m
BranchLength = 664.3 m
NumberBranches = 1940
MaxBranchOrder = 8
TotalArea = 55.28 m^2
DBHqsm = 26.01 cm
DBHcyl = 26.05 cm

Branch & data 0.4 sec. Total: 1 min 52.6 sec
Average cylinder-point distance: 5.6 7.7 5 6.3 mm
Distances 1.5 sec. Total: 1 min 54.1 sec

Figures 1 and 2 show examples of QSMs.

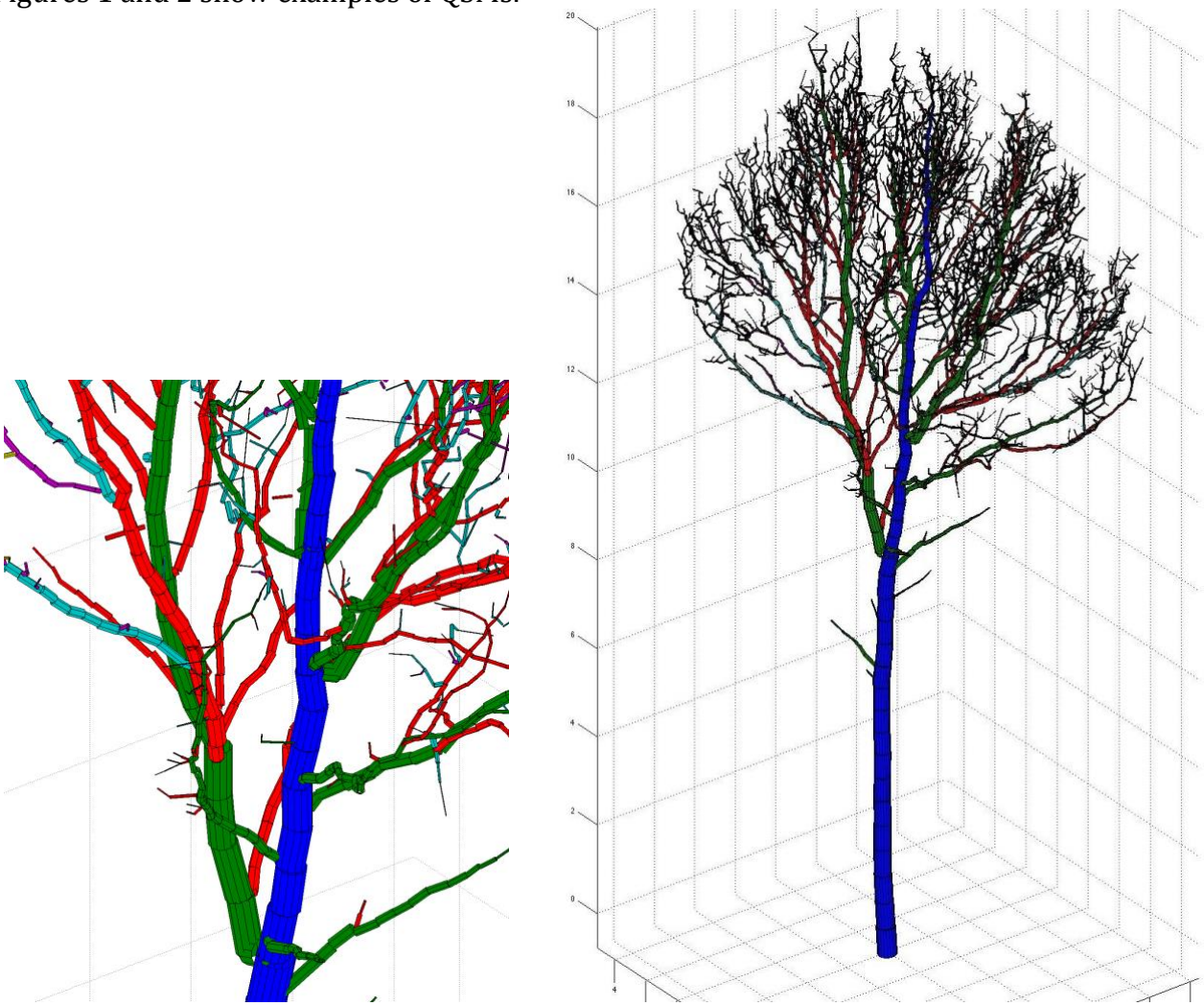


Figure 1. Example of cylinder-based QSM. The color denotes the branching order: Blue = trunk, green = 1st-order branches, red = 2nd-order branches, etc.

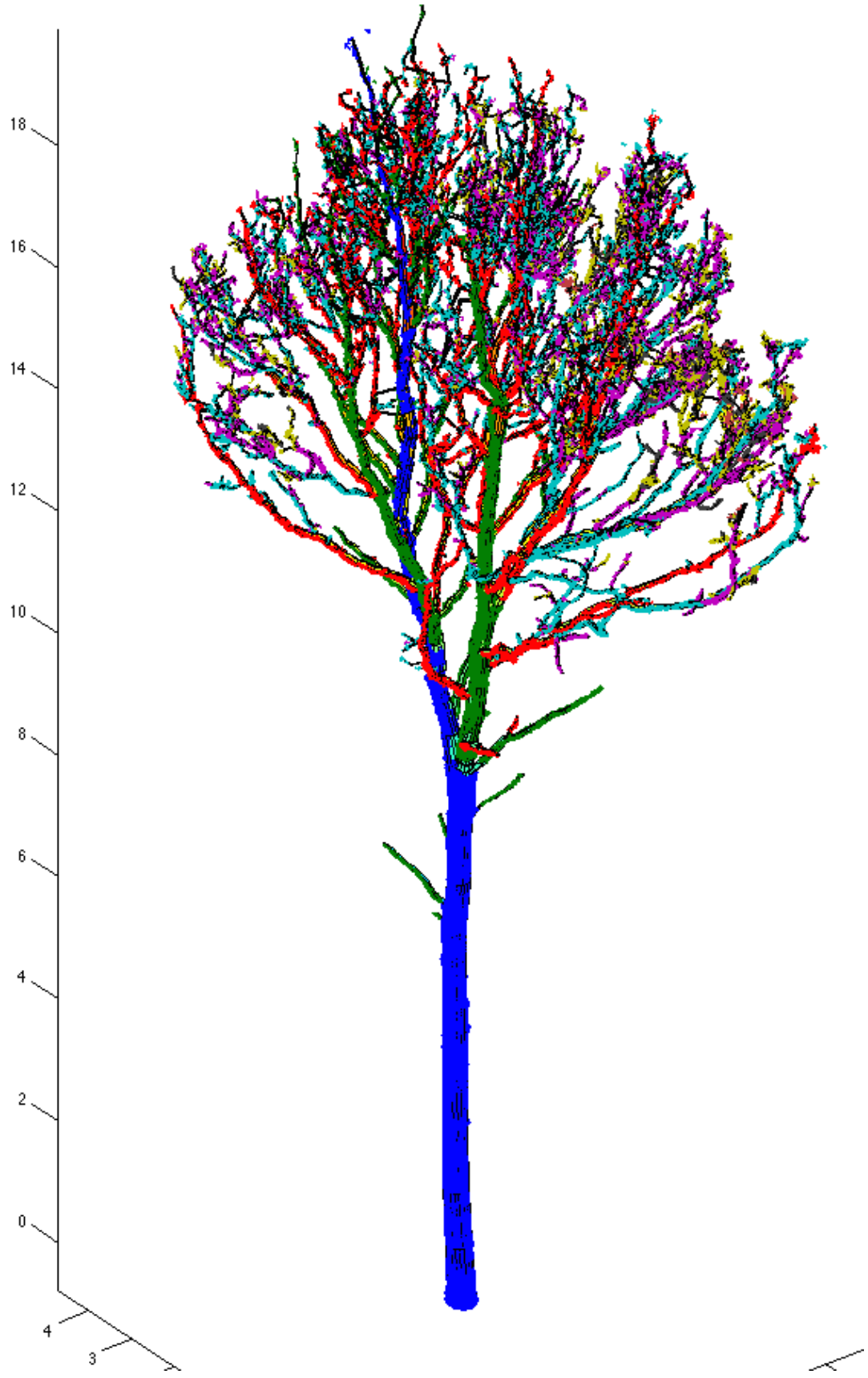


Figure 2. QSM and segmented point cloud. The color indicates the branching order: Blue = trunk, green = 1st-order branches, red = 2nd-order branches, etc.

1.7 Output structure

The output `QSM` is a structure array. It contains the structures: “cylinder”, “branch”, “treedata”, “rundata”, “pmdistance” and “triangulation”. The actual content of these is explained next.

1.7.1 cylinder

The cylinder structure contains info of each cylinder, whose name/index is the row number in the file, and the structure contains the following fields:

```
>> QSM.cylinder

          radius: [9135×1 single]
          length: [9135×1 single]
           start: [9135×3 single]
            axis: [9135×3 single]
          parent: [9135×1 uint16]
       extension: [9135×1 uint16]
           added: [9135×1 logical]
     UnmodRadius: [9135×1 single]
          branch: [9135×1 uint16]
     BranchOrder: [9135×1 uint8]
  PositionInBranch: [9135×1 uint8]
```

radius	radius of the cylinder
length	length of the cylinder
start	x,y,z-coordinates of the starting point of the cylinder
axis	x,y,z-components of the cylinder axis
parent	parent cylinder (row number)
extension	extension cylinder (row numbe)
added	if the cylinder is added after normal cylinder fitting (=1 if added)
UnmodRadius	radius of the cylinder after 1st-fitting before any possible modifications
branch	branch of the cylinder (row number in the branch structure)
BranchOrder	branch order of the branch the cylinder belongs
PositionInBranch	running number of the cylinder inside the branch it belongs

1.7.2 branch

The branch structure contains info of each branch, whose name/index is the row number in the file, and the structure contains the following fields:

```
>> QSM.branch

    order: [2016×1 uint8]
   parent: [2016×1 uint16]
   volume: [2016×1 single]
   length: [2016×1 single]
    angle: [2016×1 single]
   height: [2016×1 single]
  azimuth: [2016×1 single]
 diameter: [2016×1 single]
```

order	branch order (0 for trunk, 1 for branches originating from the trunk, etc.)
parent	parent branch (row number)
volume	volume of the branch in liters (sum of the cylinder volumes forming the branch)
length	length of the branch in meters (sum of the cylinder lengths forming the branch)
angle	branching angle in degrees (angle between the branch and its parent at the branching point)
height	height of the branch base from the ground in meters
azimuth	azimuth of the branch at the base in degrees
diameter	diameter of the branch at the base in meters

1.7.3 treedata

The treedata-structure contains a lot of single-number tree attributes (volumes, lengths, etc.) and distributions computed from the QSM, some of which can be displayed or plotted at the end of the modeling run (see the above example):

QSM.treedata

```
        TotalVolume: 1.0224e+03
        TrunkVolume: 603.8512
       BranchVolume: 418.5219
         TreeHeight: 19.2558
        TrunkLength: 19.3203
       BranchLength: 522.7181
    NumberBranches: 925
    MaxBranchOrder: 7
         TotalArea: 60.6507
           DBHqsm: 0.2634
           DBHcyl: 0.2605
          location: [-5.0404 -1.6416 -1.1854]
         StemTaper: [2×73 single]
     VolumeCylDiam: [1×34 single]
     LengthCylDiam: [1×34 single]
  VolumeBranchOrder: [144.8510 129.7607 88.1549 43.0345 10.9845 1.6854 0.0509]
  LengthBranchOrder: [0.8906 1.7360 1.5472 0.8085 0.2114 0.0325 9.9737e-04]
  NumberBranchOrder: [55 225 341 221 70 12 1]
```

TotalVolume	total volume of the tree (sum of all cylinder volumes) in liters
TrunkVolume	volume of the stem in liters
BranchVolume	volume of all the branches in liters
TreeHeight	height of the tree in meters
TrunkLength	length of the stem in meters
BranchLength	total length of all the branches in meters
NumberBranches	number of branches
MaxBranchOrder	maximum branching order
TotalArea	total surface area of the tree in square meters (sum of all cylinder surface area)
DBHqsm	DBH in meters, the diameter of the cylinder in the QSM at the right height
DBHcyl	DBH in meters, the diameter of the cylinder fitted to the height 1.1-1.5m
location	location of the tree, (x,y,z)-coordinates of the cylinder at the stem's base

StemTaper	stem taper curve, first row is the distance along the stem in meters, second row is the diameter in meters
VolumeCylDiam	volume (L) of the cylinders in diameter classes: 0-1cm, 1-2cm, 2-3cm, etc.
LengthCylDiam	length (m) of the cylinders in diameter classes: 0-1cm, 1-2cm, 2-3cm, etc.
VolumeBranchOrder	total volume (L) of the branches for each branch order
LengthBranchOrder	total length (m) of the branches for each branch order
NumberBranchOrder	total number of the branches for each branch order

When a triangulation model is reconstructed for the bottom of the stem, then also the following fields are included:

DBHtri	DBH in meters, mean length of the "diagonals" in the triangulation
TriaTrunkVolume	Volume of the triangulation in liters
MixTrunkVolume	Volume of the stem in liters, computed from triangulation (bottom) and cylinders (top)
MixTotalVolume	Total volume of the tree in liters = MixTrunkVolume + BranchVolume
TriaTrunkLength	The length (m) of the stem part triangulated

1.7.4 rundata

The rundata structure contains the following fields:

```
>> QSM.rundata
        inputs: [1×1 struct]
        time: [12×1 single]
        date: [2×6 single]
```

inputs	the input structure
time	computation times of the main reconstruction step
data	date and time of the start and end of the reconstruction

1.7.5 pmdistance

The pmdistance structure contains mean point cylinder model distances computed for each cylinder and then the median, mean, max and standard deviation of these distances for all, stem, branch, 1st-order branch, and 2nd-order branch cylinders. The content of this structure is optional and can be turned off by setting: inputs.Dist = 0.

```
>> QSM.pmdistance

       CylDist: [9135×1 single]
        median: 0.0054
          mean: 0.0072
           max: 0.0884
           std: 0.0062
   TrunkMedian: 0.0042
     TrunkMean: 0.0054
      TrunkMax: 0.0220
      TrunkStd: 0.0036
  BranchMedian: 0.0054
    BranchMean: 0.0072
     BranchMax: 0.0884
     BranchStd: 0.0062
 Branch1Median: 0.0038
```

Branch1Mean: 0.0052
Branch1Max: 0.0553
Branch1Std: 0.0048
Branch2Median: 0.0047
Branch2Mean: 0.0064
Branch2Max: 0.0884
Branch2Std: 0.0060

1.7.6 triangulation

Finally, the optional structure triangulation contains information about the triangulation model. The software tries to triangulate the stem up to the first branch, see examples in figure 3. The triangulation can be turned on by setting: inputs.Tria = 1.

```
>> QSM.triangulation

    vert: [21285×3 single]
   facet: [42572×3 uint16]
     fvd: [42572×1 single]
  volume: 1.9331e+04
  bottom: 0.3540
     top: 23.8300
   triah: 0.0695
   triaw: 0.0695
  cylind: 11
```

vert	coordinates of the vertices of the model
facet	indices of vertices forming each triangle
fvd	color information for plotting the model with patch-command
volume	volume enclosed by the model
bottom	minimum z-coordinate of the model
top	maximum z-coordinate of the model
triah	input triangle height
triaw	input triangle width
cylind	Index of the cylinder up to whose bottom the triangulation model is defined

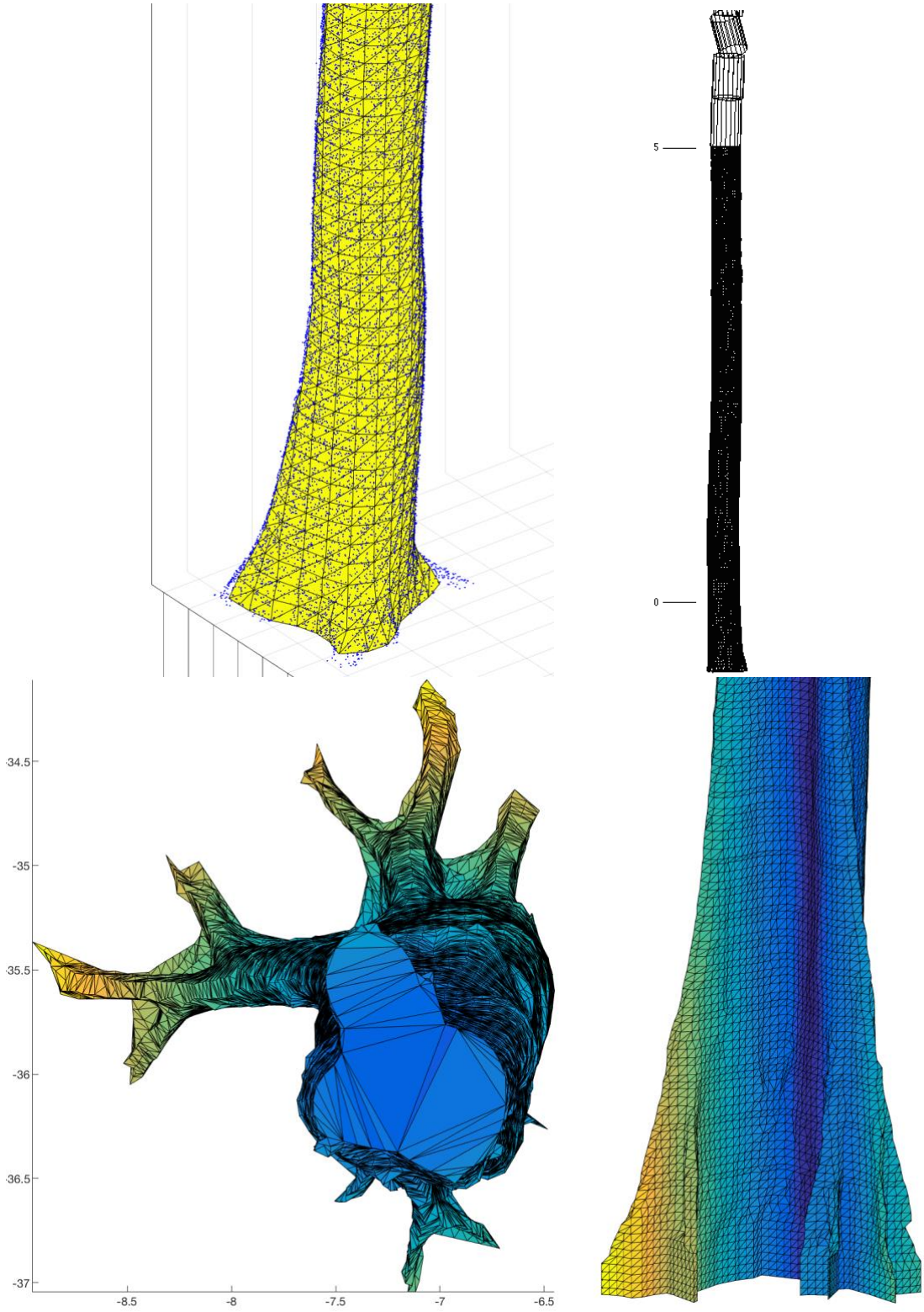


Figure 3. Triangulation models of the bottom part of trunks.

1.8 Main assumptions of the method

TreeQSM reconstruction method is based on a number of basic assumptions that must be filled to ensure reasonable results. The most important ones are:
A1. ***Single tree point cloud*****.** Except some ground and understory points that can be approximately removed by the method, the point cloud contains points from only one tree.

Moreover, lot of noise in the data, which is not filtered, is considered as true measurements from the tree and can cause erroneous reconstructions.
A2. ***3D data***. Only (x,y,z)-coordinate data of the point cloud is needed in the reconstruction. Thus, there is no need for intensity, color, etc. data. However, the additional data might be useful for filtering out leaves/needles, noise, etc.
A3. ***Only sufficiently covered tree parts can be accurately reconstructed*****.** The tree and its details must be sufficiently covered with measured points so that they can be reconstructed accurately with cylinders. The parts of the tree that are insufficiently visible/covered are not accurately reconstructed and may be not reconstructed at all and e.g. a branch with no points is not reconstructed at all. Thus, the resolution and the number of scans around the tree needs to be high enough to sufficiently catch the details of the tree. The problem of determining sufficient resolution and number of scans is not yet well studied and therefore there is no good rule of thumb which can be applied and the numbers may vary case-by-case basis. Notice that high enough scanning resolution and number of scans can result in large point cloud that can be downsampled for most parts without compromising accuracy of the resulting QSMs while speeding up the reconstruction process.
A4. ***Whole tree is wood*****.** If leaves or needles are present in the data and are not filtered, then they are used also in the reconstruction of the woody structure. This can result in a model with too thick branches or (small) non-existent branches. Thus leaf-off scans are highly recommended, also because they increase the visibility. However, some evergreen trees are possible for modeling but may have large local errors. Thus, with leaf-on scans, the user is advised to consider methods that can separate the leaf and wood points before the QSM reconstruction.
A5. ***Cylinder is an acceptable building block.*** Locally the shape of the stem and branches should be approximately cylindrical so that their diameter, volume, direction, etc. can be well approximated with right circular cylinders as the geometric primitive. However, there is the possibility for triangular mesh for modelling the bottom of the stem to capture better its volume, shape and diameters (input.Tria).
A6. ***Branches taper and are smaller than their parents*****.** There are two basic methods for controlling too large (and too small) cylinders that sometimes results in from least squares fitting due to multiple reasons: 1) The radii of the cylinders in a child branch are always smaller than the radius of the cylinder in the parent branch from which the child branch starts. 2) The taper of the branch is decreasing from the base towards the tip. A quite relaxed parabola constraint (partially linear taper for stems) is used to enforce this tapering for branches. It is described in detail later on in this document. Both of these radius controlling methods can be turned on and off with inputs.ParentCor and inputs.TaperCor. Another and possibly additional optionn for radius control to have tapering branches is to use growth volume approach (input.GrowthVolCor).
A7. ***Separate stem near the ground.*** There must be clearly separate and visible stem near the ground, because the segmentation process starts from the base of the stem. Furthermore, if the lower branches touch the ground, the reconstruction can fail badly.
A8. ***No special assumptions about tree species or size*****.** Basically, the limits come from the quality of point cloud data: if the branches are small compared to laser spot size and noise levels, then the branch cannot be resolved accurately. Furthermore, species that conform to the above assumptions should be possible for reasonable reconstruction.

There are few assumptions for the optional triangulation to work properly:
AT1. ***Sufficient point cover*****.** The bottom of the stem should be sufficiently covered with points without major gaps in the cover.
AT2. ***Sufficiently low noise level*****.** Noise should be small compared to the details of the stem (e.g. the width of a buttress root).

AT3. *Sufficiently small triangles*. The triangle size (width and height) should be small compared to the details of the stem but not smaller than the resolution of the point cover.
AT4. *Horizontal cross-sections down to the ground level*. The stem point cloud is partitioned into horizontal layers of given thickness and the boundary curves are defined, with extrapolation if necessary, to every layer from the top to the bottom.
AT5. *Only stem points and no bifurcations*. All points are considered data and thus no ground and understory points are removed by the reconstruction process. The stem cannot have bifurcations and thus it needs to be one tube-like (even if with complex shape cross-sections) structure from the top to the bottom (ground).

1.9 Reconstruction in practice

There are two practical considerations for producing good results. ***Input parameters need to be optimized*** and ***multiple models need to be reconstructed with the same input***. Furthermore, suitable downsampling of the point cloud can speed up the QSM reconstruction process considerably without compromising the accuracy of the resulting QSM. Similarly, using distributed computing with multiple processors/cores can speed up the computation of the multiple models that need to be generated.

1.9.1 Optimization of input parameters

If the values of input parameters, such as *PatchDiam2Min* and *lcyl*, are changed, then of course the resulting QSM is changed. Thus, there is a need for some kind of optimization process that selects the best or good parameters within practically meaningful limits. There are many ways to realize the optimization and it has two key ingredients, the ***optimization method*** and the ***metric/cost function*** that is minimized with the method. The optimization method depends more on the number of optimized parameters and usually there are only 3 (*PatchDiam2Max*, *PatchDiam2Min*, *lcyl*) that are most meaningful in this respect (the two next most important are *PatchDiam1* and *FilRad*). Thus, a simple grid search can work reasonably: Select a few reasonable values for each parameter and then reconstruct the models for every parameter value combination. For example, 2 values for 3 parameters makes 8 different combinations and 3 values for 4 parameters makes 81 combinations. The grid search or brute force approach is realized with `make_models`. To use parallel computing with multiple cores/processors, use `make_models_parallel` (requires Parallel Computing Toolbox).

A bigger challenge is the selection of a suitable metric. Now the default option we use is the following metric: Select the points closest to each cylinder and calculate the average distances to the cylinders. This produces one distance for each cylinder ("cylinder distance"), the average point-cylinder distance. Then the final metric-value is the mean of these cylinder distances. Function `select_optimum` realizes the selection of the optimal models based on this metric as the default option. The function selects for each tree the optimal models based on the minimum metric value (minimum average metric value over the models with the same inputs) and computes average values for each "treedata" from the models.

In version 2.3.1 there are readily available more than 30 different other metrics that the user can choose. And the user can easily modify them or add new ones. There are two types of metrics available: cylinder distance-based metrics and standard deviation-based metrics. The distance-based metrics are similar to the default option. For example, mean of trunk cylinder distances, mean of branch cylinder distances or mean of trunk cylinder distances plus mean of 1st-order branch cylinder distances. Then there are the versions of these based on the maximum distance. For example, the maximum trunk cylinder distance. And finally, there are

also the combination of the mean and the maximum distances. For example, the mean of trunk cylinder distances plus the maximum trunk cylinder distance. The standard deviation (SD) based metrics are for example SD of total volume in the models with the same inputs or SD of number of branches in the models with the same inputs. Thus, for the SD based metrics it is important to make many models with the same inputs in order to have robust SDs.

The above optimization procedure, where perhaps hundreds of models per every tree are generated, might be too excessive work in some cases, particularly if the results do not depend too much on the exact values of the input parameters close to the optimal values. Thus, in some cases where we have lot of similar trees (similar size, height, complexity) and with similar measurements (similar number of scans, scanner distances, point density, occlusion) it might be workable option just optimize the parameters with one or few trees and then apply those parameters to all the other trees. Or at least reduce the number of input parameter values used for most of the trees based on the optimization of the few trees.

1.9.2 Multiple models with same inputs

There are random elements in the reconstruction process (in the cover generation as explained in the section 2), which always results in little different models for every modelling run, even with the same inputs. This is not a bad feature but in fact allows the estimation of the uncertainty in the modelling results by making multiple models and computing the variance / standard deviation / coefficient of variation as the parameter estimating the uncertainty (or precision) of the results. We can think that there are distributions of results, e.g. total and stem volumes, with the same inputs. Although it is not true that the mean of the distribution is the correct value, we can assume that the mean is often a very reasonable value. We will show below that ***the mean is robust and about 5 models with the same inputs is already enough*** to estimate the means of the distributions within about a few percent error. The ***estimation of the standard deviations of the distributions is harder and reliable estimates would require more models, perhaps 20-30***. However, if one wants to make better and more reliable estimate of the variance/std/CV, one could ***first optimize the parameters using 5 models per inputs and then make more models with the optimal parameters to estimate the uncertainty more reliably***. The code `make_optimal_models` can be used to generate more models with a given optimal input parameters.

Next, we investigate the distribution of the modelling results with the same inputs and show how many models or repeat modelling runs are needed to achieve robust results. We do this with the following test: We reconstruct 500 QSMs with the same inputs which produces empirical distributions of modelling results, such as volumes as shown in figure 9. The empirical distributions are then taken as good approximations of the real distribution of the results. Then we randomly sample the empirical distribution with different sample sizes large number of times to estimate the distributions of sample means and sample standard deviations for each sample size (see figure 10 and 11).

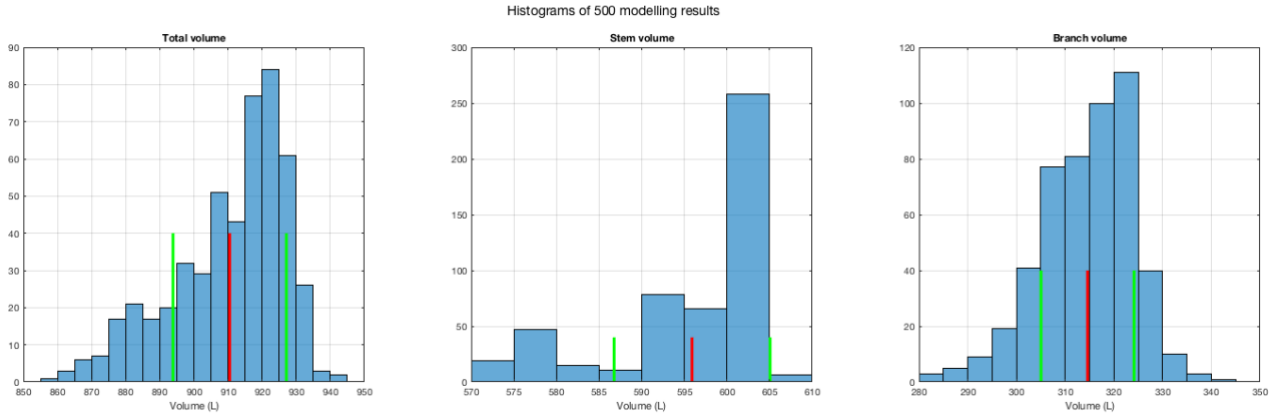


Figure 9. Volume results from 500 QSMs shown in histograms. Total (left), stem (middle) and branch (right) volumes shown. Red lines indicate the mean volumes and green lines indicate the mean +/- one standard deviation volumes.

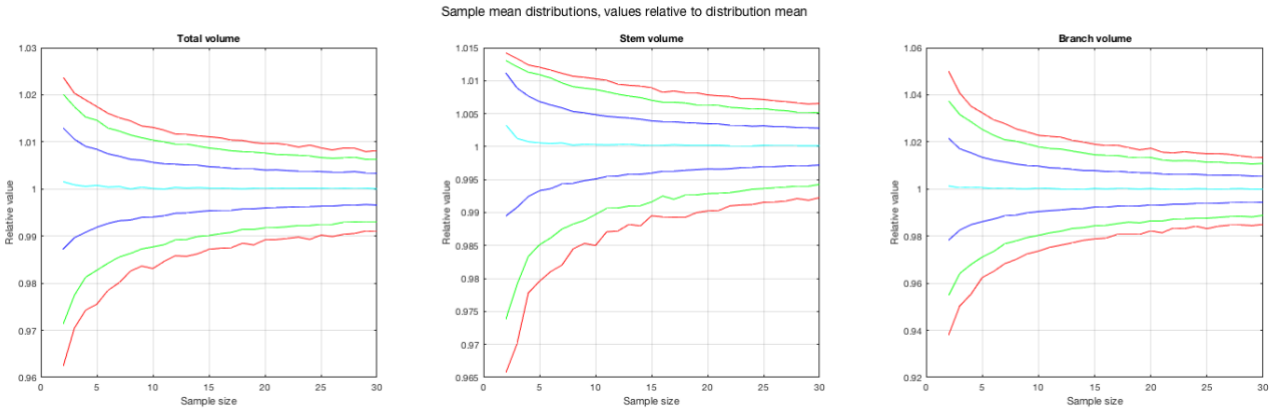


Figure 10. Sample mean distributions for different sample sizes. The values are relative to the means of the empirical distributions of 500 models. Inside the red and green lines are 99% and 95% of sample means, respectively. Light blue lines are the medians of samples and the blue lines show +/- standard deviation from the mean.

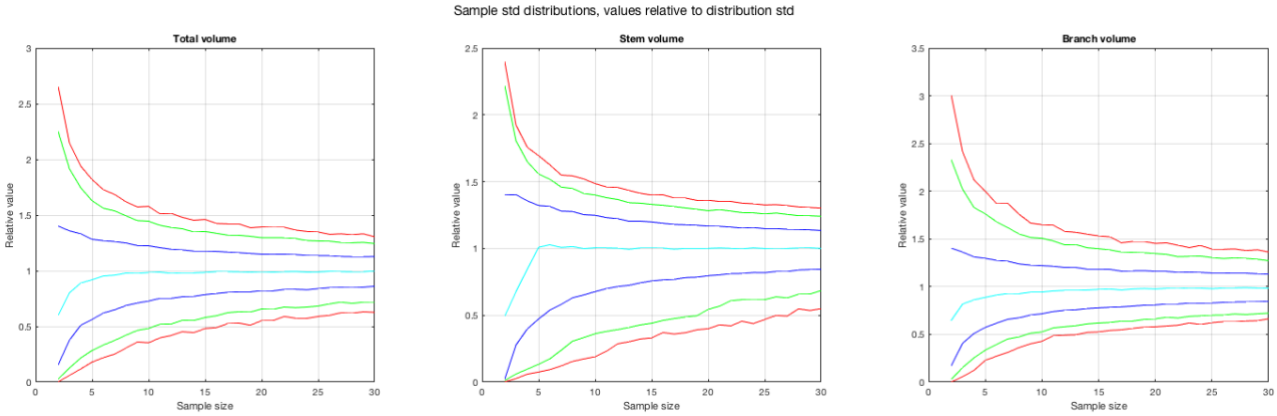


Figure 11. Sample standard deviation distributions for different sample sizes. The values are relative to the standard deviations of the empirical distributions of 500 models. Inside the red and green lines are 99% and 95% of sample means, respectively. Light blue lines are the medians of samples and the blue lines show +/- standard deviation from the mean.

Figure 10 shows the distributions of sample means compared to the mean of the empirical distribution and we can see that a very small sample size, about 5, will give reliable estimate of the mean within a few percent. In other words, if one for example generates 5 models with the same inputs and then computes the average total volume from those five models, the

resulting average differs from the real average (the mean of the empirical distribution) about 2% in 99% of cases. Of course, the exact results depend on the tree, point cloud quality, the inputs, attribute, etc. but in general we can say that with a fairly small number of models we can estimate the means (volumes, lengths, etc.) accurately and reliably.

Figure 11 shows similar results for the sample standard deviation, that is the distributions of sample SDs compared to the SD of the empirical distribution. However, unlike the mean, the standard deviation cannot be robustly estimated with a fairly small number of models. We can see from figure 11 for total volume that even with 30 models the 99% confidence interval contains samples whose error is about 30% and 95% interval have samples with errors about 25%. Again of course the exact results vary depending on the tree, attribute, etc., but reliably and accurate estimation of the standard deviation requires a large number of models.

1.9.3 Downsampling point cloud

To see all, or at least most, of the branches of the tree and cover them sufficiently with measurements, one may have to scan the tree with high resolution and/or from high number of positions around the tree. This is likely to result in a large point cloud that has high point density, particularly on the bottom of the stem and on other parts not really requiring so high point density. Naturally, the more points there are the more computation time and memory are required for the QSM reconstruction. On the other hand, it is clear that increasing the point density can only improve the accuracy of the resulting QSMs up to a limit and increasing the point density beyond this does not improve the results anymore but only requires more computational resources. Thus, in some situations it can make sense to try to downsample the point clouds before the QSM reconstruction. The function `cubical_down_sampling` is quick way to downsample point clouds by specifying cubical volume size (by giving the side length of the cubes) so that each cubical volume containing points in the original point cloud will contain only one of those points after the downsampling. Notice that this function selects a subset of the original point cloud and does not do averaging. If the user wants to do averaging, i.e. downsample the point cloud by replacing the points in the cubes with the average of the points, then he can use the function `cubical_averaging`.

1.9.4 Making QSMs in practice

Based on the above, the practical way to use the code is:

1) (Optional) Filter out noise (e.g. using `filtering`) and leaves from the point clouds
2) (Optional) Downsample the point clouds (e.g. using `cubical_down_sampling`)
3) Save the point clouds into single .mat-file.
4) Define the “inputs” structure with multiple values for the parameters to be optimized.
5) Call `make_models` (or `make_models_parallel`) with the point clouds file, “inputs” and the number of model-runs (e.g. 5-10) as the inputs.
6) Use `select_optimum` to select the optimal models and results.
7) (Optional) Use `make_models` to make additional models (about 10-25 additional models) with the optimal inputs to estimate the standard deviation (precision/uncertainty) of the results more reliably. Then use `estimate_precision` to combine all the optimal models and compute the standard deviations from all the optimal models.

1.9.5 Plotting

There are many functions in the /plotting-subfolder that the user can use in MATLAB for plotting the point clouds, the branch-segmented point cloud, cylinder models with colors denoting the branching order, etc.

1.10 QSM simplification

Reconstruction of accurate QSMs may require that the whole tree is modelled carefully. This often means short cylinders and therefore a lot of cylinders. However, in some applications the user may only be interested in the stem and bigger branches and furthermore the number of cylinders needs to be low. Fortunately, the topological and quantitative features of QSM lend itself to orderly simplification. The user can simplify a given QSM by three different ways using `simplify_qsm` function: there are two ways to remove branches and one way to decrease the number of cylinders inside branches. The first way to remove branches is by giving a maximum acceptable branching order. The second way to remove branches is by giving a minimum acceptable diameter at the base of the branch (child branches of a removed small branch are automatically always also removed). The third way to simplify was to reduce the number of cylinders inside branches: two consecutive cylinders inside a branch (i.e. cylinders 1 and 2, 3 and 4, 5 and 6, etc.) can be replaced with a single cylinder that starts from the base of the first cylinder and ends at the top of the later cylinder and whose radius is such that its volume equals the sum volumes of the replaced cylinders. The user species here the number of replacement iterations so that with one iteration the number of cylinders is about halved. Two iterations mean that the same process is applied to the cylinders resulting from the first iteration and thus the number of cylinders is again about halved and equals about one quarter of the number without replacements. Notice that currently the `simplify_qsm` function only modifies the cylinder data in the QSM-structure and the other data, particularly branch and treedata, are not modified to correspond to the changes in cylinders.

1.11 Version history

Version 2.3.0 was the initial release in 2017.

Version 2.3.1 was released October 2019. Mainly some bug fixes and changes to the optimization and “make_models” functions.

- Fixed bugs that could cause errors in some special cases in “tree_sets”, “correct_segments”, “cylinders” and “estimate_precision”.
- Fixed bug: wrong computation of cylinder starting points in “least_squares_cylinder”.
- Changed what and how is displayed during the run of “treeqsm”.
- Added many more optimization metrics to “select_optimum”
- Changes in the “make_models” and “make_models_parallel”:
    - Added try-catch structure where "treeqsm" is called, so that if there is an error during the reconstruction process of one tree, then the larger process of making multiple QSMs from multiple trees is not stopped.
    - Changed the way the data is loaded. Previously all the data was loaded into workspace, now only one point-cloud at the time is in the workspace.
    - Corrected a bug where an incomplete QSM was saved as complete QSM

Version 2.3.2 was released December 2019. Mainly some small bug fixes to handle trees without branches.

- “cylinders”: Increased the minimum number "n" of estimated cylinders for initialization of vectors at the beginning of the code.
- “point_model_distance”: Corrected the computation of the output at the end of the so that trees without branches are computed correctly.
- “estimate_precision”: Added the "name" of the point cloud from the inputs.name to the output TreeData as a field. Also, now displays the name together with the tree number.
- “select_optimum”:
    - Added the "name" of the point cloud from the inputs.name to the output TreeData as a field. Also, now displays the name together with the tree number.
    - TreeData contains now correctly fields ("location", "StemTaper", "VolumeBranchOrder", etc) from the Optimal QSMs.
- “tree_data”:
    - Bug fix: Added a statement "C < nc" for a while command that makes sure that the index "C" does not exceed the number of stem cylinders, when determining the index of cylinders up to first branch.
    - Bug fix: Changed "for i = 1:BO" to "for i = 1:max(1,BO)" where computing branch order data.
    - Added the plotting of the triangulation model
- “initial_boundary_curve”: Added "return" if the "Curve" is empty after it is first defined.
- “curve_based_triangulation”: Removed the plotting of the triangulation model at the end of the code.

2. The reconstruction method – How it works?

2.1 Overview of the main steps

There are several main steps in the QSM reconstruction method. In the highest level, there are two or three steps: First step is optional and it is *filtering noise* from the point cloud. The method assumes that most points are data from the tree and thus lot of noise in the point cloud could be, if not filtered out, a major source of error. This filtering step could be considered to include also the leaf-wood separation. Filtering of noise was considered already above and you can have your own methods for doing it. The second step is the *topological reconstruction of branching structure*, which is the segmentation of the point cloud into stem and individual branches. The third step is the *geometrical reconstruction of branch surfaces*, which is realized by fitting cylinders. The cylinders then give the surface, volume, lengths etc. for each branch.

2.2 Topological reconstruction of branching structure

There are many conceptually separate steps in the method for the segmentation of the point cloud and these have their own functions that are described in the following sections. The basis of the segmentation is cover sets that are small subsets of the point cloud and can be thought as small patches in the tree surface. The sets form a quite uniform Voronoi tessellation of the point cloud. They can be thought as the smallest “unit” for separating branches from each other in the segmentation process. They have also natural neighbor-relation, which is used for “surface growing”, where adding neighboring sets to existing set grows the set along the surface. The cover sets are generated with the function `cover_sets`.

Because of the gaps in the data due to occlusion, there are parts in the tree that form separate components in the sense that the different parts are not connected through the neighbor-relation (it is not possible to connect the parts with surface growing). Thus, the next step is to update the neighbor-relation so that the whole tree is one connected component. Moreover, in some cases the point cloud may contain points not from trees such as points from ground and understory. These points need to be removed before the segmentation process. Updating the neighbors and removing non-tree points/sets are done with the function `tree_sets`.

The next step is to segment the point cloud into segments that do not have bifurcations, that is the segmentation process finds the bifurcation points of the branching structure. Function `segments` reconstruct the initial segmentation of the tree patches into stem and branches. However, `segment` finds the bifurcations by local examination of connectivity and this does not generally result in complete branches but the segments tend to "end too soon" by making a "wrong turn" at some bifurcation point. The next step is to correct the initial segments so that they better correspond to real branches (and stem). This is realized with function `correct_segments`, that takes in a segmentation and tries to improve it by making the segments as long as possible.

To improve and accelerate the above segmentation process it is actually realized as a two-iteration process. First large and uniform size cover sets are used to remove the non-tree points if necessary and particularly to reconstruct the main branching structure and use that information for smarter and refined cover set partitioning and segmentation. The bigger sets are particularly good for segmentation of main branching structure as they are quite insensitive for small gaps in the data that are generally quite numerous. The first segmentation gives good information how the size of the cover sets should vary locally in the point cloud so that they are not too large or small for the local details. Also, the connectivity along the branches of the first segmentation can be easily enforced to the new smaller cover sets.

2.2.1 Cover sets

The QSM reconstruction method uses a "cover set" approach, where the point cloud is partitioned into small sets that correspond to small patches in the surface of the tree (see Figure 5). These sets form the smallest "unit" we use to segment the point cloud into trunk and individual branches. They are randomly generated by the input parameters *PathcDiam*, *BallRad*, *nmin*. The generation process produces a Voronoi partition of the point cloud so that the cell size (maximum diameter) is controlled and varies between PatchDiam and 2*PatchDiam. The following iterative process, that generates centers/seed points, generates the cover sets (Voronoi partition): First select a random seed point Q and define *BallRad*-ball, i.e. those points that are closer than *BallRad* to Q. If this ball has at least *nmin* points, then the ball is accepted and Q is the center of the ball and the cover set to be formed later. Next define a *PatchDiam*-ball centered also at Q. Here *PatchDiam* is usually little smaller than *BallRad*. PatchDiam is the minimum distance between nearby centers of cover sets (or seed points), so the points in the *PatchDiam*-ball will not be centers of other cover sets (seed points). Then select randomly another point R as a center of another *BallRad*-ball (seed point). This point R cannot now be in the *PatchDiam*-ball centered at Q. Similarly define the *BallRad*- and *PatchDiam*-balls for point R. Proceed this way until all points are included in some balls or are too far away from other points not be accepted in any *BallRad*-ball. Finally define the cover sets (Voronoi cells) to consist those points that are closest to the centers (seed points), i.e. each point belongs only one cover set (Voronoi cell). Because *BallRad*-balls can intersect, each

point may belong to multiple *BallRad*-balls, but it will be assigned to the cover set whose center (seed point) is the closest. This way the points are partitioned into "cover sets" or "surface patches" (Voronoi cells).

Because most of the *BallRad*-balls intersect some other *BallRad*-balls and each cover set has its own *BallRad*-ball associated with it, we define two cover sets as neighbors if their balls intersect. To ensure that cover sets next to each other are neighbors, *BallRad* should be little bigger than *PatchDiam*.

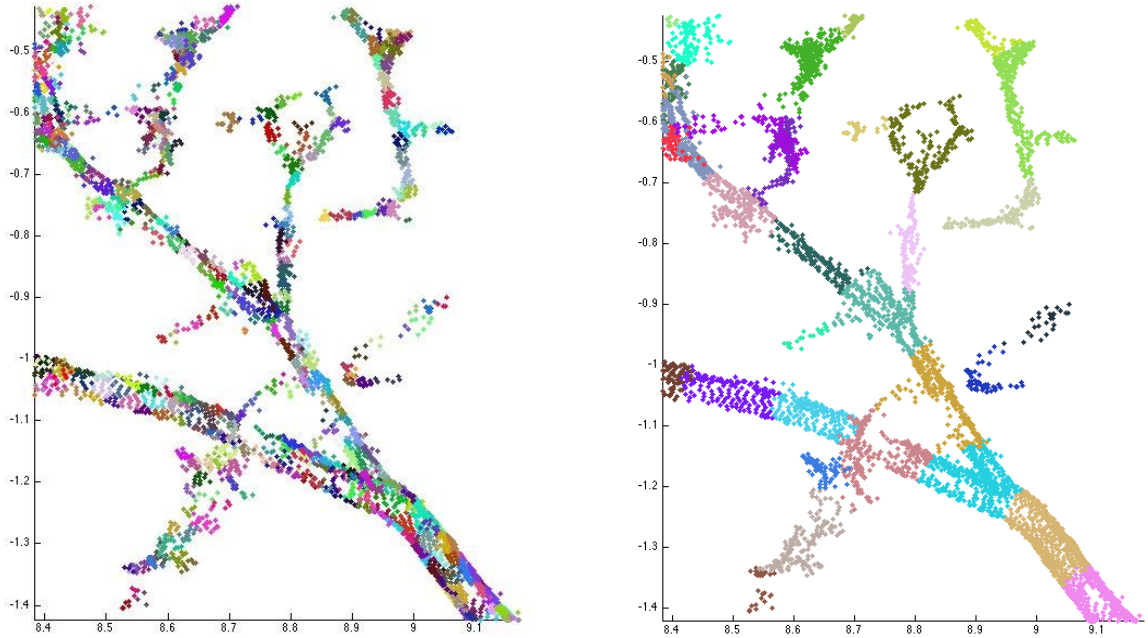


Figure 5. Comparison of the covers of a branch. The minimum diameters (PatchDiam) of the cover sets are 2 cm (left) and 10 cm (right). The smaller cover sets can capture much more detail but form more disconnected structure.

The average size of the cover sets is thus controlled by *PatchDiam*-parameter, which represent the minimum patch diameter and the maximum is two times this. There is a trade-off with the size of cover sets, as can be seen in figure 5. The smaller it is, the more details we can capture and smaller branches can be separated. However, the smaller size means more sets, which means almost quadratic increase in modeling time (half the size means about four times the cover sets and up to 4-fold modeling time). Also, memory requirements increase with decreasing cover set size. Furthermore, very small sets can segment a branch into multiple smaller branches if the branch is not covered fully with points. On the other hand, bigger sets mean faster computation and less memory required. With bigger sets, the smallest branches may not be separable. Also, the beginning of each branch may be less accurately determined which means that fitted cylinders might be too large (include points from the child branch).

The method uses two different covers. The first cover uses large and uniform size sets and the other uses smaller and variable size sets. In the above example run the first cover had *PatchDiam1* = 5 cm and often 5-15 cm is a good range for the size of the sets. The purpose of this first cover is to 1) remove the points that don't belong to the tree, e.g. ground and understory points, and 2) make initial segmentation that is used as a priori information for

the branch connections and size of the cover sets in the second cover set generation. The actual value of *PatchDiam1* for the first cover is not very important because it has little effect for the final result (this is true up to a point of course, but e.g. values 5 cm – 15 cm for some cases could work almost equally well). On the other hand, the size of the sets in the second cover is very important for the final results.

For the second cover the user specifies the minimum and the maximum patch sizes with parameters *PatchDiam2Min* and *PatchDiam2Max*. The maximum size is at the base of the stem and the minimum at the tips of the branches and the stem, see figure 6 for an example of varying patch size. The size of the sets should be small enough for the local details of the branches: The cover sets near the tips of the branches need to be small so that all the details can be seen. At the same time these small sets near the base of the trunk are too small for efficiency and may even lead wrong segmentation. Thus, the size of the cover sets should be varying and based on the first segmentation we approximately know the branching structure and the size of the branches. The local *PatchDiam* is determined for each branch base by comparing its size to the size of the stem's base. The sizes of the bases are estimated roughly by comparing the number of cover-sets near the bases (as the cover set size in the first cover is uniform everywhere, smaller branches need less cover sets to cover their surface than bigger branches). However, we compute a priori upper bound for *PatchDiam*-value based on the branching order and height of the branch: if the estimated *PatchDiam*-value is bigger than the upper bound, then the *PatchDiam* is set to the upper bound-value. Now the *PatchDiam* is set for the base and tip for every branch, including the stem. Along the branches the *PatchDiam*-value varies from the base-value to the tip-value (the minimum size) "quadratically" so that the size decreases slowly at the beginning and then faster near the tip. Finally, one more modification is done for the local patch size in the parent branch around each base of its child segments: The local *PatchDiam* is halved in order to have small enough patches for accurate separation of branches. The local size of the cover sets for the second cover is determined as a relative size by the `relative_size` function, where you can find more information.

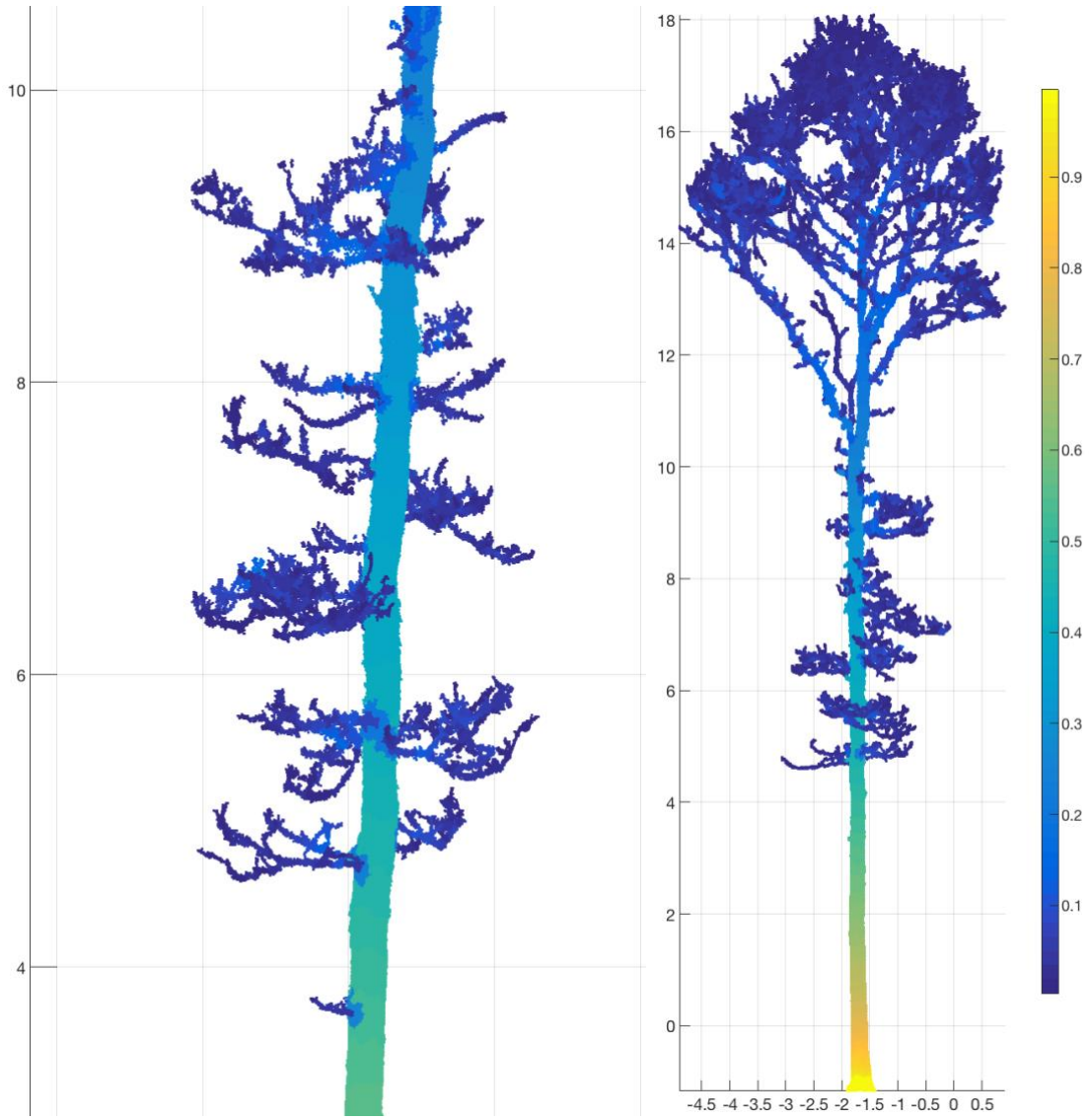


Figure 6. Relative size of the cover sets for the second cover. Value 1 corresponds to the maximum size and value 0 to the minimum size.

2.2.2 Tree sets

When the first cover is generated there are a few things that need to be done before the segmentation. First of all, the point cloud may contain points from the ground and understory and these non-tree points need to be removed. Secondly, for the segmentation we need to determine the base of the stem, which works as the starting point for the segmentation process. Finally, the segmentation process assumes that the tree cover sets form one connected whole in terms of their neighbor relation. Because of occlusion there are often lot of gaps that need to "bridge over" by modifying the neighbor relation to make the tree connected. With the second cover, when the non-tree points are already removed and first segmentation is available, the step of removing the non-tree points is replaced with the following step: The neighbor relation of the new cover sets is modified so that the branches of the first segmentation are connected and they are connected to the stem. These steps are realized by the function `tree_sets`.

Let's next see these steps with more details. First the stem and its base is located. If there no ground and understory points, then we can utilize this and set the input parameter *OnlyTree* as true or 1. Then the base of the stem will be simply a thin bottom layer of the point cloud as

we can now assume that these points must be from the bottom of the stem. If, on the other hand, there are non-tree points, such as points from the ground, then this approach does not work in general. So, if *OnlyTree* is false or 0, the code tries to locate the stem differently: The stem is assumed to be quite vertical and long so that if we project the location of the cover sets into a horizontal plane, then the highest density of sets should be very close where the base of the stem is. This is actually realized as maximizing a function defined on 1m times 1m grid of squares that considers the number of sets, the vertical layers with sets above the squares (10 layers) and the number of sets in the first two vertical layers. Thus, the grid square with the maximum function value defines the location of the stem approximately and more accurate location is estimated with cylinder fitting. When the stem is defined (the first few meters anyway), then the ground and other non-tree points are defined by region growing from the base of the stem as much as possible and finally by classifying all bottom sets (not the stem sets) as the ground sets.

The first cover is with large and uniform sets, which are good for quick estimation of the main branching structure, particularly because they are not so sensitive to small gaps in the data. With the second cover, we use the stem and branches up to 3rd-order from the previous segmentation to modify the neighbor-relation so that these branches form connected wholes and that 1st-order branches are connected to the stem. This step ensures that we can retain the branching structure from the first segmentation. However, this does not mean that the second segmentation is the same up to 3rd-order branches as the first segmentation, only that it can be.

Finally, if there are still separate components (usually there are many), then these are connected by modifying the neighbor-relation so that the whole tree is a single connected whole. First the initial connected tree is defined by the region growing as much as possible from the base of the stem. Then the separate components of the other parts are determined and they are connected to the initial tree as follows: For each component check if its nearby space has cover sets from the tree and then make a connection between the closest cover sets. Repeat this as long as new connections can be made and separate components remain. If there are still components that cannot be connected to the tree, then increase the size of the nearby search space and repeat the process again. Increase the nearby search space as long as all the components are connected to the tree. In the case of point clouds with non-tree points (*OnlyTree* = 0), there is a minimal nonzero component size depending on the distance that can be connected to the tree. This minimal component size increases with the size of the nearby search space, the idea being that small sets far away from the tree are probably from other trees or understory. Thus, components smaller than the minimal size are classified as non-tree points and excluded from the QSM reconstruction process.

2.2.3 Segmentation

Next, we segment the cover sets (i.e. the point cloud) into stem and individual branches. This process starts from the base of the trunk and in step-by-step proceeds along the stem (later along branches). At each step, possible bifurcations are determined. If there is a branch, its base is saved as new basis for later segmentation. This way the stem is segmented and its branches are separated from it. Then the same process continues from the base of the first found branch. This way the stem is first determined, then the 1st-order branches, then 2nd-order branches, etc., see figure 7 as an example.

The determination of possible bifurcations (branches) is based on a local topological analysis of cover sets, where connectedness of small regions is determined. The idea is as follows:

Start from the base of the stem (or a branch) and expand it with the neighbors a few times. This grows the base along the stem with a few layers of cover sets. Let us now assume that this resulting region of a few layers of cover sets contains only sets from the stem and is a single connected whole. When this region is moved along the stem one layer at the time and when there is a branch, the region diverges into two parts, one in the stem and the other in the branch. Thus, when moved far enough, the region becomes disconnected, which then indicates that there is a possible branch. However, due to small gaps in the data, the region may become disconnected even when being only along the stem. Therefore, very small components are considered being along the stem. If there are multiple components that are possible branches, then the biggest component is handled as the continuation of the stem. When a component is classified as a branch, its base is saved as a starting point for the segmentation of the branch later in the process and further expansion in the branch is prohibited as long as the current segment (stem) is under segmentation. When the stem is segmented, then the same process continues from the branch base saved first and the branches are segmented in the same order their bases were defined. When there are no bases of unsegmented branches, the segmentation process stops as every cover set now belongs to a segment (branch).

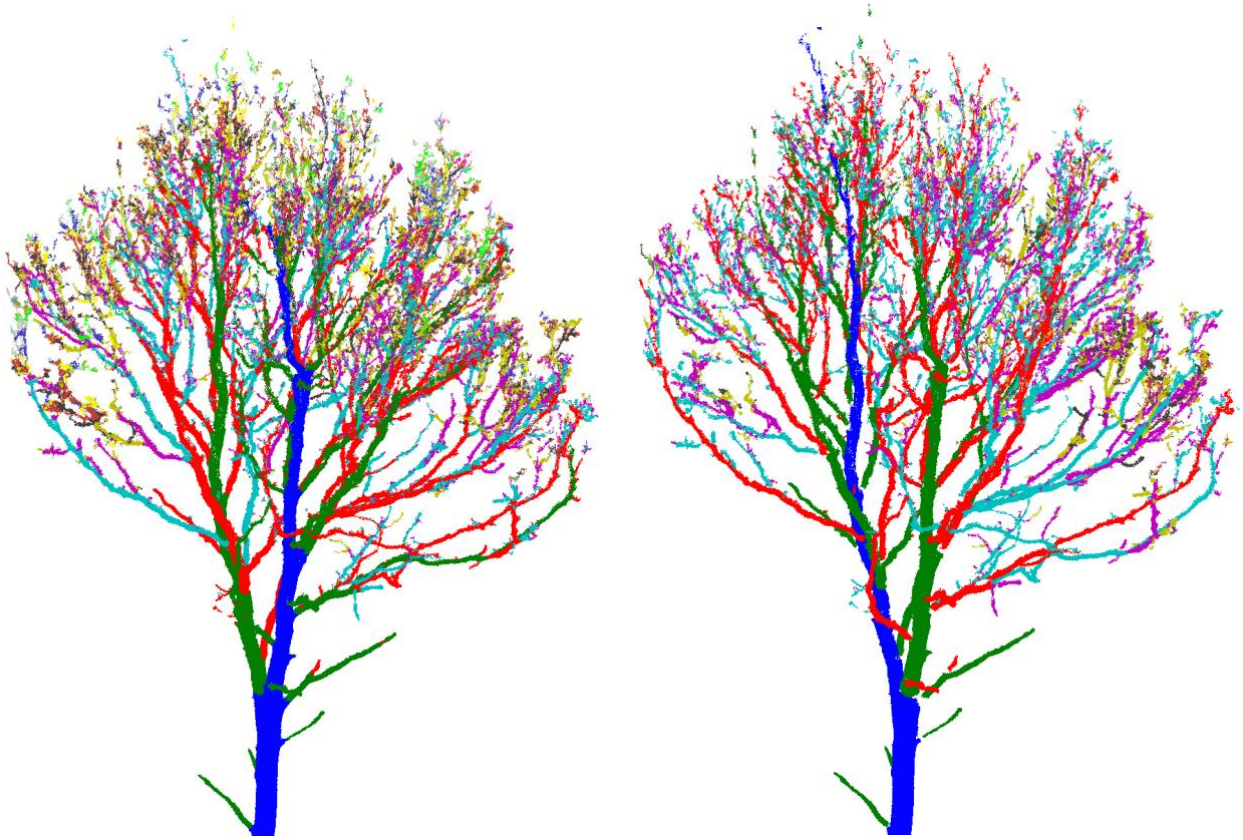


Figure 7. Initial segmentation (left) and corrected final segmentation (right). Notice how trunk (blue) and other segments are corrected and how in general branching order of the segments is reduced.

2.2.4 Correct segmentation

The initial segmentation is based on quite rough local examination and thus may result incorrect decisions, such as ending the segment too early by making wrong decisions where the segment continues at bifurcation points. Thus, the initial segmentation requires corrections so that each segment would correspond to the real branches as well as possible. Of course, due to many reasons such as noise and gaps in the data, it is not possible to

perfectly correct the segments, but the following correction operation achieves quite good results. We use the following robust heuristic property: The tip of the branch is the tip among all the tips of the child segments that is the furthest away from the base of the branch. Thus, the idea is to make the segments as long as possible and define them from the tips backwards to their bases. This process modifies the topology or the branching structure of the segmentation and next it is explained in detail.

The segments are corrected, in the increasing branching order starting from the stem (0_{th}-order), so that the segments are made to reach as far as possible. In other words, take a segment and all its current child segments and calculate the distances from the base of the segment to the tip of the segment and to the tips of all its child segments. Notice this distance is not along the branches but is the direct or the smallest distance between points. If the distance to the tip of the segment is less than to some other tip of a child segment, then the segment is corrected so that it goes to this furthest tip (and other segments are also modified suitably) (See figure 7). For the stem and 1_{st}-order branches we select the possible new tip with the restriction to the "straightness" of the segment by limiting the acceptable ratio (length of the segment)/(distance between the base and the tip).

This kind of correction makes the resulting segmentation much more robust, i.e. with different covers the resulting segmentation is usually close to identical. Thus, the volume reconstruction will also be more robust in the same sense. Also, the maximum branching order is reduced much more realistic levels. Notice that while this modification of branching structure usually makes the segmentation more correct, there are of course situations where the modification may change initial segmentation to worse. For example, if the tip of the main branch is broken off, the modification may now change the segment because its tip is not the furthest tip anymore.

Accurate cylinder fitting to the segments requires some more modifications to the segments. Firstly, in the second segmentation there may be very small segments that do not have child segments and are difficult to estimate accurately if they really are part of their parent segment or are they real segments. These segments are simply removed because their true contribution to the total volume and structure is very small but at the same they can be problematic for the cylinder fitting in some cases.

Secondly, because the branches were separated from their parent based on the disconnection of certain moving region, the branch base may not be very accurately defined and often the parent segment contains small ledge-like parts from the branches. Now these small branch base parts can be problematic when fitting cylinders because the resulting cylinder may be too large in radius as it considers these branch parts. Thus, the base of every segment is modified so that potentially some parts from the parent segment are removed. In the case of first segmentation, the part from the parent is added to the child segment. In the case of the second segmentation, the part from the parent is simply removed as this is best option for cylinder fitting.

2.3 Geometrical reconstruction of branch surfaces

After segmentation, we fit cylinders to the segments using the least squares fitting. Then optional modifications/corrections follow the fitting to prevent in some a priori sense too small or too large radii. Also, the cylinders are connected to each other and bigger gaps between child and parent segments are filled, either by extending the first cylinder in the child segment or by adding a new cylinder. All these steps are realized with the function

`cylinders`. After cylinder fitting the branch data are computed, including the length, volume and angle of each branch, in the function `branches`. Then some tree attributes such as volumes and lengths are computed in the function `tree_data`. If the input *Tria* is set to 1, then the surface and volume of the bottom part of the stem up to the lowest branch is modelled with a triangulation. The triangulation can offer better estimates of the volume and diameters for stems with e.g. big buttresses or otherwise clearly non-circular cross-sections. Finally, distances between the QSM surface and the point cloud are computed in the function `point_model_distance`. These distances can quantitatively describe the goodness of the reconstructed QSM and thus can be used in an optimization process, where the input parameter values are optimized.

2.3.1 Cylinders

After segmentation, we fit cylinders to the segments using the least squares fitting with the function `cylinders`. Here the input parameter *lcyl* controls the average relative length of the cylinders: *lcyl* = length/radii. The relative length of the fitted cylinders is not exactly *lcyl* and not even the average is. However, the regions or sub-segments, where the cylinders are fitted, are estimated to have approximately this relative length, so for most and particularly large segments the relative length of the fitted cylinders is close to *lcyl*-value. Thus, the bigger *lcyl* is, the longer the fitted cylinders are on average. Here again there is a trade-off: shorter cylinders can better model the local diameter of the branch, but at the same time, their direction can be more varying and noisy points and other local things can make the diameter too large or too small.

The regions or sub-segments used for the cylinder fitting are also used for approximating initial values needed for the least squares cylinder fitting. The axis is simply the line segment connecting the averages of points in the bottom and top layers of the region. The radius is estimated as the median distance of the points of the region to the axis. Now with the input parameter *FilRad* the user defines relative radius for outlier point filtering before the least squares fitting. For example, *FilRad* = 3.5 means that points farther than 3.5 times the estimated radius from the axis are filtered from the region. If the data is not noisy and co-registration is good, then we do not filter out points and thus *FilRad* should have large value like 3 or 3.5. But in some cases, depending on the noise level and co-registration accuracy, perhaps smaller values such as 1.5-2.5 should be used to ensure that the fitted cylinders are not too big. This is usually only important for small branches, where the noise level and co-registration accuracy are comparable for the branch diameter.

The cylinders are fitted to each region and if the first fitting is acceptable (not too different from the initial estimates), then we do a second fitting if the following three conditions hold: 1) mean distance to the fitted cylinder is over 5 mm, 2) relative to radius the mean distance to the fitted cylinder is over 5 %, and 3) maximum distance to the cylinder is over 20 % of the radius. The second fitting has the results from the first fitting as initial estimates and there are two options for the second fit: 1) if the first fitting was not too bad, then remove about 30% of the farthest points from the segment. Here the points inside the first cylinder are assigned for this purpose only half their true distance and thus are more likely to be used for the second fitting. 2) if the first fitting was bad in the sense that the radius was over 3 cm and mean distance to the fitted cylinder is over 15 mm, then try to divide the region into two parts and fit two cylinders. Try five combinations for the division with 3-7, 4-6, 5-5, 6-4, 7-3 portions. If the best division has better mean distance to the cylinder than in the first fitting, then use those two cylinders. The idea here is that there might be e.g. a “corner” in the region and thus it would be better to use two cylinders instead of one.

Because the fitted radii of the cylinders forming a branch, particularly for thinner branches, is often varying unnaturally, there are also optional controls on the radius: First, the user can enforce the empirically very universally valid fact that the radius of the child branch is always smaller than the radius of the parent branch. This is enabled with the input *ParentCor* which makes sure that the maximum radii of the cylinders in a branch cannot exceed the radius of the parent cylinder in the parent branch.

Second, we want the local value and change of the radii to have some bounds and to be generally decreasing towards the tip of the branch. We use the following approach to modify the taper, enabled with the input *TaperCor*: First fit a parabola shape taper curve to the branch length-radii data, which gives the local maximum and also the minimum radii values, which is defined by the input *MinCylRad*. We do this as follows: Divide the radii data into few sections based on the length along the branch and compute average for each section. Then scale the averages upward five percent and lastly set the radius to *MinCylRad* at the tip of the branch. The first section is about the first 10% of the branch length, then the next three are the first three quarters of the branch length. These scaled averages and the tip value are the new data where the parabola is fitted in the least squares sense. The parabola gives the maximum local radius and by scaling it downwards with 25% gives the local minimum value. If the radius of a cylinder is above or below these parabola values, then the radii is corrected to be the high or low parabola value. See figure 8 for examples. For segments with small number of cylinders, the radii are modified to be linearly decreasing. Similarly, for the stem we use partially linearly decreasing taper instead of parabolic taper.

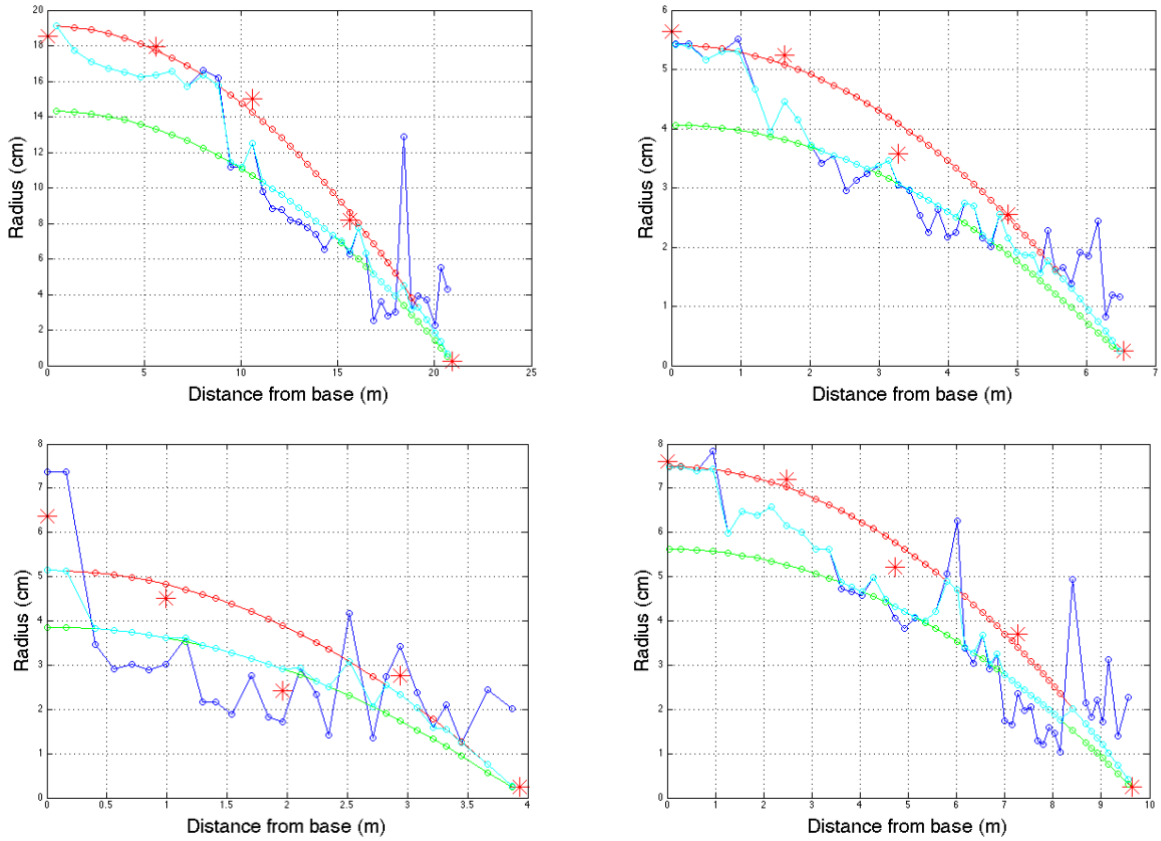


Figure 8. Cylinder radius modification with parabolas. Blue is original fitted radius, red stars are the data (scaled section averages) used for parabola fitting, red and green circled denote the maximum and minimum radius parabolas, light blue circles denote the final modified radii.

There is a third option for controlling the radii of the cylinders: The “growth volume” allometry based approach introduced by Jan Hackenberg in his SimpleTree software. It is enabled by the input *GrowthVolCor* and its effects are controlled by the input *GrowthVolFac.* In this approach, the total volume of all the cylinders following the given cylinder (“the growth volume”) is computed for every cylinder and this produces radius – growth volume data set. Then we fit the following function GrowthVol = a*Radius^b+c in the least squares sense and this function together with the factor *GrowthVolFac* gives the upper and lower bound for the radius of every cylinder.

2.3.2 Triangulation

If the input *Tria* is set to 1, then the software tries to triangulate the bottom part of the stem up to the first branch. The triangulation is based on the boundary curves of horizontal cross-sections of the stem, where the user gives the width and height of the triangles such that the thickness of the cross-section layers equals to the height and the length of the curve line elements is approximately equal to the width.