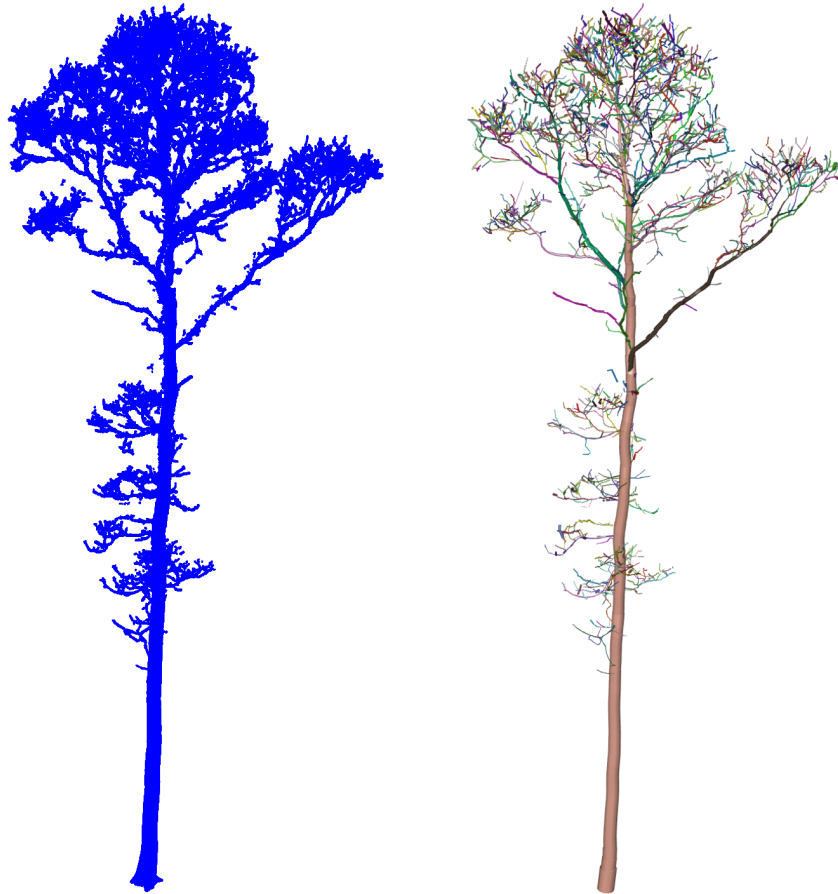


TreeQSM

Quantitative Structure Models for Single Trees from Laser Scanner Data



Instructions for MATLAB-software **TreeQSM**, version 2.4.1

Author: Pasi Raumonen, Mathematics, Tampere University

Email: pasi.raumonen@tuni.fi

Web: <https://research.tuni.fi/inverse/> and <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 R2022a 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) 2013-2022 Pasi Raunonen

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/>

Contents

1	Basics	3
1.1	What is quantitate structure model or QSM?	3
1.2	MATLAB setup	3
1.3	Importing data into MATLAB	3
1.4	The basic command	3
1.5	Inputs structure	4
1.6	Filtering point cloud	4
1.7	Test run	4
1.8	Output structure	7
1.8.1	cylinder	9
1.8.2	branch	9
1.8.3	treedata	9
1.8.4	rundata	17
1.8.5	pmdistance	17
1.8.6	triangulation	17
1.9	Main assumptions of the method	17
1.10	Reconstruction in practice	20
1.10.1	Optimization of input parameters	20
1.10.2	Multiple models with same inputs	21
1.10.3	Sensitivity of results to the input parameters	23
1.10.4	Downsampling point cloud	24
1.10.5	Making QSMs in practice	24
1.10.6	Plotting	26
1.11	QSM simplification	26
2	The reconstruction method - How it works?	27
2.1	Overview of the main steps	27
2.2	Topological reconstruction of branching structure	27
2.2.1	Cover sets	27
2.2.2	Tree sets	29
2.2.3	Segmentation	31
2.2.4	Correct segmentation	32
2.3	Geometrical reconstruction of branch surfaces	33
2.3.1	Surface coverage and surface coverage filtering	33
2.3.2	Cylinder fitting	35
2.3.3	Radius correction	35
2.3.4	Triangulation	37
3	Version history	38

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 consists 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* \rightarrow *Add with Subfolders* \rightarrow *Open* \rightarrow *Save* \rightarrow *Close* to add the subfolders, where all the codes of the software are, to the paths of MATLAB.
3. Import a point cloud of a tree into the workspace. Let us name it P.

1.3 Importing data into MATLAB

Text files can be imported into the workspace using the `import` command or tool. Popular point cloud formats `.las`- and `.laz`-files can be imported using `lasFileReader` and `readPointCloud` functions (these require the Lidar toolbox):

```
>> lasReader = lasFileReader('file_name.las');  
  
>> ptCloud = readPointCloud(lasReader);  
  
>> P = ptCloud.Location;
```

It is also good to note that the data is often in some global coordinate system, meaning that the x - and y -coordinates of the points can have large values, and this is known to cause some issues. Thus, it is recommended to transform the data into a local coordinates system for the QSM reconstruction process:

```
>> P = P-mean(P);
```

1.4 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);
```

where P is (filtered) point cloud, $(m_{points}, 3)$ -matrix, the rows give the coordinates of the points, and `inputs` structure array containing the input parameters.

`inputs` structure is created by first modifying and then running the script `create_input`. Moreover, to define reasonable reconstruction parameters ("PatchDiam"), you can use `define_input` function, which is a new addition. 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.

1.5 Inputs structure

Next the content of `create_input` script is shown and the parameters briefly explained. Notice that the parameter values are usually just examples and they can and in many cases should be changed. Also, to define the values for the important `PatchDiam` and `BallRad` parameters, you can use the new `define_input` function. Tables 1 and 2 explain briefly all the parameters.

1.6 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 filtering the point cloud. Simple filtering based on local statistical knn-distance or point density and separate cluster size can be tried with the following command:

```
Pass = filtering(P0, inputs);  
P = P0(Pass,:);
```

where `P0` is the unfiltered point cloud, `inputs` is the inputs-structure defined above and it defines the filtering parameters. There are two alternative methods to filter points locally, both based on local statistics: Statistical k-nearest neighbour distance outlier filtering computes for the user defined `k` the `k`th-nearest neighbour distance for each point, and then removes all the points whose distance is larger than the average distance + the user defined multiplier for the standard deviation. The average and the standard deviations are computed for each one meter layer of the point cloud so that the usually observed decreasing point density with height is considered. Statistical point density outlier filtering computes for the user defined ball radius the number of points ("density") in the ball for each point, and then removes all the points whose density smaller than the average density - the user defined multiplier for the standard deviation. Similarly, the average and the standard deviations are computed for each one meter layer of the point cloud. There is another method that removes small separate clusters of points. The point cloud is covered with patches (user defined patch size), and the connected components are defined. Then all the components that have less than user defined number of patches are removed.

Notice 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 is recommended to use some other filtering method for that.

1.7 Test run

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

```
oak, Tree = 1, Model = 1  
PatchDiam1 = 0.05  
BallRad1 = 0.065  
PatchDiam2Min = 0.01  
PatchDiam2Max = 0.04  
BallRad2 = 0.05  
nmin1 = 3, nmin2 = 1  
Tria = 1, OnlyTree = 1  
Progress:
```

Table 1: Inputs structure. Accessed by inputs.

QSM reconstruction parameters (can be varied and should be optimised, they can have multiple values given as vectors, e.g. [0.05 0.08]):	
inputs.PatchDiam1	Patch size of the first uniform-size cover
inputs.PatchDiam2Min	Minimum patch size of the cover sets in the second cover
inputs.PatchDiam2Max	Maximum cover set size in the stem's base in the second cover
The following parameters can be varied but can be usually kept as given (i.e. little bigger than PatchDiam parameters, inputs.BallRad1 = inputs.PatchDiam1+0.01):	
inputs.BallRad1	Ball radius in the first uniform-size cover generation
inputs.BallRad2	Maximum ball radius in the second cover generation
The following parameters can be usually kept fixed as given:	
inputs.nmin1	Minimum number of points in BallRad1-balls, generally good value is 3
inputs.nmin2	Minimum number of points in BallRad2-balls, generally good value is 1
inputs.OnlyTree	If 1, point cloud contains points only from the tree
inputs.Tria	If 1, produces a triangulation
inputs.Dist	If 1, computes the point-model distances
Different cylinder radius correction options for modifying too large and too small cylinders.	
inputs.MinCylRad	Minimum cylinder radius, used particularly in the taper corrections
inputs.ParentCor	Radii in a child branch are always smaller than the radii of the parent cylinder in the parent branch
inputs.TaperCor	Use parabola taper corrections
Growth volume correction approach introduced by Jan Hackenberg, allometry: $Radius = a * GrowthVol^b + c$:	
inputs.GrowthVolCor	Use growth volume (GV) correction
inputs.GrowthVolFac	fac-parameter of the GV-approach, defines upper and lower bound of allowed radius from the predicted radius: $1/fac * predicted_radius \leq radius \leq fac * predicted_radius$
Filtering parameters for the "filtering" function, all optional:	
inputs.filter.k	Statistical k-nearest neighbour distance outlier filtering, applied if $filter.k > 0$. The value filter.k is the number of nearest neighbours.
inputs.filter.radius	Statistical point density outlier filtering, applied if $radius > 0$. filter.radius is the radius of the ball neighbourhood.
inputs.filter.nsigma	The multiplier of the standard deviation of the kth-nearest neighbour distance/point density and points whose kth-nearest neighbour distance/point density is larger/lower than the average $\pm filter.nsigma * std$ are removed.
inputs.filter.PatchDiam1	Small component filtering, based on cover whose patches are defined by filter.PatchDiam1 and filter.BallRad1.
inputs.filter.BallRad1	Small component filtering, based on cover whose patches are defined by filter.PatchDiam1 and filter.BallRad1.
inputs.filter.ncomp	Small component filtering, applied if $ncomp > 0$. The points which are included in components that have less than filter.ncomp patches are removed.
inputs.filter.EdgeLength	The length of the cube edges in cubical downsampling, applied if $EdgeLength > 0$.
inputs.filter.plot	Plots the filtering results after the filtering if $plot > 0$.

Table 2: Inputs structure continued. Accessed by inputs.

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	Name string for saving output files and naming models
inputs.tree	Tree index. If modelling multiple trees, then they can be indexed uniquely
inputs.model	Model index, can separate models if multiple models with the same inputs
inputs.savemat	If 1, saves the output struct QSM as a matlab-file into results folder. If name = "pine", tree = 2, model = 5, the name of the saved file is "QSM_pine_t2_m5.mat"
inputs.savetxt	If 1, saves the models in .txt-files
inputs.plot	2 = plots the model, the segmented point cloud and distributions; 1 = plots the model and the segmented point cloud; 0 = plots nothing
inputs.disp	Defines what is displayed during the reconstruction: 2 = display all including QSMs, segmented point cloud, and distributions; 1 = display name, parameters and distances; 0 = display only the name

Cover sets 3.5 sec. Total: 3.5 sec
 Tree sets 0.1 sec. Total: 3.6 sec
 Initial segments 0.8 sec. Total: 4.5 sec
 Final segments 0.4 sec. Total: 4.8 sec
 Cover sets 19.4 sec. Total: 24.2 sec
 Tree sets 8.7 sec. Total: 32.9 sec
 Initial segments 12.6 sec. Total: 45.5 sec
 Final segments 3.3 sec. Total: 48.8 sec
 Cylinders 1 min 50.9 sec. Total: 2 min 39.6 sec

Tree attributes:

TotalVolume = 836.8 L
 TrunkVolume = 582 L
 BranchVolume = 254.8 L
 TreeHeight = 19.29 m
 TrunkLength = 17.89 m
 BranchLength = 668.4 m
 TotalLength = 686.2 m
 NumberBranches = 1891
 MaxBranchOrder = 8
 TrunkArea = 10.63 m^2
 BranchArea = 42.6 m^2
 TotalArea = 53.23 m^2
 DBHqsm = 0.2617 m
 DBHcyl = 0.2609 m
 CrownDiamAve = 5.282 m
 CrownDiamMax = 7.116 m
 CrownAreaConv = 25.32 m^2
 CrownAreaAlpha = 22.51 m^2
 CrownBaseHeight = 5.904 m
 CrownLength = 13.39 m
 CrownRatio = 0.6939

CrownVolumeConv = 187.3 m^2
CrownVolumeAlpha = 84.66 m^2

Tree attributes from triangulation:

DBHtri = 0.2559 m
TriaTrunkVolume = 303.4 L
MixTrunkVolume = 571.2 L
MixTotalVolume = 825.9 L
TriaTrunkArea = 4.856 m^2
MixTrunkArea = 10.6 m^2
MixTotalArea = 53.2 m^2
TriaTrunkLength = 6.096 m

Branch and data 3.7 sec. Total: 2 min 43.4 sec

PatchDiam1 = 0.05, PatchDiam2Max = 0.04, PatchDiam2Min = 0.01
Average cylinder-point distance (trunk, branch, 1branch, 2branch): 3.9 4.8 3.7 4.3 mm

Distances 1.7 sec. Total: 2 min 45.1 sec

Figures 1, 2 and 3 show examples of the branch-segmented point clouds, QSMs and triangulation model of the stem.

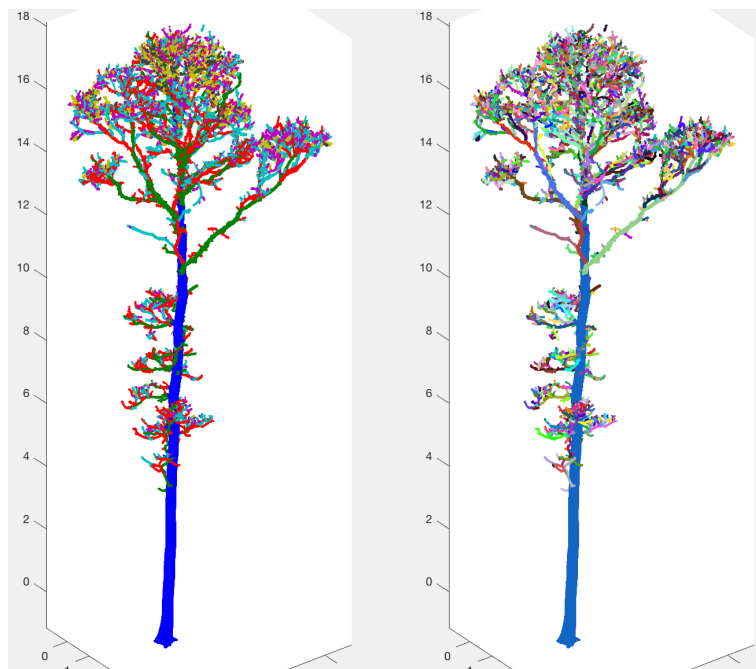


Figure 1: Branch-segmented point clouds. Left: Colors based on branching order (blue = stem, green = 1st-order branches). Right: Unique colour for each branch.

1.8 Output structure

The output QSM is a structure array. It contains the structures: `cylinder`, `branch`, `treedata`, `rundata`, `pmdistance` and `triangulation`.

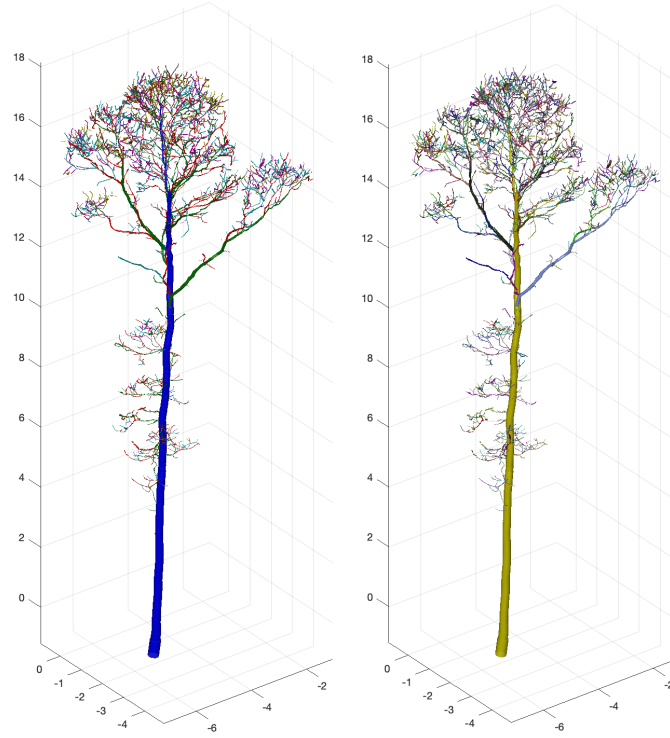


Figure 2: QSMs. Left: Colors based on branching order (blue = stem, green = 1st-order branches). Right: Unique colour for each branch.

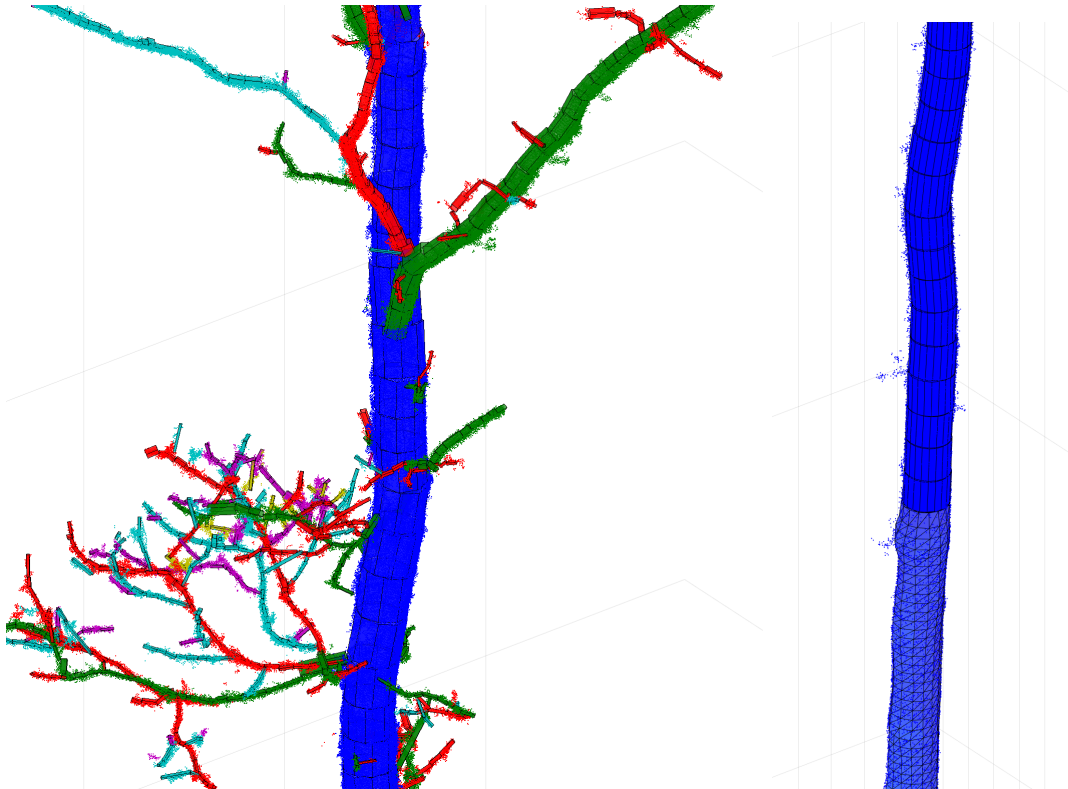


Figure 3: Close-ups of the QSM and stem model with triangulation: Left: QSM together with the point cloud. Right: Triangulated part (bottom) and cylinders (top) of the stem.

1.8.1 cylinder

The **cylinder** structure contains info of each cylinder, whose name/index is the row number in the file, and the fields contained in the structure are given in Table 3.

Table 3: Cylinder structure. Accessed by **QSM.cylinder**.

field name	explanation
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, unit vector
parent	parent cylinder (row number)
extension	extension cylinder (row number)
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
mad	mean absolute distance of points to the cylinder surface
SurfCov	surface coverage
UnmodRadius	radius of the cylinder before any possible modifications
added	if =1, then the cylinder is added after normal cylinder fitting

1.8.2 branch

The **branch** structure contains info of each branch, whose name/index is the row number in the file, and the fields contained in the structure are given in Table 4.

Table 4: Branch structure. Accessed by **QSM.branch**.

order	branch order (0 = trunk, 1 = branches starting from the trunk, etc.)
parent	parent branch (row number)
volume	volume (L) of the branch
area	area (m^2) of the branch
length	length (m) of the branch
angle	branching angle (deg) (between the branch and its parent)
height	height (m) of the branch base from the ground
azimuth	azimuth (deg) of the branch's base, angle to x-axis in the xy-plane
zenith	zenith (deg) of the branch's base, cylinder's angle to z-axis
diameter	diameter (m) of the branch at the base

1.8.3 treedata

The **treedata** structure contains a lot of single-number tree attributes (volumes, areas, lengths, etc.) and distributions computed from the QSM, some of which can be displayed or plotted at the end of the modelling run (see the above example). These fields are explained in Table 5 and 6

Crown definition for those attributes, such as crown base height, where the actual crown definition is needed: Define first major branch as a large enough branch (large enough radius and reach) whose $diameter < \min(0.05 * dbh, 5cm)$ and whose horizontal relative reach is more than the median reach of 1st-order branches (or at maximum 10). The reach is defined as the horizontal distance from the base to the tip divided by the dbh.

When a triangulation model is reconstructed for the bottom of the stem, then additional fields of **treedata** are included and these are given in Table 7.

Table 5: Treedata structure. Accessed by `QSM.treedata`.

TotalVolume	Total volume (L) of the tree
TrunkVolume	Volume (L) of the stem
BranchVolume	Volume (L) of all the branches
TreeHeight	Height (m) of the tree
TrunkLength	Length (m) of the stem
BranchLength	Total length (m) of all the branches
TotalLength	Total length (m) of all the branches and stem
NumberBranches	Number of branches
MaxBranchOrder	Maximum branching order
TrunkArea	Total surface area (m^2) of the stem
BranchArea	Total surface area (m^2) of the branches
TotalArea	Total surface area (m^2) of the tree
DBHqsm	DBH (m), the diameter of the cylinder in the QSM at the right height
DBHcyl	DBH (m), the diameter of the cylinder fitted to the height 1.1-1.5 m
CrownDiamAve	Average crown diameter (m): planar projection of the tree crown is divided into 18 cones (10 deg sector and its opposite sector) and for each cone the maximum extent is computed and averaged over all cones
CrownDiamMax	Maximum horizontal crown diameter (m)
CrownAreaConv	Area (m^2) of the crown's planar projection's convex hull
CrownAreaAlpha	Area (m^2) of the crown's planar projection's alpha shape
CrownBaseHeight	Crown's base height (m) from the ground
CrownLength	Crown's vertical length (m)
CrownRatio	Ratio of the crown length to the tree height
CrownVolumeConv	Volume (L) of the crown's convex hull
CrownVolumeAlpha	Volume (L) of the crown's alpha shape
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
spreads	Horizontal spreads (m) of the tree in 18 directions and in 20 (tree height over 10 m) or 10 (height over 2 m) or 5 (height under 2 m) height layers.
VerticalProfile	Average spread in each height layer giving kind of vertical profile of the tree, how spread the tree crown is in average at different heights

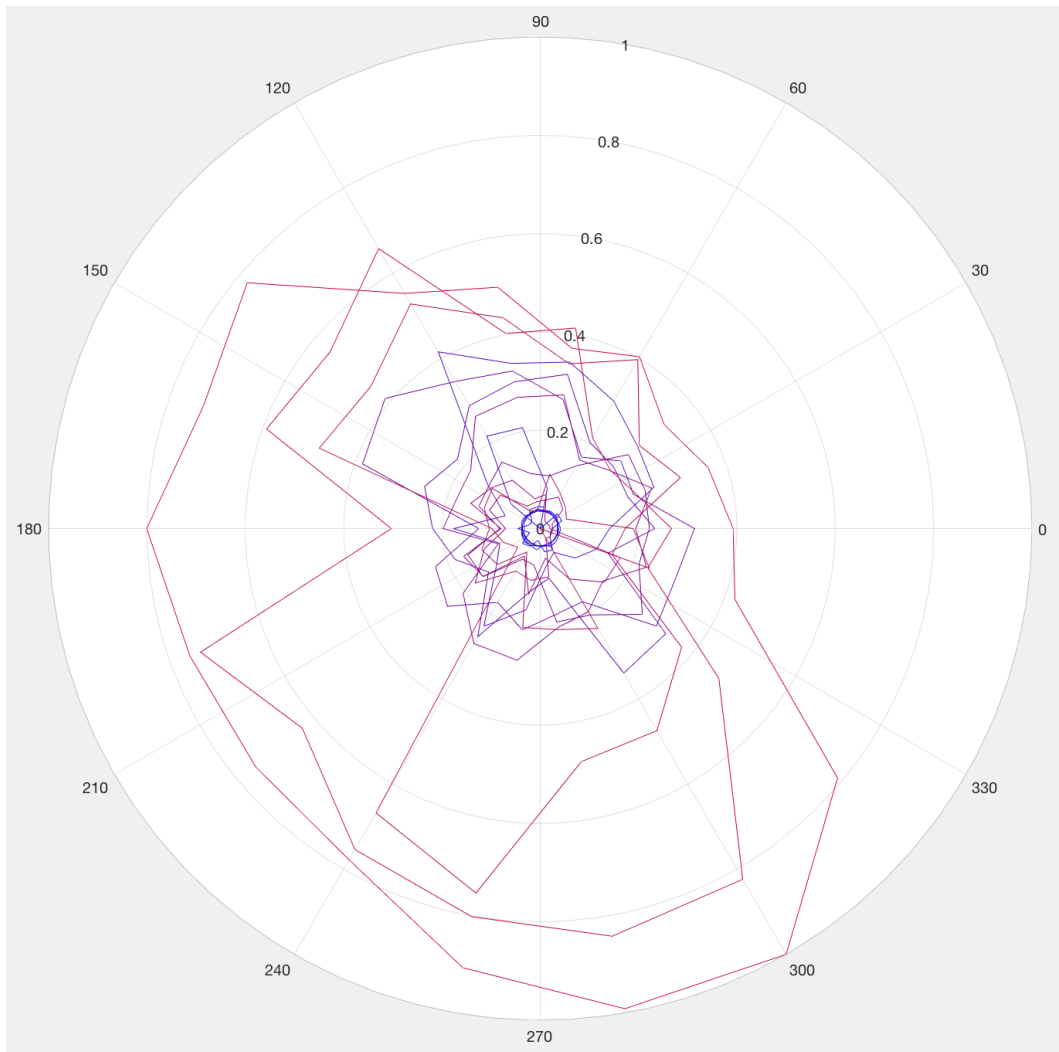


Figure 4: Relative spreads for each height layer (different colors, from blue = lowest to red = highest) shown in a polar plot. To plot this figure use command `plot_spreads(QSM.treedata,1,1,1)`.

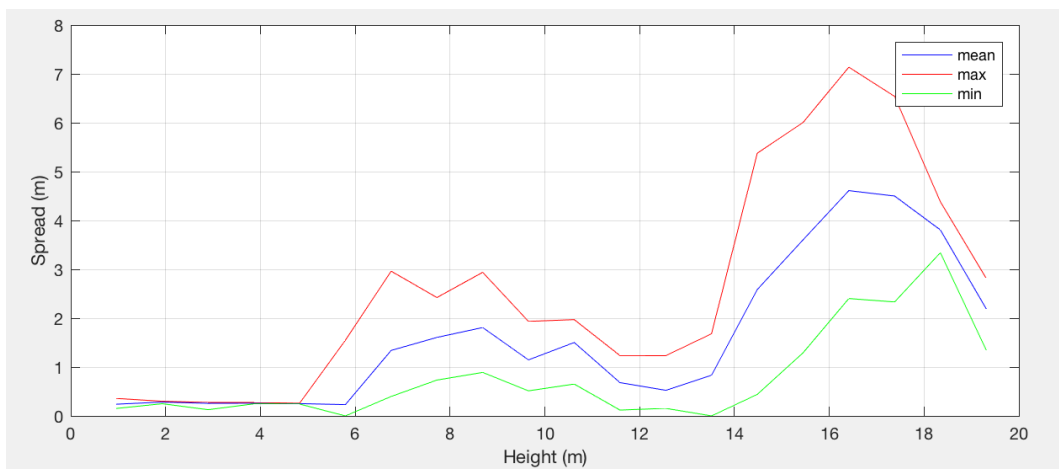


Figure 5: Average ("Vertical profile"), maximum and minimum spreads for each height layer. Notice in the first five meters there are no branches and the spread is from the stem alone that is why it is so small and the minimum and maximum are almost the same.

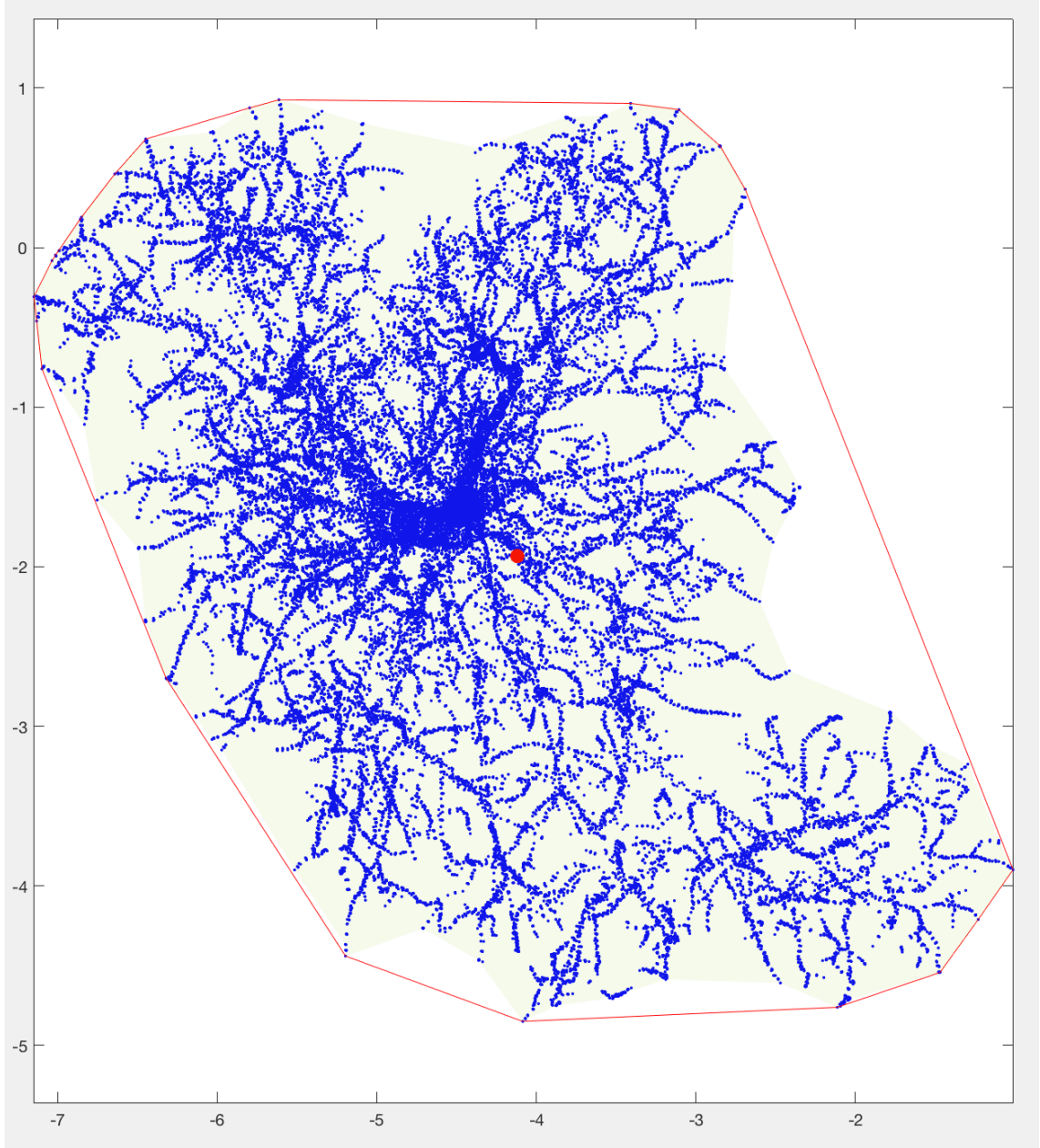


Figure 6: Crown area from the convex hull and an alpha shape. The red line shows the convex hull. The shaded area is the alpha shape. The red point in the middle is the center of gravity of the convex hull and it is used to compute the average crown diameter as the mean of the spreads into 18 directions.

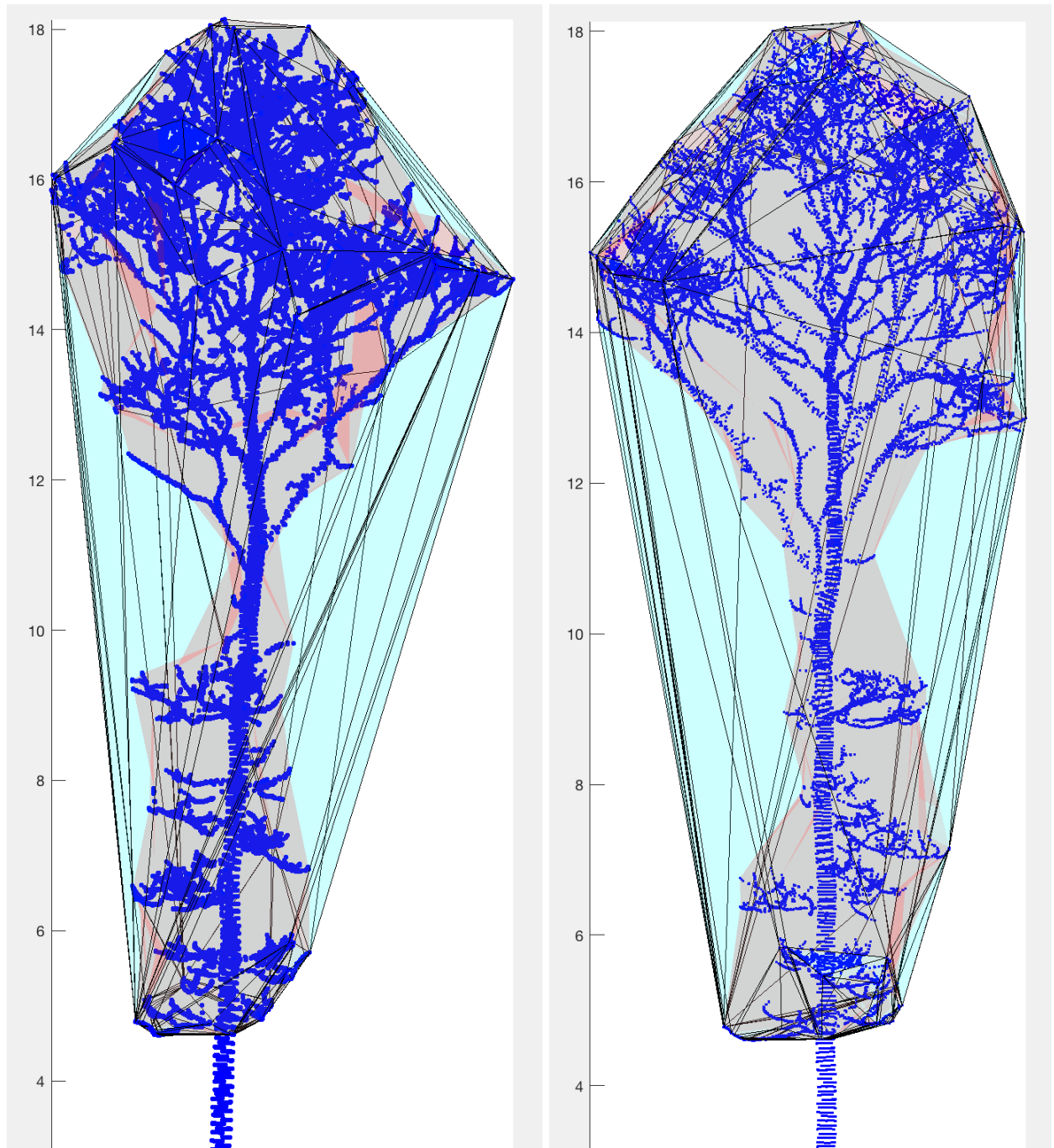


Figure 7: Crown volume from the convex hull and an alpha shape. The two figures give two different point of views into the same model.

Table 6: Treedata structure continued. Accessed by `QSM.treedata`.

VolCylDia, AreCylDia, LenCylDia		Volume, area and length of the cylinders as function of the cylinder's diameter (1 cm diameter classes)
VolCylHei, AreCylHei, LenCylHei		Volume, area and length of the cylinders as function of the cylinder's height (1 m height classes)
VolCylZen, AreCylZen, LenCylZen		Volume, area and length of the cylinders as function of the cylinder's zenith (10 deg angle classes)
VolCylAzi, AreCylAzi, LenCylAzi		Volume, area and length of the cylinders as function of the cylinder's azimuth (10 deg angle classes)
VolBranchOrd, Are-, Len-, Num-		Volume, area, length, and number of branches (not stem) as functions of branch order
VolBranchDia, Are-, Len-, Num-		Volume, area, length, and number of branches (not stem) as functions of branch diameter at the base (1 cm diameter classes)
VolBranch1Dia, Are-, Len-, Num-		Volume, area, length, and number of 1st-order branches as functions of branch diameter at the base (1 cm diameter classes)
VolBranchHei, Are-, Len-, Num-		Volume, area, length, and number of branches (not stem) as functions of branch height at the branch base (1 m height classes)
VolBranch1Hei, Are-, Len-, Num-		Volume, area, length, and number of 1st-order branches as functions of branch height at the the branch base (1 m height classes)
VolBranchAng, Are-, Len-, Num-		Volume, area, length, and number of branches (not stem) as functions of branching angle (10 deg angle classes)
VolBranch1Ang, Are-, Len-, Num-		Volume, area, length, and number of 1st-order branches as functions of branching angle (10 deg angle classes)
VolBranchAzi, Are-, Len-, Num-		Volume, area, length, and number of branches (not stem) as functions of branch azimuth angle at the branch base (10 deg angle classes)
VolBranch1Azi, Are-, Len-, Num-		Volume, area, length, and number of 1st-order branches as functions of branch azimuth angle at the branch base (10 deg angle classes)
VolBranchZen, Are-, Len-, Num-		Volume, area, length, and number of branches (not stem) as functions of branch zenith angle at the branch base (10 deg angle classes)
VolBranch1Zen, Are-, Len-, Num-		Volume, area, length, and number of 1st-order branches as functions of branch zenith angle at the branch base (10 deg angle classes)

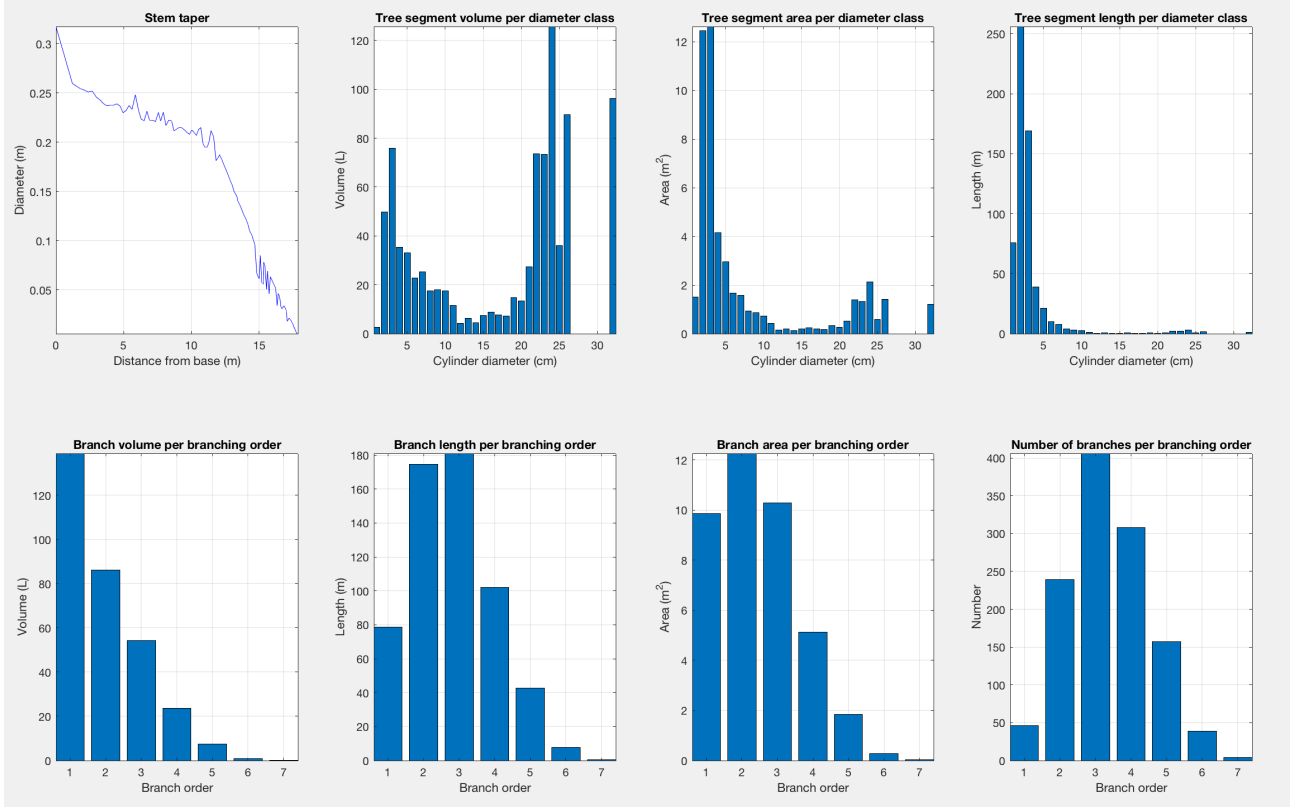


Figure 8: Stem taper, cylinder ("Tree segment") volume, area and length distributed in diameter classes, branch volume, area, length and number distributed in branching orders. To plot these distributions use the command `plot_distribution(QSM,fig,rel,cum,dis)`, where `fig` is the figure number, `rel = 1` means using relative values, `cum = 1` means cumulative distributions, `dis` is the field name of the plotted distribution. For example, `plot_distribution(QSM,1,0,0,'VolCylDia')` plots the volume-diameter distribution with absolute values into figure 1. It is possible to plot multiple different distributions in the same figure, e.g. `plot_distribution(QSM,15,1,0,'VolCylDia','AreCylDia')`, in which case relative values are used.

Table 7: Treedata structure, additional field if `inputs.Tria = 1`.

DBHtri	DBH (m), mean length of the "diagonals" in the triangulation
TriaTrunkVolume	Volume (L) of the triangulation
MixTrunkVolume	Volume (L) of the stem, computed from triangulation (bottom) and cylinders (top)
MixTotalVolume	Total volume (L) of the tree = MixTrunkVolume + BranchVolume
TriaTrunkArea	Side area (m^2) of the triangulated volume
MixTrunkArea	Area (m^2) of the stem, computed from triangulation (bottom) and cylinders (top)
MixTotalArea	Total area (m^2) of the tree = MixTrunkArea + BranchArea
TriaTrunkLength	Length/height (m) of the triangulated stem part

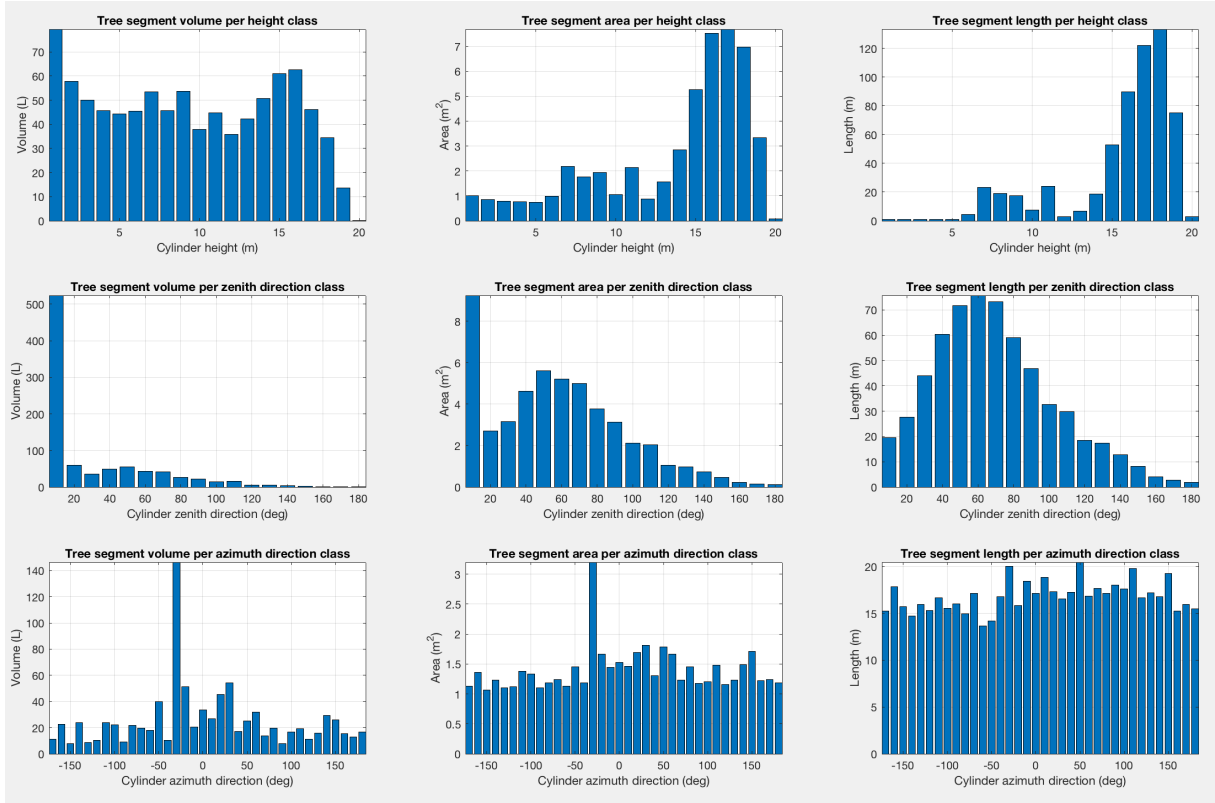


Figure 9: Cylinder ("Tree segment") volume, area and length distributed in height, zenith angle, and azimuth angle classes.

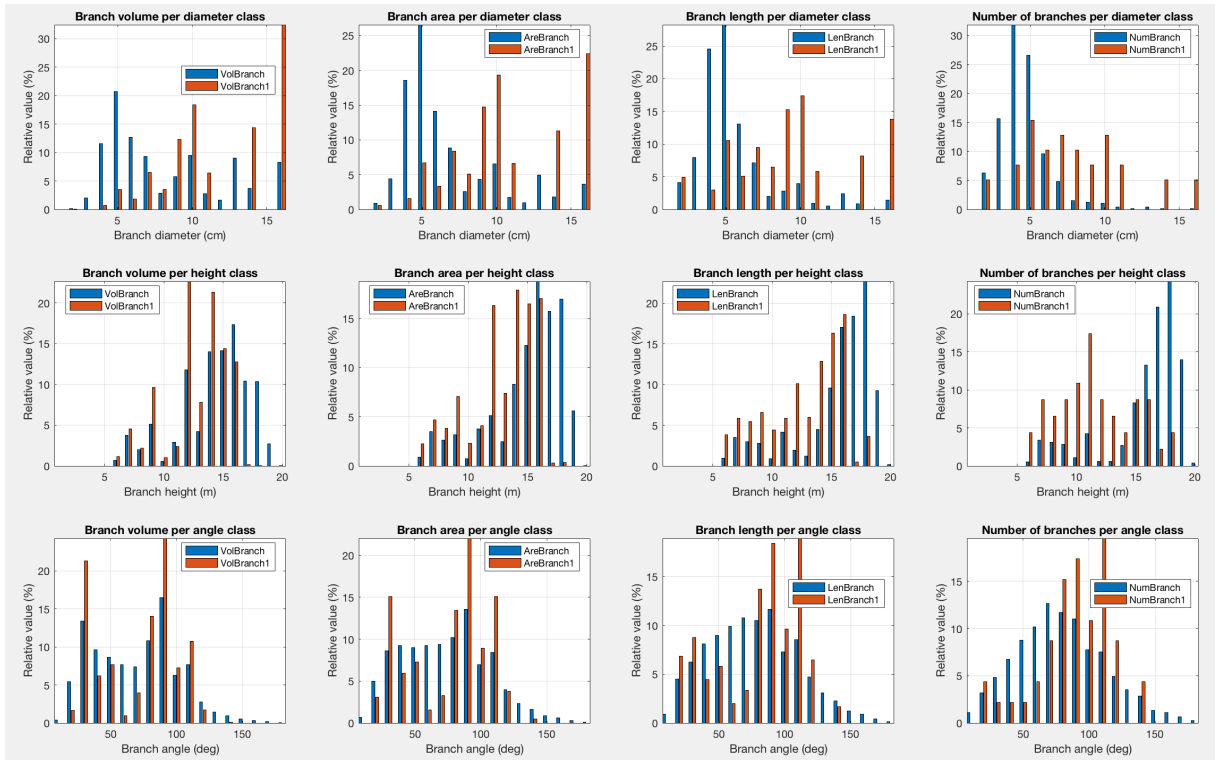


Figure 10: All branch and 1st-order branch volume, area, length, and number distributed in diameter, height and branching angle classes.

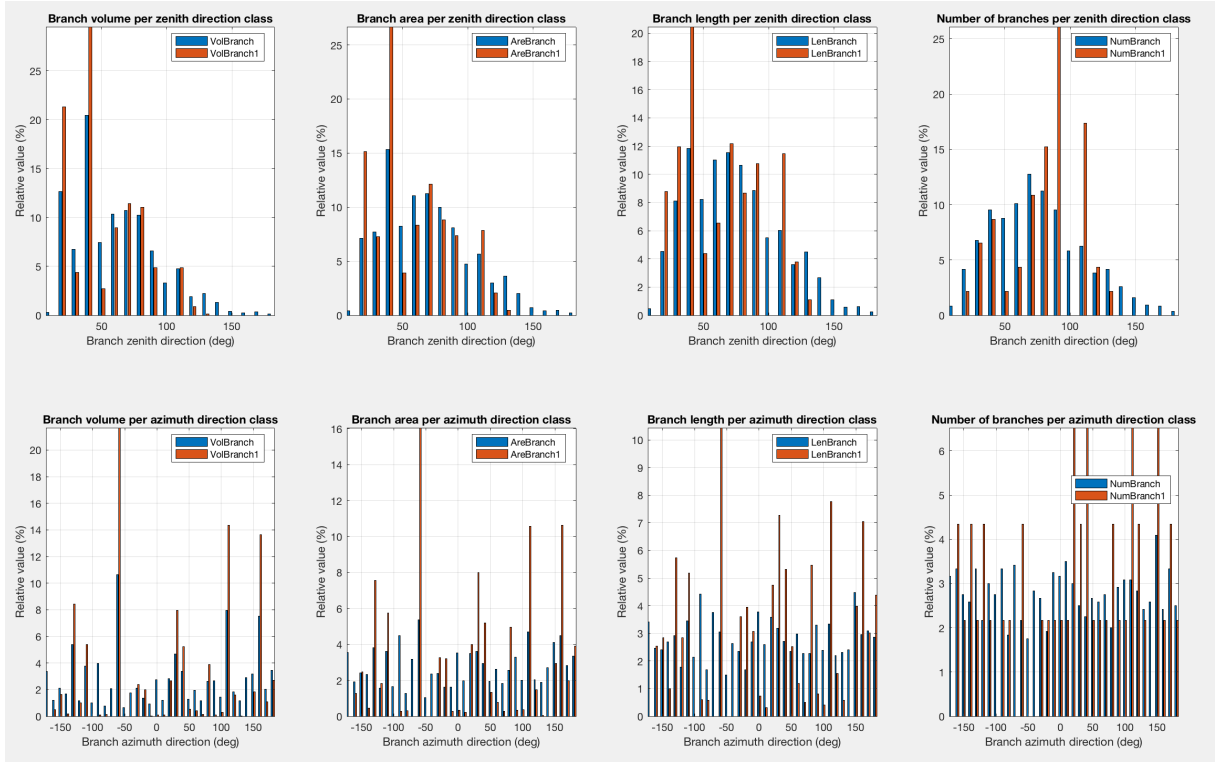


Figure 11: All branch and 1st-order branch volume, area, length, and number distributed in zenith and azimuth angle classes.

1.8.4 rundata

The `rundata` structure contains the fields given in Table 8.

Table 8: Rundata structure. Accessed by `QSM.rundata`.

inputs	the input structure
time	computation times of the main reconstruction step
date	date and time of the start and end of the reconstruction
version	the version of the <code>treeqsm</code> used for the QSM reconstruction

1.8.5 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`. Table 9 explains the fields.

1.8.6 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 12. The triangulation can be turned on by setting: `inputs.Tria = 1`. Table 10 explains the fields.

1.9 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:

Table 9: Point-model distance structure. Accessed by `QSM.pmdistance`.

CylDist	mean point-model distance (m) for each cylinder (vector)
median	median of CylDist
mean	average of CylDist
max	maximum of CylDist
std	standard deviation of CylDist
TrunkMedian	median of trunk cylinders' CylDist
TrunkMean	average of trunk cylinders' CylDist
TrunkMax	maximum of trunk cylinders' CylDist
TrunkStd	standard deviation of trunk cylinders' CylDist
BranchMedian	median of branch cylinders' CylDist
BranchMean	average of branch cylinders' CylDist
BranchMax	maximum of branch cylinders' CylDist
BranchStd	standard deviation of branch cylinders' CylDist
Branch1Median	median of 1st-order branch cylinders' CylDist
Branch1Mean	average of 1st-order branch cylinders' CylDist
Branch1Max	maximum of 1st-order branch cylinders' CylDist
Branch1Std	standard deviation of 1st-order branch cylinders' CylDist
Branch2Median	median of 2nd-order branch cylinders' CylDist
Branch2Mean	average of 2nd-order branch cylinders' CylDist
Branch2Max	maximum of 2nd-order branch cylinders' CylDist
Branch2Std	standard deviation of 2nd-order branch cylinders' CylDist

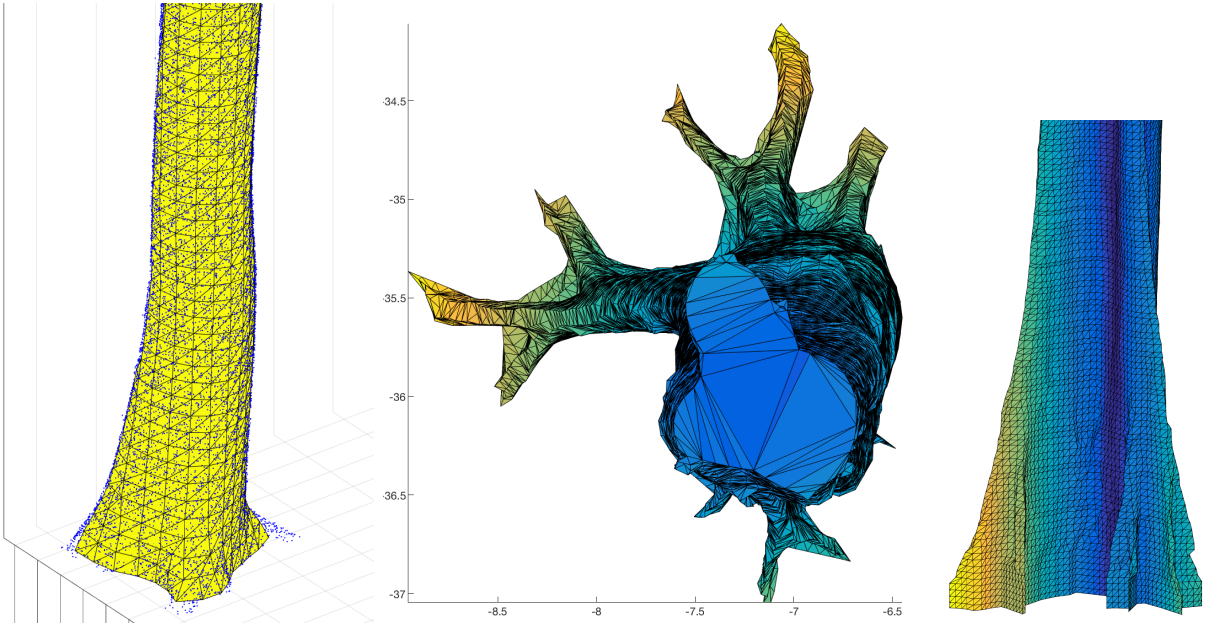


Figure 12: Triangulation models of the bottom part of trunks. Plot the triangulation plus the rest of the stem or the tree with cylinders using the code `plot_triangulation(QSM)`.

Table 10: Triangulation structure. Accessed by `QSM.triangulation`.

vert	coordinates of the vertices of the model
facet	indices of vertices forming each triangle
fvd	colour information for plotting the model with patch-command
volume	volume (L) enclosed by the model
SideArea	area (m^2) of the triangulated surface (not including the area of top and bottom)
BottomArea	area (m^2) of the triangulated bottom surface
TopArea	area (m^2) of the triangulated top surface
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

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 modelling 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 (`inputs.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 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 option for radius control to have tapering branches is to use growth volume approach (`inputs.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).

1.10 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.

1.10.1 Optimization of input parameters

If the values of input parameters, such as `PatchDiam2Min` and `PatchDiam2Max`, 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 there are only 3 (`PatchDiam1`, `PatchDiam2Max`, `PatchDiam2Min`) that are most meaningful in this respect. 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 3 parameters makes 27 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 of the Matlab).

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 2.4.0 there are readily available more than 90 different other metrics that the user can choose. And the user can easily modify them or add new ones. There are four types of metrics

available: cylinder distance, standard deviation, distribution-, and surface coverage- 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. The idea of SD metrics is that if e.g. the SD of total volume is small, then the modelling is robust which indicates some kind of reliability in the models. Thus, for the SD based metrics it is important to make many models with the same inputs in order to have robust SDs. The distribution based metrics include e.g. mean difference in volume of 1-10 cm diameter cylinder classes. That is, we compare volume distributions (mean difference in small branches) between different models and the parameters that produce the smallest difference in the distributions is the most robust and indicates again reliability in the models. Lastly, the surface coverage (SC) based metrics measure how much cylinder's surface is covered with points (high coverage indicates high reliability) and the metrics minimize 1-SC, which maximizes SC. The surface coverage is defined in section 2.3.1.

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 a lot of similar trees (similar size, height, complexity) and with similar measurements (similar number of scans, scanner distances, point density, occlusion) it might be a workable option just to 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.

1.10.2 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.

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 13. 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 a large number of times to estimate the distributions of sample means and sample standard deviations for each sample size (see Figures 14 and 15).

Figure 14 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 a 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.

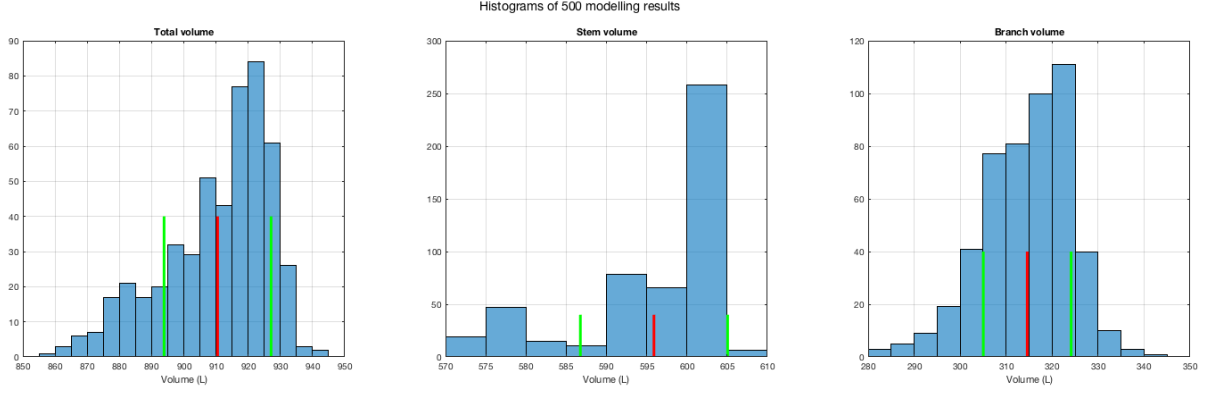


Figure 13: Volume results from 500 QSMs shown in histograms. Total (left), stem (middle) and branch (right) volumes shown. Red lines indicate the mean volumes and green lines indicate the mean \pm one standard deviation volumes.

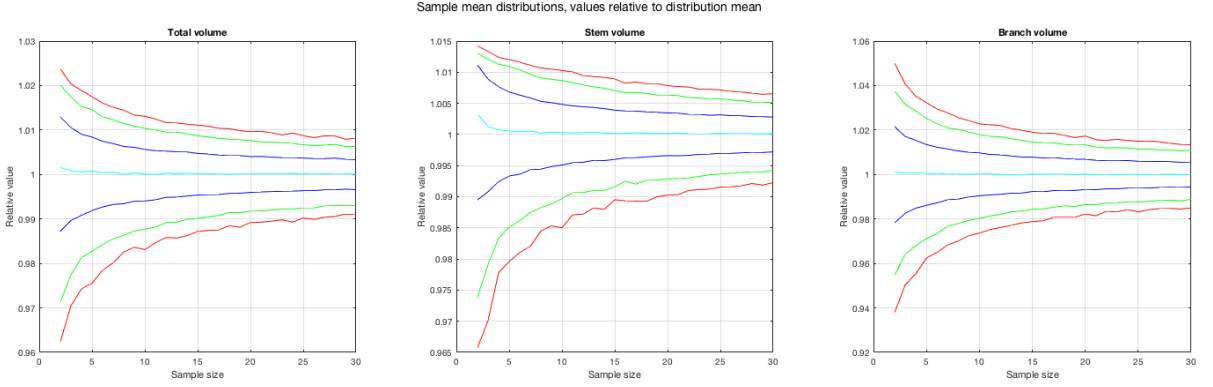


Figure 14: Sample mean distributions for different sample sizes. The values are relative to the means of the empirical distributions of 500 models. Inside the red and green lines are 99% and 95% of sample means, respectively. Light blue lines are the medians of samples and the blue lines show \pm standard deviation from the mean.

Figure 15 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 Figure 15 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.

As it is recommended to produce multiple QSMs (e.g. 5) per each input, the single number tree attributes (e.g. total volume) can be computed as the average from the QSMs. Similarly, the standard deviation of the attribute computed from the QSMs with the same inputs, is an estimate for the uncertainty or precision of the results. The function `select_optimum` not only selects the optimal inputs but also computes the "treedata" from the QSMs made with the optimal inputs and gives them in the form: *[average SD]*.

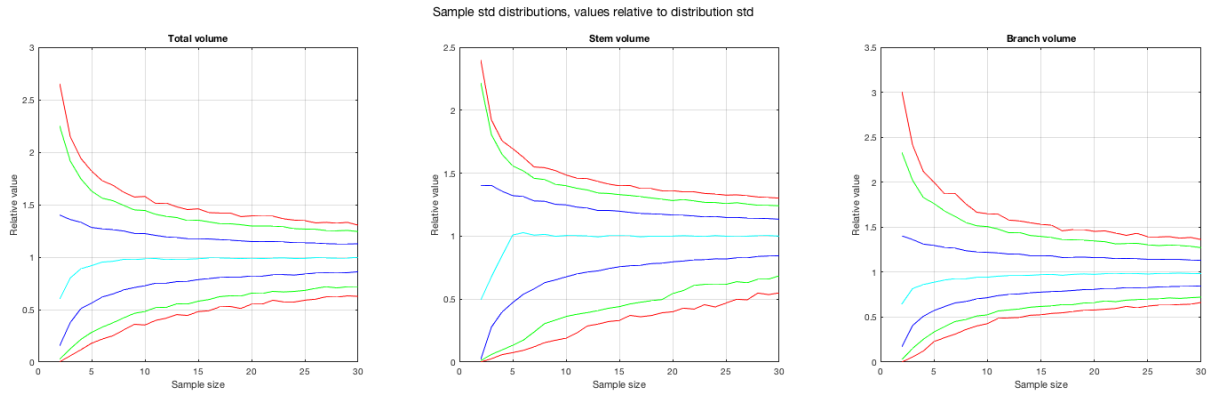


Figure 15: Sample standard deviation distributions for different sample sizes. The values are relative to the standard deviations of the empirical distributions of 500 models. Inside the red and green lines are 99% and 95% of sample means, respectively. Light blue lines are the medians of samples and the blue lines show \pm standard deviation from the mean.

1.10.3 Sensitivity of results to the input parameters

The values of the parameters `PatchDiam1`, `PatchDiam2Max`, `PatchDiam2Min` need to be optimized, and then we use the QSMs made with the optimal values. However, as the optimization is not perfect (e.g. was the values chosen reasonably, was the metric reasonable, etc.), it could be important to know if the optimal results (e.g. total volume) computed from the optimal QSMs are robust: Do the results change a lot if we change the parameter values little or not? Thus if the result do not change a lot if we had chosen other but close values for the parameters, then results are robust and do not depend much how the parameter values were chosen and optimized.

The function `select_optimum` computes the sensitivity for the "treedata" as follows: The sensitivity of an attribute is its relative change divided by the relative change of the input parameter value and the value is expressed as percentage, i.e. multiplied by 100. Let us explain how the sensitivity of total volume is computed for `PatchDiam1` as it is computed similarly for `PatchDiam2Min` and `PatchDiam2Max`. We first fix `PatchDiam2Min` and `PatchDiam2Max` to their optimal values, so we select only those QSMs that have these as inputs. Then we further select the QSMs whose `PatchDiam1` values are the closest to the optimal `PatchDiam1` value. Say, `PatchDiam1` has values `[0.1 0.2 0.3 0.4 0.5]` and the optimal value is 0.4, then we select the values 0.3 and 0.5 as they are the closest to 0.4. In this case the relative change in the input is $(0.5 - 0.4)/0.4 = (0.4 - 0.3)/0.4 = 0.1/0.4 = 0.25$. Then we compute the change in the volume for the QSMs that have `PatchDiam1` = 0.3 or 0.5, to the QSM with the optimal `PatchDiam1` = 0.4. Say $V_{0.3} = 1000$ and $V_{0.5} = 1100$ and $V_{opt} = V_{0.4} = 1060$, in which case the absolute changes are $1060 - 1000 = 60$ and $1100 - 1060 = 40$, and the maximum of these is 60. Relative to the optimal value we get $60/1060 = 0.0566$. Finally the sensitivity is the quotient of the two relative changes: Sensitivity of total volume for `PatchDiam1` parameter = relative change in volume divided by relative change in input = $0.0566/0.25 = 0.226 = 22.6\%$. This sensitivity value can be interpreted as follows: If the `PatchDiam1` value changes 1% (from the optimal value), then the total volume changes 0.226%, or equivalently, if `PatchDiam1` changes 10%, then the total volume changes 2.6%. In other words, in this example, the change of 1% or 10% of the `PatchDiam1` value from 0.4 to 0.404 or 0.44, would result in change of 0.226% or 2.6% of the volume from 1060 to 1062 or 1084. Sensitivity values of a few percents indicate very robust results.

The output "TreeData" of the function `select_optimum` contains (in version 2.4.1) five values for each attribute: `[average SD sensitivity_for_PatchDiam1 sensitivity_for_PatchDiam2Min sensitivity_for_PatchDiam2Max]`.

1.10.4 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. However, as we often want to reconstruct branches whose diameter can be even under 1 cm, then it does not make sense to downsample the whole point cloud so that the "resolution" of the point cloud is e.g. 1 or 2 cm.

The function `cubical_downsampling` 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`. Notice also that `cubical_downsampling` is now also part of the `filtering` function.

1.10.5 Making QSMs in practice

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

- (1) **(Optional) Point cloud filtering.** Filter out noise, e.g. using `filtering`, and leaves from the point clouds. NOTE: the leaf-filtering needs a separate method/code not included in `TreeQSM` package. Notice that downsampling of the point cloud is also possible with `filtering`: First define the filtering parameters with the `create_input.m` script (one needs to modify the script) and then apply the filtering:

```
>> inputs = create_input;  
>> F = filtering( P0 , inputs );  
>> P = P0( F , : );
```

- (2) **Save the point clouds into single .mat-file.** For example, save five point clouds P1, P2, P3, P4, P5 into file `trees.mat`:

```
>> save( 'trees' , 'P1' , 'P2' , 'P3' , 'P4' , 'P5' );
```

Notice that to make the process work optimally and quickly, it might be useful to save and process similar size trees and similar quality point clouds together, separately from trees with clearly different size and quality. This way one can select less and better input parameters (next step).

- (3) **Define the input parameters.** Give multiple reasonable values for the parameters `PatchDiam1`, `PatchDiam2Min` and `PatchDiam2Max` in the `create_input.m` script (and possible modify other parameters). Then run the script to produce the `inputs` structure:

```
>> create_input
```

NOTE: A recommended option to define reasonable values for `PatchDiam`- and `BallRad`-parameters for each tree separately is to use `define_input` function:

```
>> inputs = define_input( P , nPD1, nPD2Min, nPD2Max);
```

where `nPD1`, `nPD2Min`, `nPD2Max` are the number of values for the parameters. For example, reasonable choice would be `nPD1 = 2`, `nPD2Min = 3`, `nPD2Max = 2` or `nPD1 = 2`, `nPD2Min = 4`, `nPD2Max = 3`. You can generate the inputs for all the points clouds saved in .mat file

```
>> inputs = define_input( 'trees' , nPD1, nPD2Min, nPD2Max);
```

NOTE: When using `define_input` you still need to define the other inputs with the `create_input`, because `define_input` just updates the `PatchDiam`-parameters.

- (4) **Use `make_models` (or `make_models_parallel`) to produce QSMs.** Call the functions with the name (and address) of the point clouds file, the name of the file where the QSMs are saved, and the number of model-runs (e.g. 5-10), and `inputs` (created above) as the inputs:

```
>> QSMs = make_models( 'trees' , 'QSMs_trees' , 5 , inputs );
```

- (5) **Select optimal QSMs.** Use `select_optimum` function to select the optimal QSMs and to compute results:

```
>> [ TreeData , OptModels , OptInputs , OptQSM ] = select_optimum( QSMs );
```

If no metric is specified, the default average cylinder point-model distance is used. But one can and should try other metrics by specifying the metric as the second input:

```
>> TreeData = select_optimum( QSMs , 'trunk+branch_mean_dis' );
```

The output `TreeData` contains the averages and the standard deviations computed from the optimal models for each single number attribute (not distributions). It contains now also sensitivity of the attributes to the `PatchDiam`-parameters (columns 3 to 5). The output `OptModels` contains the list of indexes of the optimal QSMs (the QSMs with the optimal inputs) and the index of the single best QSM for each tree. The output `OptInputs` contains the optimal inputs for each tree. The output `OptQSM` is similar structure as `QSMs` but now contains the single best QSM for each tree.

- (6) **(Optional) Estimate uncertainty more reliably.** 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:

```
>> QSMs2 = make_models( 'trees' , 'QSMs_trees2' , 20 , OptInputs );
```

```
>> QSMs2 = make_models_parallel( 'trees' , 'QSMs_trees2' , 20 , OptInputs );
```

Then use `estimate_precision` to combine all the optimal models and compute the standard deviations from all the optimal models:

```
>> [ TreeData , OptQSMs , OptQSM ] = estimate_precision( QSMs , QSMs2 , TreeData , OptModels );
```

Now the `TreeData` contains better estimates for the averages and in particular for the standard deviation.

Convenient and perhaps most of the time reasonable way to select values for the parameters `PatchDiam1`, `PatchDiam2Min` and `PatchDiam2Max` (and the `BallRad`) is to use the `define_input` function. User only needs to determine how many values for each parameters are needed. Usually 2-3 values for `PatchDiam1` and `PatchDiam2Max` and the 3-5 values for `PatchDiam2Min` are enough.

However, if you are not using the `define_input` function, there are few rule of thumbs for selecting reasonable values for the parameters `PatchDiam1`, `PatchDiam2Min` and `PatchDiam2Max`:

1) The larger the tree and its details are, or the lower the point density of the point cloud is, or the more occlusion there are in the point cloud, particularly if the tree was scanned only from one or two positions and large parts of the tree are occluded, the larger the values for the parameters should be (larger patches). Also the larger the difference between `BallRad` and `PatchDiam` parameters should be.

2) Usually the optimal values are as small as possible so that the smaller the values are, then often the better the results are, but when the values go below certain thresholds (depending on the tree's size and complexity, resolution, occlusion, etc.) the result quickly become much worse.

3) Usually the most important parameter is `PatchDiam2Min` so it could have more values than `PatchDiam1` and `PatchDiam2Max`.

For example, if we have a 20-30 m high tree with the total volume of few cubic meters and with high enough resolution (e.g. about 1 or more points per 1 cubic centimetre), not too much occlusion and lot of small branches, then `PatchDiam1` could have values [0.05 0.07], `PatchDiam2Min` could have values [0.01 0.015 0.02] or [0.008 0.013 0.018 0.023], and `PatchDiam2Max` could have values [0.06 0.08] or [0.05 0.065 0.08] (the units are meters). In this case `BallRad` parameters could have values `BallRad1` = `PatchDiam1`+0.02 and `BallRad2` = `PatchDiam2Max`+0.01. If, on the other hand, we have a 2-7 m high tree with the total volume of few tens of liters and with high enough resolution (e.g. about 1 point per 1 cubic millimetre), not too much occlusion and lot of small branches, then

PatchDiam1 could have values [0.015 0.02], **PatchDiam2Min** could have values [0.004 0.008 0.012] or [0.003 0.006 0.009 0.012], and **PatchDiam2Max** could have values [0.015 0.02] or [0.015 0.02 0.025]. In this case **BallRad** parameters could have values **BallRad1** = **PatchDiam1**+0.007 and **BallRad2** = **PatchDiam2Max**+0.003. Or if we have a huge 40 m tall tree with the total volume of tens of cubic meters and with relative low resolution (e.g. about 1 point per 3cm x 3cm x 3cm), not too much occlusion and lot of small branches, then **PatchDiam1** could have values [0.2 0.3], **PatchDiam2Min** could have values [0.05 0.065 0.08] or [0.04 0.055 0.06 0.075], and **PatchDiam2Max** could have values [0.25 0.3] or [0.2 0.25 0.3] In this case **BallRad** parameters could have values **BallRad1** = **PatchDiam1**+0.05 and **BallRad2** = **PatchDiam2Max**+0.03.

1.10.6 Plotting

There are many functions in the plotting-subfolder that the user can use in MATLAB for plotting. Plot single point clouds using the function `plot_point_cloud` and comparing two point clouds with `plot_comparison` (e.g. comparing filtered and unfiltered point cloud). Plot branch-segmented point clouds with `plot_branch_segmentation` with colors denoting the branching order or each branch uniquely coloured (requires the point cloud, its cover and segment structures). Plot the cylinder model with `plot_cylinder_model` with colors denoting the branching order or each branch uniquely coloured. Plot the triangulation model with `plot_triangulation`. Plot distributions with `plot_distribution` and the spreads as a polar plot with `plot_spreads`.

1.11 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 specifies 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 the latest version `simplify_qsm` function modifies also the **branch** and **treedata** to make them correspond to the changes in cylinders.

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 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 branching points of the branching structure. Function `segments` reconstruct the initial segmentation of the tree patches into stem and branches. However, `segment` finds the branching points 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 branching 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 16). 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 $2 \times \text{PatchDiam}$. 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`.

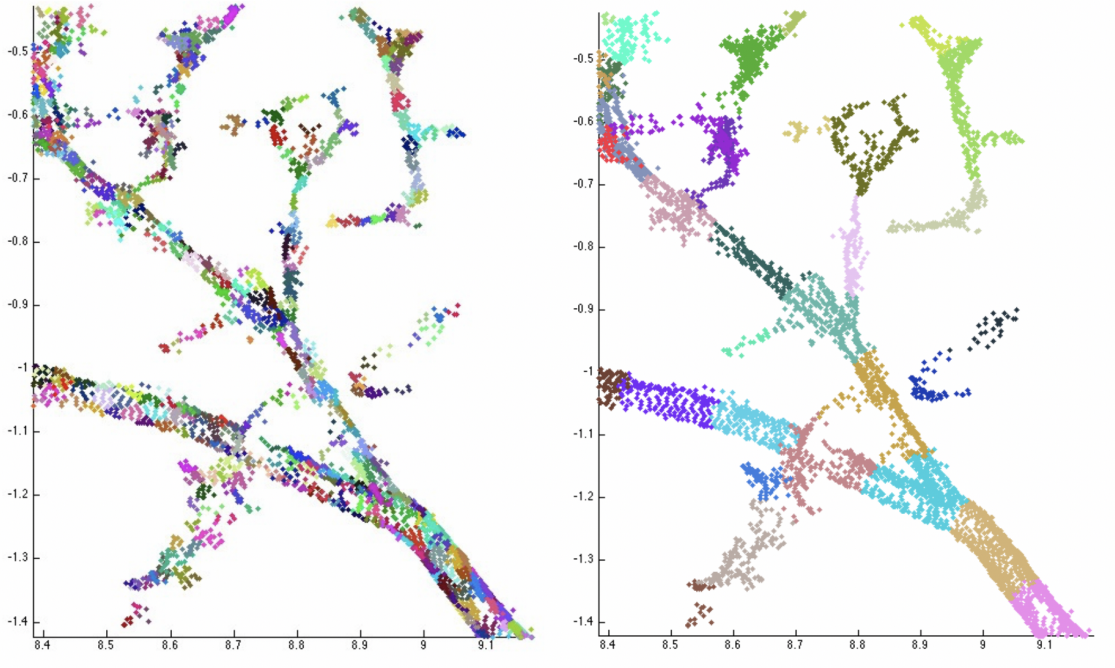


Figure 16: Comparison of the covers of a branch. The minimum diameters (`PatchDiam`) of the cover sets are 2 cm (left) and 10 cm (right). The smaller cover sets can capture much more detail but form more disconnected structure.

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 16. 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 modelling time (half the size means about four times the cover sets and up to 4-fold modelling 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 often very important because it has not so much 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 17 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.

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 are 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,

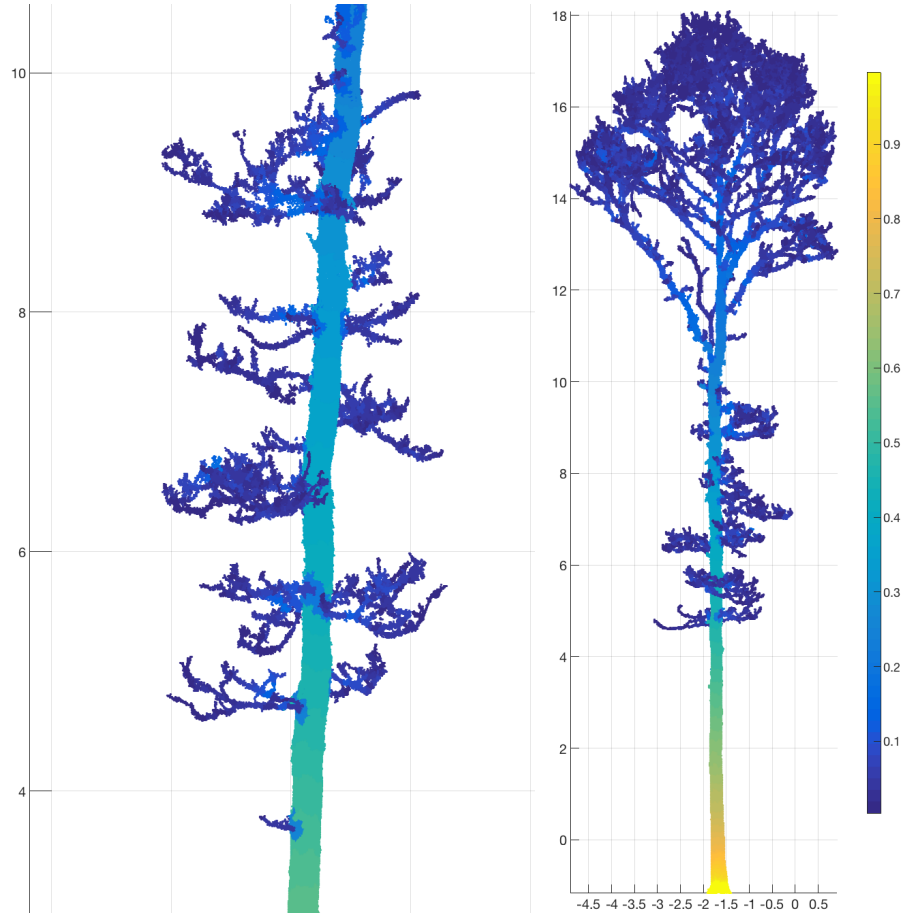


Figure 17: Relative size of the cover sets for the second cover. Value 1 corresponds to the maximum size and value 0 to the minimum size.

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 18 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).

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 (0th-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 18). For the stem and 1st-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).

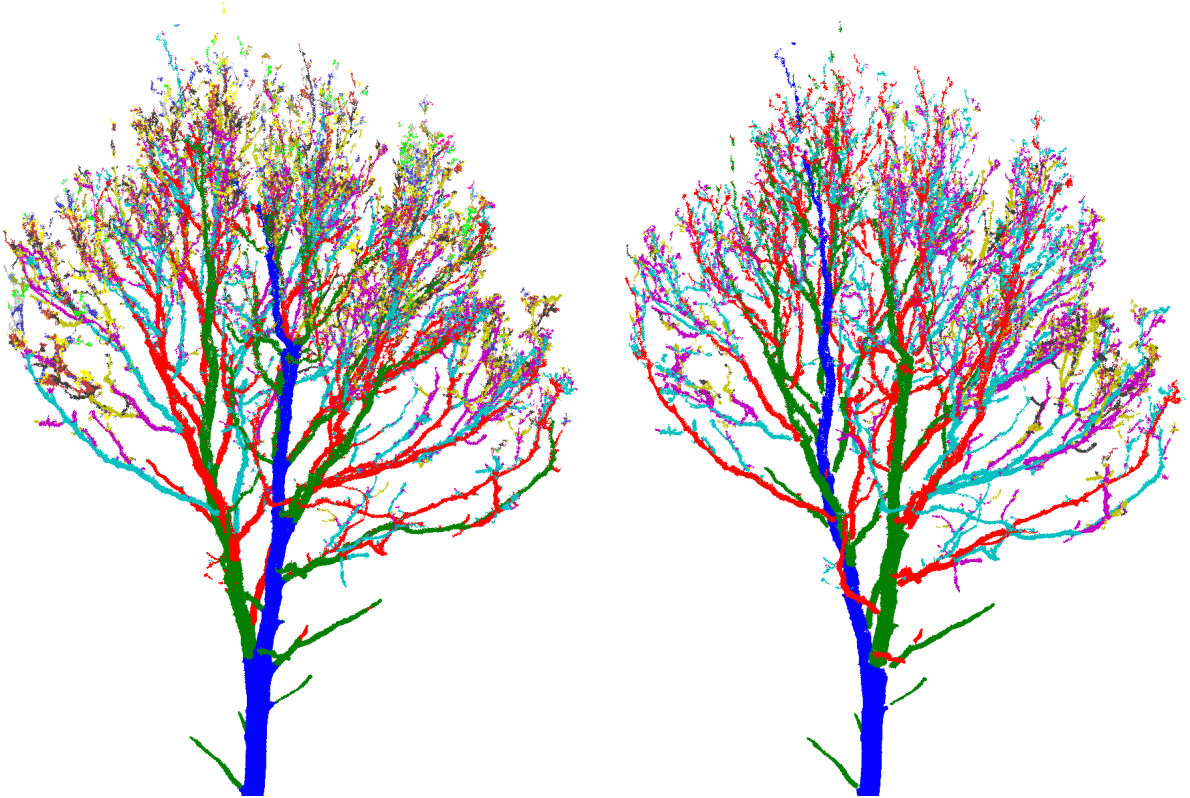


Figure 18: Initial segmentation (left) and corrected final segmentation (right). Notice how trunk (blue) and other segments are corrected and how in general branching order of the segments is reduced.

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 some in 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 Surface coverage and surface coverage filtering

A section of a segment, where a cylinder is fitted, is a point cloud whose points sample a surface that is approximately a cylinder (otherwise cylinder would not be a good "building block"). To measure how much the points cover the surface (area) of the "underlying" cylinder, we compute a value called *surface coverage* (SC). It is defined as follows: Partition the points into equal-high vertical layers in the direction of the cylinder axis, then partition the points into equal-angle sectors based on their azimuth angle, and finally take the intersections of these partitions to produce a cell-partition of the cylindrical surface. The relative number of non-empty cells is then the surface coverage. Figure 19 gives an example.

Not only this cell-partition give an estimate how much of the cylinder's surface is covered but it also gives a way to efficiently filter outliers: If there are noise or part of the child branch, as seen in Fig. 19, the cells in these locations are large in radial direction and only the points closest to the axis are needed/should be kept. Thus, we can accurately remove local outliers and we call this approach surface coverage filtering. Figure 20 gives the examples of the filtering corresponding to the Fig. 19. Moreover, as we can estimate each cell's distance to the cylinder axis, we can estimate the radius of the cylinder as mean or median of the non-zero cell distances.

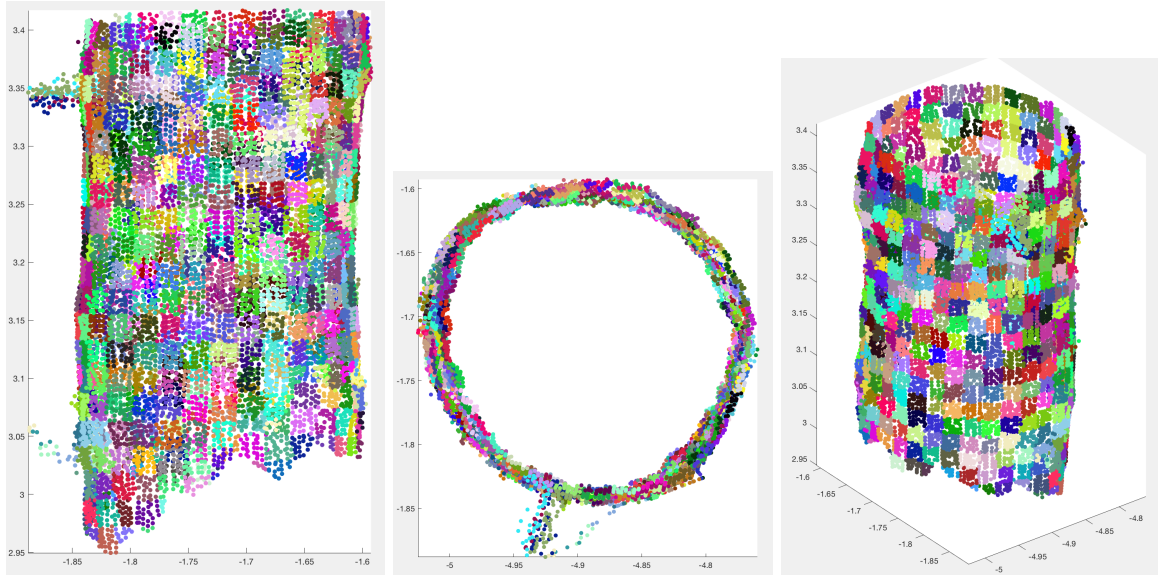


Figure 19: Surface coverage. Partition of a section of a segment into cells (intersections of layers and sectors). Different colors denote the different cells. The three figures give different point of views of the same section.

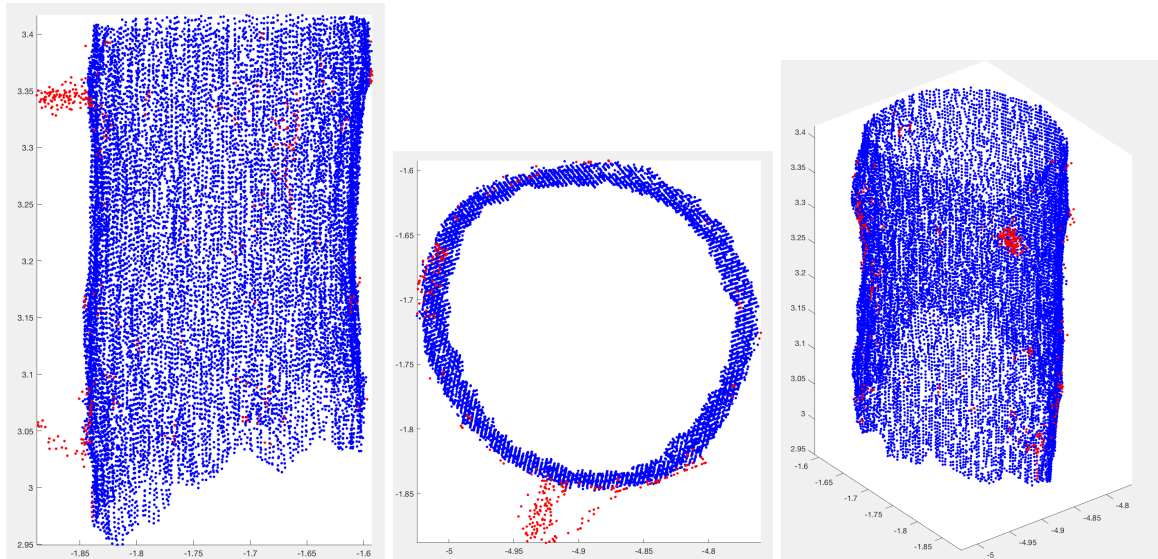


Figure 20: Surface coverage filtering corresponding to Fig. 19. Blue points denote the points passing the filtering and the red points are the filtered points.

2.3.2 Cylinder fitting

After segmentation, we fit cylinders to the segments using the least squares fitting with the function `cylinders`. The process of fitting cylinder and the optional modifying the resulting radii based on tapering are done for each segment at a time, starting from the stem and then continuing with the 1st-order branches. The process of fitting cylinders to a segment is adaptive in the sense that the length of the cylinder or the region/section of the segment used for the cylinder fitting is adapted/optimized for each cylinder. The process starts from the base of the segment and it proceeds along the segment as long as the whole segment is covered with cylinder. For each cylinder to be added to the model at least three cylinder candidates of increasing length are fitted, and the process is continued up to ten times of increasing length of cylinders to find the best cylinder to continue the segment.

For each cylinder the first step is to define the region/section of the segment where the cylinder is fitted by selecting certain number of consecutive layers of cover sets along the segment. From the section the initial values of the cylinder's radius, starting point and axis are estimated because these estimates are needed for non-linear least squares fitting: The axis is simply the line segment connecting the averages of points in the bottom and top layers of the region and the starting point is in this line. From these we can use surface coverage filtering to remove outliers from the region and estimate the radius. After fitting the first three cylinders of different length, if there is at least one cylinder whose optimization process converged reliably and whose relative length (length/radius) is over 1.5 and whose SC is over 40% (over 70% for stem cylinders), then the one of these with the highest SC is selected. If in the first three cylinder candidates there are no acceptable cylinders, then the process continues, up to ten times, as long as an acceptable candidate is found. If there are ten candidates and no one is not acceptable (reliable convergence, relative length > 1.5 , SC $> 40\%$), then the one with the highest SC is selected.

Because the cylinders are separate sections of the segment (as they should form continuous and accurate approximation of the segment's surface), they are also independent of each other. This means in particular that a section may have e.g. local occlusion in the point cloud data and thus the radius and direction of the fitted cylinder may be quite different from the previous and the subsequent cylinders even if they should not be. To make the cylinders more dependent on their neighbours we actually add points from the previous and the subsequent sections to the fitting process but with less weight than the points in the section have. The cylinder's length and starting points (and SC estimate) are computed from the points of the section.

Notice that the above process can produce in some cases, especially if the patch sizes are large compared to the segments diameter, very elongated cylinders that have large relative length (length/radius).

2.3.3 Radius correction

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, whose surface coverage are less than 70%, cannot exceed the radius of the parent cylinder in the parent branch. Or even in the case of higher than 70% of surface coverage the radius can be at most 20% larger than in the parent. We also determine the minimum radius R_{min} of the cylinders: First R_{min} is estimated from cylinders with SC over 70% but if no such cylinder exist in the segment, then the R_{min} is estimated from cylinders with SC over 40% but if even those not exists, then R_{min} is given by the input `MinCylRad`. However, if R_{min} from cylinders with SC over 70% is more than three times the minimum from all cylinders, then R_{min} is the minimum from cylinders with SC over 40%. All radii below R_{min} are set to R_{min} .

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 always use the following simple approach for smoothing radii based on comparing surface coverage of consecutive cylinders: if SC of (i-1)th and (i+1)th cylinders is over 70% but SC of ith cylinder is under 70%, then the radius of the ith cylinder is set to the mean of the radii of (i-1)th and (i+1)th cylinders.

Third, the user can also use the following approach to modify the radii to be generally tapering, enabled with the input **TaperCor**: The idea is to fit a parabola shape taper curve to the branch length-radii data from the cylinders, which gives the local maximum and minimum radii values. And then these local maximum and minimum curves guide modifications of radii based on SC: If the radius is over the predicted value, the closer to the predicted value the radius is corrected the smaller the SC is. We do this as follows: We fit the parabola $R(len) = a * len^2 + b$ to the (length to the midpoint of the cylinder from the branch base, $1.05 * \text{radius}$ of the cylinder) data using linear least squares such that the data points are weighted with the surface coverage of the cylinders. This parabola gives the local maximum radius and if a cylinder has radius R larger than $R_{parabola}$, the predicted by the parabola, and its SC is below 70%, then the radius is set to the closer the predicted maximum value $R_{parabola}$ the smaller the SC value is or precisely to $R_{parabola} + SC/0.7 * (R - R_{parabola})$. If the cylinder has SC over 70% and the radius is 33% larger than the predicted parabola value, then the radius is also set similarly but now the correction is $R_{parabola} + SC * (R - R_{parabola})$. Similarly, we scale down the parabola of the local maximum by 25% to define the local minimum radius. And similarly, if the radius is below the predicted value when SC is below 70%, the radius is corrected to the predicted parabola value. And if the radius is less than 50% of the predicted parabola value, the radius is set to the predicted value. See Figures 21 and 22 for examples. For segments with small number of cylinders, the radii are modified to be linearly decreasing. For those cylinders with large corrections/modifications on their radius the starting point and SC are estimated again to correspond better to the new radius. The starting point is estimated by first projecting the points into plane orthogonal to the axis and then fitting a circle whose radius is given to the projected points.



Figure 21: Cylinder radius modification with parabolas. Red circles and lines denote the original fitted radius, green lines denote the maximum and minimum radius parabolas, blue circles and lines denote the final modified radii, and the light blue lines denote the surface coverage (right vertical axis).

There is another 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

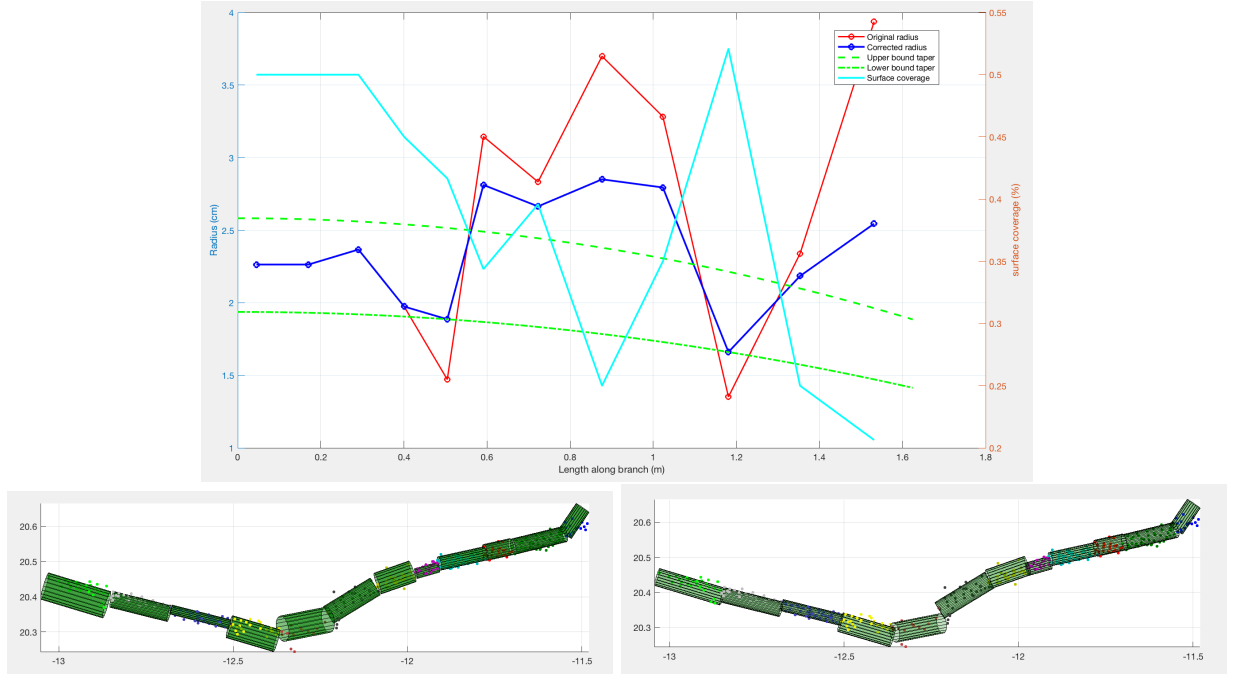


Figure 22: Cylinder radius modification with parabolas. Top: Similar figure as in Figure 21. Bottom: Actual cylinders (left: before any modification; right after the modifications) of the case shown in top figure.

volume of all the cylinders following the given cylinder ("the growth volume") is computed for every cylinder and this produces (growth volume, radius) data set. Then we fit the following function $Radius = a * GrowthVol^b + c$ 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: $1/fac * predicted\ radius \leq radius \leq fac * predicted\ radius$.

2.3.4 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 main 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.

3 Version history

Version 2.3.0 was the initial release in 2017.

Version 2.3.1 was released in 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 in 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.

Version 2.4.0 was released in August 2020. First major update. Cylinder fitting process and the taper correction has changed. The fitting is adaptive and no more `lcyl` and `FilRad` parameters. `Treedata` has many new fields: Branch and cylinder distributions; surface areas; crown dimensions. More robust triangulation of stem. Branch, cylinder and triangulation structures have new fields. More optimisation metrics. More plots of the results and more plotting functions.

- `cylinders`:
 - Many comprehensive and small changes
 - `regions` and `cylinder_fitting` are combined into `cylinder_fitting` and the process is more adaptive as it now fits at least 3 (up to 10) cylinders of different lengths for each

- region.
- `lcyl` and `FilRad` parameters are not used anymore
- Surface coverage (`SurfCov`) and mean absolute distance (`mad`) are added to the cylinder structure as fields.
- Surface coverage filtering is used in the definition of the regions and removing outliers
- `adjustments` has many changes, particularly in the taper corrections where the parabola-taper curve is fitted to all the data with surface coverage as a weight. Adjustment of radii based on the parabola is closer the parabola the smaller the surface coverage. For the stem the taper correction is the same as for the branches. The minimum and maximum radii corrections are also modified.
- Syntax has changed, particularly for the `cyl`-structure
- **tree_data:**
 - Totally newly written code, structure has more sub-functions
 - Changed the setup for **triangulation**:
 - * The size of the triangles is more dependent on the dbh
 - * The height of the stem section is defined up to the first major branch but keeping the stem diameter above 25% of dbh.
 - Makes now more tries for triangulation, also changes triangle size and the length of the stem section if necessary.
 - Changed the names of some fields in the output:
 - * `VolumeCylDiam` → `VolCylDia`
 - * `LengthCylDiam` → `LenCylDia`
 - * `VolumeBranchOrder` → `VolBranchOrd`
 - * `LengthBranchOrder` → `LenBranchOrd`
 - * `NumberBranchOrder` → `NumBranchOrd`
 - Added a lot of new fields into the output **treedata** structure array, including:
 - * `TrunkArea`, `BranchArea`, `TotalLength`
 - * Crown dimensions (spreads, areas, volumes, vertical profile)
 - * Distribution of branches (volumes, areas, lengths, number) in terms of branch order, diameter, height, branching angle, azimuth and zenith angle
 - * Distribution of cylinder (volumes, areas, lengths) in terms of cylinder diameter, height, azimuth and zenith angle
 - * Displays the distributions if `inputs.disp == 2`
 - Added new area-related fields into the output triangulation: side, top and bottom area
 - Added new triangulation related fields to the output **treedata**:
 - * `TriaTrunkArea`: side area of the triangulation
 - * `MixTrunkArea`: trunk area from triangulation and cylinders
 - * `MixTotalArea`: total area where the `MixTrunkArea` used instead of `TrunkArea`
 - Changed the coding for cylinder fitting of DBH to conform new output of the `least_square_cylinder`
- **select_optimum:**
 - Major change in the structure: sub-functions
 - Removed the two cylinder fitting parameters `lcyl` and `FilRad` from the optimisation.
 - Added more choices for the optimisation criteria or cost functions ("metric") that are minimised. There are now 91 metrics and the new ones include some area related metrics, branch and cylinder distribution based metrics and cylinder surface coverage based metrics.
- **tree_sets:**
 - **make_tree_connected:**
 - * Removed "`Trunk(Base) = false;`" at the beginning of the function as unnecessary and to prevent errors in a special case where the Trunk is equal to Base.
 - * Removed from the end the generation of "`Trunk`" again and the new call for the function if "`Trunk`" had nonzero elements
 - * Increased the minimum distance of a component to be removed from 8m to 12m.
 - **define_base_forb:** Changed the base height specification from `0.1*aux.Height` to `0.02*aux.Height`

- **define_base_forb**: changed the cylinder fitting syntax corresponding to the new input and outputs of **least_squares_cylinder**
- **correct_segments**:
 - **modify_topology**: "if size(ChildSegs,1) > size(ChildSegs,2)" → "if isempty(ChildSegs) && size(ChildSegs,1) > size(ChildSegs,2)"
 - **modify_topology**: added "if isempty(SegChildren)" before "SegInd = SegChildren(1);" in order to prevent error in a special case.
- **branches**:
 - Changed the coding to simplify and shorten the code
 - Added branch area and zenith direction as new fields in the branch-structure array
 - Removed the line where **ChildCyls** and **CylsInSegment** fields are removed from the cylinder-structure array
- **treeqsm**:
 - Removed the for-loops for **lcyl** and **FilRad**.
 - Changed what is displayed about the quality of QSMs (point-model-distances and surface coverage) during reconstruction
 - Added version number to **rundata**
 - Added remove of the field **ChildCyls** and **CylsInSegment** of **cylinder** from **branches** to **treeqsm**
- **least_squares_cylinder**:
 - Changed the input parameters of the cylinder to the struct format.
 - Added optional input for weights
 - Added optional input "Q", a subset of "P", the cylinder is intended to be fitted in this subset but it is fitted to "P" to get better estimate of the axis direction and radius
- **make_models_parallel**: Changed "m = m+n;" to "m = m+n(j);" at the end of the function.
- **save_model_text**:
 - Added the new fields of cylinder, branch and treedata structures
 - Added header names to the files
 - Changed the names of the files to be saved
 - Changed the name of second input from "string" to "savename"
 - Changed the rounding of some parameters and attributes
- **plot_models_segmentations**: Plots segmented point clouds and cylinder models with two different colourings, one based on branching order and the other on branch

Version 2.4.1 was released in May 2022. Includes an update of the filtering process, code for selecting automatically PatchDiam parameter values for the optimization process, sensitivity estimates of the results, a smoother and more 3D look for the cylinder model plots, small bug fixes and code streamlining:

- **filtering**: Major changes and additions.
 - Added two new filtering options: statistical kth-nearest neighbor distance outlier filtering and cubical downsampling .
 - Changed the old point density filtering, which was based on given threshold, into statistical point density filtering, where the threshold is based on user defined statistical measure
 - All the input parameters are given by "inputs"-structure
 - Streamlined the coding and what is displayed
- **create_input**: Added filtering parameters, so that they are included in the same inputs-structure.
- **define_input**: Completely new code for defining the PatchDiam and BallRad parameters for QSM reconstruction process.
- **select_optimum**:
 - Added estimation of (relative) sensitivity of the single number attributes in TreeData for the inputs PatchDiam1, PatchDiam2Min, PatchDiam2Max. Now TreeData contains also

- these values as the columns 3 to 5.
- Corrected a small bug in the subfunction "collect_data" that caused error for QSMs whose maximum branch order is less than 2.
- Bug fix for 3 lines (caused error for some cases and for other cases the optimal single model was wrongly selected).
- **cylinders:**
 - Added the growth volume correction option back, which was removed from the previous version by a mistake.
 - Added the fields "branch", "BranchOrder", "PositionInBranch" to the output structure "cylinder"
 - Removed the fields "CylsInSegment" and "ChildCyls" from the output structure "cylinder"
- **least_squares_cylinder:**
 - Included the Gauss-Newton iterations into this function (removed the call to nlssolver function)
 - Changed how the update step is solved from the Jacobian
 - Simplified some expressions and added comments
 - "mad" is computed only from the points along the cylinder length in the case of the optional input "Q" is given.
 - Changed the surface coverage estimation by filtering out points whose distance to the axis is less than 80% of the radius
- **tree_data:**
 - Small changes in "crown_measures" when computing crown base to prevent errors in special cases.
 - Small change for how to compute the "first major branch" in "triangulate_stem".
 - Modified code so that "n" cannot be empty in "branch_distribution" and cause warning
 - Decreased the minimum triangle sizes in "triangulate_stem"
- **branches:** Changed the code such that the input "segment" and output "cylinder" are not needed anymore, which simplified the code in many places. Cylinder info is now computed in "cylinders" function.
- **tree_sets:** Added new lines of code at the end of the "define_main_branches" to make sure that the "Trunk" variable defines connected stem
- **correct_segmentation:** Added "if isempty(SegChildren)... " statement to the "modify_topology" subfunction where next branch is selected based on the increasing branching order to prevent a rare bug.
- **cover_sets:** Added comments and changed some variable names and enforced that input parameters are type double.
- **curve_based_triangulation:**
 - Increased the radius of the balls at seed points from TriaWidth to 2*TriaWidth in the input of "boundary_curve"
 - Added triangle orientation check after the side is covered with triangles so that the surface normals are pointing outward
 - Modified the check if the new boundary curve changes only a little and then stop reconstruction
 - Added halving the triangle height if the boundary curve length has increased three times.
 - Changed the bottom level from the smallest z-coordinate to the average of the lowest 100 z-coordinates.
- **estimate_precision:** Added "TreeData", the output of "select_optimum", as an input, and now it is updated.
- **cubical_partition:** Changed the determination of EL and NE so that the while loop don't continue endlessly in some cases.
- **point_model_distance:** Changed the determination of NE, the number of empty edge layers, so that is now limited in size, before it is given as input for "cubical_partition" function.
- **surface_coverage_filtering:** Small changes to make the code little faster

- **plot_cylinder_model:**
 - Changed the surface plot ("patch") so that the edges are not plotted with separate colour, so the surface looks more smooth. Added also shading.
 - Added cylinder branch "Bran" and branch order "BOrd" vectors where the colouring options are defined to prevent some errors.
- **simplify_qsm:**
 - Added modification of branch and treedata structures based on the modified cylinders
 - Added input for plotting and displaying the results
 - Corrected some bugs that could cause errors in some special cases
- **growth_volume_correction:** Changed the roles of RADIUS and GROWTH VOLUME in the allometry, i.e. the radius is now predicted from the growth volume. Also does not increase the radius of the branch tip cylinders.