**Quantitative Structure Models of Trees from Laser Scanner Data**

Pasi Raumonen

Department of Mathematics, Tampere University of Technology

Email: [pasi.raumonen@tut.fi](mailto:pasi.raumonen@tut.fi)

Web: <http://math.tut.fi/inversegroup/>

Published paper: <http://www.mdpi.com/2072-4292/5/2/491>

**Instructions for MATLAB-software *Tree\_model\_2.0***

**NOTICE! This software is only for non-commercial use and cannot be distributed without the permission of P. Raumonen.**

**How to begin:**

1. Start MATLAB and set the main path to the root folder, where *qsm\_tree.m* is located.
2. Use *Set Path* 🡪 *Add with Subfolders 🡪 Open 🡪 Save 🡪 Close* to add the subfolders, where all the codes of the software are, to the paths of MATLAB.
3. Import a point cloud from a tree into the workspace. Let us name it *P0*.

**How to make quantitative structure models (QSM) of the point cloud:**

We have been using “quantitative structure models” and the abbreviation QSM in our latest publications to refer the reconstructed tree models. I would recommend this term also for you.

I first explain the basic command that produces the QSM and then give more details how the algorithm works. So the basic command is:

qsm\_tree(P0,dmin0,rcov0,nmin0,dmin,rcov,nmin,lcyl,NoGround,string, rfil1,nfil1,rfil2,nfil2);

P0 Point cloud, (npoints x 3)-matrix of (x,y,z)-coordinates, contains only one tree

dmin0 Minimum diameter of cover set in the first cover, given in the same units as the coordinates in P0

rcov0 Radius of balls used to generate the cover sets in the first cover, affects also which cover sets are neighbors

nmin0 Minimum number of points in a ball to be accepted in the first cover

dmin Minimum diameter of cover set in the second cover, given in the same units as the coordinates in P0

rcov Radius of balls used to generate the cover sets in the second cover, affects also which cover sets are neighbors

nmin Minimum number of points in a ball to be accepted in the second cover

lcyl Relative length (length/radius) of the cylinders

NoGround “True” or “1” if no ground points in the point cloud P0, “False” or “0” if ground points in P0

string String that is attached to the saved model file names

rfil1 Radius of balls used in the filtering of low density points

nfil1 Minimum number of points for rfil1-balls to pass the filtering

rfil2 Radius of balls used in the filtering of small separated clusters

nfil2 Minimum number balls in the clusters to pass the filtering

Often it is better first to make the filtering separately to see, which filtering result is acceptable and then use the filtered point cloud for QSM reconstruction. So first run the filtering code

I = filtering(P0,rfil1,nfil1,rfil2,nfil2);

P = P0(I,:);

Here the output of the filtering is logical vector of point passing the filtering. And then make the QSM:

qsm\_tree(P,dmin0,rcov0,nmin0,dmin,rcov,nmin,lcyl,NoGround,string);

Here is an example. The filtering produces the following result:

I = filtering(double(P0),0.02,3,0.03,20);

All points: 1783261, First filtering: 67358, Points left: 1715903

110400 cover sets, points not covered: 0

All points: 1715903, Second filtering: 116412, Points left: 1599491

All points: 1783261, All filtered points: 183770, Points left: 1599491

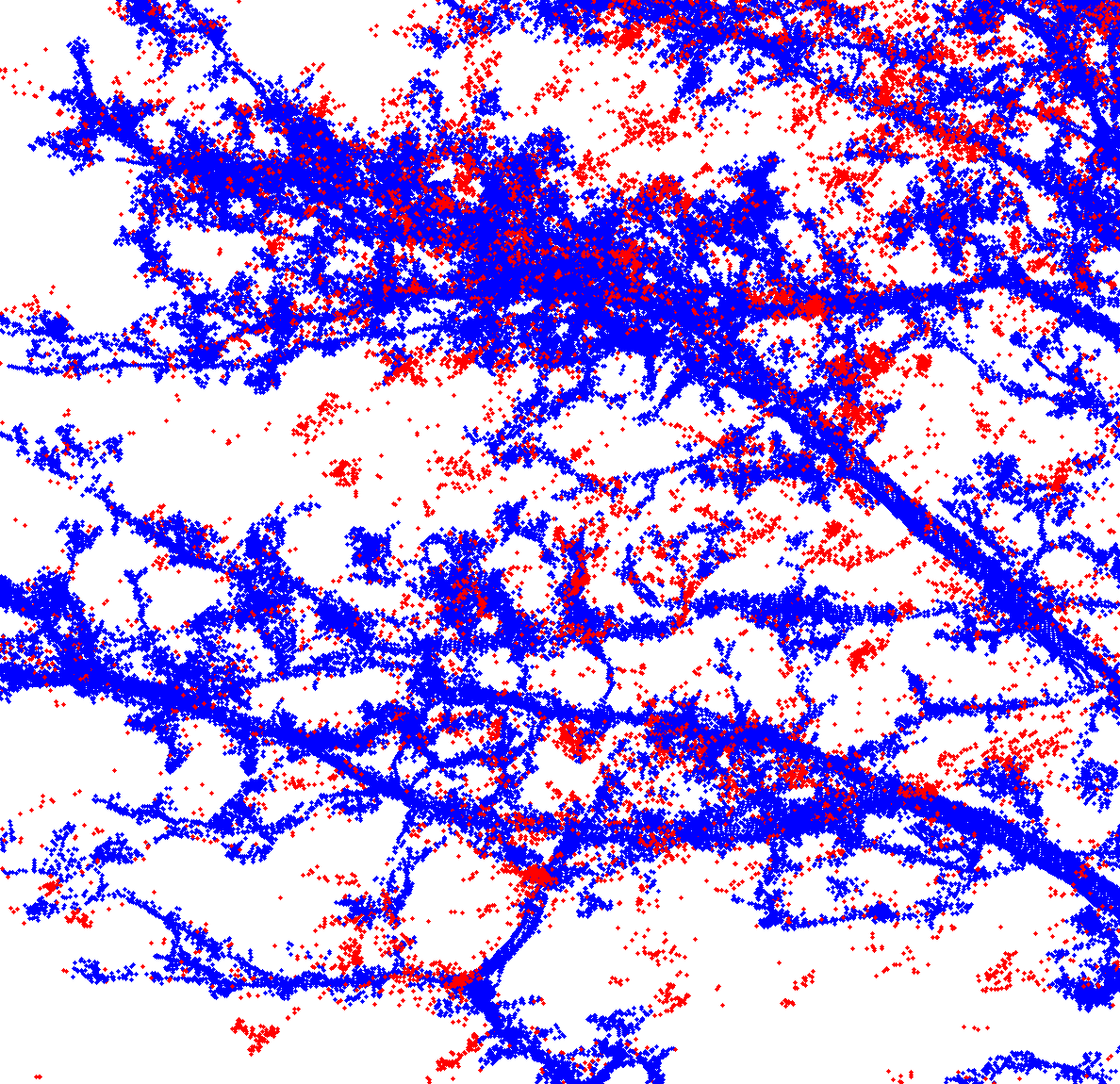
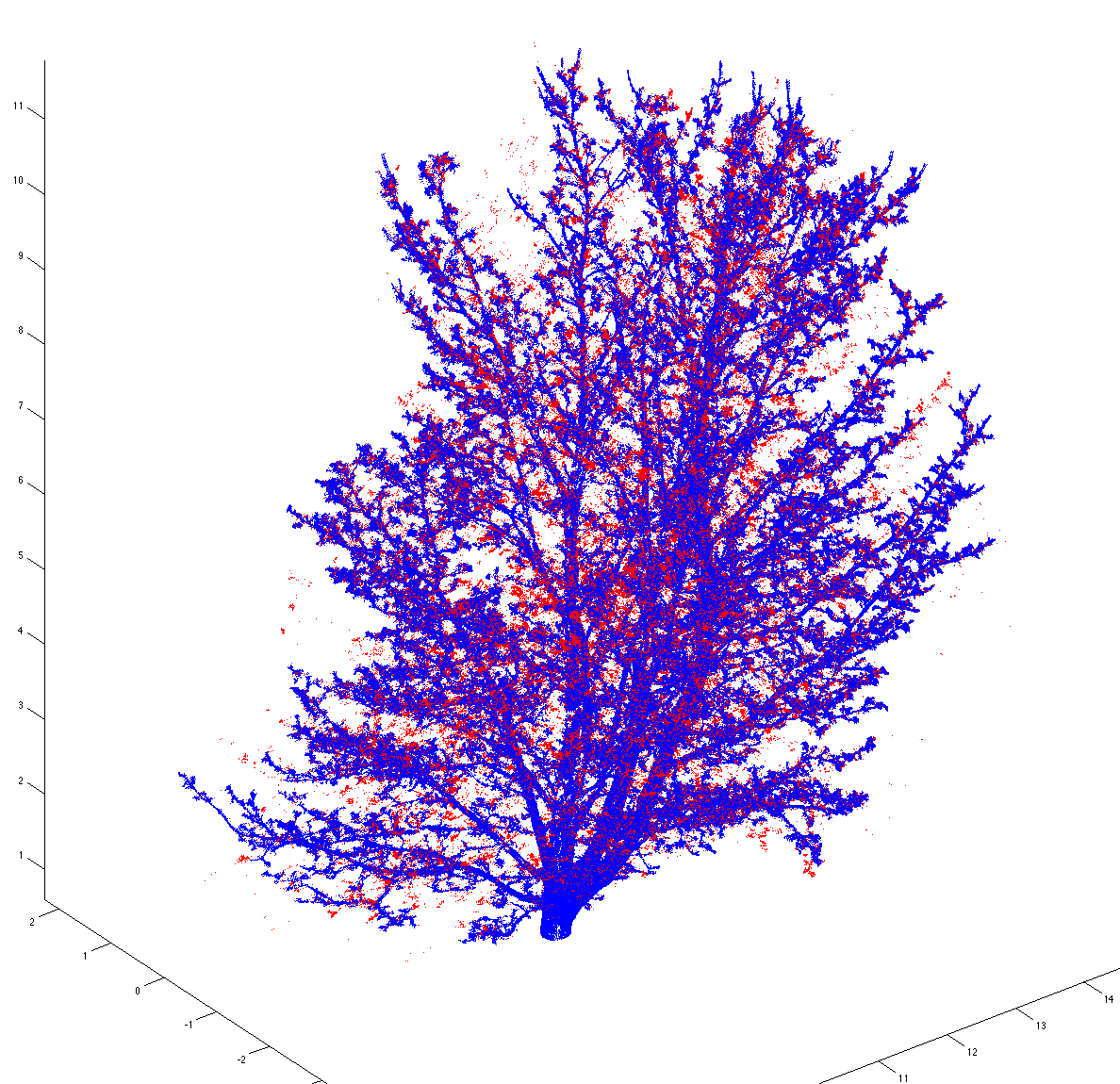


Figure 1. Left: Original point cloud, where red points are filtered out points and blue points are the points that passed the filtering, i.e. the point cloud *P*. Right: A close-up of the left hand side figure.

Notice that there are more filtering options to use and also that the filtering uses covers which are randomly generated, meaning that the results are little different for every run even with the same input parameters.

Then we make the QSM:

>> qsm\_tree(P,0.08,0.09,5,0.03,0.035,3,5,true,'test');

---------------

test

dmin0 = 0.08, rcov0 = 0.09, nmin0 = 5

dmin = 0.03, rcov = 0.035, nmin = 3, lcyl = 5

Progress:

GENERATING COVER...

17599 cover sets, points not covered: 0

Cover sets 0 min 12 sec, total: 0 min 12 sec

DETERMINING TREE SETS...

Tree sets 0 min 1.1 sec, total: 0 min 13.1 sec

SEGMENTING...

1959 segments found

Maximum branch order before correction: 7

Maximum branch order after correction: 7

791 segments after correction

Segments 0 min 6.8 sec, total: 0 min 19.9 sec

GENERATING NEW COVER...

239983 cover sets, points not covered: 1052

Cover sets 1 min 48 sec, total: 2 min 7.9 sec

DETERMINING TREE SETS...

Tree sets 0 min 32.7 sec, total: 2 min 40.7 sec

SEGMENTING...

8535 segments found

Maximum branch order before correction: 17

Maximum branch order after correction: 9

3141 segments after correction

Segments 1 min 43.6 sec, total: 4 min 24.2 sec

CONSTRUCTING CYLINDER MODEL...

9827 cylinders fitted

Median relative cylinder length: 4.89

Cylinders 0 min 57 sec, total: 5 min 21.2 sec

------------

Tree attributes:

Total volume = 2529 liters

Trunk volume = 1032 liters

Branch volume = 1497 liters

Total height = 20.5 meters

Trunk length = 20.9 meters

Branch length = 960 meters

Number of branches = 1935

Maximum branch order = 9

Total cylinder area = 125 square meters

Dbh (cylinder) = 35.5 centimeters

Dbh (triangulation) = 35.5 centimeters

Trunk volume (cylinders) = 764 liters

Trunk volume (triangulation) = 761 liters

Trunk length (cylinders) = 8.55 meters

Trunk length (triangulation) = 8.55 meters

-----

Branch order data:

Number of 1st-order branches = 25

Number of 2nd-order branches = 180

Number of 3rd-order branches = 505

Number of 4th-order branches = 618

Number of 5th-order branches = 395

Number of 6th-order branches = 162

Volume of 1st-order branches = 484 liters

Volume of 2nd-order branches = 494 liters

Volume of 3rd-order branches = 309 liters

Volume of 4th-order branches = 149 liters

Volume of 5th-order branches = 44.6 liters

Volume of 6th-order branches = 14.6 liters

Length of 1st-order branches = 88.3 meters

Length of 2nd-order branches = 209 meters

Length of 3rd-order branches = 284 meters

Length of 4th-order branches = 231 meters

Length of 5th-order branches = 100 meters

Length of 6th-order branches = 39.2 meters

------------

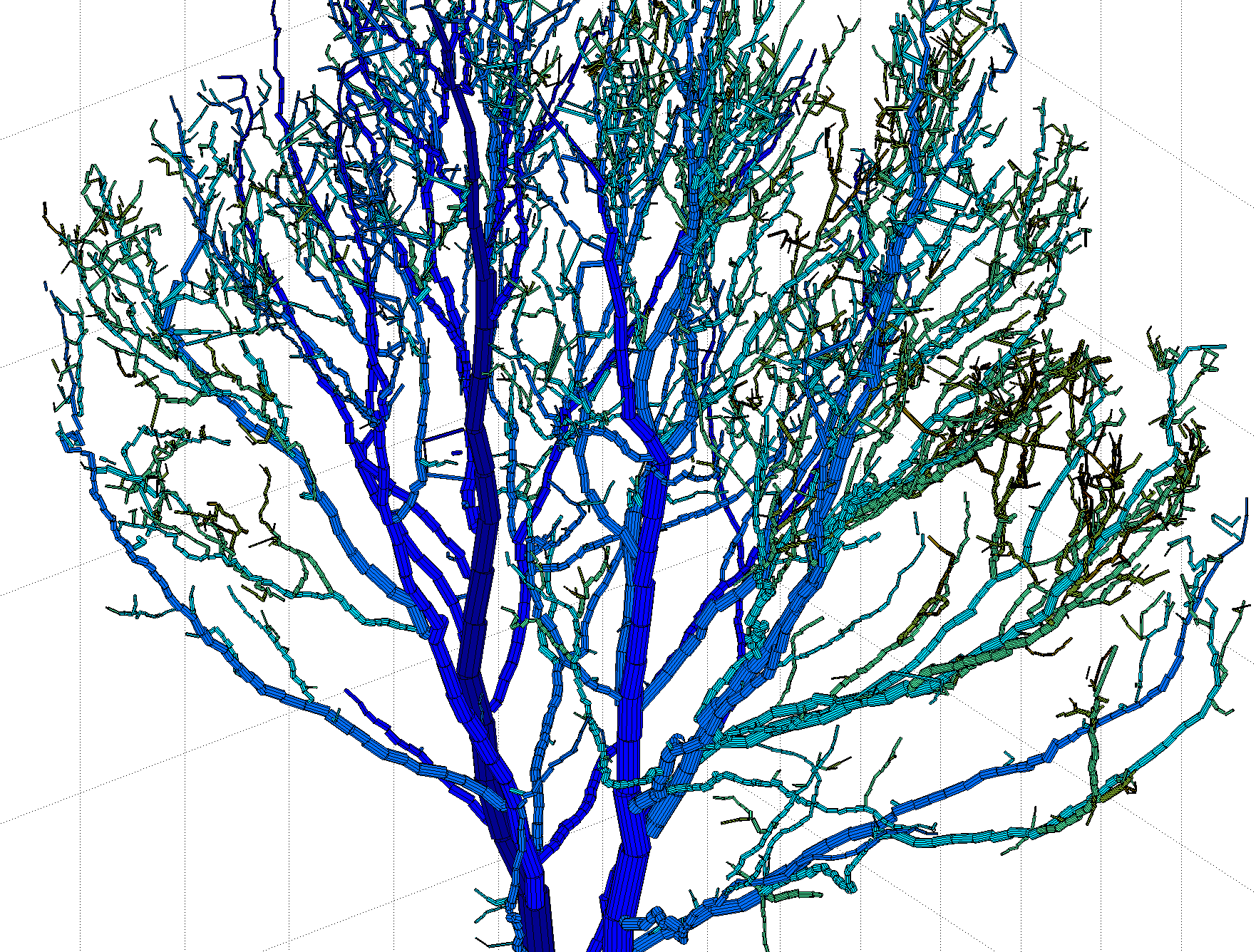
This produces a cylinder model of the tree and saves the model into “cyl\_data\_test.txt” and “branch\_data\_test.txt”-files. Also some the tree attributes displayed above are saved in “tree\_data\_test.txt”-file. The files are saved /results folder.

cyl\_data files contain info of each cylinder, whose name/index is the row index/number in the file and info is given by the following columns:

1. radius
2. length
3. x-coordinate of the staring point
4. y-coordinate of the staring point
5. z-coordinate of the staring point
6. x-component of the cylinder axis
7. y-component of the cylinder axis
8. z-component of the cylinder axis
9. index (row number in this file) of the parent cylinder
10. index (row number in this file) of the extension cylinder
11. branch (row number in the branch\_data-file) of the cylinder
12. branch order of the branch the cylinder belongs
13. running number of the cylinder in the branch it belongs
14. Indication if the cylinder is added after normal cylinder fitting (=1 if added)

branch\_data files contain info of each branch, whose name/index is the row index/number in the file and info is given by the following columns:

1. branch order (0 for trunk, 1 for branches originating from the trunk, etc.)
2. index (row in this file) of the parent branch
3. volume of the branch in liters (sum of the volumes of the cylinders forming the branch)
4. length of the branch in meters (sum of the lengths of the cylinders)
5. branching angle in degrees (angle between the branch and its parent at the branching point)



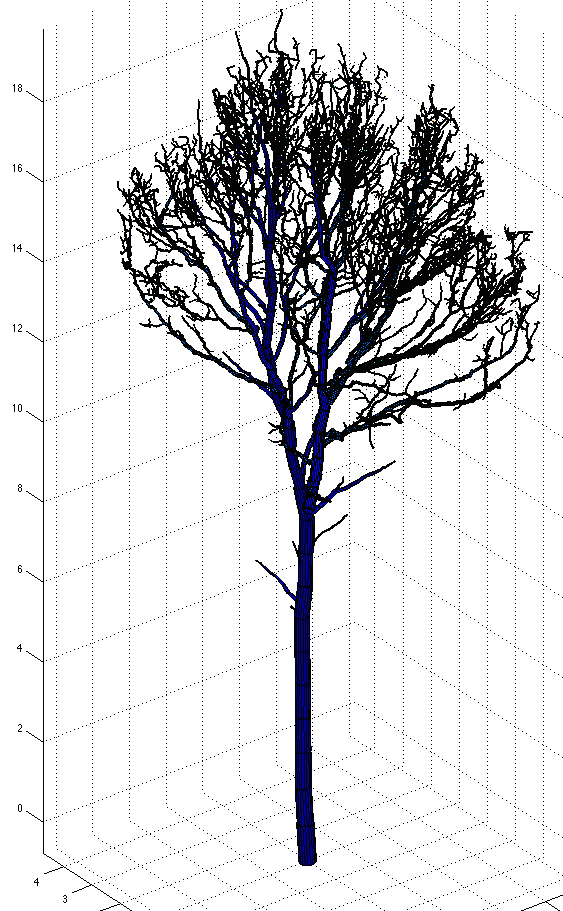
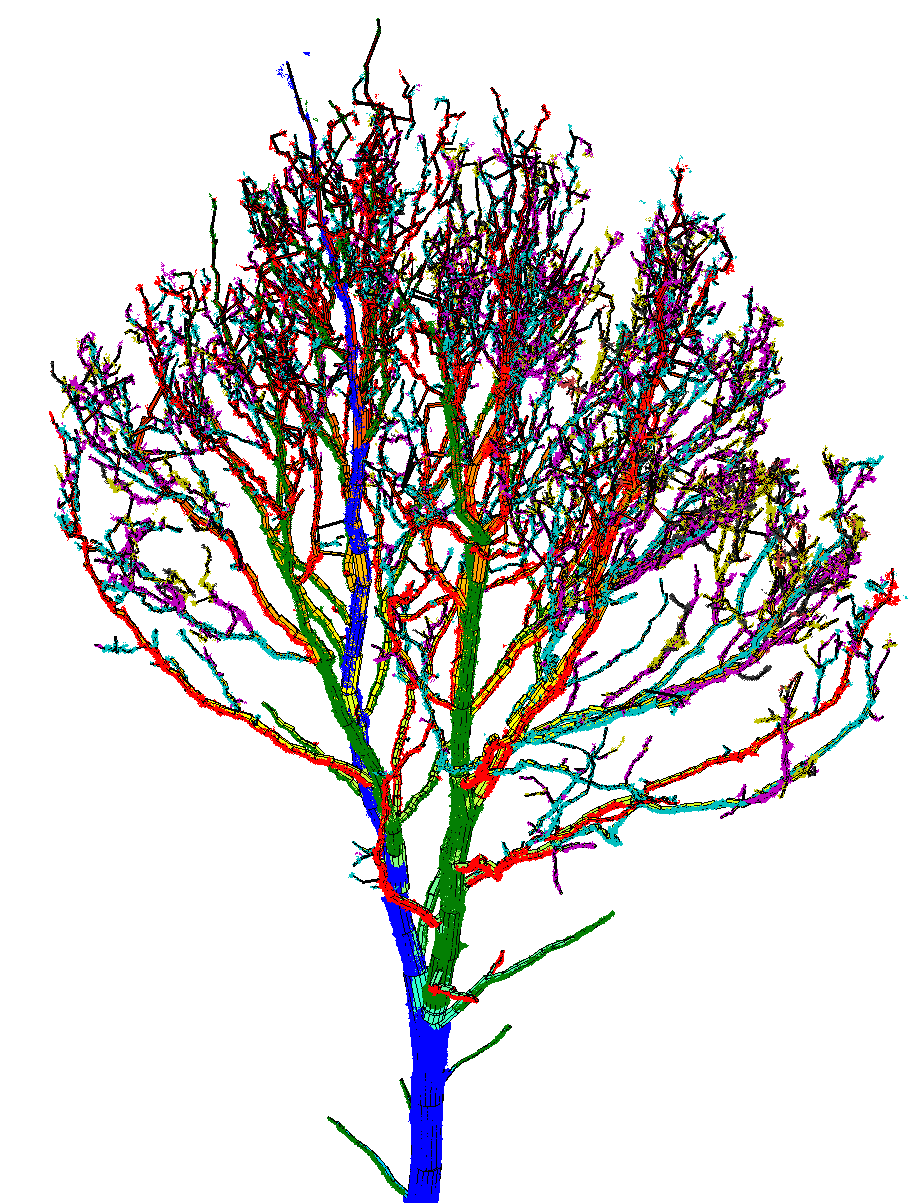


Figure 2. QSM-model. The cylinder color marks the branch order.



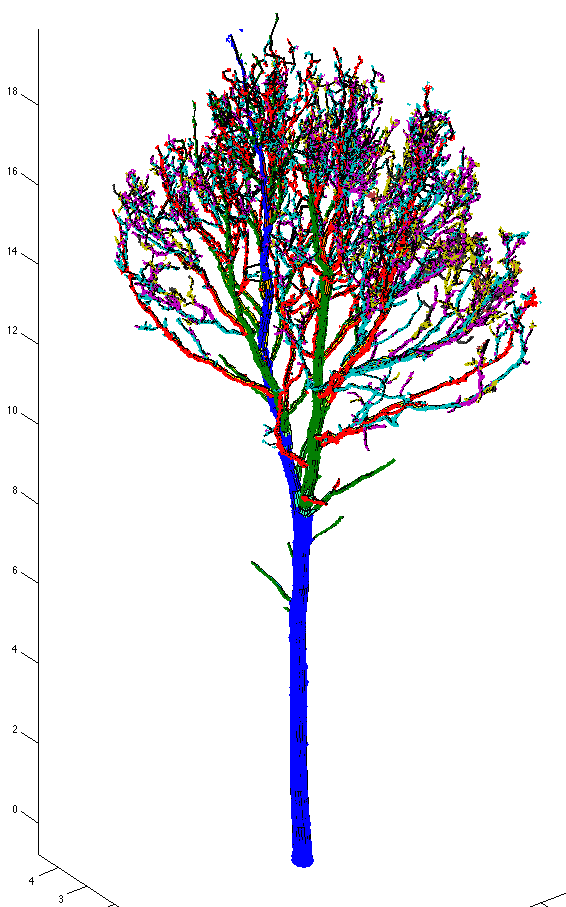


Figure 3. QSM and segmented point cloud. The color indicates the branch order: Blue = trunk, green = 1st-order branches, red = 2nd-order branches, etc.

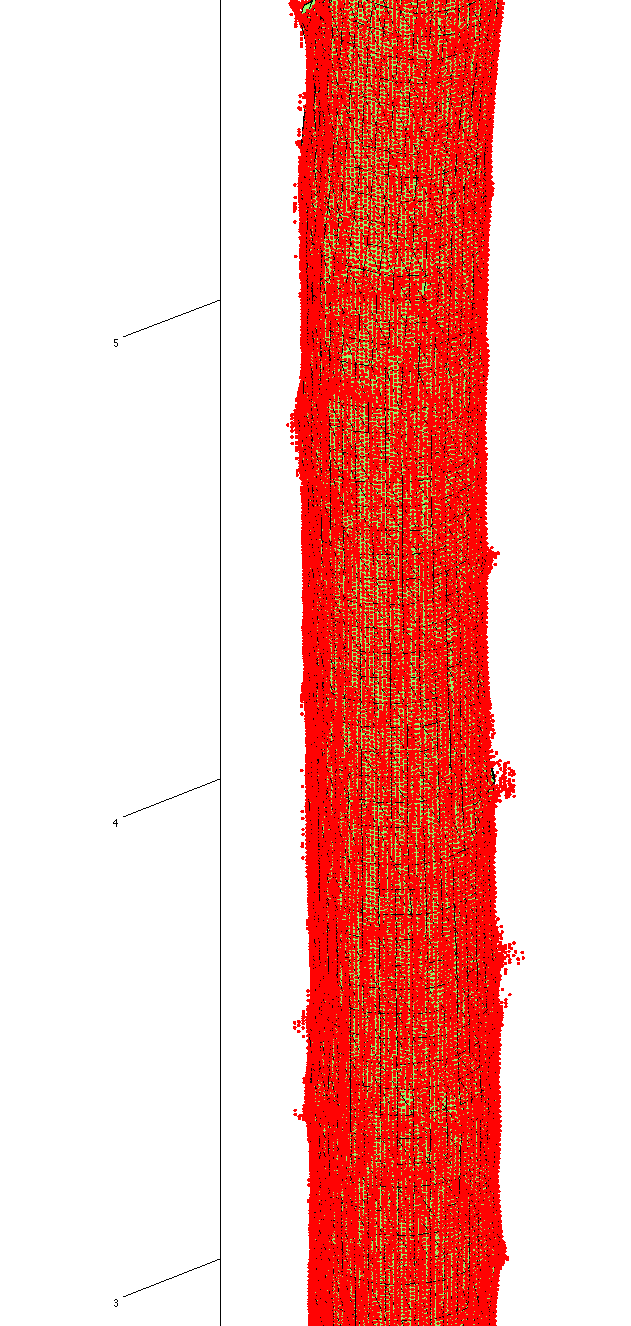
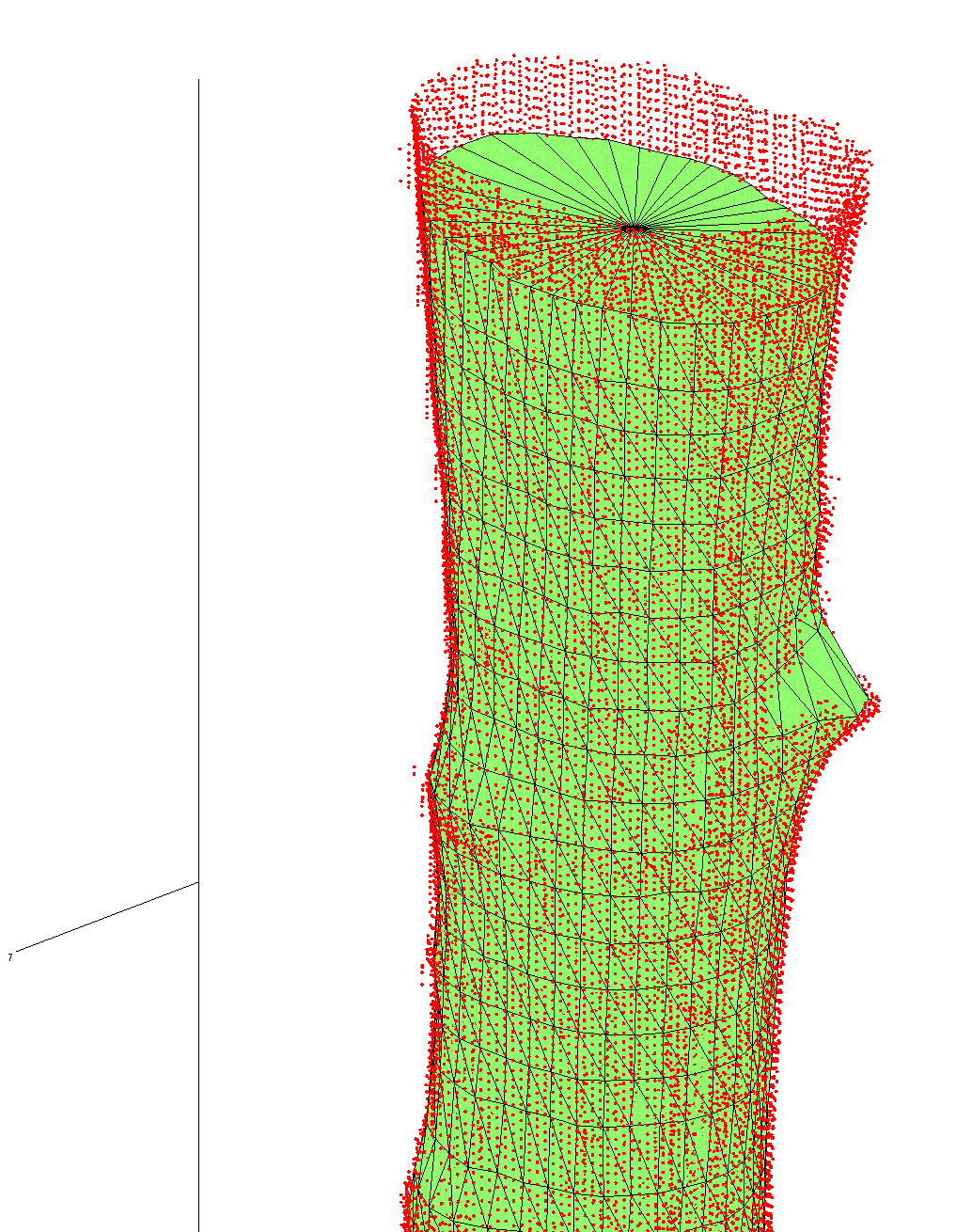


Figure 4. Triangulation model of the bottom part of the trunk.

**How it works:**

The 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 6). These form the smallest “unit” we use to segment the point cloud into trunk and individual branches. They are randomly generated by the input parameter *dmin*, *rcov*, *nmin*: First select a random point Q and define *rcov*-ball, i.e. those points that are closer than *rcov* for Q. If this has at least *nmin* points, then the ball is accepted and Q is the center of the ball and a cover set to be formed later. Next define a *dmin*-ball centered also at Q. Here *dmin* is usually little smaller than *rcov*. dmin is the minimum distance between nearby centers of cover sets, so the points in the *dmin*-ball will not be centers of other cover sets. Then select randomly another point R as a center of another *rcov*-ball. This point R cannot now be in the *dmin*-ball centered at Q. Similarly define the *rcov*- and *dmin*-balls for point R. Proceed this way until all points are included in some balls or are too far a way from other points not be accepted in any *rcov*-ball. Finally define the cover sets to consist those points that are closest to the centers, i.e. each point belongs only one cover sets. Because *rcov*-balls can intersect nearby points, each point may belong to multiple *rcov*-balls, but it will be assigned to the cover set whose center is the closest. This way the points are partitioned into “cover sets” or “surface patches”.

Because the *rcov*-balls intersect and each cover set has its own *rcov*-ball associated with it, we use this intersection of balls as the definition if two cover sets are neighbors. This is why *rcov* should be little bigger than *dmin*, so that we can assure that cover sets next to each other are neighbors.

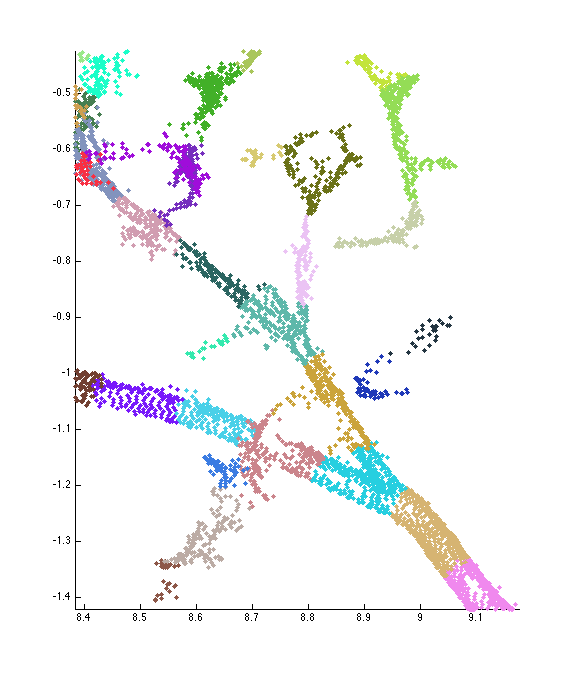
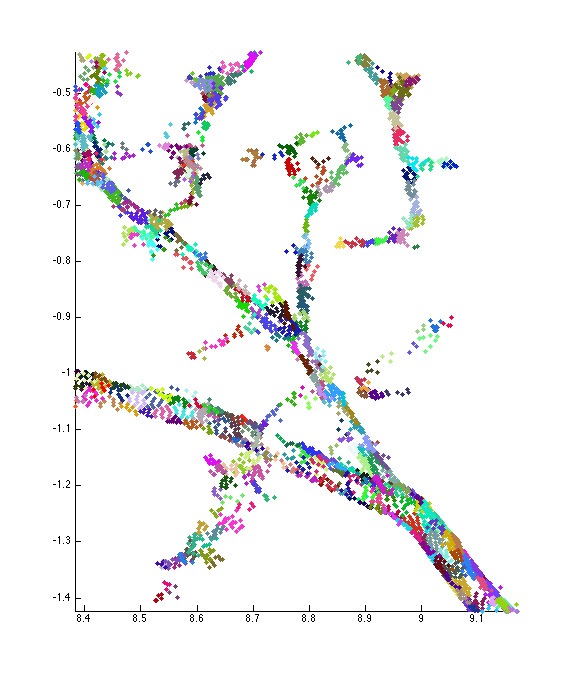


Figure 6. Comparison of the covers of a branch. The minimum diameters (dmin) of the cover sets are 2 cm (left) and 10 cm (right). The smaller cover sets can capture much more detail.

The average (also the minimum and maximum) size of the cover sets is thus controlled by *dmin*-parameter. There is a trade-off with the size of cover sets. The smaller it is, the more details we can capture and smaller branches can be separated. However, the smaller size means more sets, which means almost quadratic increase in modeling time (half the size means about four times the cover sets and up to 4-fold modeling time). Also memory requirement 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 means 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 current version of the method explained here uses now two different covers. The first cover has large size cover sets, e.g. 8 cm as in above example model run. The purpose of this first cover is to i) remove the points that don’t belong to the tree, e.g. ground and understory points, and ii) make initial segmentation that is used as a priori information for the second cover sets generation as the size of the cover sets. 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 with the first segmentation we now know approximately the branching structure. So for the second cover with smaller sets (e.g. 3 cm as in the above example run) we also determine varying size so that the maximum size is at the base of the trunk and decreasing linearly to the top of the trunk so that the size at the top is half in diameter as at the base (e.g. 1,5 cm in the above example run). Similarly the size at the base of the branches decrease as the branch order increase and then the size linearly decrease to the tip of the branch. Here the input *dmin* gives the maximum size at the base of the trunk and then at the tips of the branches and trunk the size is half of this. The relative size of the cover sets is determined in “relative\_size” step of the code, where you can find more information.

So after the first cover generation, the next step is to remove automatically the points/cover sets not included in the tree using heuristics. This step in the code is “tree\_sets”. Thus if the point cloud contains measurements from the ground and the understorey, they are removed. So if these are already removed manually, use NoGround = true as the input parameter, otherwise NoGround = false. At this step we also define the base of the trunk, which is used as the staring point for the segmentation process. Furthermore, some parts of the point cloud are clearly separated from other parts, and this is reflected in the neighbor relation of cover sets: the sets in this separated parts are not neighbors of set in other parts. Thus to make the tree one connected whole (in the sense of the neighbor relation), we determine these separated parts and connect them to each others by updating the neighbor relation (we use the closest distances between the clusters to determine the connections).

Next we segment the cover sets (point cloud) into trunk and individual branches. This process starts from the base of the trunk and in step-by-step proceeds along the trunk (later along branches) and each step finds out if there is bifurcation. If there is branch, its base is saved as new basis for later segmentation. This way the trunk is segmented and its branches are separated from it. Then we continue the same process from the base of the first branch found. This way we first determine the trunk, then the 1st-order branches, then 2nd-order branches, etc.

After this initial segmentation process, we try to correct this many ways. The initial segmentation is based on local examination and thus may result incorrect decisions, such as ending the segment too early or making wrong decisions in the bifurcation points. So here we first remove very small segments whose distance from the parent segment is about the same as approximate radius of the parent segment. These are removed because they are small and unclear in their classification (are they real branches or part of the parent segment?).

Then the major correction operation is to study the segments in the increasing branching order starting from the trunk so that each segment is made to reach as far as possible. In other words, we take a segment and all its current child segments and calculate the distances from the base of the segment to the tip of the segment and to the tips of all its child segments. Notice this distance is not along the branches but is the direct or the smallest distance between points. If the distance to the tip of the segment is less than to some other tip of a child segment, then the segment is corrected so that it goes to this furthest tip (and other segments are also modified suitably) (See figure 7). For the trunk we select the highest tip.

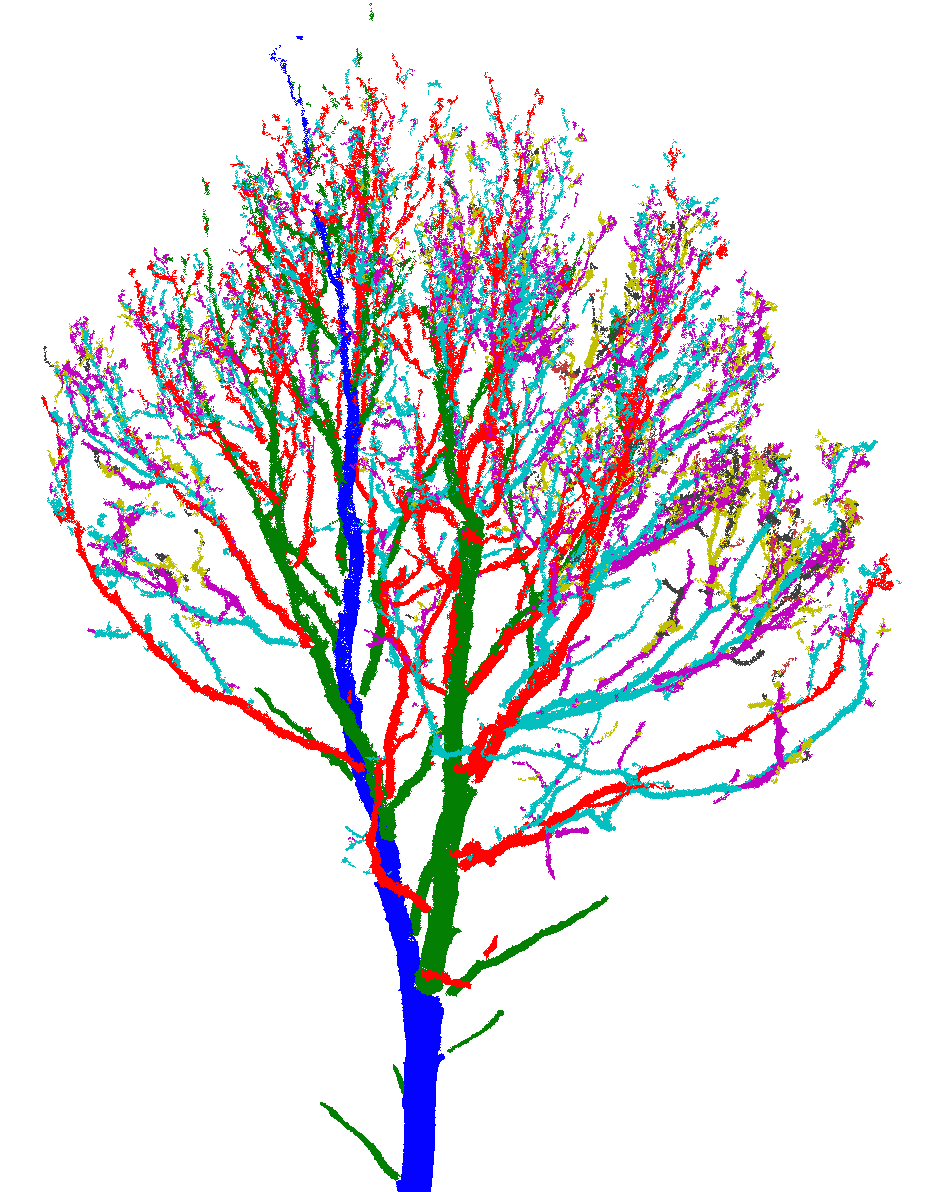
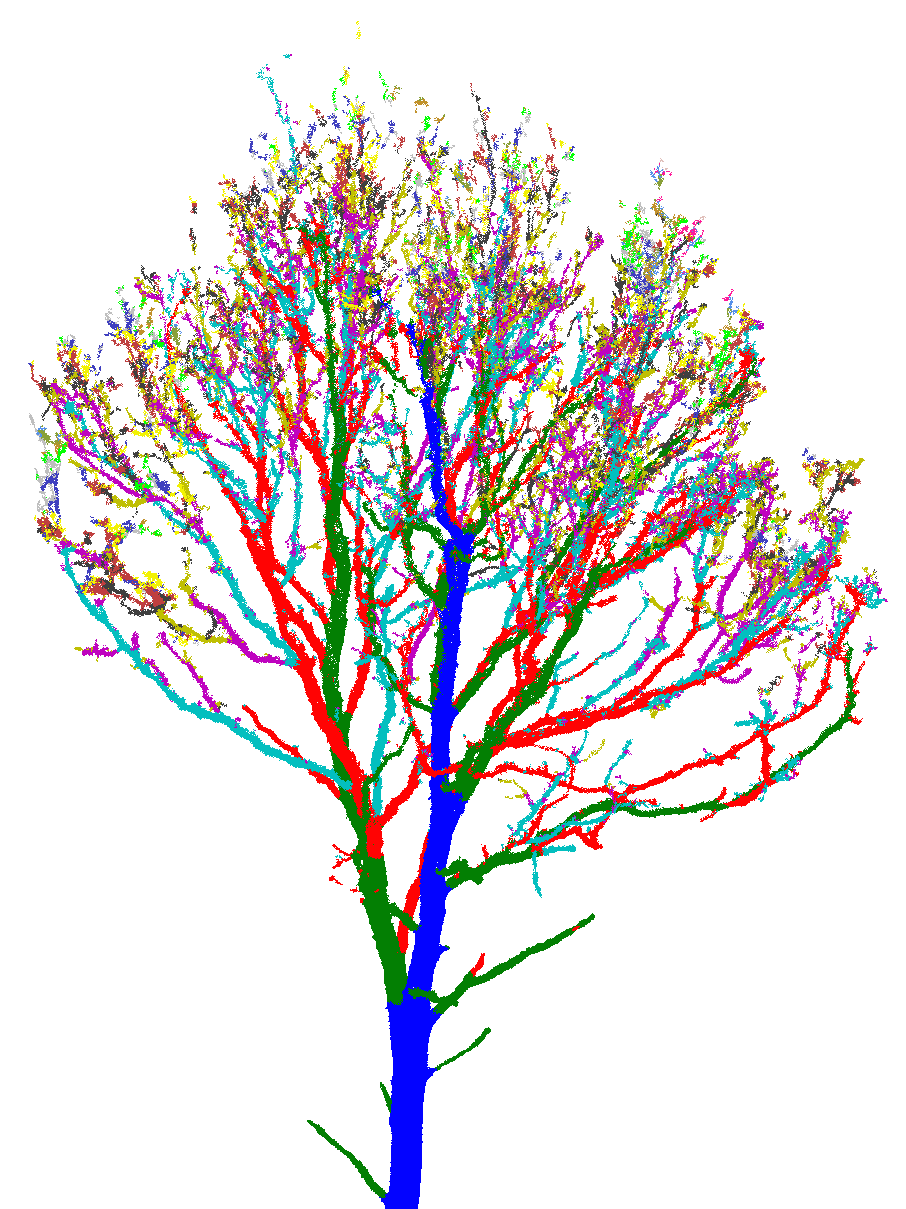


Figure 7. Initial segmentation (left) and corrected final segmentation (right). Notice how trunk (blue) and other segments are corrected and how in general branching order of the segments is reduced.

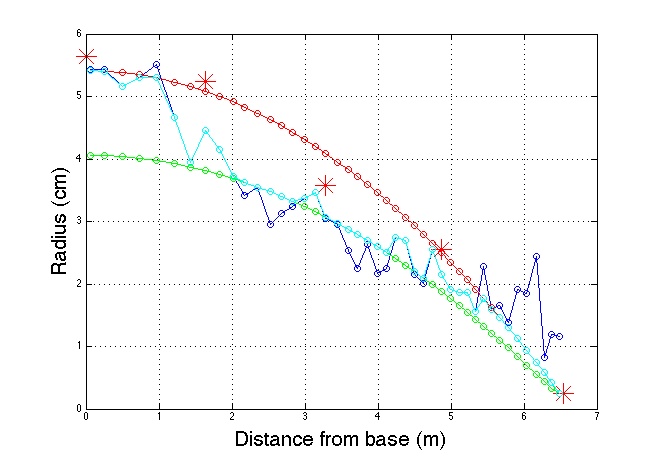
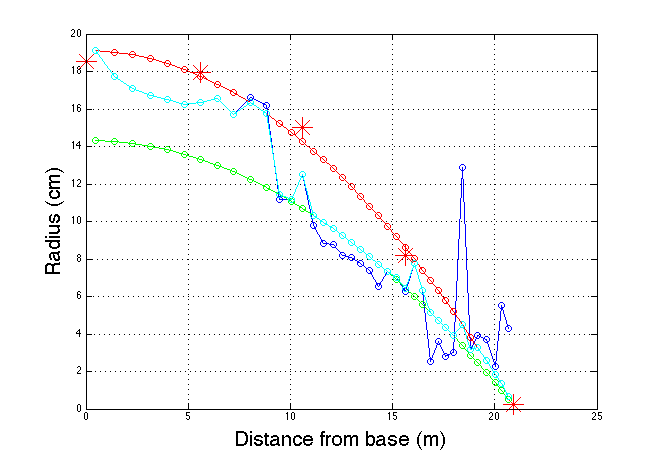
This kind of correction makes the resulting segmentation much more robust, i.e. with different covers the resulting segmentation is usually almost identical. Thus the volume reconstruction will also be more robust in the same sense. Also the maximum branching order is reduced much more realistic levels. For example, in the above example run we see that with the second cover with small sets the initial maximum branching order was 17 and after the correction it was 9. 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 anymore the furthest tip.

Finally, we remove some cover sets, from the parent segment, where the child segments originate. This way we try to ensure that small parts of the child segments are not part of the parent segments and that way make the fitted cylinders too big. After all these corrections also the number of segments is reduced considerably, for example in the above example run, with the second cover, from the initial 8535 to the final 3141 segments.

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

Because the fitted radii of the cylinders forming a branch is, particularly for thinner branches, varying unnaturally, there is also some control on the radii: First of the maximum radius cannot exceed the radius of the parent cylinder in the parent branch. Then we want that the local size and chance of the radii has some bounds and is generally decreasing as we approach the tip of the branch. To achieve this we use the following approach to modify fitted radii values: First all the radii larger than the maximum given by the parent branch are reduced to this maximum. Then we fit a parabola shape taper curve to the branch length-radii data, which gives the local maximum and also the minimum radius values. We do this as follows:

We take average radii of few sections of the branch and scale these upward five percent, and lastly set the radius to some small value at the tip of the branch (it is now set to 2,5mm). The first section is the about the first 10% of the branch length, then the next three are the first three quarters of the branch length. These scaled averages and the tip value are the new data where the parabola is fitted in least square sense. The parabola gives the maximum local radius and by scaling it downwards with 25% gives the local minimum value. If the radius of a cylinder is above or below these parabola values, then the radius is corrected to be the high or low parabola value. See figure 8 for examples. For segments with small number of cylinders, the radii are modified to be linearly decreasing.



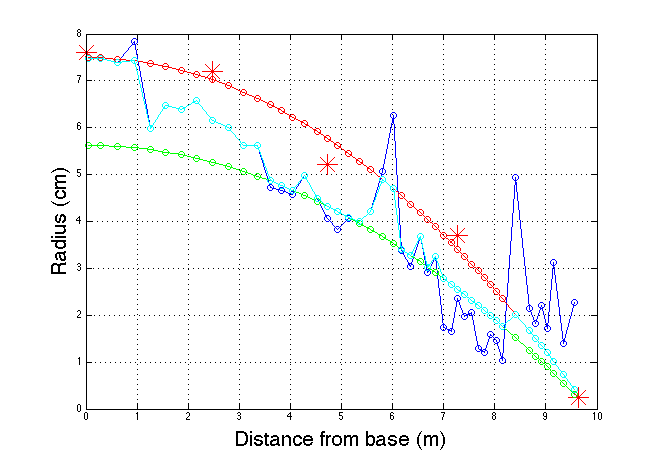
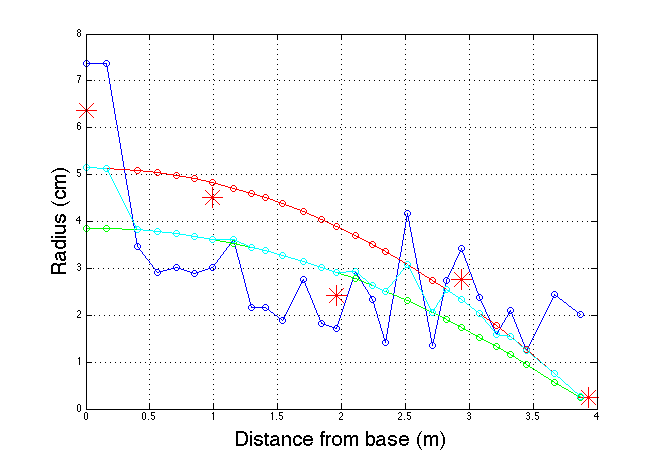


Figure 8. Cylinder radius modification with parabolas. Blue is original fitted radius, red stars are the data (scaled section averages) used for parabola fitting, red and green circled denote the maximum and minimum radius parabolas, light blue circles denote the final modified radii.

Finally the branches are determined and tree attributes are computed. One of these attributes is the trunk volume computed another way with a triangulation (see Figure 4): We select the bottom part of the trunk segment based on the fitted cylinders: cut the segment where the radius is under one third of the first cylinder or the direction changes too much. Then use the points of this shortened segment and triangulate its surface using a “cylindrical support”. The length and volume of this triangulated trunk part is computed. The same attributes computed from the cylinders for the same part are also given for comparison:

Trunk volume (cylinders) = 764 liters

Trunk volume (triangulation) = 761 liters

Trunk length (cylinders) = 8.55 meters

Trunk length (triangulation) = 8.55 meters

When the model reconstruction is done, the code produces a summary report (pdf), which contains many different attributes and distributions. The report and other result files should be saved into /result –folder. There maybe some problems with the report generation for various reasons. If so, just comment out the following lines in *qsm\_tree.m*:

publish('make\_report','pdf');

close all

str = ['results/results\_report\_',string,'.pdf'];

movefile('report/html/make\_report.pdf',str)

**How to use:**

The best way to select the input parameters at the beginning is to just try different parameter values to see how the results change. Notice that for the first cover with large cover sets, the actual values used is not so important, but they should be quite large compared to the values used in the second cover so that the first segmentation is done quickly. For the second cover with smaller sets, the values are more important because they affect the resulting QSM much more.

Because the cover generation is random, each cover is different, and this means that if you run the code with the same inputs, the results will be little different each time. Sometimes the variations in the results can be quite large. Thus if we are after some attributes such as volumes, it is desirable to make multiple models with the same inputs, e.g. 5-20 models, and then take average of these model results. From these models we can also compute standard deviations, maximums and minimums to give some reliability intervals.