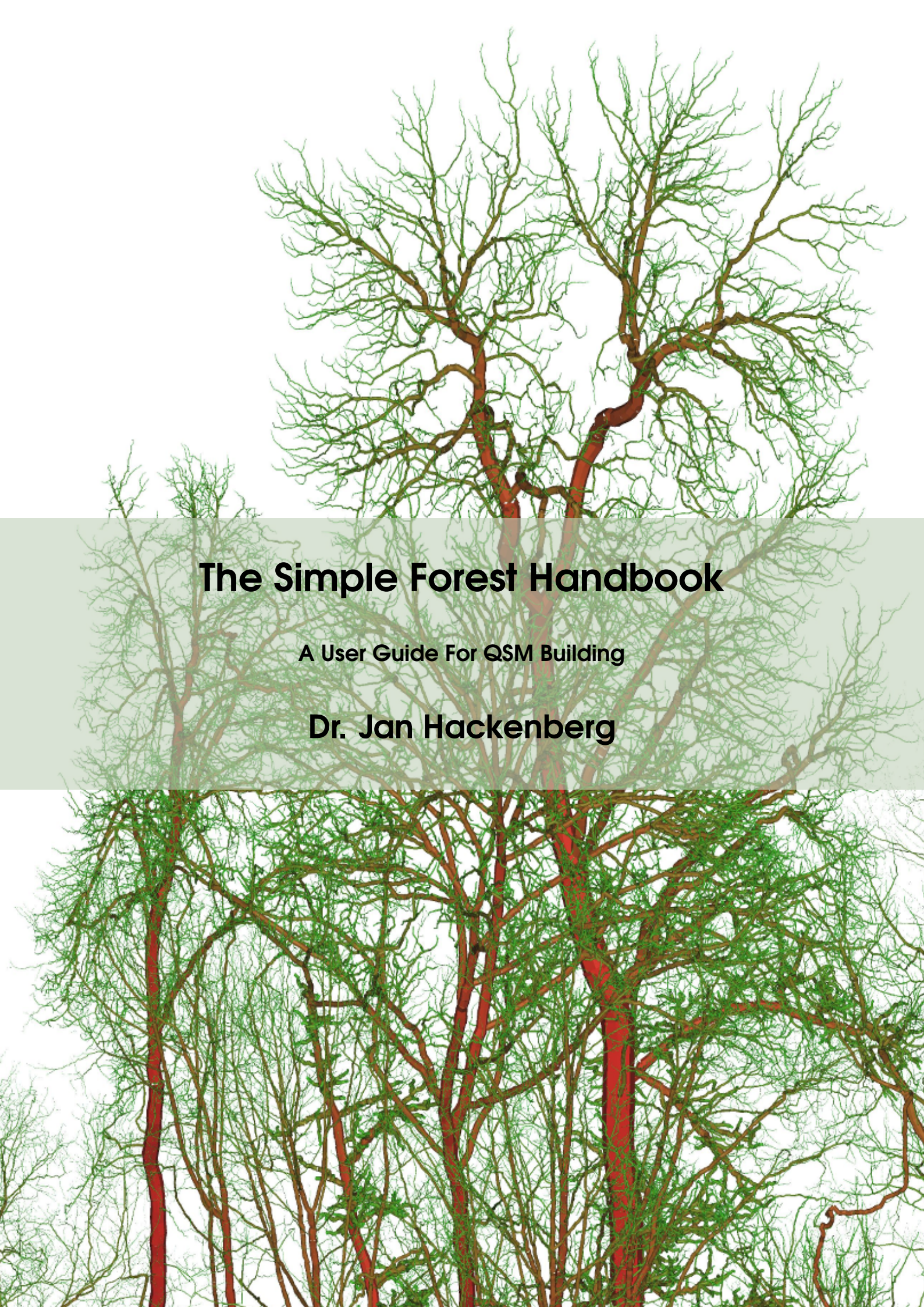


The Simple Forest Handbook

A User Guide For QSM Building

Dr. Jan Hackenberg

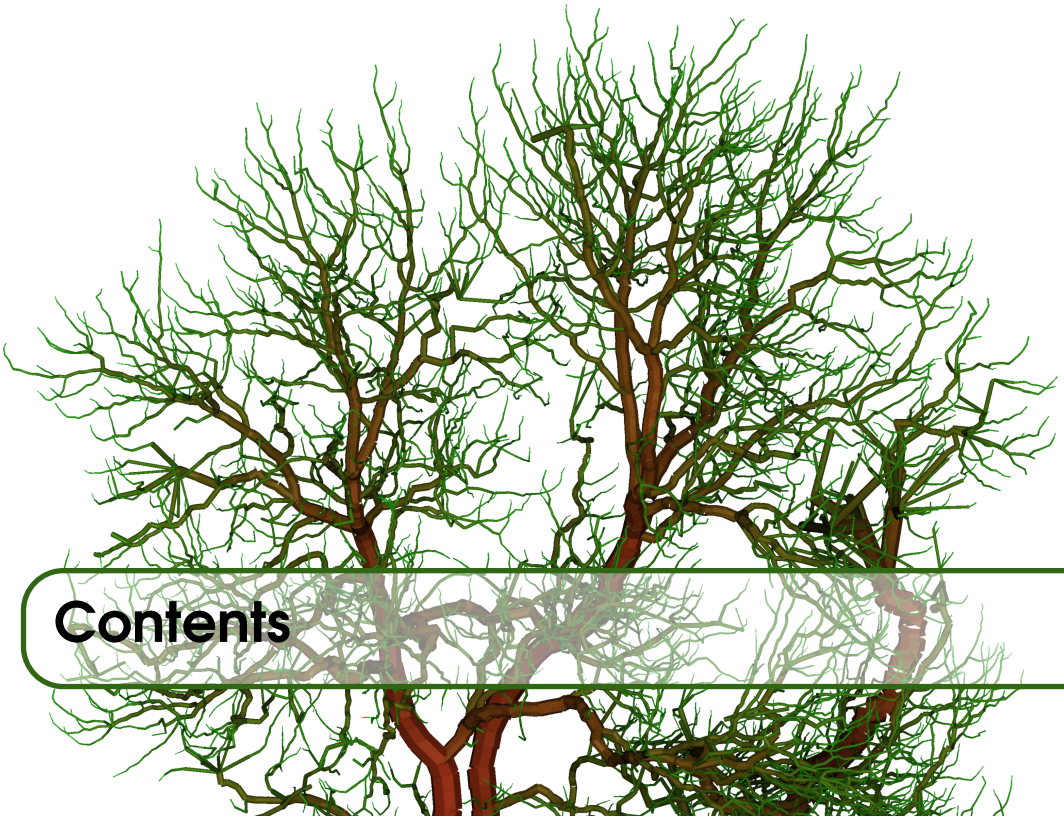
Copyright © 2019 Dr. Jan Hackenberg

FREE SOFTWARE DEVELOPER

HTTPS://SIMPLEFOREST.ORG/

Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported License (the "License"). You may not use this file except in compliance with the License. You may obtain a copy of the License at http://creativecommons.org/licenses/by-nc/3.0. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, December 2019



Contents

I Part One - definitions and statistics

1 Definitions 8
1.1 QSM 8
1.2 Growthparameters 10
1.2.1 Growthlength 10
1.2.2 Growthvolume 11
1.3 Branchorder 12
1.3.1 Branchorder 12
1.3.2 Reverse Branchorder 12
1.3.3 Reverse Pipe Area Branchorder 13
1.3.4 Reverse Pipe Radius Branchorder 13
1.4 Computational metrics 14

2 Statistics 15
2.1 Rscript 15

II Part Two - The step description

3 Source Code SimpleForest 18

4 Binary Point Cloud Filters 19
4.1 Cut cloud above DTM 20
4.1.1 Screenshots 20

4.1.2 Step Placement . . . 20
4.1.3 IO . . . 20
4.1.4 Description . . . 20
4.2 Euclidean Clustering Filter 21
4.2.1 Screenshots . . . 21
4.2.2 Step Placement . . . 21
4.2.3 IO . . . 21
4.2.4 Description . . . 21
4.3 Ground Point Filter 22
4.3.1 Screenshots . . . 22
4.3.2 Step Placement . . . 22
4.3.3 IO . . . 22
4.3.4 Description . . . 22
4.4 Radius Outlier Filter 23
4.4.1 Screenshots . . . 23
4.4.2 Step Placement . . . 23
4.4.3 IO . . . 23
4.4.4 Description . . . 23
4.5 Statistical Outlier Filter 24
4.5.1 Screenshots . . . 24
4.5.2 Step Placement . . . 24
4.5.3 IO . . . 24
4.5.4 Description . . . 24
4.6 Stem Filter 25
4.6.1 Screenshots . . . 25
4.6.2 Step Placement . . . 25
4.6.3 IO . . . 25
4.6.4 Description . . . 25
4.7 Stem Filter RANSAC 26
4.7.1 Screenshots . . . 26
4.7.2 Step Placement . . . 26
4.7.3 IO . . . 26
4.7.4 Description . . . 26
4.8 Filter By Min Height 27
4.8.1 Screenshots . . . 27
4.8.2 Step Placement . . . 27
4.8.3 IO . . . 27
4.8.4 Description . . . 27
5 Clustering . . . 28
5.1 Segmentation Euclidean Clustering 29
5.1.1 Screenshots . . . 29
5.1.2 Step Placement . . . 29
5.1.3 IO . . . 29
5.1.4 Description . . . 29

5.2 Dikstra Based Tree Segmentation 30
5.2.1 Screenshots . . . 30
5.2.2 Step Placement . . . 30
5.2.3 IO . . . 30
5.2.4 Description . . . 30
5.3 Voronoi Based Tree Segmentation 32
5.3.1 Screenshots . . . 32
5.3.2 Step Placement . . . 32
5.3.3 IO . . . 32
5.3.4 Description . . . 32
5.4 QSM based tree clustering 33
5.4.1 Screenshots . . . 33
5.4.2 Step Placement . . . 33
5.4.3 IO . . . 33
5.4.4 Description . . . 33

6 DTM/DEM steps . . . 34
6.1 Dtm Pyramidal Mlesac Fit 35
6.1.1 Screenshots . . . 35
6.1.2 Step Placement . . . 35
6.1.3 IO . . . 35
6.1.4 Description . . . 35

7 QSM steps . . . 36
7.1 QSM Spherefollowingbasic 37
7.1.1 Screenshots . . . 37
7.1.2 Step Placement . . . 37
7.1.3 IO . . . 37
7.1.4 Description . . . 37
7.2 QSM SpherefollowingAdvanced 41
7.2.1 Screenshots . . . 41
7.2.2 Step Placement . . . 41
7.2.3 IO . . . 41
7.2.4 Description . . . 41
7.3 Dijkstra Modelling 43
7.3.1 Screenshots . . . 43
7.3.2 Step Placement . . . 43
7.3.3 IO . . . 43
7.3.4 Description . . . 43
7.4 QSM Median Filter 45
7.4.1 Screenshots . . . 45
7.4.2 Step Placement . . . 45
7.4.3 IO . . . 45
7.4.4 Description . . . 45

7.5 QSM Correct Shoots **46**
7.5.1 Screenshots . . . 46
7.5.2 Step Placement . . . 46
7.5.3 IO . . . 46
7.5.4 Description . . . 46
7.6 QSM Allometric Correction **47**
7.6.1 Screenshots . . . 47
7.6.2 Step Placement . . . 47
7.6.3 IO . . . 47
7.6.4 Description . . . 47
7.7 QSM Allometric Correction Manual **51**
7.7.1 Screenshots . . . 51
7.7.2 Step Placement . . . 51
7.7.3 IO . . . 51
7.7.4 Description . . . 51
7.8 QSM Reverse Pipe Model Filter **52**
7.8.1 Screenshots . . . 52
7.8.2 Step Placement . . . 52
7.8.3 IO . . . 52
7.8.4 Description . . . 52

III **Part Three Evaluation**

7.9 Evaluation **55**
7.9.1 Results . . . 55
7.9.2 Interpretation . . . 55
7.9.3 Recommondatation . . . 56

IV **Part Four FAQ**

7.10 FAQ **58**

V **Part Five Change Log**

7.11 Version 5.2.2 **60**
7.12 Version 5.2.1 **60**
7.13 Version 5.1.4 **60**
7.14 Version 5.1.3 **60**

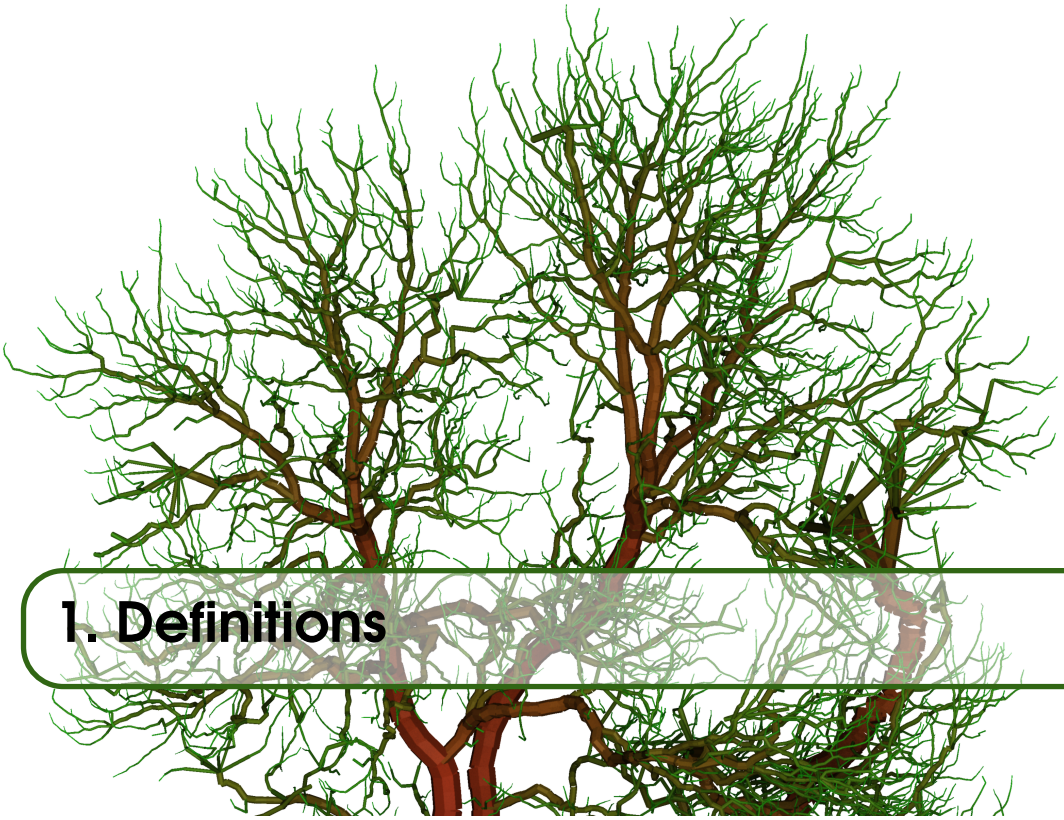
VI **Part Six Citations**

Bibliography . . . 62
Articles **62**
Index . . . 65

Part One - definitions and statistics

1 **Definitions** 8
1.1 QSM
1.2 Growthparameters
1.3 Branchorder
1.4 Computational metrics

2 **Statistics** 15
2.1 Rscript



1. Definitions

1.1 QSM

A quantitative structure model (QSM) is a topological orederd structure of building bricks, in our case cylinders.

Definition 1.1.1 — Segment. Cylinders between two neighboring branching juntions are stored within a segment. The cylinders are an ordered list with the first cylinder being the cylinder closest to the root and the last cylinder the furthest one.

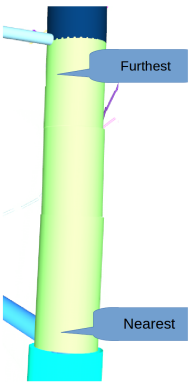


Figure 1.1: Cylinders between two branch junctions colored in bright green.

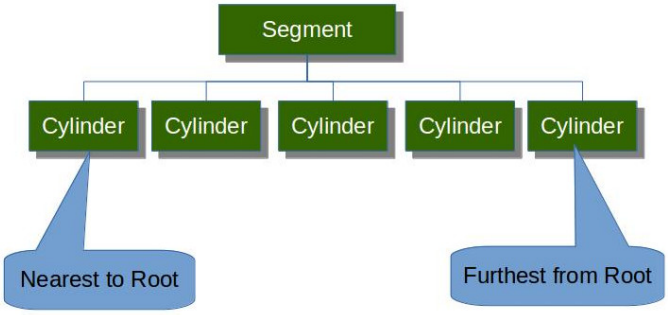


Figure 1.2: The abstract segment presentation.

Each Segment has at least 2 child segments connected to it, if it ends in a branch juntion or zero children if it is a tip segment. We can see examples in figures 1.1, 1.1 and 1.1.

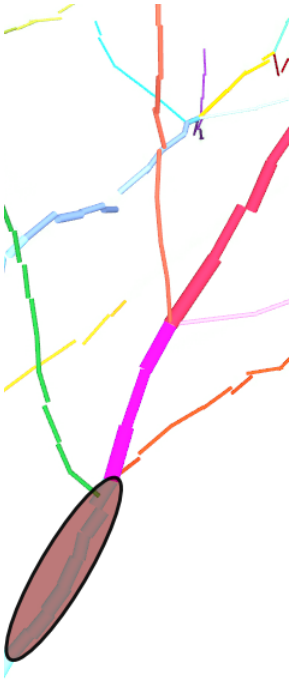


Figure 1.3: A cylinder screenshot with a highlighted cylinder.

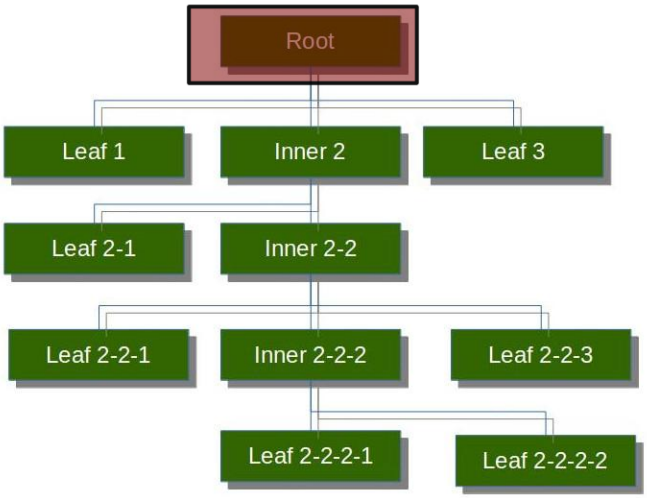


Figure 1.4: The highlighted cylinder in the abstract tree represenation.

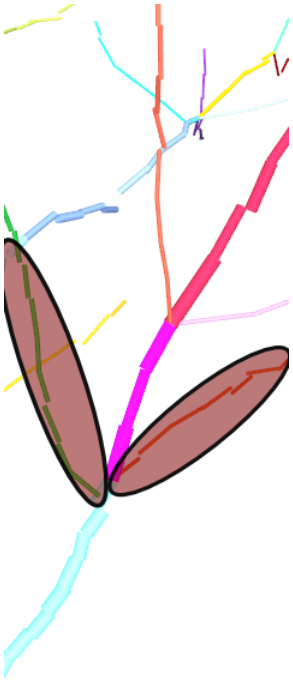


Figure 1.5: A cylinder screenshot with two highlighted cylinders.

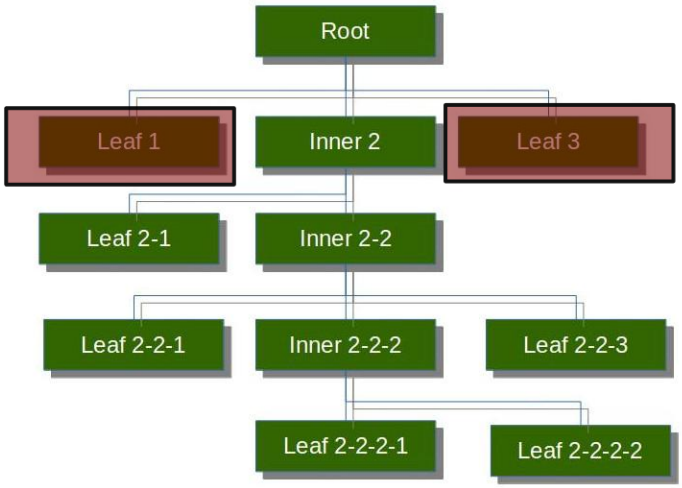


Figure 1.6: The highlighted cylinders in the abstract tree represenation.

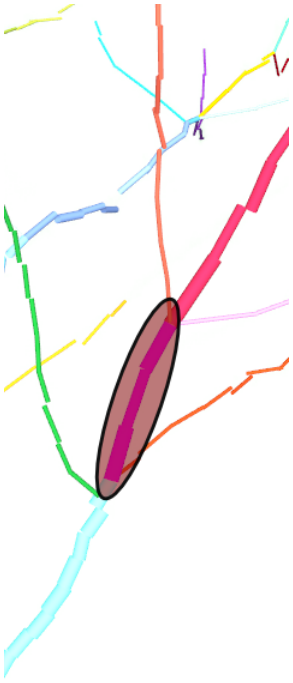


Figure 1.7: A cylinder screenshot with a highlighted cylinder.

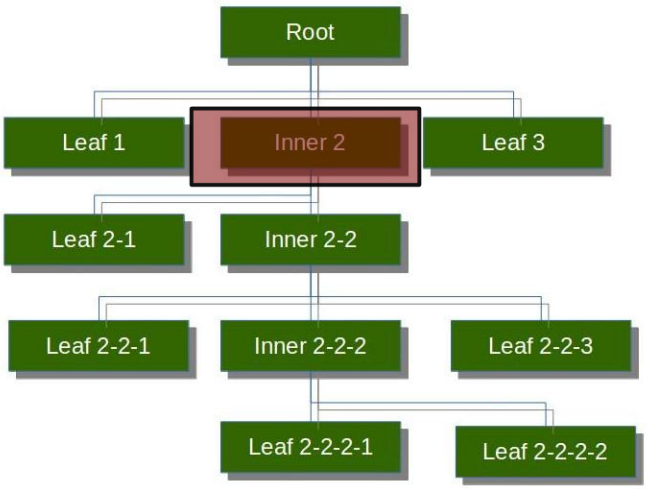


Figure 1.8: The highlighted cylinder in the abstract tree represenation.

As the segments have a topological parent child relation, also its contained cylinders have one.

1.2 Growthparameters

In *Hackenberg et al.* **2015b** [9] recursive parameters as the following have been firstly defined as output.

We use here in the following:

1.2.1 Growthlength

Definition 1.2.1 — Growthlength. The growthlength of a cylinder is the cylinder's length plus its childrens' growthlength.

We can see an example in figure 1.9.

Figure 1.9: The cylinder of interest is the green one. Its growthlength is the summed up length of all green and yellow cylinders.

In figure 1.10 we see the modeled relation between growthlength and radius.

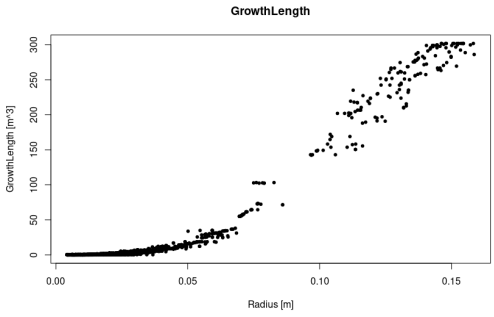


Figure 1.10: We plot for a *Quercus Petraea* radius vs growthlength. A clear s-shaped pattern is visible.

1.2.2 Growthvolume

The growthvolume is defined in the same manner as the growthlength, but utilizes volume instead of length

Definition 1.2.2 — Growthvolume. The growthvolume of a cylinder is the cylinder's volume plus its childrens' growthvolume.

In figure 1.11 we see the modeled relation between growthvolume and radius.

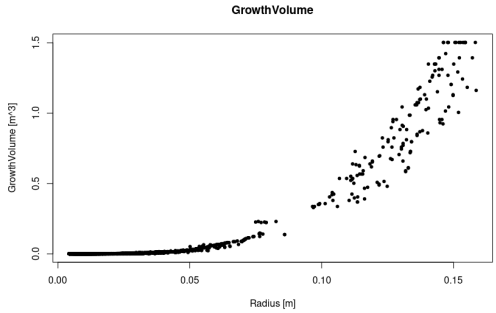


Figure 1.11: We plot for a *Quercus Petraea* radius vs growthvolume. A clear power function is visible.

1.3 Branchorder

SimpleForest has implemented various ways to express the branch order of the underlying branch
We use here in the following:

1.3.1 Branchorder

The traditional way to express the branch order. The stem is initialized with branch order 0. Branches - major branches as well as shoots - splitting from the stem have the branch order 1. At each branch juntion the branch order of side branches is increased by 1.

The benefit of this branch order is that it can be also measured in the field. But it does not correlate well with the radius, as we can see in figure 1.12.

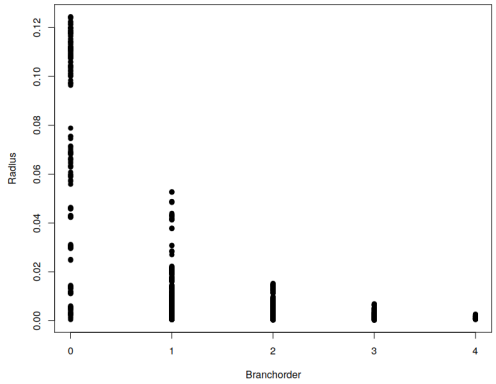


Figure 1.12: Each branch order contains large as well as small radii.

1.3.2 Reverse Branchorder

In constrast to that we output as well:

Definition 1.3.1 — reverse Branchorder. The reverse Branchorder is defined as the maximal number of branch junctions which can be passed before reaching a tip.

We can clearly see, that this one correlates a lot better with the radius:

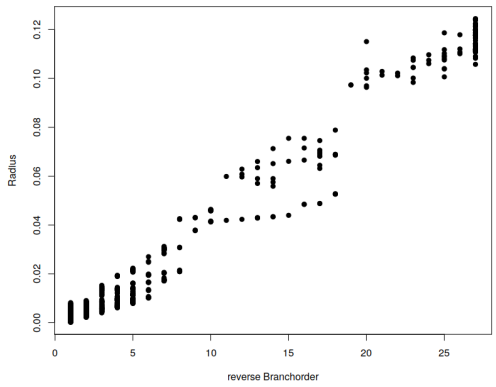


Figure 1.13: The reverse Branchorder has a close to linear relation to the radius.

1.3.3 Reverse Pipe Area Branchorder

According to the metabolic scaling theory the number of pipes/vessels of a tip is constant. So the area of a típ can be seen as constant. If two or more tips merge into a common parent segment, their number of pipes merges as well.

Definition 1.3.2 — reverse Pipe Area Branchorder. A tip receives the abstract unit 1 as reverse Pipe Area Branchorder. A parent section receives the sum of its children's reverse Pipe Area Branchorder. The reverse Pipe Area Branchorder of a cylinder therefore equals the number of supported tips.

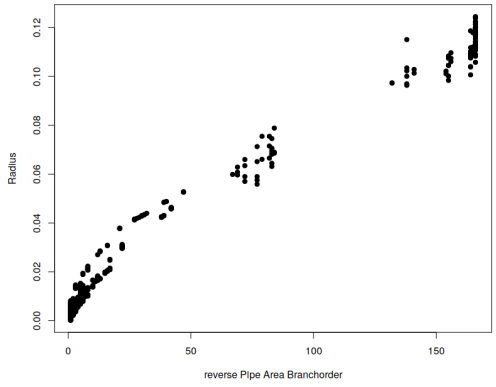


Figure 1.14: We can see a strong non linear relation between reverse Pipe Area Branchorder and the radius.

1.3.4 Reverse Pipe Radius Branchorder

Definition 1.3.3 — reverse Pipe Radius Branchorder. The reverse Pipe Radius Branchorder is simply the square root of the reverse Pipe Branchorder.

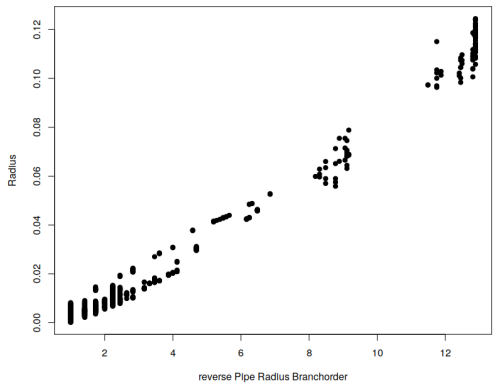


Figure 1.15: We can see a strong linear relation between reverse pipe radius Branchorder and the radius.

1.4 Computational metrics

For the optimization routines the cloud to model distance is used as a metric.

Definition 1.4.1 — Point to Cylinder distance. We define the distance between a single point and a single cylinder as the eulidean distance between the point and the finite cylinder hull.

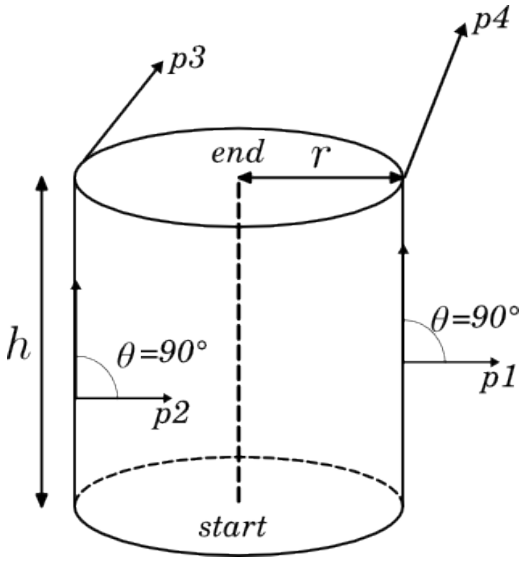
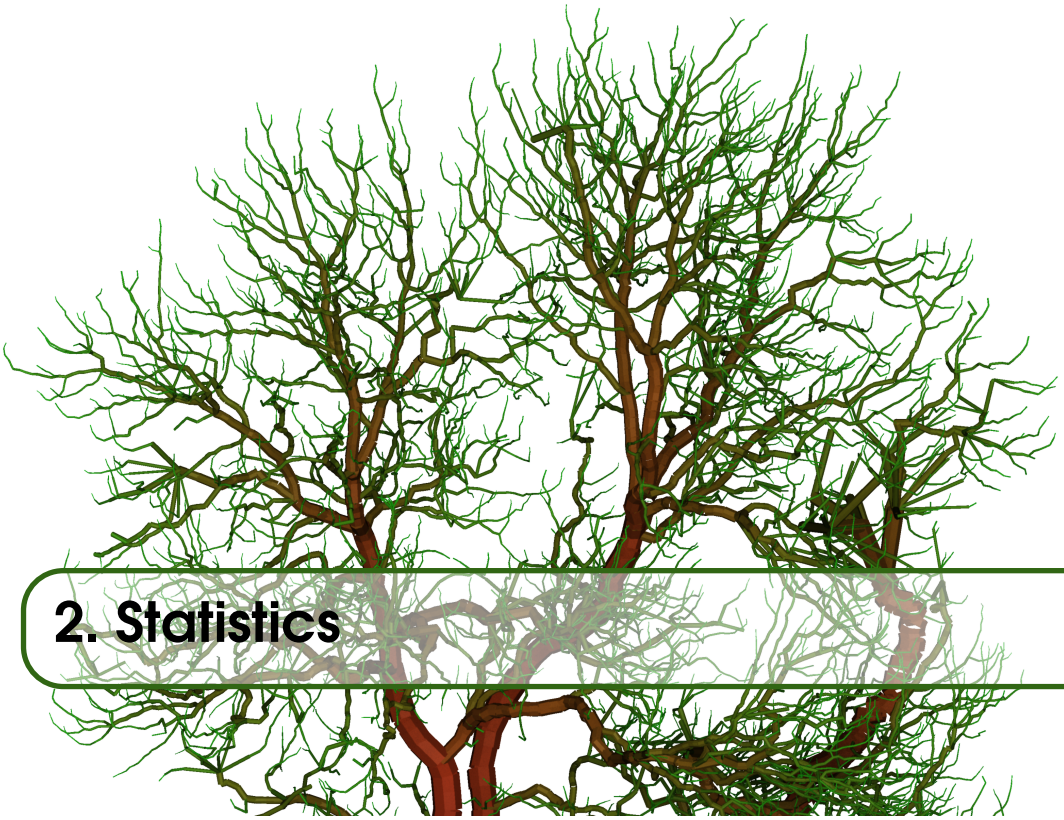


Figure 1.16: We see for points and a cylinder, the distance is always positive.

Definition 1.4.2 — Point to Model distance. A QSM consists of n cylinders. A single point has therefore n Point to Cylinder distances. The minimum of those n distances is the Point to Model distance.

Definition 1.4.3 — Cloud to Model distance. Depending on which metric is finally used we can either build the average Point to Model distance for all points or used the root summed squared metric here.



2. Statistics

2.1 Rscript

A gitlab repository with an R-Script is available `https://gitlab.com/SimpleForest/simpleForestR`. The script allows for powerful analysis of QSM csv output files. We can see various plots generated with that script.

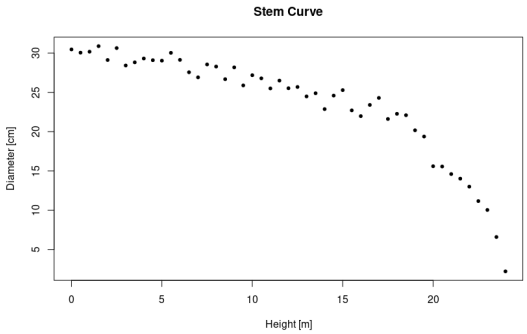


Figure 2.1: The stem curve of a Quercus Petraea.

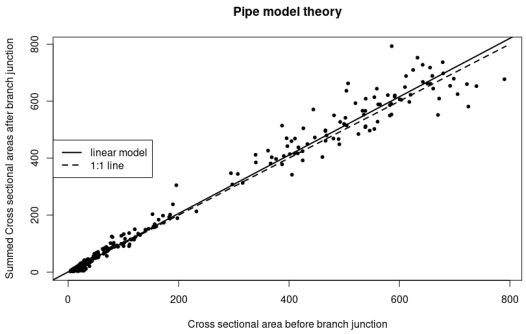


Figure 2.2: Applied pipemodel theory on a Quercus Petraea.

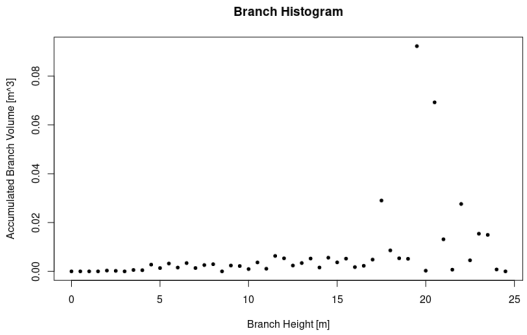


Figure 2.3: An histogram about the volume of branches along the stem height.

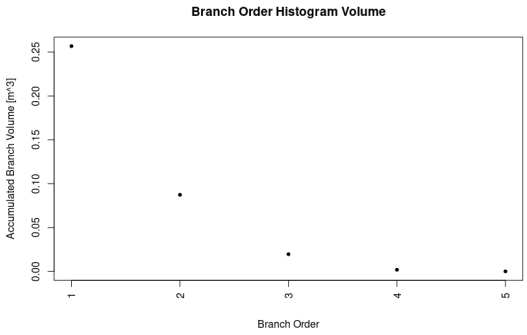


Figure 2.4: The volume of branches depending on the branch order.

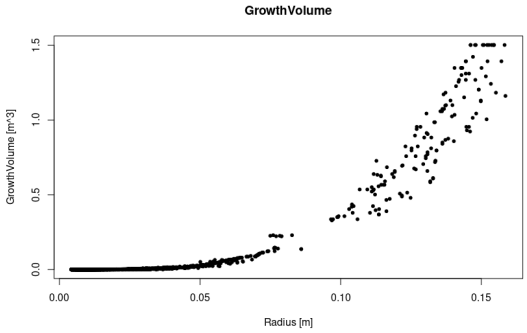


Figure 2.5: The growthvolume radius relation.

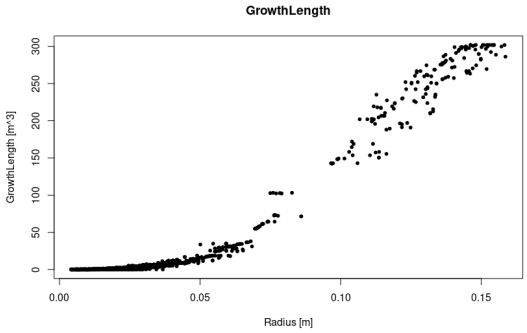


Figure 2.6: The growthlength radius relation.

II Part Two - The step description

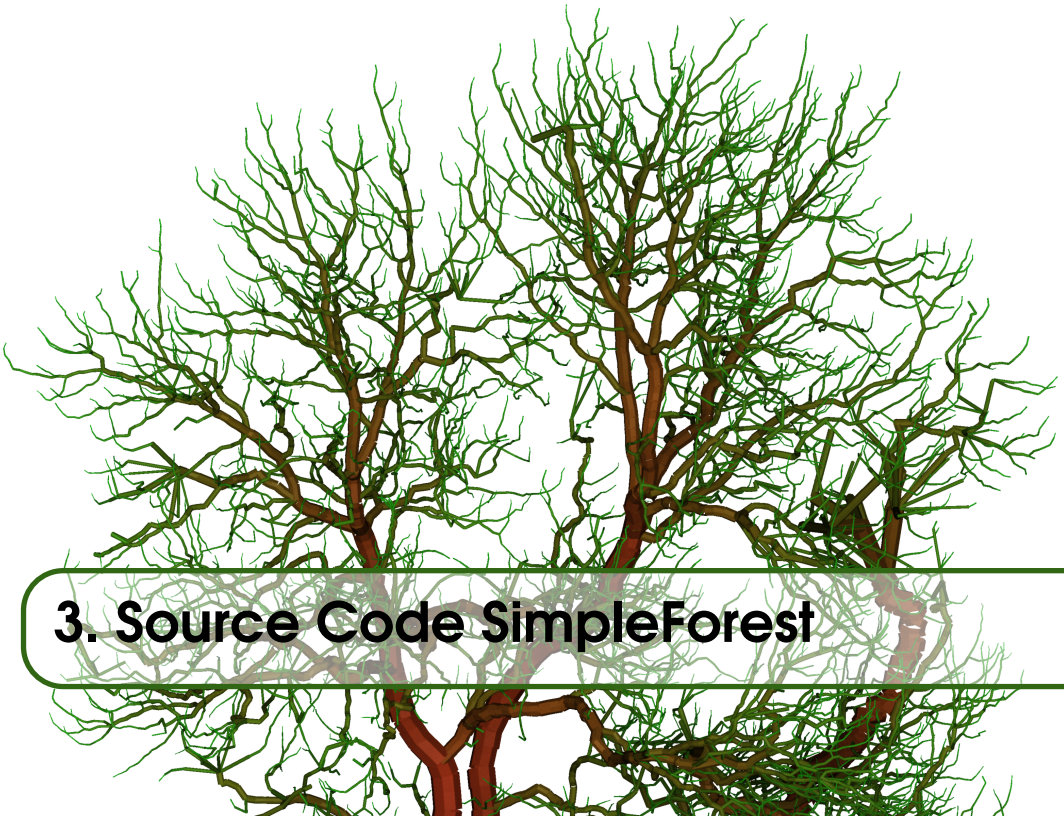
3 Source Code SimpleForest 18

4 Binary Point Cloud Filters 19
4.1 Cut cloud above DTM
4.2 Euclidean Clustering Filter
4.3 Ground Point Filter
4.4 Radius Outlier Filter
4.5 Statistical Outlier Filter
4.6 Stem Filter
4.7 Stem Filter RANSAC
4.8 Filter By Min Height

5 Clustering 28
5.1 Segmentation Euclidean Clustering
5.2 Dikstra Based Tree Segmentation
5.3 Voronoi Based Tree Segmentation
5.4 QSM based tree clustering

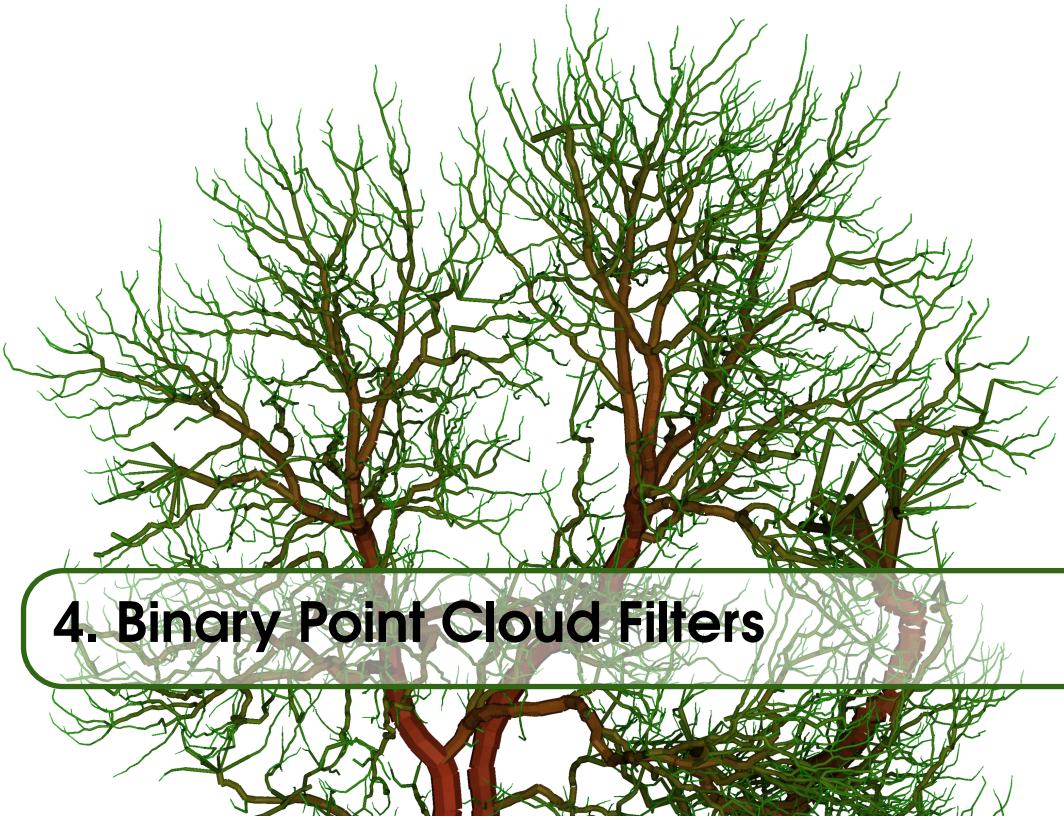
6 DTM/DEM steps 34
6.1 Dtm Pyramidal Mlesac Fit

7 QSM steps 36
7.1 QSM Spherefollowingbasic
7.2 QSM SpherefollowingAdvanced
7.3 Dijkstra Modelling
7.4 QSM Median Filter
7.5 QSM Correct Shoots
7.6 QSM Allometric Correction
7.7 QSM Allometric Correction Manual
7.8 QSM Reverse Pipe Model Filter



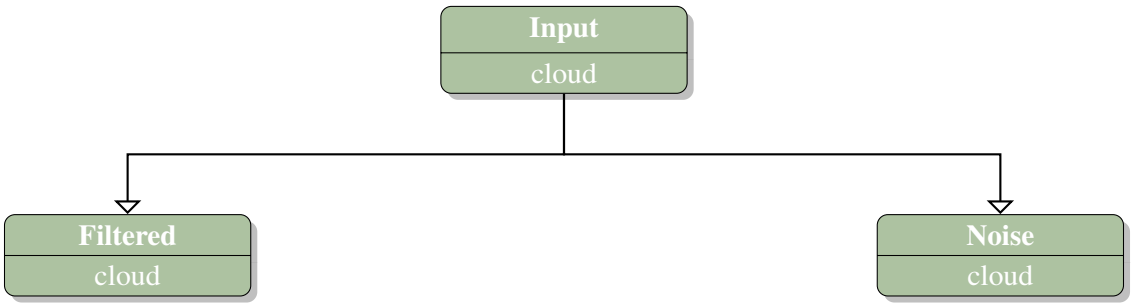
3. Source Code SimpleForest

A gitlab repository with the complete SimpleForest plugin `https://gitlab.com/SimpleForest/pluginSimpleForest` is available.



4. Binary Point Cloud Filters

Here point cloud filters are described which split up an input cloud into a denoised cloud and a noise cloud.



4.1 Cut cloud above DTM

4.1.1 Screenshots

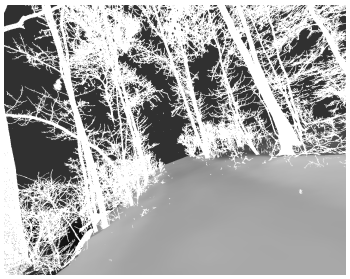


Figure 4.1: The input point cloud and the DTM.

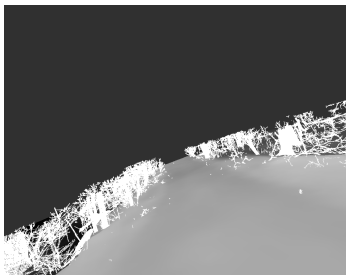


Figure 4.2: The lower points output and the DTM.

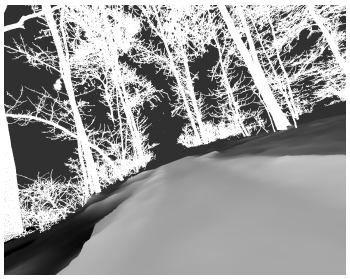


Figure 4.3: The upper points output and the DTM.

4.1.2 Step Placement

- Points
  - Filter

4.1.3 IO

Input — A point scene.

Input — The according DTM.

Output — Lower - the lower points.

Output — Upper - the upper points.

4.1.4 Description

The step imports a cloud and its according digital terrain model (DTM) as input. For each point its height above the DTM is computed. If the height is lower than a **threshold height**, it is put into the first output cloud, otherwise into the second.

The step can be used to cut out an improved vegetation cloud from an unclassified plot scenerie. Or it can be used to generate a seed slice for segmentation which afterwards can be clustered.

4.2 Euclidean Clustering Filter

4.2.1 Screenshots

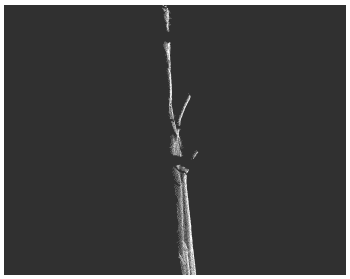


Figure 4.4: The input point cloud.

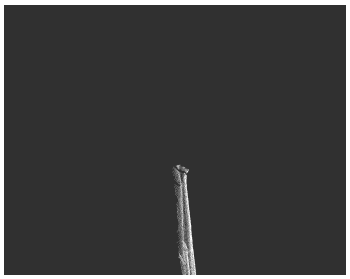


Figure 4.5: The largest cluster in good output.

Figure 4.6: All other points in noise output.

4.2.2 Step Placement

- Points
  - Filter

4.2.3 IO

Input — A point cloud.

Output — Filtered Cloud - the largest clusters combined in one cloud.

Output — Noise - all other points.

4.2.4 Description

The step imports a cloud as input. The cloud is downscaled with the voxel grid downscale routine from the PCL library [10]. The **voxel size** of the downscale routine is a user parameter - standard value is 2 (*cm*).

Next the PCL euclidean clustering routine is performed. The distance between two clusters has to be larger than a user given **range**, otherwise the clusters are merged. We use a standard value of 5 (*cm*).

If a cluster less than **minimum percentage** of the downscaled cloud size or more than **maximum percentage** of the downscaled cloud it is contained in the noise output. All other clusters' points are stored in the filtered cloud output. We dont apply this two thresholds in standard values as we use 0% and 100% respectively.

Instead we store the **n** largest clusters fullfilling this threshold in the filtered cloud output only, with **n** set to 1 as standard.

4.3 Ground Point Filter

4.3.1 Screenshots

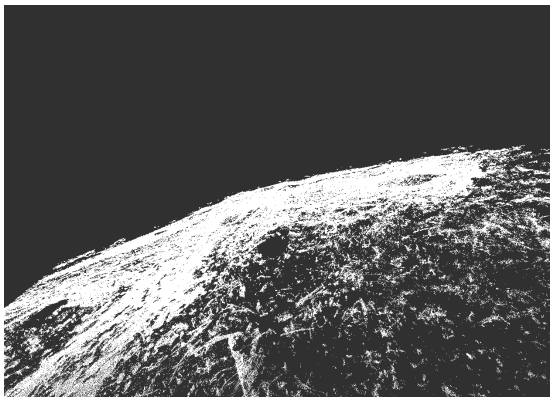


Figure 4.7: The ground points output.

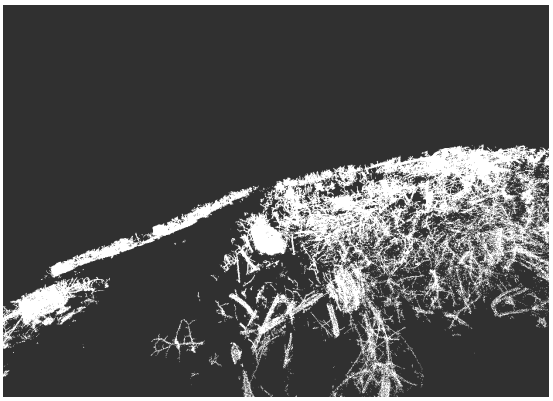


Figure 4.8: The noise points output.

4.3.2 Step Placement

- Points
  - Filter

4.3.3 IO

Input — A point scene.

Output — Ground Cloud - the ground points.

Output — Noise - all other points.

4.3.4 Description

The step imports a cloud as input. The cloud is downscaled with the voxel grid downscale routine from the PCL library [9, 10]. The **voxel size** of the downscale routine is a user parameter - standard value is 4 (*cm*). This downscaling is done to assure an averaged point density over the input cloud as well as to fasten up the following procedure.

For each downscaled point the normal is computed. A plane is fitted into the point‘s neighborhood and the plane normal is the normal of the point. For retrieving the point‘s neighbors we need to give a **search range**. The **search range** has to be larger than **voxel size** as otherwise no points would be included in the neighborhood. The standard value is 20 (*cm*).

Then the angle between the point‘s normal and the z-axis is computed. If the angle is smaller than a user given **threshold** it is accounted as a ground point and as noise otherwise.

Lastly a nearest neighbor search is performed for each point in the original input cloud to the classified downscaled cloud. The original point will inherit its classification from its nearest downscaled neighbor.

4.4 Radius Outlier Filter

4.4.1 Screenshots

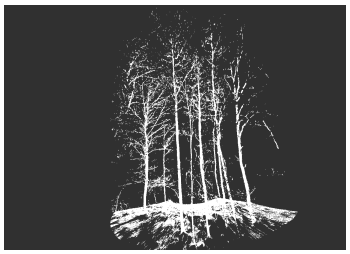


Figure 4.9: The input point cloud.

Figure 4.10: The filtered points output.

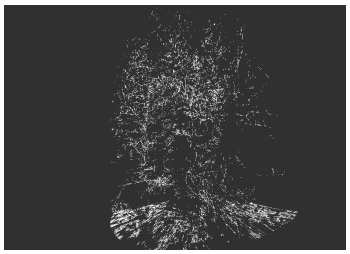


Figure 4.11: The noise points output.

4.4.2 Step Placement

- Points
  - Filter

4.4.3 IO

Input — A point scene.

Output — Filtered Cloud - the filtered points.

Output — Noise - all points having not enough neighbors.

4.4.4 Description

The step imports a cloud as input. On the cloud the radius outlier filter routine from the PCL library [9, 10] is performed. For each point the number of neighbors within a user given **search range** are retrieved. If the number of neighbors is smaller than a **minimum number** of points, the point is considered noise. Otherwise it will remain in the filtered cloud.

The step‘s procedure can also be used as a clear sky filter by applying a large **search range** of for example one meter and looking for a large number of **minimum number** points, e.g. 1000.

4.5 Statistical Outlier Filter

4.5.1 Screenshots

Figure 4.12: The input point cloud.

Figure 4.13: The filtered points output.

Figure 4.14: The noise points output.

4.5.2 Step Placement

- Points
  - Filter

4.5.3 IO

Input — A point scene.

Output — Filtered Cloud - the filtered points.

Output — Noise - all points having a too large distance to the neighbors.

4.5.4 Description

For each point the **n** closest neighbors are retrieved [9, 10]. The average distance to those neighbor points is computed. Of all points' average distances mean and standard deviation are calculated. If a points average distance is larger than the mean plus a **factor** multiplied with the standard deviation, the point is considered noise.

After an execution of the filter the point distribution changes, so the procedure can be repeated a user desired number of **iterations**.

It is recommended to choose a large number of **iterations**, such as 15, and a large number of standard deviation muliplication **factor**, such as 4. The number of **n** closest neighbors can be choosen best first small, e.g. 2, and then the step is exectuted a second time with a larger number of **n**.

4.6 Stem Filter

4.6.1 Screenshots

Figure 4.15: The input point cloud.

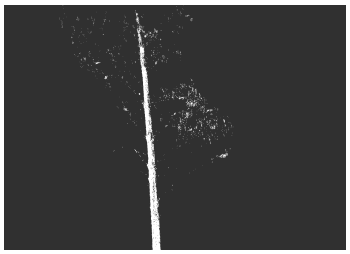


Figure 4.16: The stem points output.

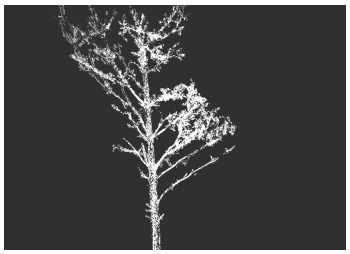


Figure 4.17: The non stem points output.

4.6.2 Step Placement

- Points
  - Filter

4.6.3 IO

Input — A point scene.

Output — Stem Cloud - the stem points.

Output — Noise - all other points.

4.6.4 Description

The step imports a cloud as input. The cloud is downscaled with the voxel grid downscale routine from the PCL library [10]. The **voxel size** of the downscale routine is a user parameter - standard value is 2 (*cm*). This downscaling is done to assure an averaged point density over the input cloud as well as to fasten up the following procedure.

For each downscaled point the normal is computed. A plane is fitted into the point‘s neighborhood and the plane normal is the normal of the point. For retrieving the point‘s neighbors we need to give a **search range**. The **search range** has to be larger than **voxel size** as otherwise no points would be included in the neighborhood. The standard value is 5 (*cm*).

Now each point has a precomputed normal. We again apply a second range search with a **second search range** - has to be larger than **search range**. Teh standard value is 12 (*cm*). On all normals of the points within the **second search range** a Principal Component Analysis is performed. The Eigenvector of the largest Eigenvalue is returned as the so called principal direction. This principal direction is alligned with the growth direction of the underlying branch or stem segment [11].

Then the angle between the point‘s growth direction and the z-axis is computed. If the angle is smaller than a user given **threshold** it is accounted as a stem point and as noise otherwise. We use 20 as a standard value here.

Lastly a nearest neighbor search is performed for each point in the original input cloud to the classified downscaled cloud. The original point will inherit its classification from its nearest downscaled neighbor.

4.7 Stem Filter RANSAC

4.7.1 Screenshots

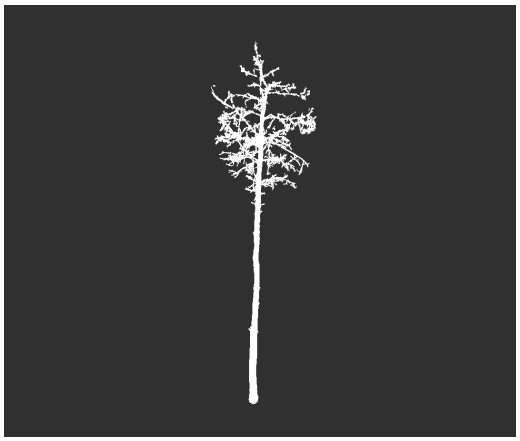


Figure 4.18: The input point cloud.

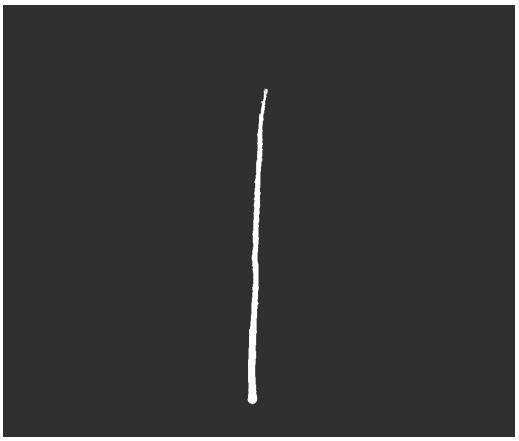


Figure 4.19: The stem points output.

4.7.2 Step Placement

- Points
  - Filter

4.7.3 IO

Input — A point scene.

Output — Stem Cloud - the stem points.

Output — Noise - all other points.

4.7.4 Description

The step imports a cloud as input. The cloud is downscaled with the voxel grid downscale routine from the PCL library [10]. The **voxel size** of the downscale routine is a user parameter - standard value is 2 (*cm*). This downscaling is done to assure an averaged point density over the input cloud as well as to fasten up the following procedure.

For each downscaled point the normal is computed. A plane is fitted into the point‘s neighborhood and the plane normal is the normal of the point. For retrieving the point‘s neighbors we need to give a **search range**. The **search range** has to be larger than **voxel size** as otherwise no points would be included in the neighborhood. The standard value is 3 (*cm*).

The cloud is sliced into 1 meter height slices. Into each slice with an **inlier range** a RANSAC cylinder is fitted. The standard value is 10 (*cm*). If the angle betwen the cylinder axis and the z-axis is smaller than **max degrees** all inliers are considered stem points. Otherwise the points are considered noise.

4.8 Filter By Min Height

4.8.1 Screenshots

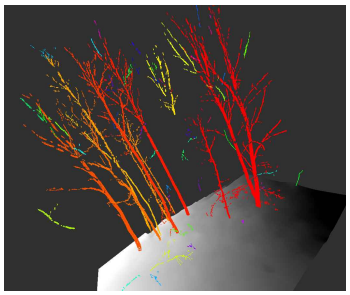


Figure 4.20: The input point cloud as well as a DTM.

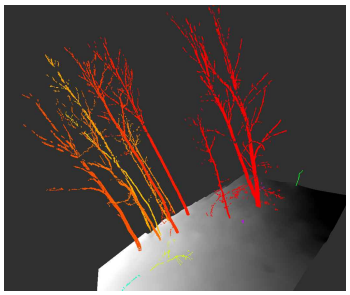


Figure 4.21: The valid clusters close enough to the DTM.

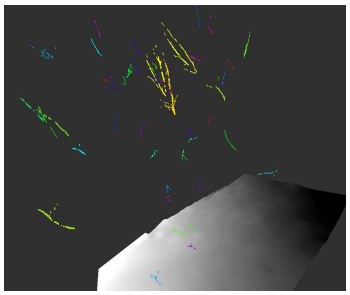


Figure 4.22: The noise clusters.

4.8.2 Step Placement

- Points
  - Filter

4.8.3 IO

Input — A point scene.

Input — The DTM.

Output — Tree Cloud - the tree points.

Output — Noise Cloud - all other points.

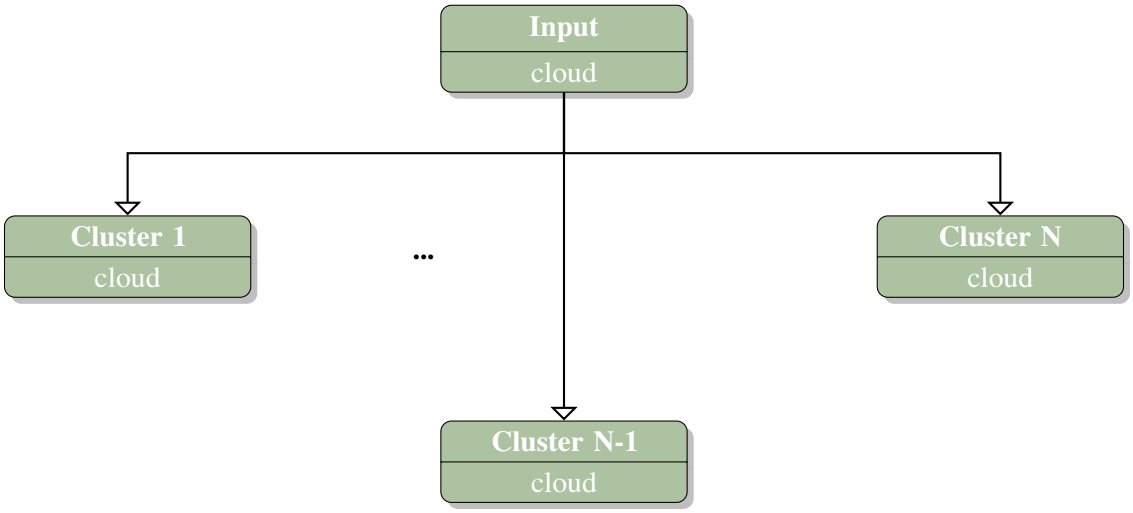
4.8.4 Description

Within the vegetation cloud after segmentation and filtering artefact clouds of trees not belonging to any tree within the plot can remain. While the stem and other major parts of those trees are outside of the plot borders crown artefacts can be within the point cloud. Those artefacts sub clouds are far above the ground model.

This steps helps to remove those artefacts. It searches within each input cloud for that clouds' lowest point. Then it computes the distance of the lowest point to the ground. If the distance is larger than **maximal MinHeight**, the cloud is considered an artefact, e.g. noise. Otherwise it is classified as useful vegetation.

5. Clustering

Here clustering routines are described which split up one cloud into multiple other clouds.



5.1 Segmentation Euclidean Clustering

5.1.1 Screenshots

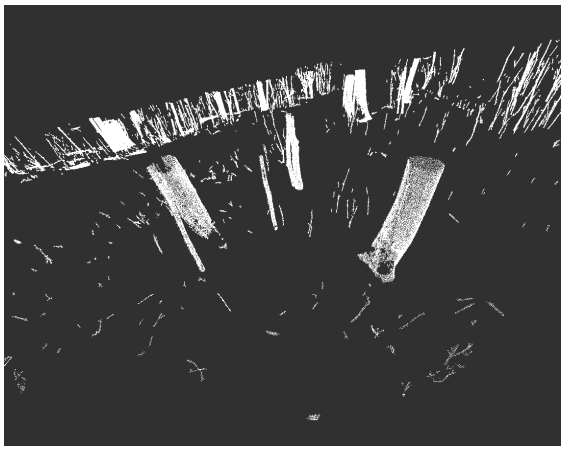


Figure 5.1: The points stored into a single cloud.

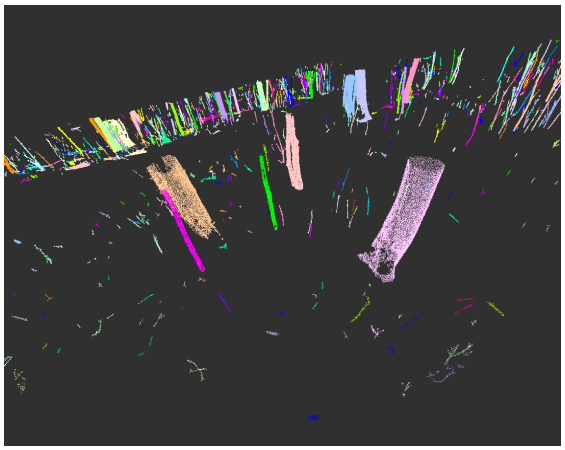


Figure 5.2: The input cloud split into clusters.

5.1.2 Step Placement

- Points
  - Clusterize

5.1.3 IO

Input A point cloud.

Output Euclidean clusters - input cloud split into multiple clusters.

5.1.4 Description

The step imports a cloud as input. The cloud is downscaled with the voxel grid downscale routine from the PCL library [10]. The **voxel size** of the downscale routine is a user parameter - standard value is 2 (*cm*).

Next the PCL euclidean clustering routine is performed. The distance between two clusters has to be larger than a user given **range**, otherwise the clusters are merged. We use a standard value of 5 (*cm*).

A cluster has to contain more than **n** points, otherwise it will be removed.

5.2 Dikstra Based Tree Segmentation

5.2.1 Screenshots

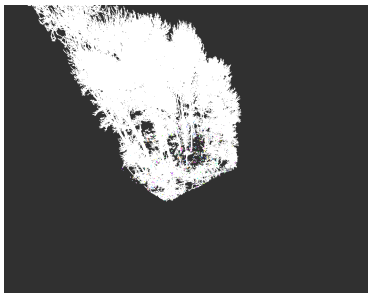


Figure 5.3: A vegetation cloud stored into a single cloud.

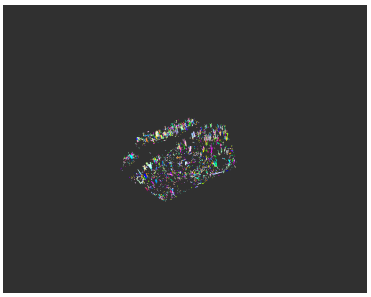


Figure 5.4: A slice near root height split into clusters.

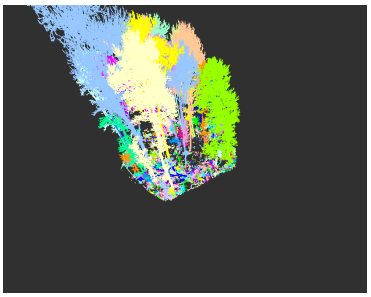


Figure 5.5: The vegetation cloud split into different trees.

Figure 5.6: A vegetation cloud in low quality.

Figure 5.7: A slice near root height split into clusters.

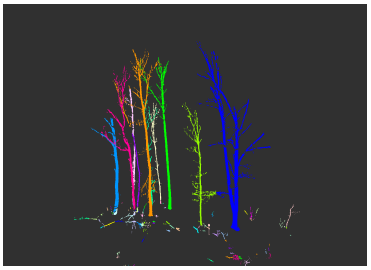


Figure 5.8: The vegetation split into incomplete trees.

5.2.2 Step Placement

- Points
  - Clusterize

5.2.3 IO

Input — Vegetation cloud - A point cloud of a forestry scene.

Input — Seed clusters - The seed clusters close to the root of the trees.

Output — Dijsktra Segmented - The vegetation cloud split into multiple clusters consisting of trees.

5.2.4 Description

The step imports a vegetation cloud as input. Optionally this cloud can be scaled along the z-axis with a **factor** between 0.1 and 1. Scaling should be applied for lower quality point clouds. This enables the algorithm to jump over occlusion gaps along the stem. Those gaps get smaller while horizontal gaps between two neighboring trees remain their size and the algorithm does not jump over to other trees.

Then the cloud is downscaled with the voxel grid downscale routine from the PCL library [10]. The **voxel size** of the downscale routine is a user parameter - standard value is 3 (*cm*).

The seed clusters are treated in the same manner. Each cluster here already has a different tree ID. Dijkstra's algorithm [4] is then initialized. We perform a nearest neighbor search from the seed clusters to the forestry scene. The nearest forestry scene points retrieve the cluster ID of their seed neighbor point and a Dijkstra distance of zero, all other points are initialized with unkown ID and distance infinity. Between all forest scene point pairs with a distance smaller than a user given **range** edges are generated. When Dijkstra's algorithm terminates, most vegetation scene points are connected to one of the seed points. The connection is along a path on the tree surface if **range** is choosen small enough. Nevertheless **range** should be larger than **voxel size**. We choose in general a 2-3 times larger value here.

Also Dijkstra's algortithm cannot jump over occlusion gaps which are larger than **range**. The parameter **factor** can help here. Otherwise both **voxel size** and **range** can be enlarged.

It is sufficient to have segmented information for the major branching structure, see figure 5.8. After the segmentation of major branch structure the Voronoi based segmentation 5.3 is able to fully segment the remaining cloud.

Lastly a nearest neighbor search is performed for each point in the original vegetation cloud to the classified downscaled cloud. The original point will inherit its classification from its nearest downscaled neighbor.

5.3 Voronoi Based Tree Segmentation

5.3.1 Screenshots

Figure 5.9: A vegetation cloud stored into a single cloud.

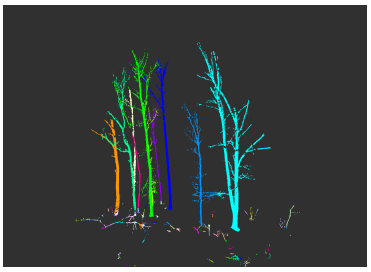


Figure 5.10: An incomplete Djikstra segmentated scene.

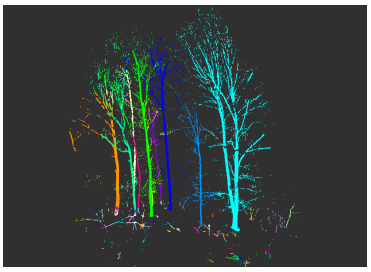


Figure 5.11: The vegetation cloud split into different trees.

5.3.2 Step Placement

- Points
  - Clusterize

5.3.3 IO

Input — Vegetation cloud - A point cloud of a forestry scene.

Input — Seed clusters - the seed clusters produced by an incomplete segmentation of the Dijkstra algorithm.

Output — Voronoi Segmented - vegetation cloud split into multiple clusters consisting of trees.

5.3.4 Description

The step imports a vegetation cloud as input. Optionally this cloud can be scaled along the z-axis with a **factor** between 0.1 and 1. Scaling should be applied for lower quality point clouds. This enables the algorithm to jump over occlusion gaps along the stem. Those gaps get smaller while horizontal gaps between two neighboring trees remain their size and the algorithm does not jump over to other trees.

Then a nearest neighbor search is performed for each point in the unsegmented cloud to the Dijkstra seeds. If the distance to the nearest neighbor is smaller than **threshold**, the vegetation point inherits the Dijkstra tree ID.

5.4 QSM based tree clustering

5.4.1 Screenshots

Figure 5.12: A tree cloud stored into a single cloud.

Figure 5.13: A qsm produced with this cloud.

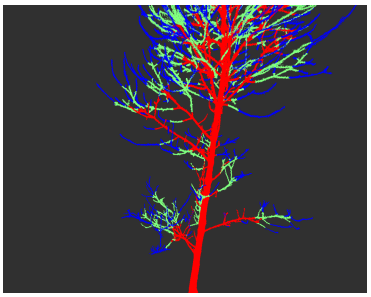


Figure 5.14: The tree cloud split into multiple clusters.

5.4.2 Step Placement

- Points
  - Clusterize

5.4.3 IO

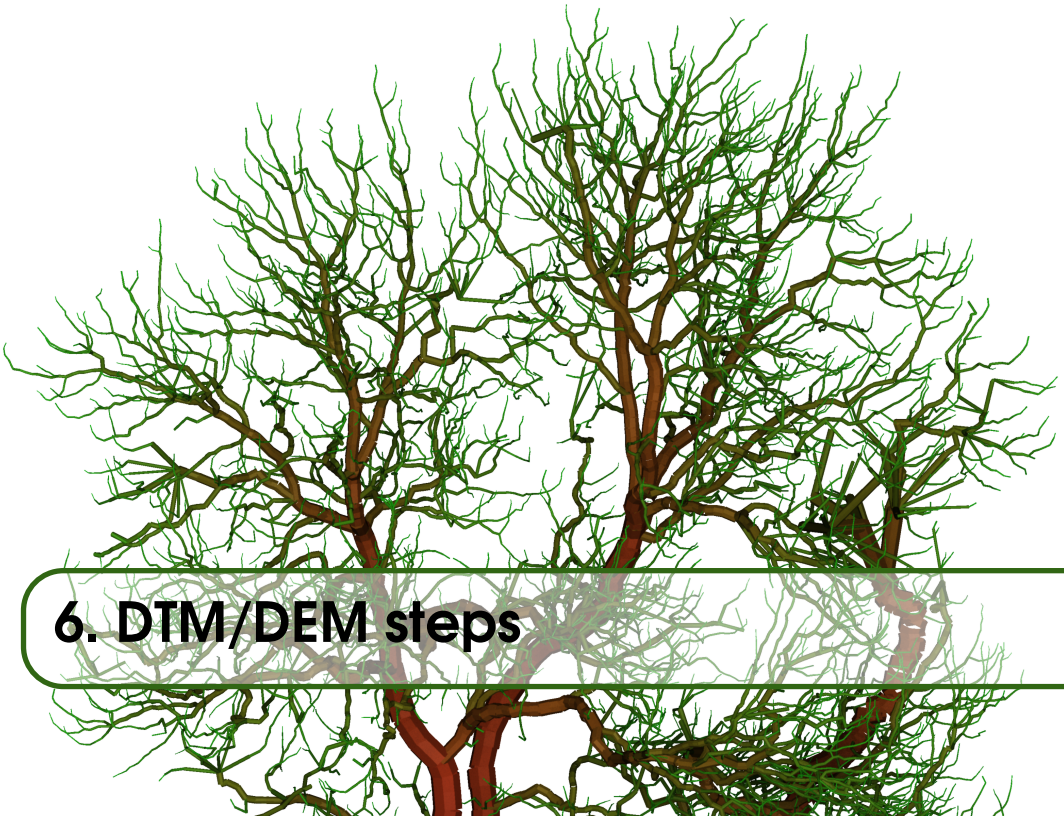
Input — Tree cloud - A point cloud of a segmented tree.

Input — QSM - A qsm produced with this cloud.

Output — ClusterID - A vector storing the cluster ID for each point of the tree cloud.

5.4.4 Description

The step imports a tree cloud and a QSM as input. Each point of the cloud is alligned to its closest cylinder. First the cylinder inherits the growthlength of the cylinder as a temporary parameter. The points are then sorted by their growthlength and then split into **number of Clusters**. The points with the largest growthLength are stored in the first cluster, the point with the smallest in the last.



6. DTM/DEM steps

Here steps are described which produce a digital terrain model (DTM) or a digital elevation model(DEM).

6.1 Dtm Pyramidal Mlesac Fit

6.1.1 Screenshots

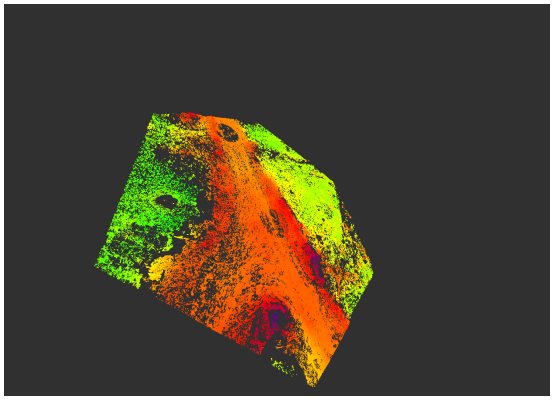


Figure 6.1: The ground points colored by height.

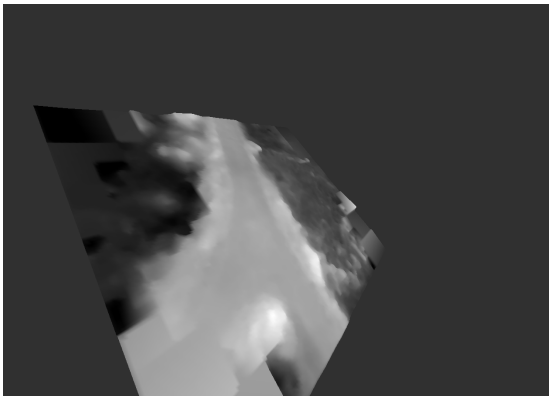


Figure 6.2: The Digital terrain model colored grayscale by height.

6.1.2 Step Placement

- Rasters/Images
  - Digital Elevation Model

6.1.3 IO

Input — Ground cloud - A point cloud of the ground points.

Output — Dtm - The digital terrain model of the ground.

Output — Ground points - A point cloud of the ground points.

6.1.4 Description

The step imports a ground cloud as input. The cloud is downscaled with the voxel grid downscale routine from the PCL library [10]. The **voxel size** of the downscale routine is a user parameter - standard value is 5 (*cm*).

For each downscaled point the normal is computed with a user given **search range**. This **search range** has to be at minimum 2-3 times larger than the **voxel size** with default value 20 (*cm*).

A digital terrain model (Dtm) is then generated and will receive a chosen **cell size** - standard value is 20 (*cm*). The **cell size** is supposed to be in the order of **search range**, but can also be larger. A smaller value will not improve the accuracy of the Dtm.

First into all points a Mlesac [12] plane is fitted. Then the area is split into four even sized squares, so are so points subdiveded into four clusters. Each cluster receives another Mlesac plane fit. The new plane is accepted is the angle between the new plane's normal and the old plane's normal does not deviate more than **angle**. Otherwise the plane coefficient of the subdivision are inherited from the parent division. This routine is repeated iteratively until a subdivision size smaller or equal **cell size** is reached.

The small subdivision build a raster and each raster cell receives the height value of the plane's center point.

All subdivisions are then median filtered. Lastly a new raster is build with the exact **cell size**. The cell values are retrieved via an Inverse Distance Weighted interpolation.

7. QSM steps

Here steps are described which fit or improve a QSM.

7.1 QSM Spherefollowingbasic

7.1.1 Screenshots

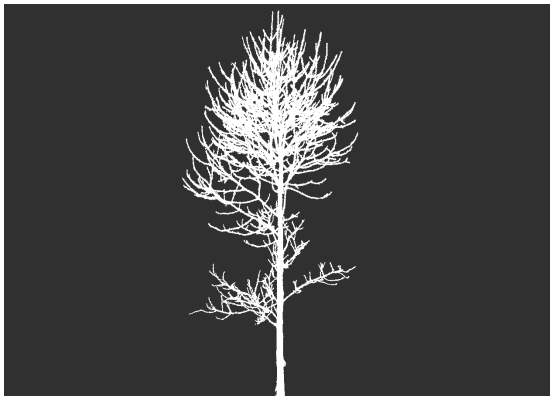


Figure 7.1: The input points stored in a single cloud.

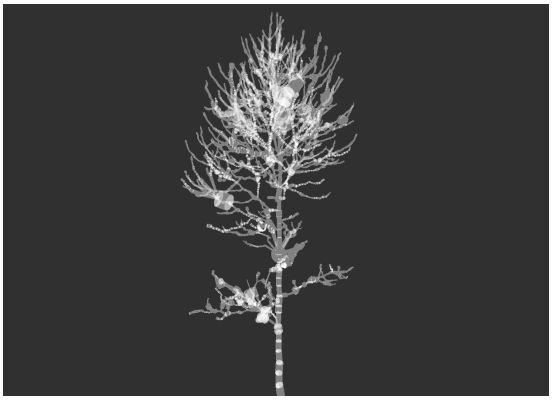


Figure 7.2: The QSM without postprocessing improvements.

7.1.2 Step Placement

- 3D geometry
  - QSM

7.1.3 IO

Input Input cloud - A denoised point cloud of a tree.

Output QSM Cylinder - The QSM stored in Computree format.

Output SphereFollowing QSM - An internal QSM structure used to import into other SimpleForest steps.

Output SphereFollowing parameters - The optimized sphereFollowing parameters.

7.1.4 Description

The method is used to fit cylinders into a de-noised point cloud and to build the tree model. Spheres are utilized to follow the branching structure of the tree from the root to its tips [7, 8, 9].

The cloud is downscaled with the voxel grid downscale routine from the PCL library [10]. The **voxel size** of the downscale routine is a user parameter. Additionally only the largest cluster of an euclidean clustering routine with **clustering range** to be set will be processed. This improves the parameter optimization, as non reachable points will not effect the cloud to model distance.

The algorithm:

A sphere with a center point on a skeleton axis cuts the point cloud. All points within a distance of **sphere Epsilon** to the sphere-surface are considered to be used to detect the next sphere and are put into a sub point cloud P_{sub}.

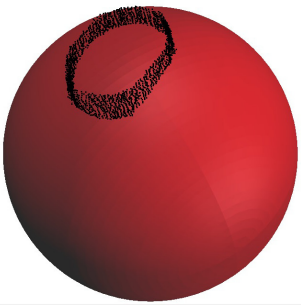


Figure 7.3: P_{sub} on the sphere surface.

P_{sub} is clustered then with **euclidean clustering distance** into i clusters P_i, sorted by their number of points decreasingly. Each cluster represents a cross-sectional area of the stem/branch.

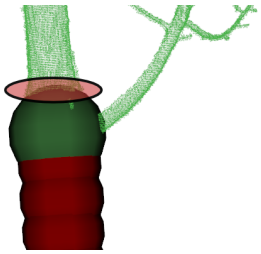


Figure 7.4: The first cluster.

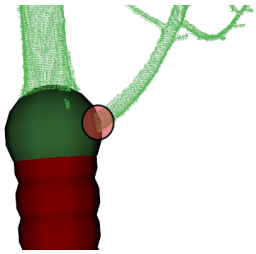


Figure 7.5: The second cluster.

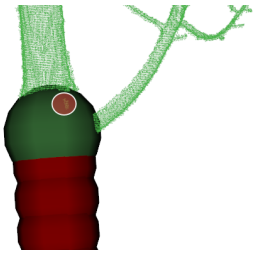


Figure 7.6: The third cluster.

A circle is fitted with one of the following methods

- **Gauss Newton Least Squares**
- **Random Sample Consensus (RANSAC)** [5]
- **Maximum Likelihood Estimator SAmple Consensus (MLESAC)** [12]
- **M-estimator SAmple Consensus (MSAC)** [12]
- **Randomized M-estimator SAmple Consensus (RMSAC)** [12]
- **Progressive Sample Consensus (PROSAC)** [1]
- **Randomized RAndom SAmple Consensus (RRANSAC)** [2]
- **Least Median of Squares (LMEDS)** [6]

into P_i, if the number of points in P_i exceeds **minPts**. The **inlier distance** for the method is set as a user parameter as well as the number if **iterations** of this routine.

The center point of the circle, the center point of the sphere and the circle radius are chosen as cylinder parameters.

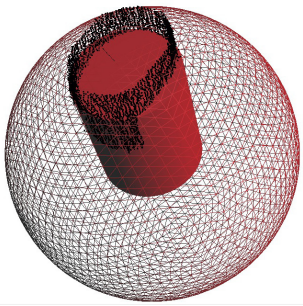


Figure 7.7: The detected cylinder.

The circle is enlarged with **sphere multiplier** and transformed to a three dimensional sphere, but the sphere radius is never allowed to be smaller than **min global radius**.

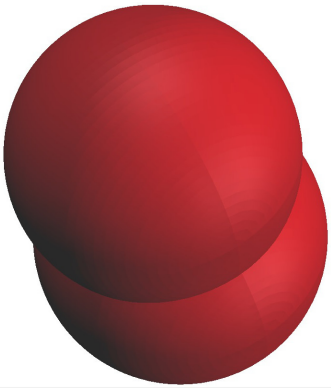


Figure 7.8: The next search sphere.

The procedure is repeated recursively until no more cross sectional areas can be found.

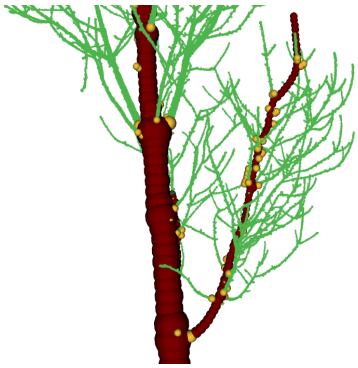


Figure 7.9: The first part of the tree is modelled.

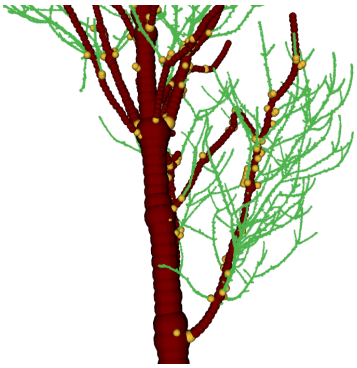


Figure 7.10: More parts of the tree is modelled.

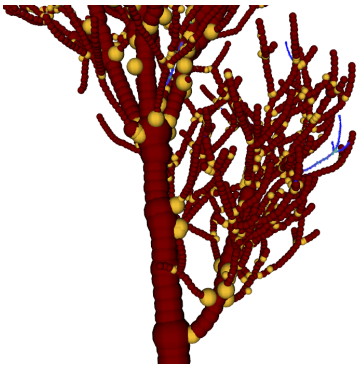


Figure 7.11: Close to the complete tree is modelled.

The algorithm is initialized on a slice at ground height with the thickness **initialization height**.

Three parameters, namely **sphere multiplier**, **sphere epsilon** and **euclidean clustering distance** can be searched for with the **auto parameter** search [9]. Additionally all three parameters are optimized internally. Three vectors are choosen which contain percentage numbers. The named parameters are multiplied with the percentage numbers and each potential parameter combination is tested.

The best model is chosen via the cloud to model distance. The parameter set with the smallest cloud to model distance is chosen after parameter optimization. The following options to compute the cloud to model distance are available (**distance method**):

- **Second Momentum Order** - The distance of each point to the model is squared. This method is the most accurate for perfect point clouds, but also the least robust.
- **Second Momentum Order MSAC** - Same as Second Momentum Order, but here the distance is cropped at a maximum value named **crop distance**. This allows the algorithm to ignore smaller branches better and gain some robustness.
- **First Momentum Order** - The point distances are not squared and simply summed up. More robust than the above ones.
- **First Momentum Order MSAC** - Same as First Momentum Order, but again the distances are cropped.
- **Zero Momentum Order**. Here the number of inlier points of the input cloud to the model is counted. An inlier is a point closer than **crop distance**. This method maximises the number of inliers in contrast to the other minimization methods and is the most robust one.

7.2 QSM SpherefollowingAdvanced

7.2.1 Screenshots

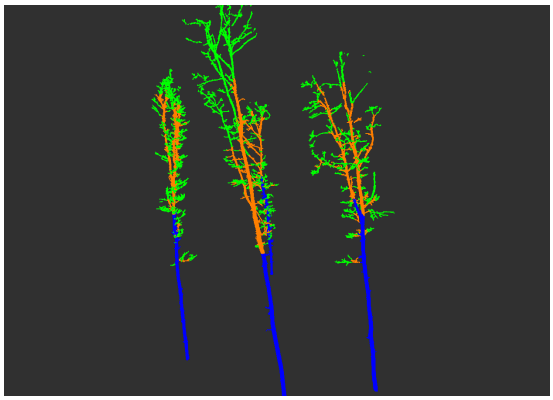


Figure 7.12: The input points clustered by their branch order.

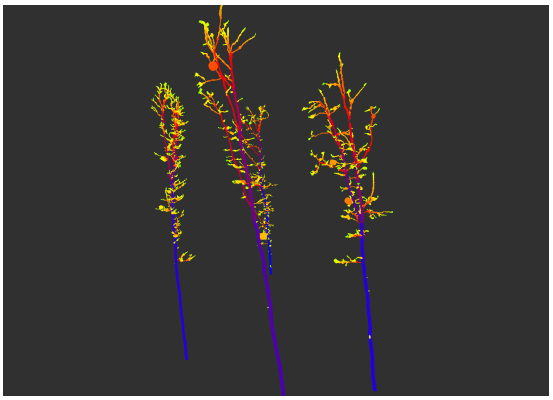


Figure 7.13: The QSM without postprocessing improvements.

7.2.2 Step Placement

- 3D geometry
  - QSM

7.2.3 IO

Input — Result SphereFollowing Parameters - Parameters computed with SphereFollowing Basic, section 7.1.

Input — Input Cloud - The input cloud.

Input — Input Cluster ID - A vector storing for each point of Input Cloud its cluster ID computed with QSM based tree clustering, section5.4.

Output — QSM Cylinder - The QSM stored in Computree format.

Output — SphereFollowing advanced QSM - An internal QSM structure used to import into other SimpleForest steps.

Output — SphereFollowing advancedparameters - The optimized sphereFollowing parameters.

7.2.4 Description

The method is used to fit cylinders into a de-noised point cloud and to build the tree model. Spheres are utilized to follow the branching structure of the tree from the root to its tips [7, 8, 9].

The input tree cloud has been preprocessed with QSM based tree clustering, section 5.4. This routine did split the isolated tree cloud into n clusters. The first cluster contains the stem and major branches, the last cluster the twig points. Clusters in between represent the intermediate branching structure ordered from close to the stem to close to the twigs. Exemplarily we use $n = 3$ in the following description.

The optimized parameters **Result SphereFollowing Parameters** computed from SphereFollowing Basic, section 7.1, utilizing the same but unclustered input cloud are imported as well.

The preprocessing with the voxel grid downscale and the euclidean clustering routine uses simply the imported parameters and those cannot be modified.

Then only the first cluser, e.g. the stem and major branches is used for modelling. An optimization routine starts with the imported method parameters and optimizes **sphere Epsilon**, **euclidean clustering distance** and **sphere multiplier** - all described in SphereFollowing Basic 7.1 - only for those stem points. After the optimization finishes the second cluster is added to the first cluster. A new modelling run starts for the extended cloud using the optimized parameters from the first run for both the first and the second cluster as initial parameters. Yet **sphere Epsilon**, **euclidean clustering distance** and **sphere multiplier** are optimized here differently for the first and the second cluster. Then the last cluster is added, e.g. the full tree is modelled. The stem cluster receives the stem parameters from the second run and both the intermediate and the twig cluster receive the parameters of the second cluster from the second run. All three clusters receive different optimized parameters in the end. In our case with three clusters we have optimzied 3x3, e.g. 9 parameters. This leads to an improved QSM quality in comparison to the QSM Spherefollowingbasic, section 7.1, which only optimizes one set of 3 parameters. The downside is that the calculations take more time, as we optimize a higher parameter space. We can reduce the compuation time by either enlarging the convergence cireterium of the down hill simplex **min size**. The **min size** roughly represents how much parameters change from one to the next iteration. Or we simply limit the maximal **number if iterations**.

The best model is chosen via the cloud to model distance [9]. The parameters are also imported from **Result SphereFollowing Parameters** with the following options set in the exection of SphereFollowing Basic, section 7.1:

- **Second Momentum Order** - The distance of each point to the model is squared. This method is the most accurate for perfect point clouds, but also the least robust.
- **Second Momentum Order MSAC** - Same as Second Momentum Order, but here the distance is cropped at a maximum value named **crop distance**. This allows the algorithm to ignore smaller branches better and gain some robustness.
- **First Momentum Order** - The point distances are not squared and simply summed up. More robust than the above ones.
- **First Momentum Order MSAC** - Same as First Momentum Order, but again the distances are cropped.
- **Zero Momentum Order**. Here the number of inlier points of the input cloud to the model is counted. An inlier is a point closer than **crop distance**. This method maximises the number of inliers in contrast to the other minimization methods and is the most robust one.

7.3 Dijkstra Modelling

7.3.1 Screenshots

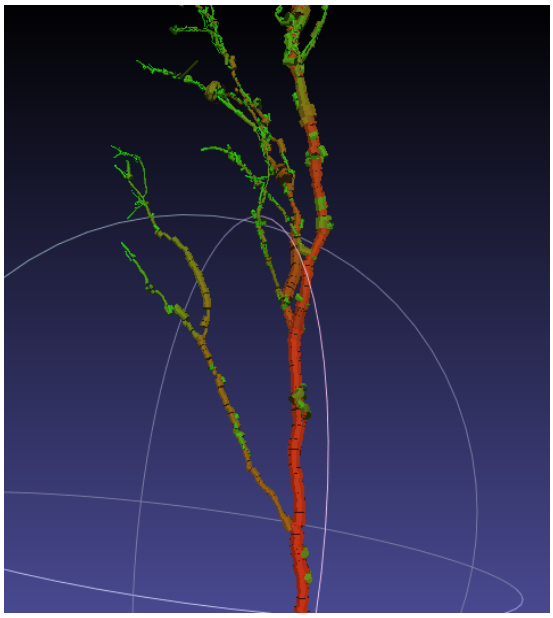


Figure 7.14: A result cylinder model with unimproved cylinders visualized in meshlab.

7.3.2 Step Placement

- 3D geometry
  - QSM

7.3.3 IO

Input — Input cloud - A denoised point cloud of a tree.

Output — QSM Cylinder - The QSM stored in Computree format.

Output — Dijkstra QSM - An internal QSM structure used to import into other SimpleForest steps.

7.3.4 Description

The method is used to fit cylinders into a de-noised point cloud and to build the tree model. The method relies on Dijkstras algorithm. The algorithm parameters are estimated internally via an optimization method, which can be adapted by the user.

The cloud is downscaled with the voxel grid downscale routine from the PCL library [10]. The **voxel size** of the downscale routine is a user parameter. Additionally only the largest cluster of an euclidean clustering routine with **clustering range** to be set will be processed. This improves the parameter optimization, as non reachable points will not effect the cloud to model distance.

Aft

After a skeleton is build with the dijkstra algorithm [3, 4] each skeleton unit gets its nearest input points allocated to build a subcloud P_i. A cylinder is fitted with one of the following SACModel methods

- **Random Sample Consensus (RANSAC)** [5]
- **Maximum Likelihood Estimator SAmple Consensus (MLESAC)** [12]
- **M-estimator SAmple Consensus (MSAC)** [12]
- **Randomized M-estimator SAmple Consensus (RMSAC)** [12]
- **Progressive Sample Consensus (PROSAC)** [1]
- **Randomized RAndom SAmple Consensus (RRANSAC)** [2]
- **Least Median of Squares (LMEDS)** [6]

into P_i, if the number of points in P_i exceeds **minPts**. The **inlier distance** for the method is set as a user parameter as well as the number if **iterations** of this routine.

The dijkstra parameters are chosen in an optimization routine [9]. The qsm is here modelled muliple times.

The best model is chosen via the cloud to model distance. The parameter set with the smallest cloud to model distance is chosen after parameter optimization. The following options to compute the cloud to model distance are available (**distance method**):

- **Second Momentum Order** - The distance of each point to the model is squared. This method is the most accurate for perfect point clouds, but also the least robust.
- **Second Momentum Order MSAC** - Same as Second Momentum Order, but here the distance is cropped at a maximum value named **crop distance**. This allows the algorithm to ignore smaller branches better and gain some robustness.
- **First Momentum Order** - The point distances are not squared and simply summed up. More robust than the above ones.
- **First Momentum Order MSAC** - Same as First Momentum Order, but again the distances are cropped.
- **Zero Momentum Order**. Here the number of inlier points of the input cloud to the model is counted. An inlier is a point closer than **crop distance**. This method maximises the number of inliers in contrast to the other minimization methods and is the most robust one.

7.4 QSM Median Filter

7.4.1 Screenshots

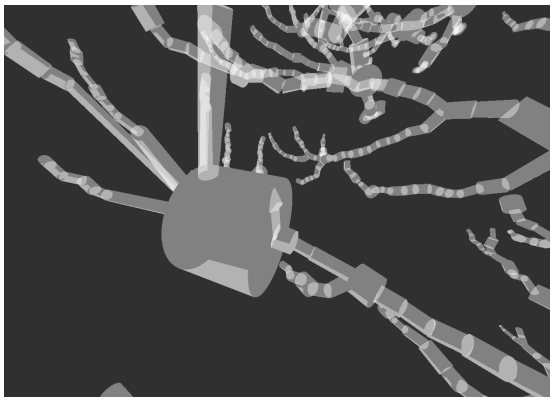


Figure 7.15: The unimproved QSM in close up view.

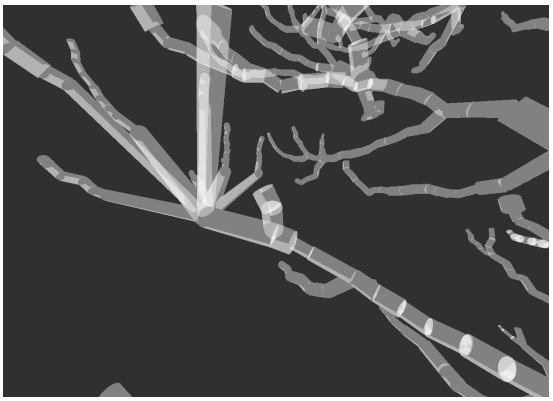


Figure 7.16: The improved QSM in close up view.

7.4.2 Step Placement

- 3D geometry
  - QSM

7.4.3 IO

Input — Internal QSM - An internal QSM structure used to import into other SimpleForest steps.

Output — QSM Cylinder - The QSM stored in Computree format.

Output — QSM sphereFollowing median filtered - An internal QSM structure used to import into other SimpleForest steps.

7.4.4 Description

The method imports a QSM. The radii of the QSM's cylinders are filtered via median filtering. A list of child, grandchild, grandgranchild, parent, grandparent and grandgrandparent with the cylinder itself is generated. The median radius of this list is computed. If the cylinder's radius deviates more than **percentage** of the median radius to this median radius it will receive the median radius as a new radius.

7.5 QSM Correct Shoots

7.5.1 Screenshots

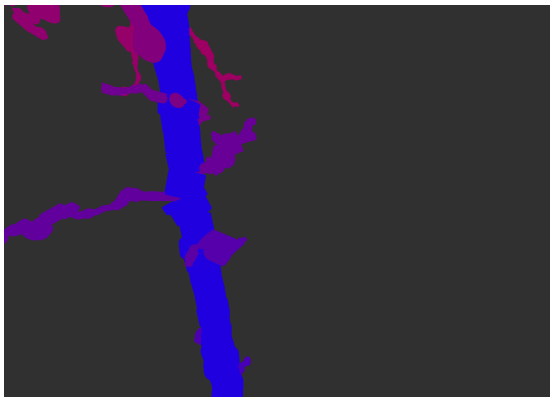


Figure 7.17: The input QSM.

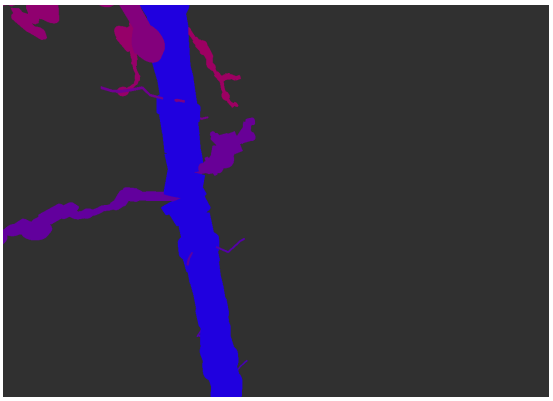


Figure 7.18: The QSM with corrected shoots.

7.5.2 Step Placement

- 3D geometry
  - QSM

7.5.3 IO

Input — Internal QSM - A precomputed QSM.

Output — QSM Cylinder - The QSM stored in Computree format.

Output — Shoot corrected QSM - An internal QSM structure used to import into other SimpleForest steps.

7.5.4 Description

Often shoots have a largely overestimated radius. This method detects all shoots growing out of the stem.

All branches growing out of the stem are analyzed. If the analyzed branch does not split at all, it is detected as a shoot. If **apply to second order branches also** is checked, then it might split exactly but not more than one time to be detected as a shoot.

All detected shoots get a new user choosen as **correction radius**.

7.6 QSM Allometric Correction

7.6.1 Screenshots

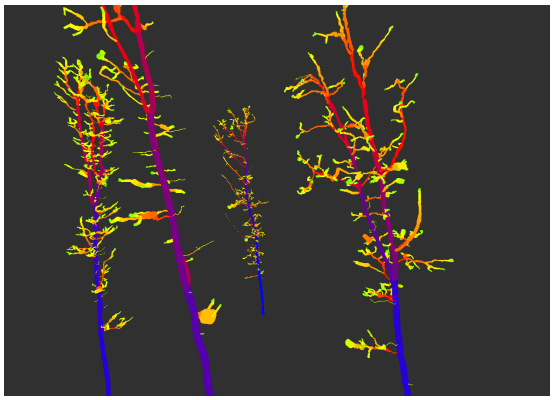


Figure 7.19: The input QSM.

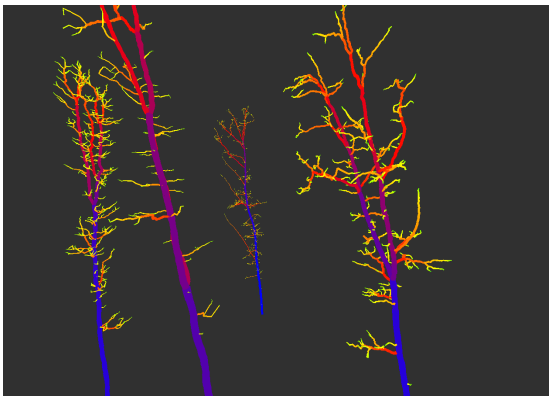


Figure 7.20: The QSM allometric corrected.

7.6.2 Step Placement

- 3D geometry
  - QSM

7.6.3 IO

Input — Internal QSM - an input QSM.

Output — QSM Cylinder - The QSM stored in Computree format.

Output — Allometric Corrected QSM - An internal QSM structure used to import into other SimpleForest steps.

7.6.4 Description

The method corrects under or overestimated radii among the complete QSM [9].

A nonlinear model of the form $y = a * x^b$ is build with y being the radius and x is a user chosen **growth parameter**, either growthVolume or growthLength. If a cylinders radius is outside of the **correction range** it will be corrected to the predicted radius y.

The power model is build as described, here with the growthVolume as an example:

Each cylinder has a radius and a growthVolume as can be seen in figure 7.21.

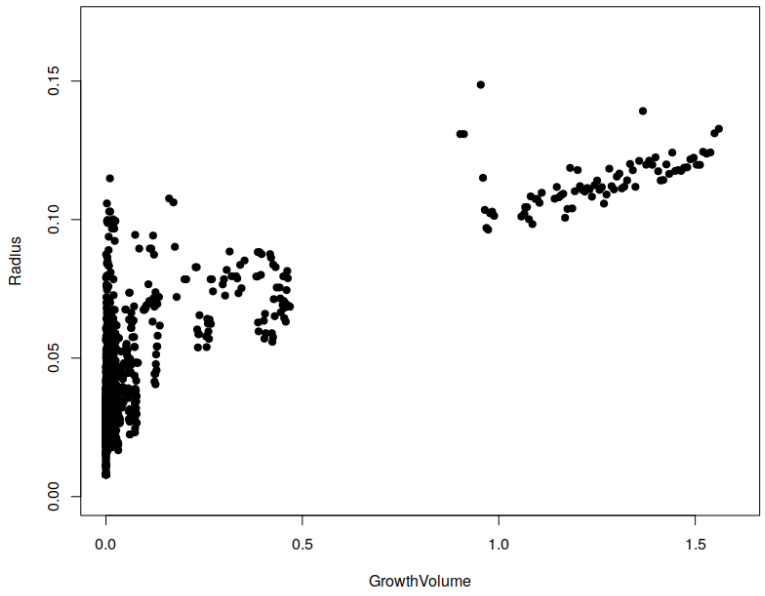


Figure 7.21: Radius and growthVolume for each cylinder of a QSM.

Only the radii estimated by a modelling routine are acounted to. Radii corrected in a postprocessing routine such as the QSM Median Filter, section 7.4 are ignored, see figure 7.22.

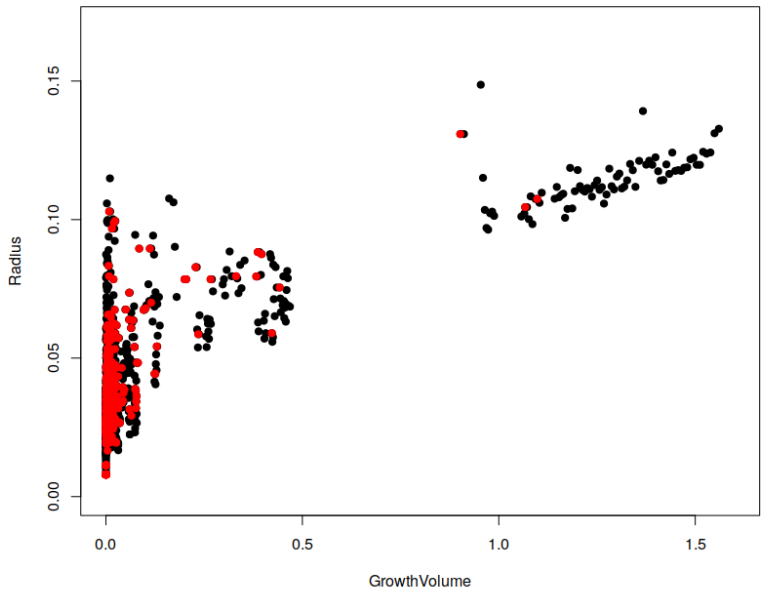


Figure 7.22: Postprocessed cylinders are ignored, here colored in red.

We do not want to use cylinders near the twigs or near the root for the prediction. Also cylinders having a too small or too large growthVolume are ignored, see figure 7.23.

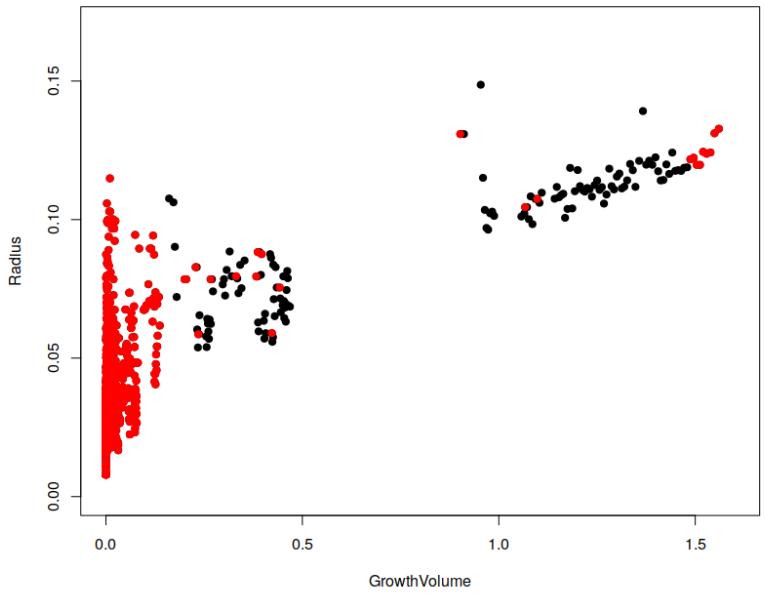


Figure 7.23: Postprocessed cylinders are ignored as well as root or twig cylinders, here colored in red.

All other cylinders are used to make a power fit, see figure 7.24.

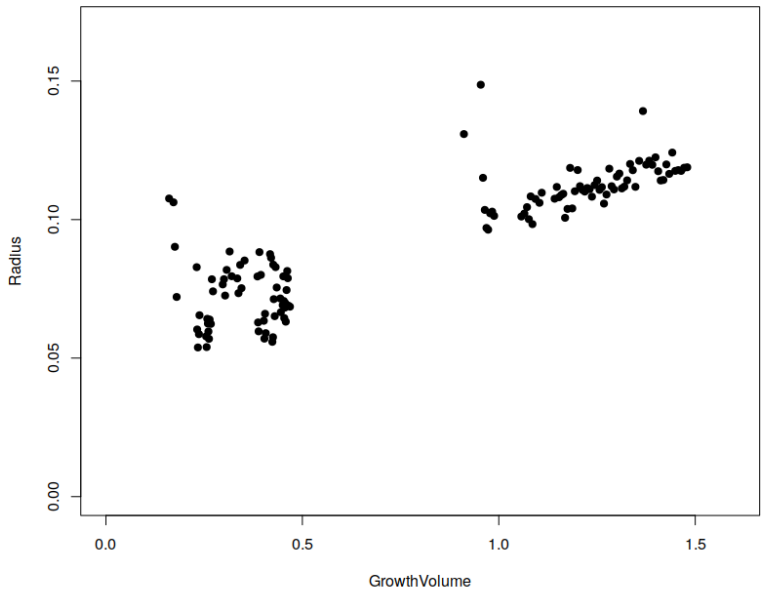


Figure 7.24: Cylinders used to build the power model.

After the prediction is finished wrongly fitted cylinders can be corrected and the result plot can be seen in figure 7.25.

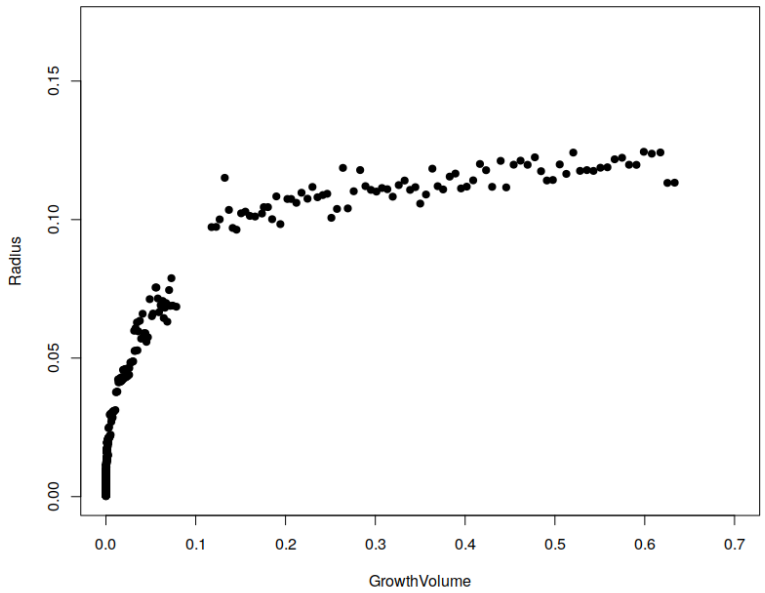


Figure 7.25: Corrected QSM.

If twigs have been lost during denoising a minimum radius r can be searched for by enabling the switch **estimate average radius**. In that case the equation is changed to $y = a * x^b + r$.

If the prediction of the model fails, the algorithm can fall back on a user given **power parameter** b, The algorithm can fail for example if less than **minimum number of measurements** are used to build the model.

7.7 QSM Allometric Correction Manual

7.7.1 Screenshots

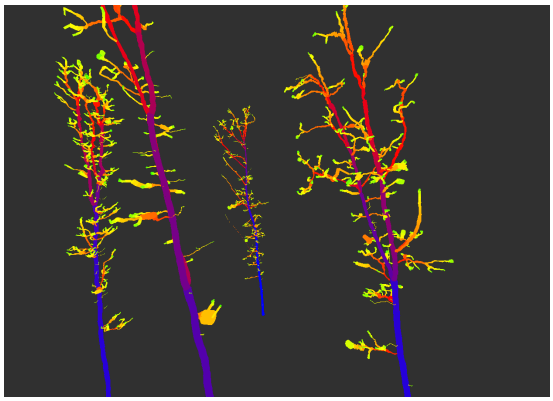


Figure 7.26: The input QSM.

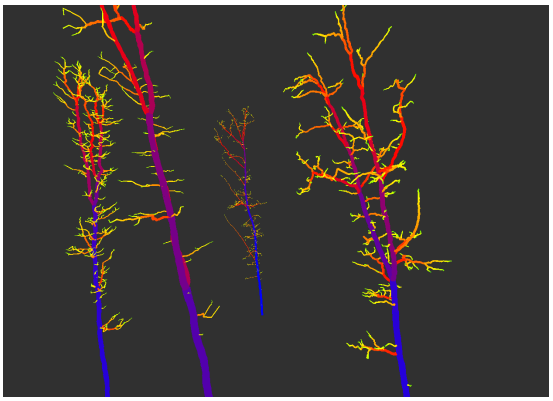


Figure 7.27: The QSM allometric corrected.

7.7.2 Step Placement

- 3D geometry
  - QSM

7.7.3 IO

Input — Internal QSM - an input QSM.

Output — QSM Cylinder - The QSM stored in Computree format.

Output — Manual allometric Corrected QSM - An internal QSM structure used to import into other SimpleForest steps.

7.7.4 Description

The method corrects under or overestimated radii among the complete QSM. It is aimed to do the same correction as 7.6, but the power parameters are user chosen and not estimated automatically.

A nonlinear model of the form $y = a * x^b$ is imported with y being the radius and x is a user chosen **growth parameter**, either growthVolume or growthLength. The user has to estimate the parameters **a** and **b** externally utilizing statistical software like R with the uncorrected QSM as user input. If a cylinders radius is outside of the **correction range** it will be corrected to the predicted radius y. The predicted radius can never be smaller than **min Radius**.

7.8 QSM Reverse Pipe Model Filter

7.8.1 Screenshots

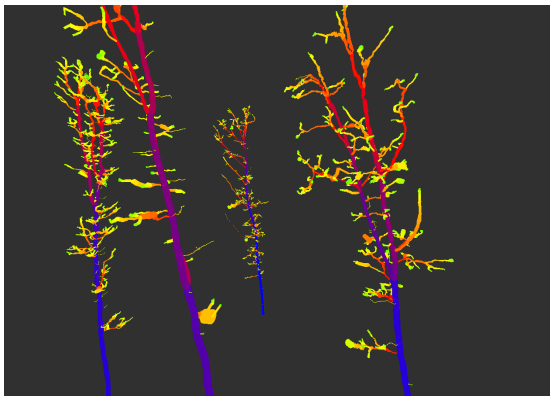


Figure 7.28: The input QSM.

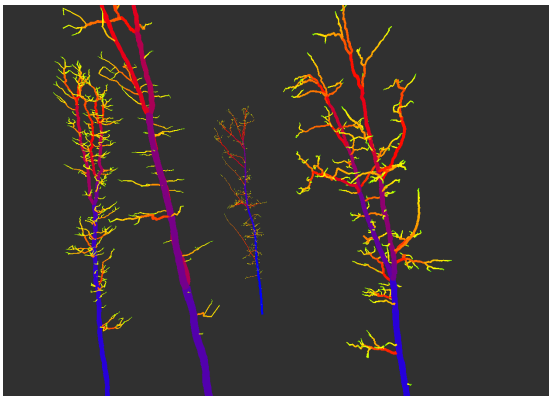


Figure 7.29: The QSM pipe model corrected.

7.8.2 Step Placement

- 3D geometry
  - QSM

7.8.3 IO

Input — Internal QSM - an input QSM.

Output — QSM Cylinder - The QSM stored in Computree format.

Output — Reverse Pipe Model Filtered QSM - An internal QSM structure used to import into other SimpleForest steps.

7.8.4 Description

The method corrects under or overestimated radii among the complete QSM. According to the pipemodel theory the area of a branch segment before a branch junction equals the summed area of all the child segments. As the area at twig level is supposed to be equal for all twigs, the area of a segment can be seen as the sum of all supported twig areas. Therefore the radius of a segment has to correlate with the squared root of the number of supported twigs.

We defined in section 1.2 the reverse pipe branch order, which is exactly the squared root of the number of supported twigs.

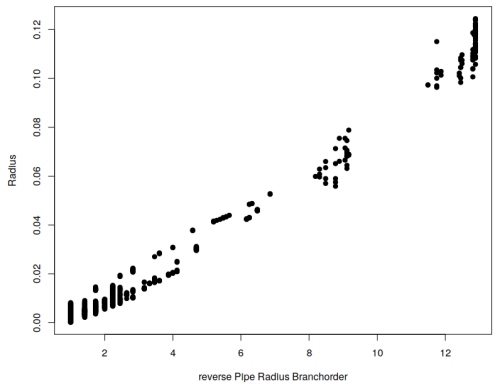


Figure 7.30: We can see a strong linear relation between reverse pipe radius Branchorder and the radius.

A linear model of the form $y = a * x$ is imported with y being the radius and x is the reverse pipe radius branch order. If a cylinders radius is outside of the **correction range** it will be corrected to the predicted radius y. The predicted radius can never be smaller than **min Radius**.

Part Three Evaluation

7.9 Evaluation

7.9.1 Results

For version 5.1.1 I recently denoised all 36 trees published in Hackenberg 2015a [8]. I did then run the modeling with the SphereFollowing method 7.2:

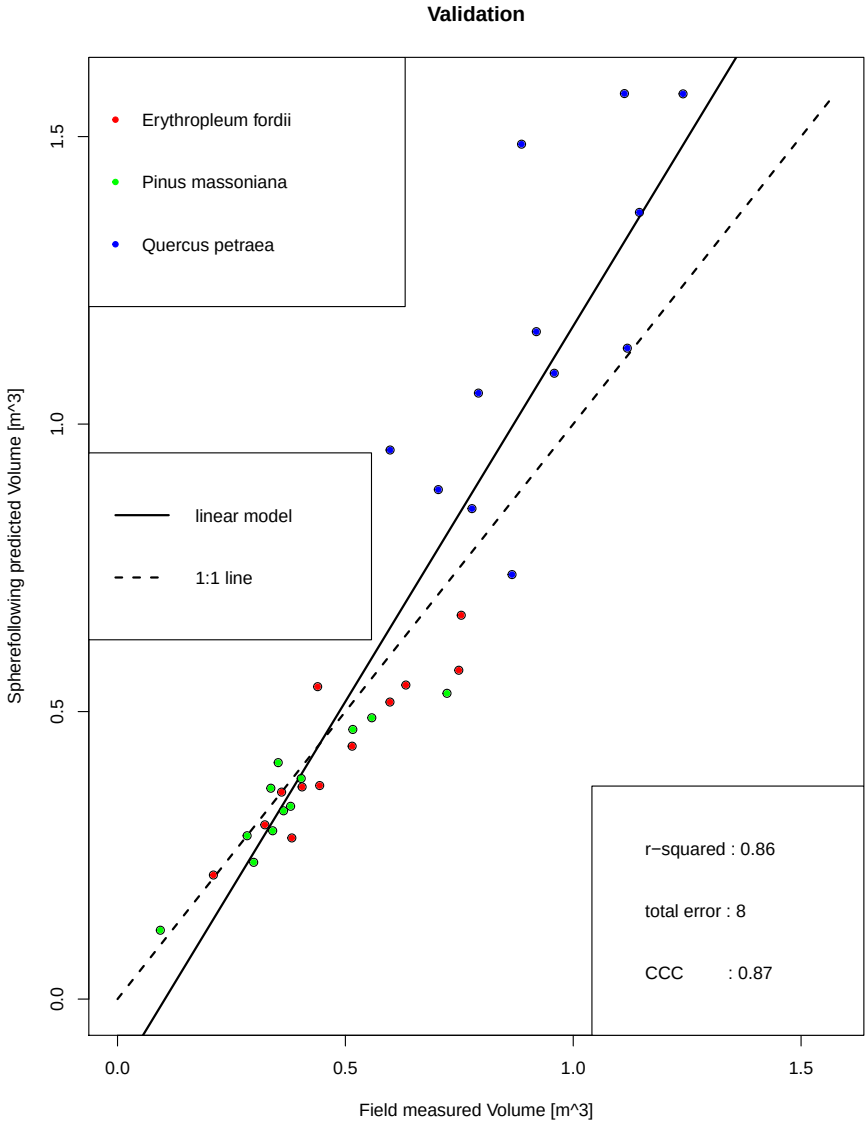


Figure 7.31: Evaluation of the Spherefollowing method.

I will work on a peer-reviewed publication with hopefully extended and improved results.

7.9.2 Interpretation

The results are good, but worse than those in in Hackenberg 2015b [9]. This might have several reasons:

- While the sphere following method can be parameterized in SimpleForest, the algorithm parameters have been estimated fully by SimpleForest and not adapted by myself.
- I did not use those tree clouds and their field data during development of the plugin, neither

did I observe this data during denoising and preparation of the clouds. I simply might have done less overfitting and provide more "true" results. Please understand that I did not fake results by purpose in the named publication, but I might have done this a bit being unaware of an overfitting potential.

- The current version can model a lot more clouds than the old software was capable. As I constantly got access to user data, most times of lot lower quality, I did adapt my method to perform better on clouds of different quality. Making an algorithm in that sense more generic comes with the downside of having less accurate results in the spezialized case.
- I did not use intensity values while denoising. This is a clear limit in denoising capabilities. Yet I did not want to rely on a feature which varies a lot from scanner to scanner and not all cloud data has even intensity feature. Again, more generic means less accurate.

You should always download the data and play around with my evaluation script available at simpleforest.org. Compare visually the quality of your own data to decide how much you can rely on my evaluation.

7.9.3 Recommondatation

While there is multiple ways to produce a QSM I give you the following guideline:

- The spherefollowing method 7.2 is more robust and accurate in comparison to the Dijkstra based method 7.3. I also evaluated Dijkstra method locally, and two out of 36 trees did fail completely. For the remaining 34 tress the results have been worse.
- You can already achieve good results with the basic spherefollowing step 7.1. Putting on top the spherefollowing advanved step 7.2 most likely will improve your results. But it does not have to do so in every case. For low quality clouds I observed this one to perform less accurate. For high resolted clouds this was not observed, but the modelling time is between 5-20 times longer. If you cannot efford to use up so much time, ignore the advanced step.
- I still recommmend to use the Dijkstra modelling step 7.3 in the following scenario. You have a huge set of segmented clouds and want to rely on full automatism without visually checking the QSMs' visually. You can also efford to drop some of your clouds. For example you have 1000 clouds of one species, your task is to make an allometric model and you are satisfied with also having only 900 trees here. You can in such a case run both algorithms and just stick to those QSMs which have similar volume measures. Having two comparable results originating from different algorithms can be used as a full automatic validation check for the quality. Then you should rely on the first point in that list and use the spherefollowing result.
- For coniferous trees you should ignore the branches. Denoising is really difficult and left over branches seem to be of such a low quality that the errors outweigh the benefit here.

IV

Part Four FAQ

7.10 FAQ

- **I struggle with setting up a pipeline**: Read this user guide and watch my video tutorials. I also have example scripts at simpleForest.org. If this does not help, please put your question to the Computree forum.
- **I run your pipeline, but it crashes**: please check if this happens during the QSM export step. Generating ply models can sometimes crash the whole pipeline. It seems to happen randomly and I was not able to fix this yet. You can uncheck the ply generation in that case, but if you do so, you cannot review your results in cloudcompare.
- **I run your pipeline, but it crashes and not in the ply export**: You can do a bug report in the Computree forum. Even better would be to create an issue on my Gitlab.

V

Part Five Change Log

7.11 Version 5.2.2
7.12 Version 5.2.1
7.13 Version 5.1.4
7.14 Version 5.1.3

7.11 Version 5.2.3

- Improved the optimizer related cloud to model distance. The quality of the qsms should be smoother now. That means the fitting quality improved. Runtime was also improved by about 2 percent.
- Made QSM SpherefollowingAdvanced 7.2 inherit the cloud to model distance parameters from its basic counterpart.
- Implemented a Gauss Newton circle fit in the least squares sense.

7.12 Version 5.2.2

- Five percent runtime improvement SphereFollowing method.
- Uploaded an R-script in a new repository and added to this handbook a statistic section 2.1.

7.13 Version 5.2.1

- Implemented a new cloud filter, the Filter By Min Height 4.8.
- Adjusted an internal routine of the DTM modeling 6.1. DTMs of plots with terrain changing the slop frequently within the plot should be modelled better/correctly.
- Fixed a bug in the RANSAC stem filter 4.7 leading to a crash in rare occurences.
- Minor code improvements.

7.14 Version 5.1.4

- Fixed a bug in the Dtm Pyramidal Mlesac Fit 6.1. An internal threshold prevented the correct modeling of ground with a decent amount of curvature. The threshold has been removed and the modeling should be now correct also for non flat ground.
- Fixed a bug in the Stem Filter RANSAC 4.7. This step could crash under certain circumstances and the cause of error is now removed.

7.15 Version 5.1.3

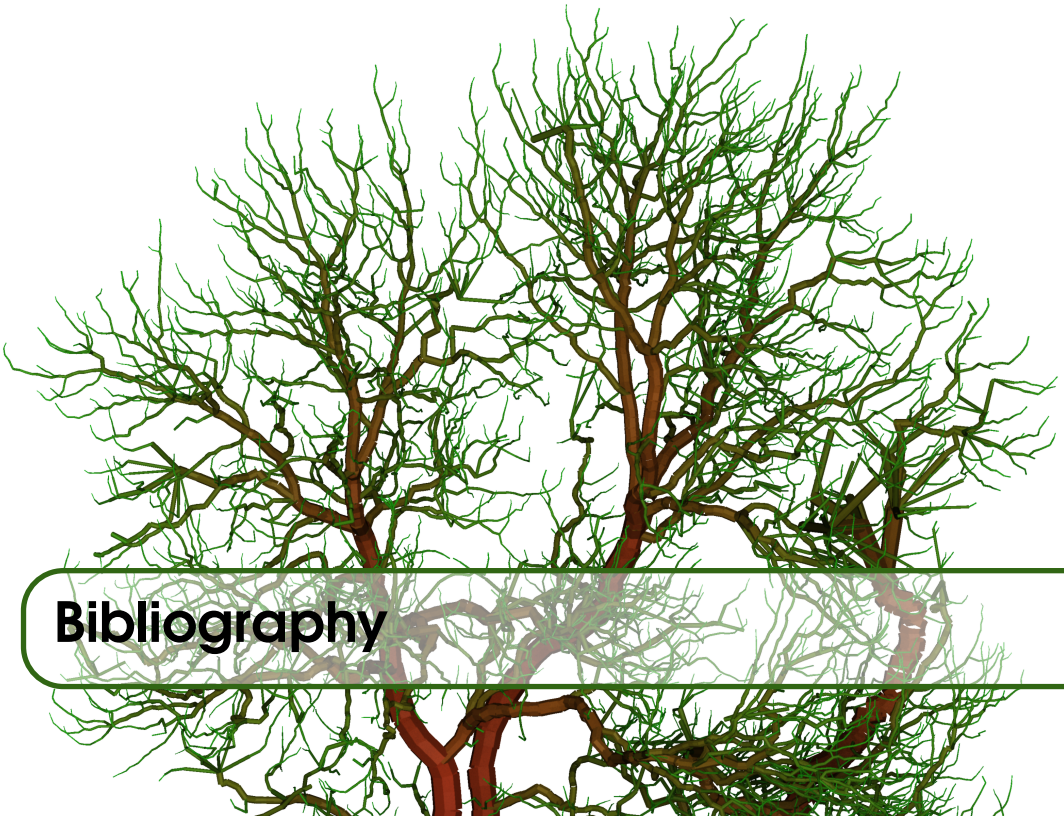
- Fixed a bug in QSM SpherefollowingAdvanced 7.2. This bug made results of the method worse and now the quality is improved.
- Fixed a bug in the ply generation of the export step. DTM is not modelled correctly. Also added a smoothing step to the QSM ply generation.
- Added a new stem filter, the Stem Filter RANSAC 4.7.
- Improvement of QSM Allometric Correction 7.6. For clouds where the twigs were lost during denoising, a switch was added to automatically estimate the average diameter of the branches at the point where they have been cut.
- Various code quality improvements.

VI

Part Six Citations

Bibliography **62**
Articles

Index **65**



Bibliography

Articles

[1] Ondrej Chum and Jiri Matas. “Matching with PROSAC-progressive sample consensus”. In: 1 (2005), pages 220–226 (cited on pages 38, 44).

[2] Ondrej Chum and Jırı Matas. “Randomized RANSAC with Td, d test”. In: 2 (2002), pages 448–457 (cited on pages 38, 44).

[3] Jean-François Côté, Richard Fournier, and Richard Egli. “An architectural model of trees to estimate forest structural attributes using terrestrial LiDAR”. In: *Environmental Modelling & Software* 26.6 (2011), pages 761–777 (cited on page 43).

[4] Edsger W Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische mathematik* 1.1 (1959), pages 269–271 (cited on pages 31, 43).

[5] Martin A Fischler and Robert C Bolles. “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography”. In: *Communications of the ACM* 24.6 (1981), pages 381–395 (cited on pages 38, 44).

[6] Andrea Fusiello. “Elements of geometric computer vision”. In: *homepages.inf.ed.ac.uk* (2006) (cited on pages 38, 44).

[7] Jan Hackenberg et al. “Highly accurate tree models derived from terrestrial laser scan data: A method description.” In: *Forests* 5 (2014), pages 1069–1105 (cited on pages 37, 41).

[8] Jan Hackenberg et al. “Non destructive method for biomass prediction combining TLS derived tree volume and wood density.” In: *Forests* 6 (2015), pages 1274–1300 (cited on pages 37, 41, 55).

[9] Jan Hackenberg et al. “SimpleTree —An Efficient Open Source Tool to Build Tree Models from TLS Clouds.” In: *Forests* 6 (2015), pages 4245–4294 (cited on pages 10, 22–24, 37, 40–42, 44, 47, 55).

[10] Rusu Radu Bogdan and Steve Cousins. “3D is here: Point Cloud Library (PCL)”. In: 6 (May 2011), pages 1–8 (cited on pages 21–26, 29, 30, 35, 37, 43).

[11] Pasi Raumonen et al. “Fast Automatic Precision Tree Models from Terrestrial Laser Scanner Data”. In: *Remote Sensing* 5.2 (2013), pages 491–520. ISSN: 2072-4292. DOI: 10.3390/rs5020491 (cited on page 25).

[12] Philip HS Torr and Andrew Zisserman. “MLESAC: A new robust estimator with application to estimating image geometry”. In: *Computer vision and image understanding* 78.1 (2000), pages 138–156 (cited on pages 35, 38, 44).

Index

B

Branchorder 12

C

Computational metrics 14
Cut cloud above DTM 20

D

Dijkstra Modelling 43
Dikstra Based Tree Segmentation 30
Dtm Pyramidal Mlesac Fit 35

E

Euclidean Clustering Filter 21
Evaluation 55

F

FAQ 58
Filter By Min Height 27

G

Ground Point Filter 22
Growthparameters 10

Q

QSM 8
QSM Allometric Correction 47
QSM Allometric Correction Manual 51
QSM based tree clustering 33
QSM Correct Shoots 46
QSM Median Filter 45
QSM Reverse Pipe Model Filter 52
QSM SpherefollowingAdvanced 41
QSM Spherefollowingbasic 37

R

R-script 15
Radius Outlier Filter 23

S

Segmentation Euclidean Clustering 29
Statistical Outlier Filter 24
Stem Filter 25
Stem Filter RANSAC 26

V

Version 5.1.3 60
Version 5.1.4 60
Version 5.2.1 60
Version 5.2.2 60
Voronoi Based Tree Segmentation 32