

# STRUCTURING LASER-SCANNED TREES USING 3D MATHEMATICAL MORPHOLOGY

Ben Gorte and Norbert Pfeifer

Section of Photogrammetry and Remote Sensing, TU Delft, Kluyverweg 1, 2629HS, The Netherlands  
{b.g.h.gorte,n.pfeifer}@lr.tudelft.nl

Commission V, Working Group 2

**KEY WORDS:** Laser scanning, Reconstruction, Forestry, Segmentation, Virtual Reality

## ABSTRACT:

The task addressed in this paper is 3D modelling and reconstruction of (real world) trees on the basis of terrestrial laser scans. To identify the structure of a tree in terms of stem and branches, an algorithm has been designed in 3D voxel space, based on a selection of basic and advanced 2D raster (image) processing algorithms, transferred into the 3D domain. The selection includes filtering, mathematical morphology, skeletonization, connected component labeling and shortest route computation.

## 1 INTRODUCTION

The purpose of the paper is to analyse point clouds of terrestrial laser scans that are recorded inside a production forest. The purpose is to reconstruct trees in 3D, in order to measure lengths and thicknesses of the tree stem and of the main branches, to estimate the corresponding wood volumes. (In the remainder we will not mention the stem anymore explicitly, but only speak of branches).

Reconstruction of real world trees on the basis of terrestrial laser scans allows to automate the estimation of forest wood mass for valuation purposes. Also to make ecological assessments detailed knowledge of tree structures is considered helpful in forestry.

Several authors report modelling from laser scans, but these are usually made with airborne scanners and provide rough estimates of tree canopies, albeit over large areas [Pyysalo and Hyypäe, 2002, Sun, 2000]. Using 2D video and 2.5D laser pulse range measurements, [Clark, 2002] builds a 3D model of a tree stem by combining multiple vantage points. In our paper we will build the model from 3D measurements directly.

Datasets were provided by the Institute for Forest Growth in Freiburg, Germany. One plot (i.e. one location in a forest) is captured by one or more positionings of the laser scanner. The datasets are oriented relative to each other (i.e. registered). The data is captured with Zoller+Fröhlich laser scanners and provided in the form of point clouds (sequence of  $(x, z, y)$ -triples).

A tree to be reconstructed is scanned from all sides by locating the scanner at a number of viewpoints around the tree and merging the resulting scans into a single point cloud. Because of the nature of laser scanning, it may be expected that the coordinates of the reflected points are located at the surfaces of the branches. Scans are taken during the winter season when the trees are without leaves.

At the final stage of the reconstruction process a cylinder fitting process is carried out for each of the significant branches: a cylinder is fitted through those points that are at the surface of a particular branch. Before this is possible, however, it is necessary to have a rough idea which points belong to that particular branch, in other words to know the initial subset of points that should be involved in the fitting process. This process itself will be able to identify erroneous points (that do not fit) within the initial subset, and possibly to add points to the subset that are in the vicinity and fit on the same cylinder.

The crucial point in the above is to find the initial subset of points

for each branch. This is the problem addressed in this paper. We are presenting an algorithm that subdivides (segments) a laser point cloud of a tree into subsets that correspond to main branches (those that have a significant amount of wood). Besides, points that do not seem to correspond to any main branch are identified as noise.

The algorithm segments a point cloud according to branches and at the same time identifies how the branches are connected to each other. As a result, a "tree" data structure is constructed, which represents the topology of the branches within a tree.

Basically, the task of the algorithm boils down to detecting elongated, cylindrical structures in a point cloud. However, in general point cloud segmentation can be considered a difficult subject, especially in presence of the before-mentioned noise, in addition to gaps within branches (missing points, e.g. caused by occlusions by other branches) and varying point densities (as a result of varying distance from the scanner). To establish the topology between branches on the basis of point clouds is an unsolved problem as well.

Therefore we decided to perform this part of the analysis in the 3-dimensional raster domain. Connectivity and neighborhood relations between voxels in a 3D raster can be established much easier than between the original  $(x, y, z)$ -points, using morphological operations, which are well-known from 2D image processing. Their extension to the 3D raster domain is described here. When defining connectivity in 3D, a choice has to be made between 6, 18 and 26-adjacency, depending on the number of voxels that are considered neighbors of a voxel: only those with a common face, also those with a common edge, or even also the ones with a common corner.

The 3D reconstruction is done in several steps. After converting the point cloud to 3D raster, closing and opening are applied to close gaps and holes, and to remove isolated points, respectively. Next, 3D skeletonisation is applied, which (temporarily) reduces the stem and branches to single voxel thickness. Then, Dijkstra's minimum spanning tree algorithms ensures that the skeleton from the previous step becomes a tree in the data structuring sense. Topology can now be established, in which the stem and the braches obtain unique IDs. Finally, 3D distance transform is applied to assign to each point in the original set the ID of the nearest branch. As a result, the points are grouped by ID according to the tree structure.

Summarizing, the reconstruction process has two phases:

1. Segmentation of laser points into the tree stem and major individual branches (and at the same time removal of points that do not belong to these tree elements). This phase is executed in the 3D raster (voxel) domain, and is described in detail in this paper.
2. Fitting of 3D geometrical primitives, e.g cylinders, to the segmented point sets. This takes place in the  $(x,y,z)$  point cloud domain, where points are labeled on the basis of the results in the first phase. It is described in [Pfeifer and Gorte, 2004].

## 2 VOXEL DOMAIN

Analysis for tree reconstruction from terrestrial laser point clouds described in this paper is performed in the 3-dimensional raster domain, which is a discrete 3-dimensional space with elements called *voxels*.

Conceptually, a 3D raster data set records voxel *values* only, since the voxel *locations* are defined implicitly by the position of a voxel in the data set. This is in contrast to a vector data set, where point locations (coordinates) are recorded explicitly, in addition to values and other kinds of information, such as topology. In our implementation, however, voxels with value 0 (usually the vast majority) are not stored on disk, which in turn requires that locations of non-zero voxels are stored explicitly.

Within a 3D raster different 3-dimensional phenomena can be represented, with different meanings attached to voxel values. In the simplest raster representation of laser points the voxel value range may be limited to 0 (this voxel is empty) and 1 (it contains laser points).

The overall purpose of raster processing during tree reconstruction is to segment the set of points into different trees (if appropriate), and within trees into different branches. At the end of this phase, therefore, each point will have obtained a unique branch identification number. In addition, topological information, describing how branches are connected to each other in terms of ancestors and descendants, will become available during this phase.

## 3 ALGORITHM

The algorithm has the following steps (Fig 1):

1. Point cloud to 3D raster conversion
2. 3D neighborhood operations
3. Skeletonization
4. Skeleton segmentation
5. Connected component labelling
6. Component Separation
7. Raster tree segmentation
8. Point cloud segmentation (in the continuous domain)

### 3.1 Point cloud to 3D raster conversion

During this step a 3-dimensional raster space is created and all the laser points are transformed to voxels in that space. The most important parameter is **spatial resolution**, which denotes the size of a single voxel (in meter). This is the step size that is used for quantization of the 3D space.

The 3D raster space is subdivided in planes, lines and columns. The location of a voxel is defined by a  $(p,l,c)$  coordinate triple,

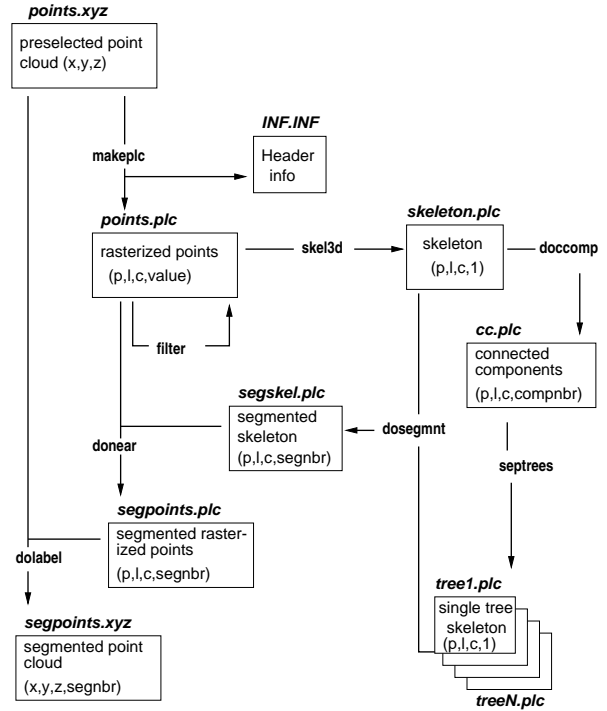


Figure 1: Raster processing work flow. The arrows represent processing steps, labelled by the names of the corresponding routines.

where  $p$ ,  $l$  and  $c$  are integer numbers  $[0, 1, 2, \dots]$ . The number of planes, lines and columns depends on the chosen resolution and on the minimum and maximum  $z$ ,  $y$  and  $x$ -coordinates (respectively) that occur in the laser point cloud (Fig. 2). It is useful to specify the thickness of a layer of empty voxels to surround the whole block, to ensure that neighborhood operators (see below) do not expand outside the voxel space.

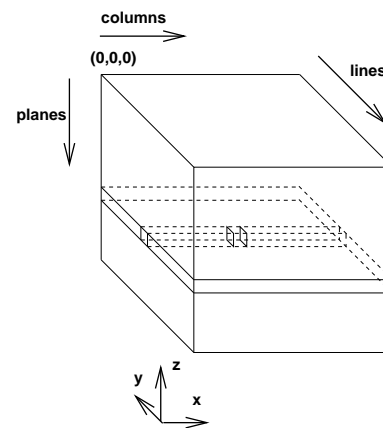


Figure 2: Voxel space with  $(x,y,z)$  and  $(p,l,c)$  coordinate systems

The choice of a suitable resolution is based on conflicting requirements, concerning:

**Laser point density:** Points belonging to a single tree should form a connected set of voxels — or at least a set that can be made connected in subsequent processing steps. With a too fine resolution (small voxel size), holes will appear that may

cause problems later. When the resolution is too coarse, on the other hand, it will happen many times that several laser points get transformed into a single voxel. Then, these points are indistinguishable in the voxel space and details like small branches may be lost. Furthermore, different branches (or trees) may get connected, *i.e.* pass through the same voxel or through neighboring voxels. This may cause errors when building topology in a later stage. It should be noted that the laser point density decreases with the distance to the scanner, and therefore is not uniform over the 3D space. This further complicates the choice for an optimal resolution.

**Space and time:** The number of voxels in a 3D raster increases with the 3rd power of the resolution (when expressed in voxels/distance) and may easily become impracticable when resolution is chosen too fine. Memory capacity problems may occur in software that needs random access within the 3D space. Smart implementations might prevent this situation in many cases, but not always. Moreover, processing times usually increase with the 3rd power of the resolution as well.

During experiments, resolution between 2 cm and 5 cm were used. 2 cm seems to preserve the details of the tree accurately enough, although this should be further investigated.

According to the laser scanning model, laser points occur on the surface of objects (trees). After rasterization, only voxels that are intersected by the tree-air surface will have non-zero values. The actual voxel value assigned is the number of points falling inside the cube in  $(x, y, z)$ -space corresponding to a voxel. Therefore, a voxel value can be regarded as a point density measure.

### 3.2 3D neighborhood operators

After transferring the laser points to a 3D raster, *neighborhood operators* are applied to enhance the data. The neighborhood operators are 3D extensions of 2D raster operators, such as linear filtering (convolution), distance transforms [Borgefors, 1996], [Svensson and Borgefors, 2002] and mathematical morphology [Serra, 1985].

The purpose of enhancement using neighborhood operators is threefold:

**Noise reduction:** In the context of tree reconstruction, isolated laser points, or isolated small groups of adjacent laser points, can be considered noise. Neighborhood operations are able to reduce this kind of noise.

**Repairing gaps caused by occlusions:** The software provides morphological *dilation* and *erosion* with user-defined *structuring elements* (Fig. 3). *Closing* (dilation followed by erosion) is used to fill small holes and gaps between laser point, which may be caused by branches occluding other branches. The maximum size of the holes and gaps that can be repaired in this way is controlled by the structuring element size, and is also related to the chosen resolution in the previous step. By using different shapes of structuring elements it is possible to indicate a preferred direction for the closing, for example to connect points above each other rather than points at the same height.

**Filling ‘hollow’ trees:** Laser points are located on tree surfaces rather than inside trees. Initially, trees are hollow. However, we are going to extract the *skeleton* (see 3.3), allowing to reveal the topology within the tree. Hollow trees have to be filled, by a second morphological closing (Fig. 4). The structuring element size is given by the maximum trunk thickness expected in a given raster resolution. The

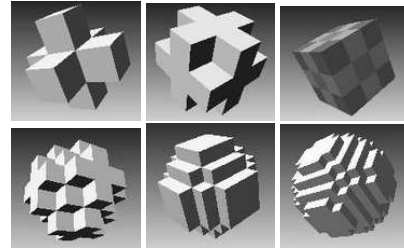


Figure 3: 3D Filter kernels and structuring elements

structuring element should be oriented horizontally, since filling is needed mostly in the (vertical) trunk.

This should work well when points are located completely around the tree, as was the intention of recording the scene from four different viewpoints. This would also reduce the above-mentioned occlusion problems. Unfortunately, the four datasets from the different scanner positionings could not be properly co-registered. Therefore, they had to be processed one by one, which limits the application of this hollow tree filling method.

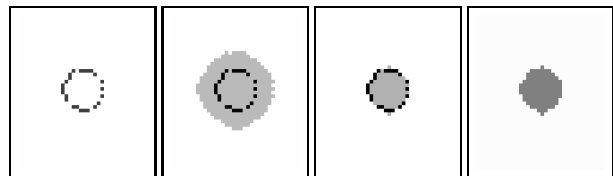


Figure 4: 2D illustration of dilation and erosion to fill ‘hollow’ trees. From left to right: input cross-section; dilation; erosion; filled cross section.

### 3.3 Skeletonization

The next step in tree reconstruction is skeletonization, the reduction of the thickness of tree trunks and branches to one voxel. After that, it will be possible to identify different branches and reveal topological relations between them.

Skeletonization is done by iteratively removing voxels from the outside of objects, up to the point where only a single-voxel wide linear structure is left. Strictly speaking, this is *line skeletonization*, which is to be distinguished for *surface skeletonization*, where volumes to are reduced to surfaces of single-voxel thickness. At that point, removal of one more voxel would cause a branch to break (be separated into disconnected parts), assuming 26-adjacency.

Skeletonization has been a difficult problem in 2D and even more so in 3D, but nowadays various solutions can be found in literature [Borgefors *et al*, 1999], [Lohou and Bertrand, 2001], [Palágyi *et al*, 2001], [Telea and Vilanova, 2003]. Skeletonization algorithms iteratively ‘peel’ layers from the outside of objects, and some approaches can be subdivided according to the number of subiterations within each iteration. This number can be 6, 8 or 12, depending on whether a cubic object to be skeletonized would be ‘attacked’ from the faces, corners or edges, respectively. For this project we implemented an 8-subiteration method, described by [Palágyi and Kuba, 1999], which shows excellent performance.

### 3.4 Skeleton segmentation

The next step finds the structure of the tree as represented by the skeleton, in terms of branches and junctions that connect



Figure 5: Skeletonization. Left: rasterized points; right: skeleton

branches. As suggested by [Palágyi, 2003] this step is based on Dijkstra's algorithm, a well-known method for finding shortest routes in a weighted directed graph (representing, for example, a road network).

First, the graph is created. Its nodes are the voxels that participate in the skeleton, and any pair of adjacent voxels generates *two* arcs between the nodes, one in either direction (see Fig. 6.B, but there each pair of arcs is represented by only one line). Arcs are assigned weights of either 3, 4 or 5, depending on whether the two voxels share a face, an edge or a point respectively [Verwer, 1991].

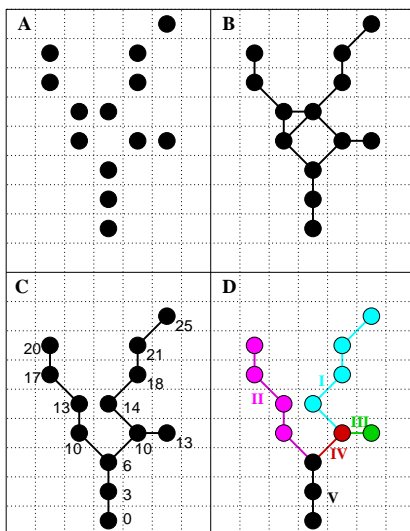


Figure 6: 2D illustration of skeleton segmentation. A. input skeleton; B. graph nodes and edges; C. spanning tree with shortest distances; D. segments and topology.

Given such a graph, Dijkstra's algorithms finds the shortest route to a destination node (the root!) from all other nodes in the graph. In our application, the lowest voxel of the tree trunk typically serves as the destination. The collection of shortest routes leading

from anywhere to the root forms a spanning tree of the graph. The term 'tree' in the graph-theoretical sense is not chosen by accident: the spanning tree that we find provides a logical model for a tree in a rasterized laser point cloud. Any ambiguities, such as loops caused by falsely-connected branches, which may still exist after skeletonization, are resolved now (Fig. 6.C) — how realistic the results are depends mostly on the success of previous steps.

After the algorithm finishes, each node (except the root) knows its parent, the next node along the shortest route to the root. By traversing the tree once more it is easy to find branches and their junctions, being nodes (voxels) that appear as a parent more than once. During this traversal each voxel is assigned a unique branch identification number. Each time a junction is found, a new identification number is given to this voxel and to the other voxels in the same branch closer to the root.(Fig. 6.D).

### 3.5 Connected component labeling

A problem occurs during skeleton segmentation (3.4) if the scene contains several (disconnected) trees. Dijkstra's algorithm needs to be provided with a single root voxel, after which it will find shortest routes only for voxels that are connected to that root. All other trees will be omitted.

To overcome this problem, connected component labeling has to be applied, followed by component separation (3.6). Connected component labeling uses a slightly modified segmentation algorithm, which scans the entire voxel space, starting at the bottom (*i.e.* at the plane with the highest number). Each time an unassigned voxel is encountered, Dijkstra's algorithm is started with that voxel as root. One tree is then traversed and all voxels involved will be assigned a label, which is unique for that tree. After that, the algorithm continues to scan for unassigned voxels *et cetera*, until the top of the space is reached.

This algorithm can be regarded a general 3D connected component labelling, since it does not rely on the input data being skeletonized trees.

### 3.6 Component Separation

In case of a scene with multiple trees, Connected Component Labelling (3.5) had to be applied to label the trees uniquely. Subsequently, the data set has to be subdivided into several sets, such that each of these contains only one tree, which can then be segmented into branches, as described in 3.4.

During the component separation step, components are sorted according to descending number of voxels and output into different data sets one by one. It may happen that certain components do not contain trees, but "loose" branches or other kinds of noise. The user may request that only a certain number of components is generated, or that only components with more than a certain number of voxels are considered.

### 3.7 Raster tree segmentation

At this stage the skeleton has been subdivided into trees and branches, and each skeleton voxel has been labelled with a unique branch identification. Now we will label all other tree voxels according to the nearest skeleton label.

There would be several ways to accomplish this by raster operations, such as iterative dilation or distance transform [Borgefors, 1996]. We use a Nearest Neighbor search algorithm, as it is known from database environments, where the problem is sometimes to find *k* nearest neighbors (database records) of a given query record, in a vector space of arbitrary dimension.

Usually this is a feature space, and the purpose is to find those  $k$  records in a database that are most similar to a query specification; the number of considered attributes specifies the dimensionality.

In our case the space is Euclidian with just 3 dimensions. The skeleton provides the data base, and we need only 1 neighbor (skeleton voxel) in each query (tree voxel). In this way all tree voxels will be labeled (Fig. 7).

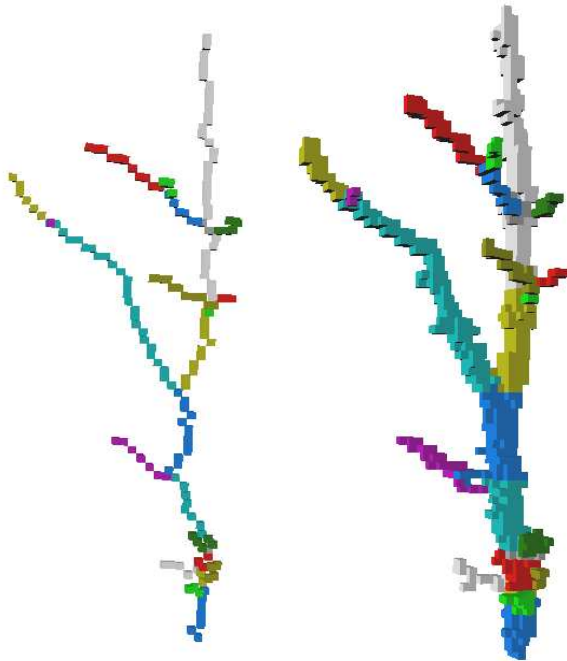


Figure 7: Segmentation. Left: segmented skeleton; right: segmented voxel space

### 3.8 Point cloud segmentation

Finally, we use once again the routine that originally transferred all the laser points into the voxel space. This time the purpose is to see into which voxel each laser point would be transferred, in order to assign the voxel's label to the point (Fig. 8).

## 4 CONCLUSION

In a rather straightforward fashion we transferred a carefully chosen selection of basic and advanced 2D raster (image) processing algorithms into the 3D domain. The result is a flexible set of tools, which we applied to segment a terrestrial laser data set of a scene in a forest according to the individual tree branches.

## REFERENCES

[Borgefors, 1996] Gunilla Borgefors, On digital distance transforms in three dimensions, *Computer Vision and Image Understanding*, Vol.64 No.3 pp.368-376.

[Borgefors et al, 1999] Gunilla Borgefors, Ingela Nyström and Gabriella Sanniti Di Baja, Computing skeletons in three dimensions, *Pattern Recognition* 32 (1999) 1225-1236.

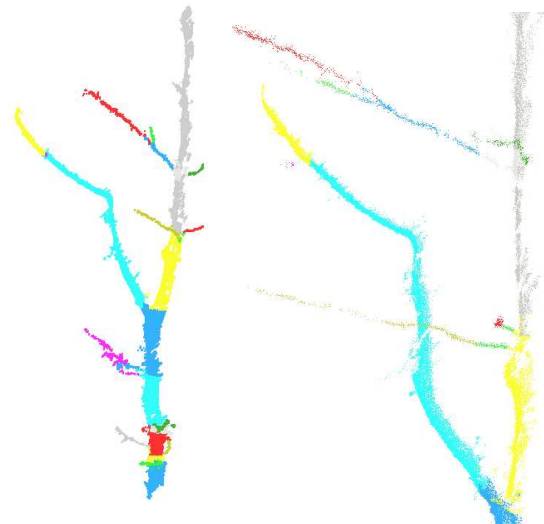


Figure 8: Point cloud segmentation. Left: segmented skeleton; right: detail

[Clark, 2002] N. E. Clark, 3D reconstruction of a tree stem using video images and pulse distances, *Symposium on Statistics and Information Technology in Forestry*. September 8-12, 2002 Blacksburg, Virginia USA

[Lohou and Bertrand, 2001] Christophe Lohou and Gilles Bertrand, A new 3D 12-subiteration thinning algorithm based on P-simple points, *IWCIA*, 2001.

[Palágyi, 2003] K. Paliágyi, J. Tschirren, M. Sonka: Quantitative analysis of intrathoracic airway trees: methods and validation, *LNCS 2732*, Springer, 2003, 222-233.

[Palágyi et al, 2001] K. Palágyi, E. Sorantin, E. Balogh, A. Kuba, C. Halmi, B. Erdöhelyi and K.Hausegger, A sequential 3D thinning algorithm and its medical applications, in M. F. Insana and R. M. Leahy (Eds.): *IPMI 2001, LNCS 2082*, Springer 2001, pp.409-415.

[Palágyi and Kuba, 1999] Kálmán Palágyi and Attila Kuba, Directional 3D thinning using 8 subiterations, *LNCS 1568*, Springer, 1999, 325-336.

[Pfeifer and Gorte, 2004] Norbert Pfeifer and Ben Gorte, Automatic reconstruction of single trees from terrestrial laser, *IAPRS 35*, Istanbul.

[Pyysalo and Hyypäe, 2002] Ulla Pyysalo, Petri Rönholm, Hannu Hyypäe Henrik Hagggré, Reconstructing tree crowns from laser scanner data for feature extraction. *IAPRS 34 Part 3-B* pages B-218 ff (4 pages).

[Serra, 1985] J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, London, 1982.

[Sun, 2000] Guoqing Sun, Modeling lidar returns from forest canopies, *TGARS* Vol.38, No.6.

[Svensson and Borgefors, 2002] Stina Svensson and Gunilla Borgefors, Distance transforms in 3D using four different weights, *Pattern Recognition Letters* 23 (2002) 1407-1418.

[Telea and Vilanova, 2003] Alexandru Telea and Anna Vilanova, A robust level-set algorithms for centerline extraction, *Joint EUROGRAPHICS-IEEE Symposium on Visualisation (2003)*

[Verwer, 1991] B.Verwer, Local Distances for Distance Transformations in Two and Three Dimensions *Pattern Recognition Letters* Vol.12, No. 11 (1991), 671-682