Robust approximation of the Medial Axis Transform of LiDAR point clouds as a tool for visualisation

Peters, Ravi; Ledoux, Hugo

DOI
10.1016/j.cageo.2016.02.019

Publication date
2016

Document Version
Accepted author manuscript

Published in
Computers & Geosciences: an international journal

Citation (APA)
Peters, R., & Ledoux, H. (2016). Robust approximation of the Medial Axis Transform of LiDAR point clouds as a tool for visualisation. *Computers & Geosciences: an international journal*, *90*, 123-133. https://doi.org/10.1016/j.cageo.2016.02.019

Robust approximation of the Medial Axis Transform of LiDAR point clouds as a tool for visualisation

Ravi Peters
r.y.peters@tudelft.nl

Hugo Ledoux
h.ledoux@tudelft.nl

Ravi Peters and Hugo Ledoux (2016). Robust approximation of the Medial Axis Transform of LiDAR point clouds as a tool for visualisation. *Computers & Geosciences* 90(A), March 2016, pp. 123-133.
DOI: http://dx.doi.org/10.1016/j.cageo.2016.02.019

Governments and companies around the world collect point clouds (datasets containing elevation points) because these are useful for many applications, e.g. to reconstruct 3D city models, to understand and predict the impact of floods, to monitor dikes, etc. We address in this paper the visualisation of point clouds, which is perhaps the most essential instrument a practitioner or a scientist has to analyse and understand such datasets. We argue that it is currently hampered by two main problems: (1) point clouds are often massive (several billion points); (2) the viewer's perception of depth and structure is often lost (because of the sparse and unstructured points). We propose solving both problems by using the Medial Axis Transform (MAT) and its properties. This allows us to: (1) smartly simplify a point cloud in a geometry-dependent way (to preserve only significant features), and (2) to render splats whose radii are adaptive to the distribution of points (and thus obtain less "holes" in the surface). Our main contribution is a series of heuristics that allows us to compute the MAT robustly for noisy real-world LiDAR point clouds, and to compute the MAT for point clouds that do not fit into the main memory. We have implemented our algorithms, we report on experiments made with point clouds (of more than one billion points), and we demonstrate that we are able to render scenes with much less points than in the original point cloud (we preserve around 10%) while retaining good depth-perception and a sense of structure at close viewing distances.

1 Introduction

With recent and ongoing advances in remote sensing to collect 3D elevation information, e.g. aerial LiDAR (Mallet and Bretar, 2009) and dense image matching (Haala and Rothermel, 2012), we are able to acquire samples of the Earth in unprecedented quantities (up to hundreds of points per square meter) and with very high accuracy. A prime example is the Dutch national elevation dataset AHN[1] which has a total of over 600 billion points. The collected elevation points represent both natural (e.g. vegetation, mountains and valleys) and man-made (e.g. buildings, dikes and bridges) surfaces—we refer to such point-based datasets, independent of their acquisition technique, as *point clouds*. Such geo-referenced point clouds are currently being collected by many governments and organisations because they allow us to reconstruct 3D city models (Rottensteiner, 2003), to better understand and predict the impact of floods (Fewtrell et al., 2011) or wind (Ujang et al., 2013), to monitor dikes (Krüger and Meinel, 2008), and can help improve several applications such as precision farming (Koenig et al., 2015), forest mapping (van Leeuwen and Nieuwenhuis, 2010) and infrastructure management (Snyder, 2013).

We address in this paper the effective visualisation of massive point clouds, which is perhaps the most essential instrument a scientist has to analyse and understand a point cloud. As argued by Dykes et al. (2005), visualisation can and should support the entire geoscientific process from the intitial data exploration to synthesis, analysis, evaluation and presentation. However, the visualisation of point clouds is currently hampered by two main problems: (1) due to their massive size they fit neither a computer's main memory nor a computer's graphics memory; and (2) how to achieve a visually pleasing rendering that strengthens the viewer's perception of depth and her sense of structure when only sparse and unstructured points are rendered. As we further describe in Section 2, it is indeed possible to visualise point clouds that exceed the capacity of a computer's internal memory through the use of out-of-core spatial indexing schemes and by applying methods such as view-frustum culling and multi-resolution hierarchies to select a subset of the points (Kreylos et al., 2008; Wimmer and Scheiblauer, 2006; Richter and Döllner, 2010). Thus a high framerate can be achieved by limiting the number of points that is sent to the graphics card of a computer. We argue that the visual quality is linked to the spatial distribution of points on the screen and the applied point rendering technique. However, current point cloud visualisation methods often use the most basic point rendering techniques and always apply a regular grid-based point simplification scheme that fails to take into account the geometry of the sampled surface. See for instance Figure 1 that illustrates how the viewer's sense of depth and structure is distorted at closer viewing distances because of

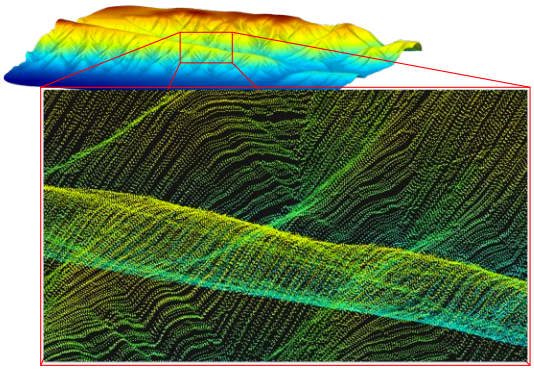


Figure 1: Point cloud rendered with shaded fixed-sized points. When zoomed in it is hard to perceive structure and depth, due to the large screen distances between points.

[1] http://www.ahn.nl

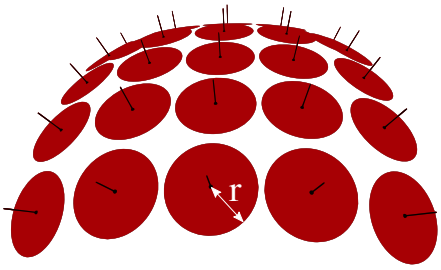


Figure 2: A splat is defined for each point as a normal-oriented disk with a radius r. Usually r is chosen such that there are no holes.

the large gaps between points.

Furthermore, because the geometry of the sampled surface is not taken into account during regular grid-based simplification, fine details can not be adequately represented.

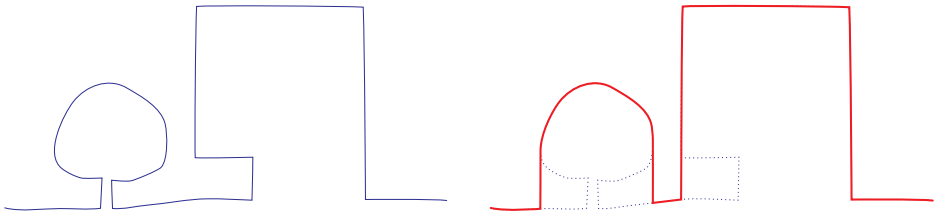
In this paper we solve both problems by using a point cloud visualisation approach based on the Medial Axis Transform (MAT), a skeleton-like representation of shape that we describe further in Section 3. In Section 4 we demonstrate how to use the *local feature size*, a property derived from the MAT, to achieve both a geometry-dependent point cloud simplification and a geometry-dependent point rendering. The point distribution that we then obtain is adaptive to the geometry of surface features: small surface features are represented with relatively more points than large features. To render points we use surface splatting (see Figure 2), i.e. a rendering technique were points are rendered as circular disks (splats) that are oriented and shaded using the point normals (see also Gross and Pfister (2011)). By relating the splat radii to the local feature size, we are able to draw bigger splats for large features that do not contain fine details and small splats for finer geometry with a smaller feature size. Combined with our geometry-dependent point simplification, we are able to render scenes with much less points while retaining the same visual quality and we achieve good depth-perception and sense of structure also at close viewing distances.

Furthermore, to our knowledge we are the first to robustly estimate the MAT for real-world LiDAR point clouds. In Section 3.3 we explain how we extend an existing algorithm to estimate the MAT (Ma et al., 2012) to deal robustly with the noise that is typically present in LiDAR point clouds. We also show in Section 3.4 that our method to obtain MAT approximations from point clouds can be scaled to massive datasets through the application of a simple partitioning scheme. Finally, in Section 5, we describe our experimental results with real-world datasets containing up to 1.3 billion points.

2 Related Work

We review in this section two main topics related to the work presented in this paper: 1) point cloud simplification, and 2) visualisation approaches for big point clouds.

We do not elaborate on more traditional approaches of simplification and visualisation that are based on raster grids or TINs (see for example Lee (1989); Garland and Heckbert (1995); Kraus and Pfeifer (1998)), because these are all based on the assumption that the area of interest can be adequately represented with a so-called 2.5D surface or elevation field, i.e. a surface monotone to the horizontal plane (Li et al., 2005; Kumler, 1994). We believe this assumption does not apply for existing urban point clouds that contain many inherently 3D objects such as trees, balconies and even vertical walls (see Figure 3). By representing these objects in 2.5D, valuable information on their shape is lost. The resulting misrepresented shapes may sub-



(a) Actual boundary (b) Boundary after 2.5D projection

Figure 3: Profile view of a DSM. Information is lost when the surface must be uniquely projectable to a horizontal plane.

sequently lead to errors in any follow-up analysis.

2.1 Point simplification

Pauly et al. (2002) implement and review two approaches for point cloud simplification of densely-sampled smooth shapes: clustering and iterative simplification. Notice, however, that the point cloud datasets that we consider are not necessarily smooth and the sampling density may vary greatly.

Clustering subdivides the point cloud into clusters that are each replaced by one representative sample. The cluster may be defined by the non-empty cells of a regular grid that is superimposed on the input point cloud, in which case the clusters are replaced by the center points of these cells. Grid-based clustering is widely used for LiDAR point clouds because it is simple to implement and fast to compute. However, because of the fixed cell size the resulting points are uniformly distributed, which makes it impossible to achieve a sampling density that is adaptive to the geometry of the sampled surface.

Iterative simplification, which can be considered a generalization for 3D of the work of Lee (1989), reduces the number of points based on an error metric that quantifies the error that results from the removal of a point. Points are removed in order of increasing error, and every point removal affects the error of surrounding points. This is a global algorithm that is difficult to scale to massive point clouds.

Pauly et al. (2002) use the surface-variation metric to quantify geometric detail for each point as the variation along the approximated normal with respect to the tangent plane of a point. They show that surface-variation closely resembles curvature, but argue that surface-variation is a more meaningful metric for point cloud simplification, because when two surfaces come close together, i.e. closer than the smallest enclosing sphere of the k-neighborhood of the point in question, this also leads to a higher surface-variation. The local feature size metric that we use in this paper also possesses these favourable properties.

2.2 Visualisation of massive point clouds

Kreylos et al. (2008) have implemented a multi-resolution out-of-core octree-based renderer. Their octree-based downsampling scheme, similar to clustering as described in Section 2.1, is constructed in a pre-processing phase and designed to achieve a uniform point distribution at every level of detail, so it does not consider the geometry of the sampled objects in any way. At any time a subset of the input point cloud is displayed and points are rendered as simple fixed-

sized squares with optional shading. While fast and simple to implement, this results in a distorted sense of depth and structure at closer viewing distances (see Figure 1). This is due to the presence of holes in the surface when the distances between points become too large on the screen. It is especially a problem for sparsely sampled areas such as vertical surfaces (walls) in aerial point clouds. Wand et al. (2008), Wimmer and Scheiblauer (2006); Scheiblauer (2014), Richter and Döllner (2010, 2014) and Elseberg et al. (2013) all showcase comparable out-of core octree-based visualisation frameworks for large point clouds with uniform point downsampling. Elseberg et al. (2013) visualise coarser level of details presumably by rendering sets of octree cells as points located in the cells' center rather than using a subset of the original point cloud.

As illustrated by Figure 2, splatting is a point rendering technique where points are rendered as surface aligned disks that are parameterised by some radius r (see also Gross and Pfister (2011)). Usually r is chosen such that the splats are overlapping each other so that holes are absent and the point cloud appears to be a closed surface on the screen regardless of the viewing distance, because r is defined in object space (i.e. in the coordinate system of the point cloud).

Wand et al. (2008) briefly discusses the possibility of using rectangular point splats that are aligned according to the two greatest principal components of a k-neighbourhood. Scheiblauer (2014) and Richter and Döllner (2014) both implement a form of point splatting. But Scheiblauer (2014) does not use normals, and the size of the splats either depends on the rendered level of detail or on a local point density estimate.

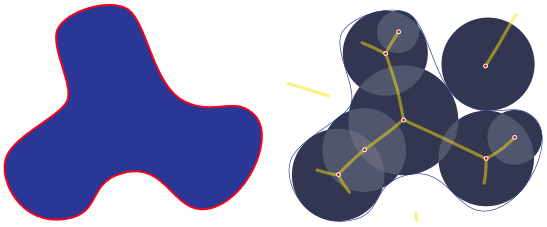
Kovač and Žalik (2010) propose to use a precomputed quadtree index that facilitates on-the-fly point loading and normal computation, but assumes good spatial coherence and is designed to work only for 2.5D surfaces. Points are rendered as oriented splats using the approach of Botsch and Kobbelt (2003). They apply random subsampling and fixed splat radii, which does not necessarily result in a hole-free visualisation of the scene. As the authors themselves note, the holes are partly caused by inadequate sampling densities in some parts of the point cloud. Indeed, vertical and transparent surfaces are often relatively sparsely sampled in airborne LiDAR point clouds.

We conclude that many of the described approaches have solved the issue of managing huge point clouds using out-of-core spatial indexes, and that some of the approaches apply splatting, but none of them takes the geometry of sampled objects in consideration.

3 Computing the medial axis transform

3.1 The MAT

The medial axis transform (MAT) is an alternative representation of shape that explicitly encodes a shape's topology and geometry as a skeletal structure. The MAT can be computed from a boundary representation of a shape, and vice versa, and it can be defined as a set of *medial balls*. Given a shape boundary $\mathcal{S}$ embedded in the three-dimensional Euclidean space ($\mathbb{R}^3$), we define a medial ball as a maximal ball that touches $\mathcal{S}$ on at least two points and does not contain any points of $\mathcal{S}$ on its interior. The MAT, denoted $\mathcal{M}[\mathcal{S}]$, is defined as the set of medial balls (see Figures 4a and 4b for a 2D example); likewise the *medial axis* of $\mathcal{S}$ is defined as the set of centers of medial balls. Consequently, all points of the medial axis of $\mathcal{S}$ are closest to at least two points on $\mathcal{S}$. In $\mathbb{R}^3$ the medial axis is a set of manifolds with



(a) A shape (b) Its medial axis and some medial balls.

Figure 4: The Medial Axis Transform for a two-dimensional shape

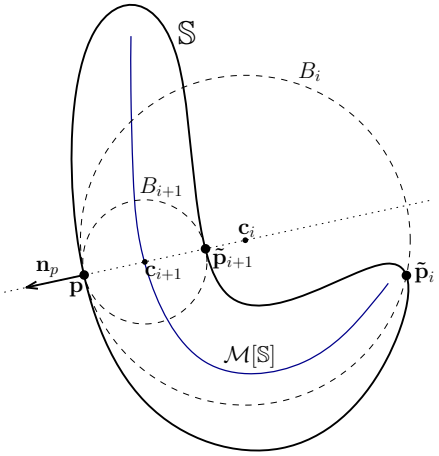


Figure 5: The two final iterations in the ball-shrinking process for the point $\mathbf{p}$ to approximate the medial axis $\mathcal{M}[\mathcal{S}]$.

boundaries (called *sheets*) that meet along a set of Y-intersection curves (see Siddiqi and Pizer (2008)) and form a skeleton-like structure. The medial balls in the interior of $\mathcal{S}$ form the *interior* MAT, and the balls on the exterior of $\mathcal{S}$ form the *exterior* MAT.

3.2 Approximating the MAT

The exact computation of the MAT is not possible for all shapes and is computationally infeasible in practice, especially for 3D objects (Attali et al., 2009). Fortunately, approximations of the MAT give satisfactory results for most shapes and are computationally feasible. Various algorithms exist to approximate the MAT from either a set of sample points on $\mathcal{S}$, often using the Voronoi diagram (Attali and Montanvert, 1997; Amenta et al., 2001; Dey and Zhao, 2004), or from a voxelised representation of $\mathcal{S}$ using the distance transform (Foskey et al., 2003; Hesselink et al., 2005; Chaussard et al., 2011). However, Voronoi methods are complex to implement robustly and using a voxelisation introduces unwanted inaccuracies and is not scalable (Sobiecki et al., 2013). In this paper we therefore choose to use the ball-shrinking algorithm that was introduced by Ma et al. (2012). This algorithm is simple, robust, memory efficient and easy to parallelize (see Ma et al. (2012); Jalba et al. (2012)). And, as we demonstrate in Sections 3.3 and 3.4 it can be modified to deal effectively with noisy real-world inputs and to process massive dataset.

The ball-shrinking algorithm approximates the medial axis from an oriented point cloud P that samples $\mathcal{S}$. It works as follows (see Figure 5): for each point $\mathbf{p} \in P$ with corresponding normal $\mathbf{n_p}$, a ball $B_0(\mathbf{c}_0, r_0)$ with a large radius r_0 and center $\mathbf{c}_0 = -r_0\mathbf{n_p} + \mathbf{p}$ is initialized. By definition this ball is centered on the straight line through $\mathbf{n_p}$. This large initial ball is iteratively shrunk to find and approximate a medial ball. Let $i > 0$ be an iteration counter. A new ball B_i is found by performing a nearest neighbour query from $\mathbf{c_{i-1}}$ to the points in P (but excluding $\mathbf{p}$ itself). The resulting point $\mathbf{\tilde{p}_i}$ is used to compute r_i and $\mathbf{c_i}$ such that B_i passes through $\mathbf{p}$ and $\mathbf{\tilde{p}_i}$ and remains centered on the straight line that passed though $\mathbf{n_p}$. The iteration is stopped when $\mathbf{\tilde{p}_{i+1}}$ equals $\mathbf{p_i}$ or $\mathbf{p}$, implying that B_i is empty. It clearly also touches P on two points and can therefore be considered a medial ball; and $\mathbf{c_i}$ is therefore a point on the medial axis of P. If this ball-shrinking procedure is executed for every $\mathbf{p} \in P$, an approximate inte-

rior medial ball for every $\mathbf{p}$ is obtained. By running the algorithm a second time, but now with flipped normals, we find also an approximate *exterior* medial ball for every $\mathbf{p} \in P$. The set of the interior and exterior medial balls forms an approximate MAT of $\mathcal{S}$.

3.3 Denoising heuristics

The original ball-shrinking algorithm of Ma et al. (2012) was designed to handle well-sampled point clouds with very little noise. However, in practice, point clouds such as LiDAR datasets typically contain significant noise and have a highly varying point density which leads to a distorted MAT approximation. While it is common to filter and remove noisy MAT points after approximation (see for instance Amenta et al. (2001)), this has an unwanted side effect: a filtered MAT contains less points and is therefore a sparse approximation of the true MAT. In our experience, the resulting MAT approximation may in fact be so sparse that the MAT is hardly perceivable at all. We therefore introduce heuristics that do not remove noisy MAT points, but move them towards a stable MAT instead. They are extensions to the ball-shrinking algorithm of Ma et al. (2012) which, as described in Section 3.2, computes a series of n balls $B_0, \ldots, B_{n-1}$ for every point in P. We have found that it is often possible to recognize an unstable medial ball by analyzing the progression of ball metrics as a function of the ball iteration. Furthermore, we make the observation that the last ball in the ball-shrinking sequence may not always be the best approximate medial ball when noise is taken into consideration (see Figure 6).

Based on these observations we propose two simple heuristics that both use the scale invariant separation angle θ_i (the angle $\angle \mathbf{p}\mathbf{c_i}\tilde{\mathbf{p}}_\mathbf{i}$, see Figure 6) that is defined for each iteration i in the ball-shrinking process for a single point.

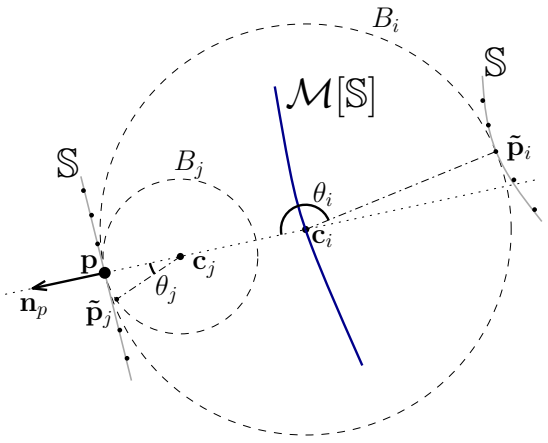


Figure 6: Two ball iterations for $\mathbf{p}$, where $j = i+1$. The noisy point $\tilde{\mathbf{p}}_\mathbf{j}$ can be detected by the small separation angle $\theta_\mathbf{j}$ of B_j.

stable ball preservation Whenever θ_{i+1} drops below a threshold t_{preserve}, we stop the ball-shrinking process and set B_i to the approximate medial ball for the current surface point. As illustrated in Figure 6, B_i ignores the noisy point of iteration $i + 1$, and is therefore a better approximate medial ball.

plane detection when the separation angle of the first ball θ_1 is lower than a threshold t_{planar}, we do not assign a medial ball to the current surface point. This typically occurs when approximating the medial axis for a planar feature whose point samples are slightly perturbed. Since a ball that touches a plane in two points should have an infinite radius, we should ignore these balls.

Thus, by exploiting the information that is captured in the sequence of balls of a surface point, rather than only considering the final ball, we find significantly more robust approximate medial balls. Note that this does not add to the

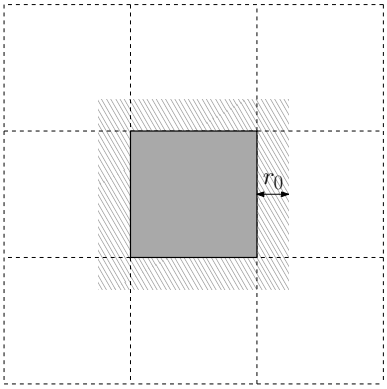


Figure 7: Simple tiling scheme. Each tile is buffered with the initial ball radius r_0.

computational cost of the ball-shrinking algorithm, since the average number of ball iterations per point is lower as a result of these heuristics.

3.4 Approximating the MAT for large datasets

To be able to handle massive datasets (i.e. billions of points) that do not fit a computer's main memory (in-core), we propose a simple partitioning scheme for the ball-shrinking algorithm that is able to sequentially process subsets of the dataset. This is possible because the only global operation performed by the algorithms described in this paper, the nearest neighbour query, is actually bounded by the initial ball radius r_0 which is a user-defined parameter. Only medial balls with a radius up to r_0 are constructed, and the radius of the largest medial ball of an object typically depends on the approximate size of that object. A sensible choice for the value of r_0 is therefore an approximate size of the largest object in the input. For our datasets we found that a value for r_0 of approximately 100 m suffices. It is therefore possible to spatially subdivide and process a massive dataset with a limited amount of main memory.

To partition, we subdivide the dataset into square tiles of fixed dimensions. The tile-size is chosen such that the contained points easily fit main memory. Additionally, every tile is buffered with the value of r_0 (see Figure 7). Then, the tiles are processed one by one. First we compute a kd-tree for the complete tile (including the buffer) to speed up nearest neighbour queries. Then we approximate the point normals and the MAT itself for the points that are inside the tile but not in the buffer region.

With this simple out-of-core scheme we can obtain identical outputs compared to the in-core approach. In Section 5 we demonstrate how we compute the MAT for a dataset with 1.3 billion points.

4 MAT-based simplification and splat-based visualisation

We propose to use the MAT derived *local feature size* (LFS) for both the point cloud simplification and the splat radius determination. The LFS $f(\mathbf{p})$ of a point on $\mathbf{p} \in P$, where P represents the point cloud, is defined as the shortest distance between $\mathbf{p}$ and the MAT of P (see Figure 8). It captures the curvature at $\mathbf{p}$ and the proximity of other parts of $\mathcal{S}$, since in both these cases the medial axis is close to $\mathcal{S}$. Given a point cloud P, we can eliminate much of its redundancy by removing points that have relatively high LFS. We use the ϵ-sampling criterion (Amenta et al., 1998) to achieve this, similarly to Dey et al. (2001) and Ma et al. (2012). P is called an ϵ-sample if each point $\mathbf{p} \in P$ has another point of P within a distance of $\epsilon f(\mathbf{p})$ (see Figure 8). An ϵ-sample can be approximated by iteratively removing points that do not break the ϵ-sampling criterion. Similar to Ma et al. (2012) we compute an approximate ϵ-sample from an oversampled input point cloud P by testing for

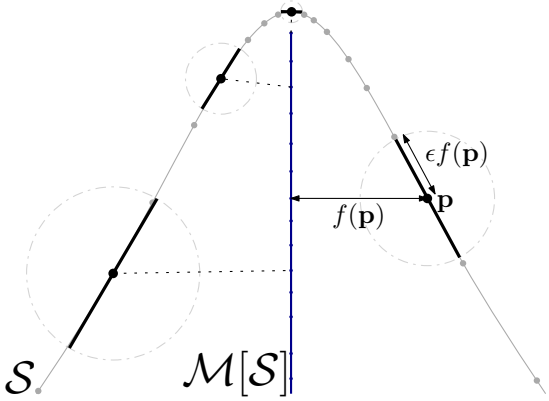


Figure 8: The local point density and the radius of the splat for $\mathbf{p}$ (a point sampled on $\mathcal{S}$) are based on the local feature size $f(\mathbf{p})$ which is the shortest distance from $\mathbf{p}$ to the point approximation of $\mathcal{M}[\mathcal{S}]$. For this figure $\epsilon \approx 0.5$.

each $\mathbf{p} \in P$ whether the ball $B(\mathbf{p}, \epsilon f(\mathbf{p}))$ contains any point from P other than $\mathbf{p}$ itself. If it does, $\mathbf{p}$ is removed from P. In our version of this algorithm we process the points in a random order, because an order in which subsequent points are too close may cause gaps in the simplified point cloud. The LFS thus gives a simple and effective definition of the size of a feature, and with the ϵ-sampling criterion we are able to relate the LFS directly to the local sampling density. We thus obtain a geometry dependent simplification of the point cloud where areas with a large LFS are represented with relatively fewer points than areas with a small LFS.

To complete our visualisation approach we also make the splat radius adaptive to the LFS. Each point is rendered as a splat with a radius set to $\epsilon f(\mathbf{p})$ (see Figure 8). Points in areas with a lower point density in the ϵ-sample are therefore drawn with larger splats. Moreover, because both the splat radii and the ϵ-sample are both based on the distance $\epsilon f(\mathbf{p})$, the resulting visualisation is such that a surface-like impression is obtained where holes are minimised despite the non-homogeneous geometry-aware point simplification.

5 Implementation, experiments and discussion

We have implemented the algorithms described in Sections 3 and 4 and tested them using two aerial LiDAR datasets. The first is a publicly available dataset of a mountain range in California, USA (National Science Foundation, 2005). From this 'mountain' dataset we took an area of 942 × 898 meters, containing 1 632 040 points. The reported point density is 2.98 points per square meter. The second dataset is from the municipality of Rotterdam, The Netherlands, and has a reported point density of 30 points per square meter. From this 'urban' dataset we selected an area of 200 × 250 meters, containing 746 351 points, to compare with the mountain dataset and an area of 1.3 billion points to test the scalability of our method.

Since our ball-shrinking algorithm requires a normal vector for every input point, we compute these beforehand using a principal component analysis of the nearest 15 neighbours of every point. To speed up nearest neighbour queries for the normal computation, the MAT estimation, the LFS estimation and the ϵ-sample simplification, we use a kd-tree. Furthermore, to make the computation of the LFS more robust, we compute the LFS (the distance from a surface point to the MAT) as the median distance of the 15 nearest MAT points. To perform splatting we implement the splat rendering method of Botsch and Kobbelt (2003) using OpenGL shaders.

We have released our Python implementation under an open-source license and made it publicly available (Peters, 2015).

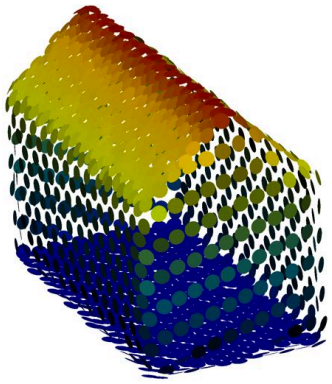


Figure 9: Simulated aerial LiDAR point cloud for a house model (dimensions: 10 × 20 × 15 m, 1751 points).

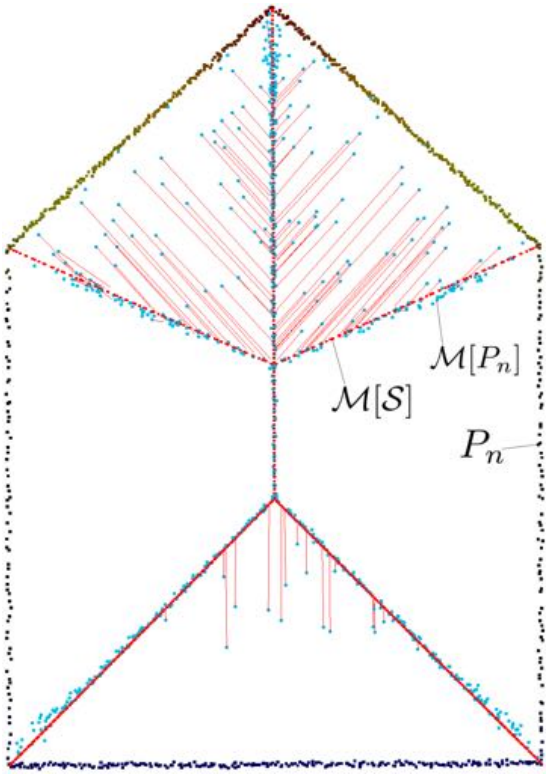
5.1 MAT approximation and noise handling

In order to quantitatively analyse the effect of the novel denoising heuristics that we propose in Section 3.3 we use a point cloud that was obtained by simulating an aerial LiDAR scan[2] on a surface model of a house (see Figure 9). Noise is simulated by adding a normally distributed noise component with a standard deviation of 2 cm along the scanning direction and an additional 2 cm in the position of the scanner. We ran the unmodified ball-shrinking algorithm (Ma et al., 2012) (denoted $\mathcal{M}$) and our improved variant with denoising heuristics (denoted $\mathcal{M}_d$) on this point cloud with added noise (denoted P_n) and without added noise (denoted P). To quantify the quality of the resulting MAT approximation we measure the distance to a reference MAT, that is directly derived from the surface model geometry (denoted $\mathcal{M}[\mathcal{S}]$). From Figure 10b we observe an overall decrease of 31% (at a conservative $t_{\text{preserve}} = 20°$) in the error of $\mathcal{M}[P_n]$ as a result of our ball-preservation denoising heuristic that affected 11% of the interior MAT points. These points are no longer close to the main MAT branches, as a result of the small perturbations in P_n for which the unmodified ball-shrinking is particularly sensitive. Figure 10a illustrates how these points are moved towards $\mathcal{M}[\mathcal{S}]$ as a result of our ball-preservation heuristic, thereby obtaining a denser MAT approximation than possible when these points would have been removed (e.g. Amenta et al. (2001)). From our experiments with real-world LiDAR datasets we observe the same behaviour (see Figures 11 and 12). As a result of the plane detection heuristics, 96% of the exterior MAT points are filtered at $t_{\text{planar}} = 30°$ (similar to the plane detection in Figure 11). Furthermore, from the plot of $\mathcal{M}_d[P]$, we see that our denoising heuristics have a negligible effect on the approximation of the MAT of the noise-free pointcloud P.

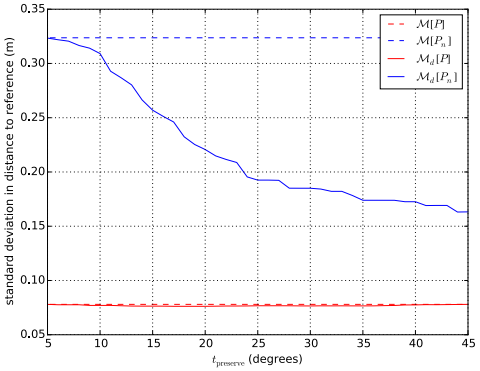
It can also be observed (especially from Figure 12) that some of the minor side-branches disappear or shrink in the denoised MAT. We can conclude that there is a trade-off between the amount of detail captured by the MAT and the robustness to the noise present in the input point cloud. Where our denoising approach distinguishes itself is that it delivers a much denser MAT approximation than previous methods (where noisy points are removed). Also, despite the dependence of two user-defined thresholds, we find that the same parameter values ($t_{\text{preserve}} = 20°$ and $t_{\text{preserve}} = 32°$) are adequate for both the urban and the mountain dataset.

Ultimately a much more distinctive and more useful MAT approximation is obtained. Without our denoising heuristics, computing the LFS would not be feasible since there are many MAT points near the sampled surface that distort the LFS computation (see Figure 13). Therefore, it would not be possible to successfully perform our geometry-dependent point cloud simplification and splat-radius determination for visualisation.

[2] Using industry standard parameters (flight height 400 m, flight line spacing: 400m) and the BlenSor software (Gschwandtner et al., 2011)



(a) Cross section of simulated point cloud with noise (P_n), reference MAT ($\mathcal{M}[\mathcal{S}]$), and $\mathcal{M}[P_n]$. Red lines indicate how points MAT are moved as a result of our denoising heuristics ($t_{\text{preserve}} = 30°$ and $t_{\text{preserve}} = 30°$)



(b) Overall error in MAT approximation with respect to reference MAT with and without our denoising heuristics.

Figure 10: Sequence of shrinking balls on a 2D dataset for a single input point

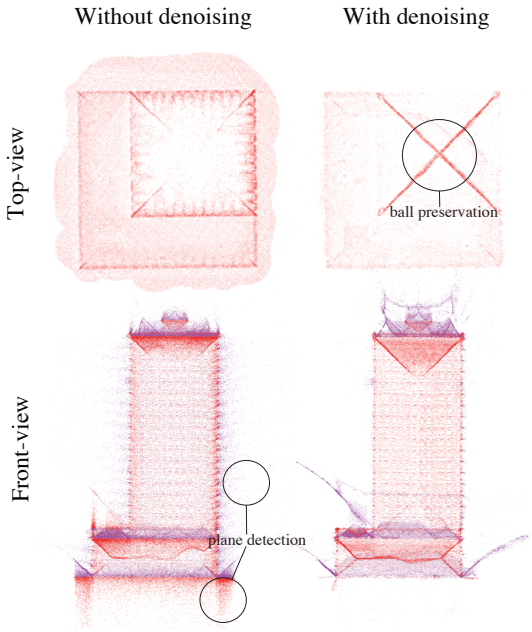


Figure 11: Medial axis approximation dataset B without denoising (Ma et al., 2012) and with denoising (our method) for $t_{\text{preserve}} = 20°$ and $t_{\text{planar}} = 32°$. Shown are top views (top) and side views (bottom). Interior points in red, Exterior points in purple

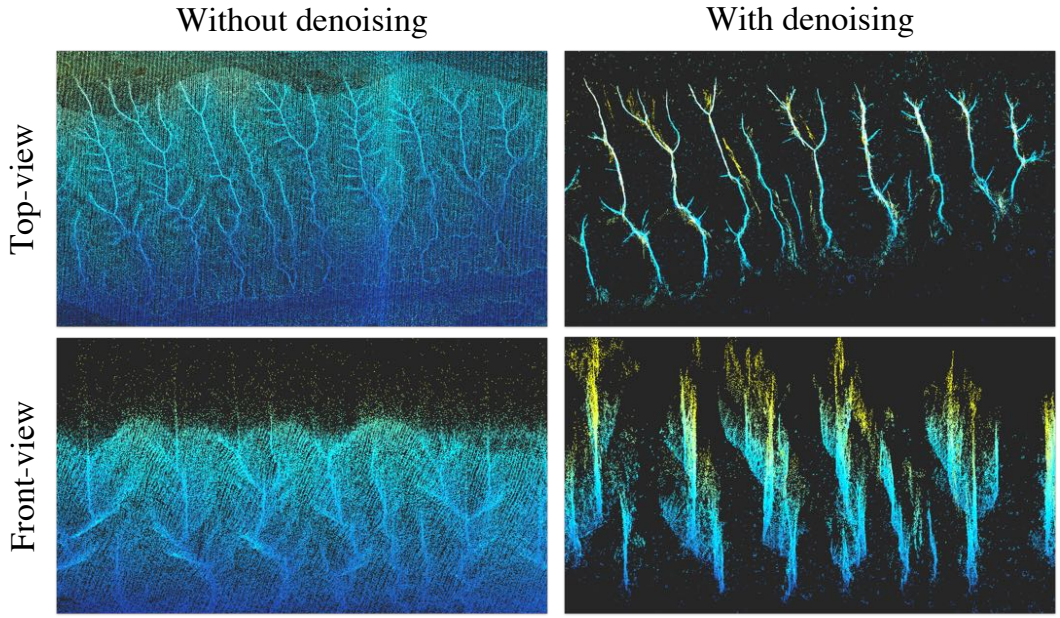


Figure 12: Exterior MAT approximation with (our method) and without denoising (Ma et al., 2012) heuristics. $t_{\text{preserve}} = 20°$ and $t_{\text{planar}} = 32°$

Figure 14 illustrates very clearly the skeleton-like nature of the MAT of the sample dataset, where the ridges and valleys of the mountain range are translated to branches of the interior and exterior MAT respectively.

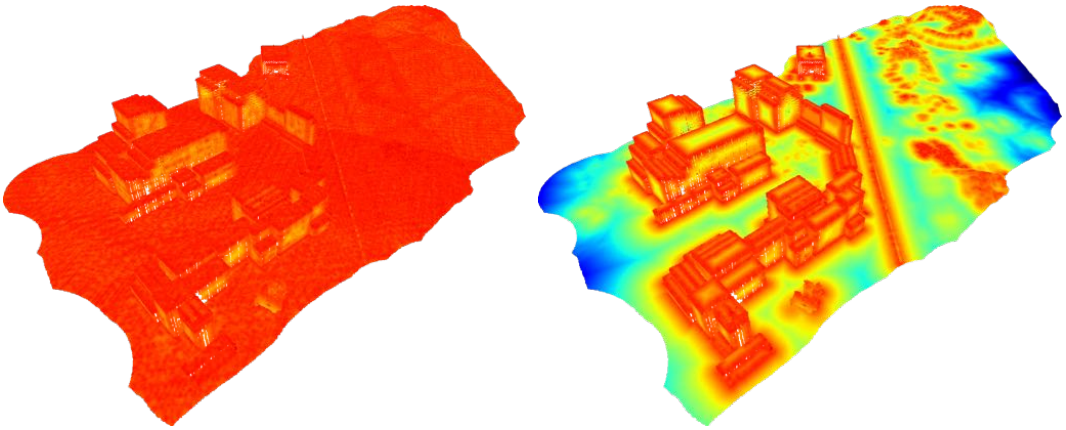
5.2 Scaling

Figure 15 shows the memory usage of our simple partitioning scheme during the MAT computation of a 1.3 billion point dataset. We must note that our current implementation has inefficient memory management, which is inherent to the used programming language (Python). The memory measurements in Figure 15 are therefore exhibiting a slightly increasing trend over time (i.e. as the number of processed points increases) that is independent for the theoretical memory requirements of the algorithm. Nonetheless, we make the key observation that the amount of required memory is successfully limited. The amount of required memory is now bounded by the largest number of points inside a tile rather than the total point count of the dataset. As a result we are able to process massive datasets.

Furthermore, others have shown that highly efficient parallel implementations of the ball-shrinking algorithm are feasible (Ma et al., 2012; Jalba et al., 2012) (our added denoising heuristics do no alter the memory requirements of the ball-shrinking algorithm). We therefore conclude that our approach to obtain a robust MAT approximation of a LiDAR point cloud is well scalable in terms of both memory and computational cost.

5.3 MAT-based simplification and splat-based visualisation

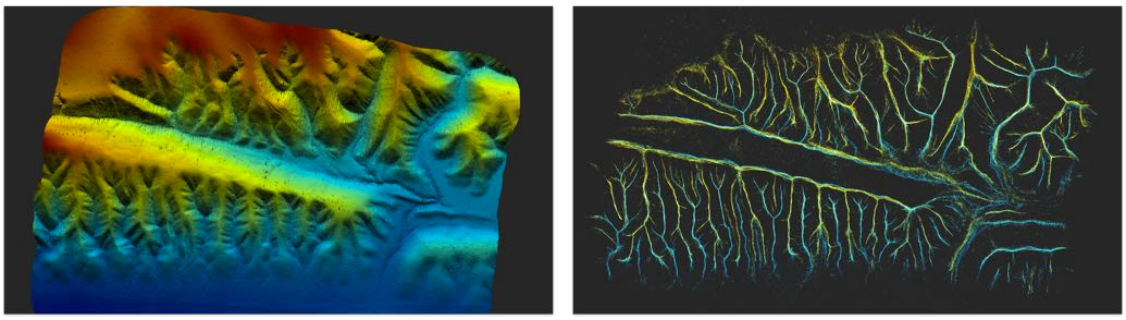
Figures 16 and 17 demonstrate the effect of the MAT-based simplification and splats with their



(a) Without denoising.

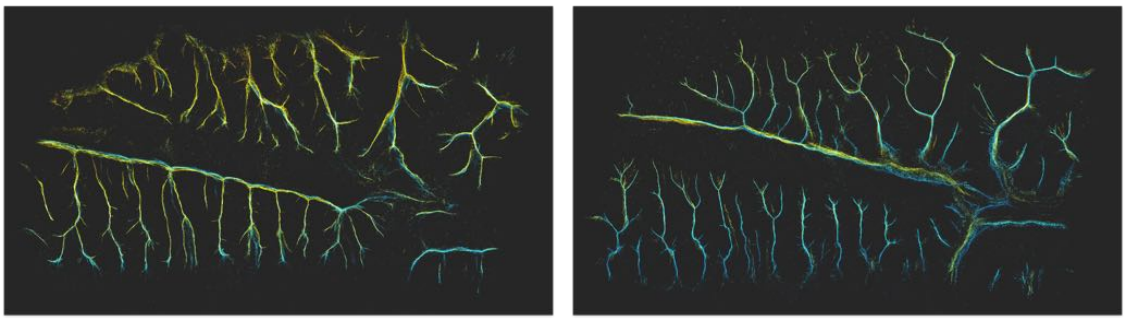
(b) With denoising. $t_{\text{preserve}} = 20^\circ$, $t_{\text{planar}} = 32^\circ$

Figure 13: Local feature size approximations for an urban dataset with and without denoising. Red indicates low local feature size, blue high local feature size.



(a) Raw point cloud.

(b) Interior and exterior MAT



(c) Interior MAT

(d) Exterior MAT

Figure 14: The denoised MAT for the mountain dataset. Colors indicate elevation.

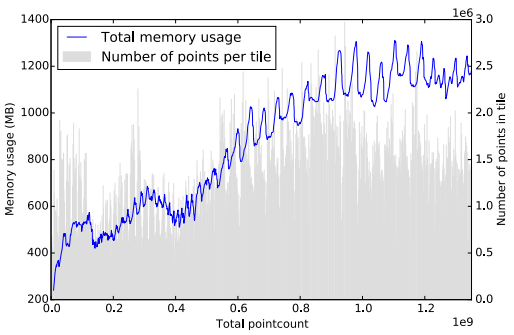


Figure 15: Memory consumption as a function of the number of processed points for the MAT approximation of the 1.3 billion point urban dataset.

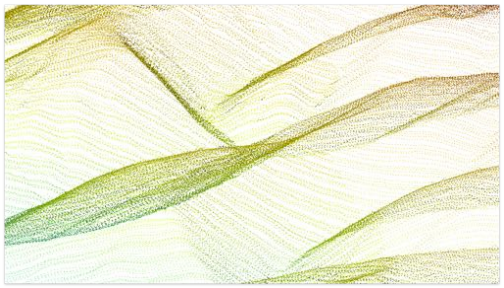
radii adapted to the LFS.

In both cases the simplification removed 90% of the original points ($\epsilon = 0.4$), yet when rendered with LFS-sized splats the resulting visualisation is similar to the original splatted point cloud with fixed splat-radii. While the simplified LFS-splatted rendering is not absolutely free from holes for the urban dataset (see mark 2 in Figure 17), we also observe that, despite the reduction in points, the sparsely sampled vertical surfaces (walls) now appear as solid surfaces (see mark 1 in Figure 17d). This is a notable improvement over fixed-sized splats, because this amplified the viewer's sense of structure and depth at all viewing distances.

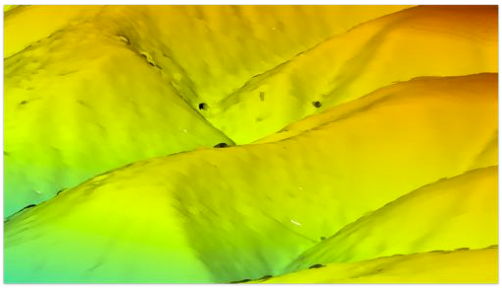
Figure 18 illustrates further how the distribution of points in the simplified point cloud respects the geometry of the sampled surface. Splats are drawn there with a decreased radius so that it is clearly visible that 1) more points are drawn in areas with a relatively high curvature such as the creases in the valleys and 2) the corresponding splats have a smaller radius when compared to the planar areas with fewer and larger splats (also apparent in Figure 17d). Finally, in Figure 19a we compare fixed-sized splats with LFS-sized splats for the simplified mountain dataset. The flat region has fewer samples due to the relatively high LFS. But, in the case of LFS-sized splats, the larger splat radii effectively compensate for the coarser point distribution, leading to a virtually hole-free visualisation.

6 Discussion

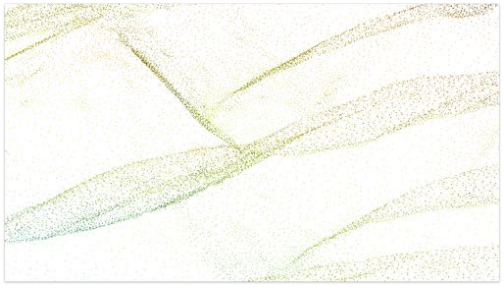
We have shown that a robust MAT can be computed from LiDAR point clouds, and that we can use the MAT to perform geometry-dependent simplification and to improve splat-based visualisation of LiDAR point clouds. A key motivation for the development of this approach was the distorted depth-perception and sense of structure due to the presence of holes when a point cloud is viewed up close (see Figure 1). From Figure 17 it is obvious that splat-based rendering gives significantly less and smaller holes. Furthermore, with our MAT-based point simplification and splat-radii, we can maintain this benefit while retaining only 10% of the points. However, not all holes are eliminated and some new holes are even introduced. This has two causes. First, the largest holes are mainly a result of insufficient sampling of the LiDAR point cloud due to occlusion (e.g. missing walls) and the material and orientation of the surface with respect to the laser scanner (also noted by Kovač and Žalik (2010)). Without either making explicit assumptions on the shape of objects or acquiring more samples in the field, little can be done about this in our opinion. Second, while our simplification procedure guarantees that areas are not oversampled according to the ϵ-sample criterion, it does not guarantee that sufficient points are preserved in all cases. Thus, although the local point densities are approximately the same, depending on the (randomized) order in which points are processed during the simplification, holes may appear in some places. To solve this issue, the *local* point distri-



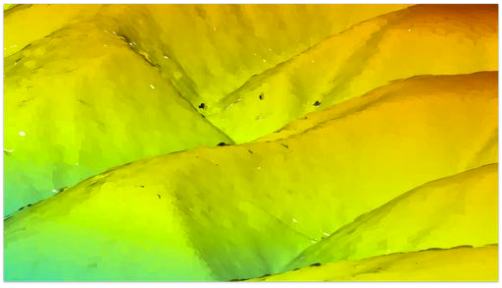
(a) Full point cloud with simple points



(b) Full point cloud with fixed-radius splats

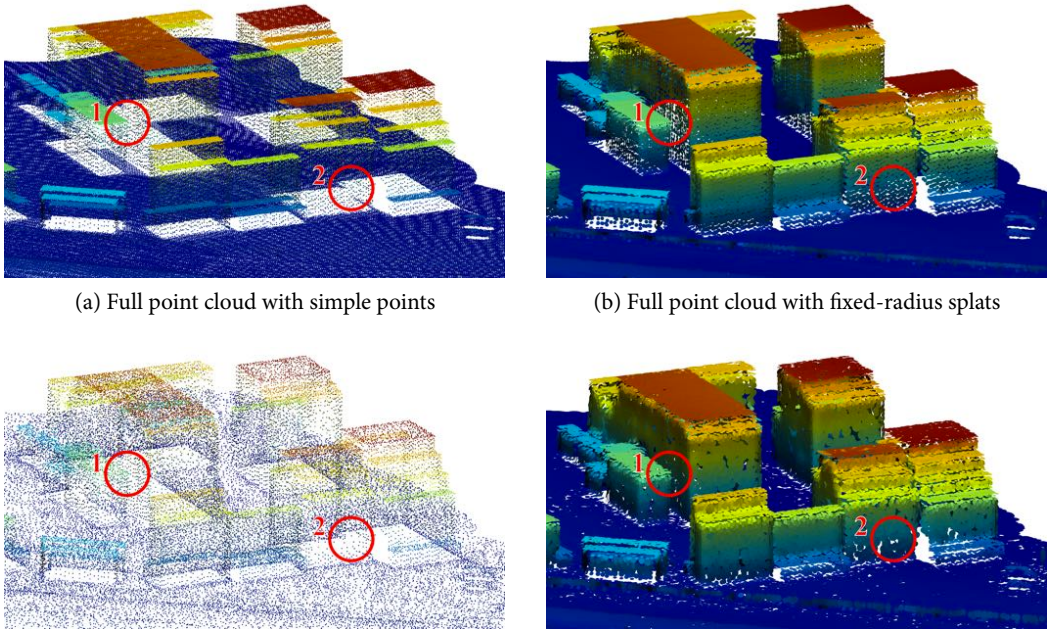


(c) Simplified point (90% of points removed) cloud with simple points



(d) Simplified point (90% of points removed) cloud with lfs-radius splats

Figure 16: Visualisation results for the Mountain dataset ($\epsilon = 0.4$).



(a) Full point cloud with simple points

(b) Full point cloud with fixed-radius splats

(c) Simplified point (90% of points removed) cloud with simple points

(d) Simplified point (90% of points removed) cloud with lfs-radius splats

Figure 17: Visualisation results for the Urban dataset ($\epsilon = 0.4$). Note the different point-densities on vertical and horizontal surfaces (marked 1 and 2).

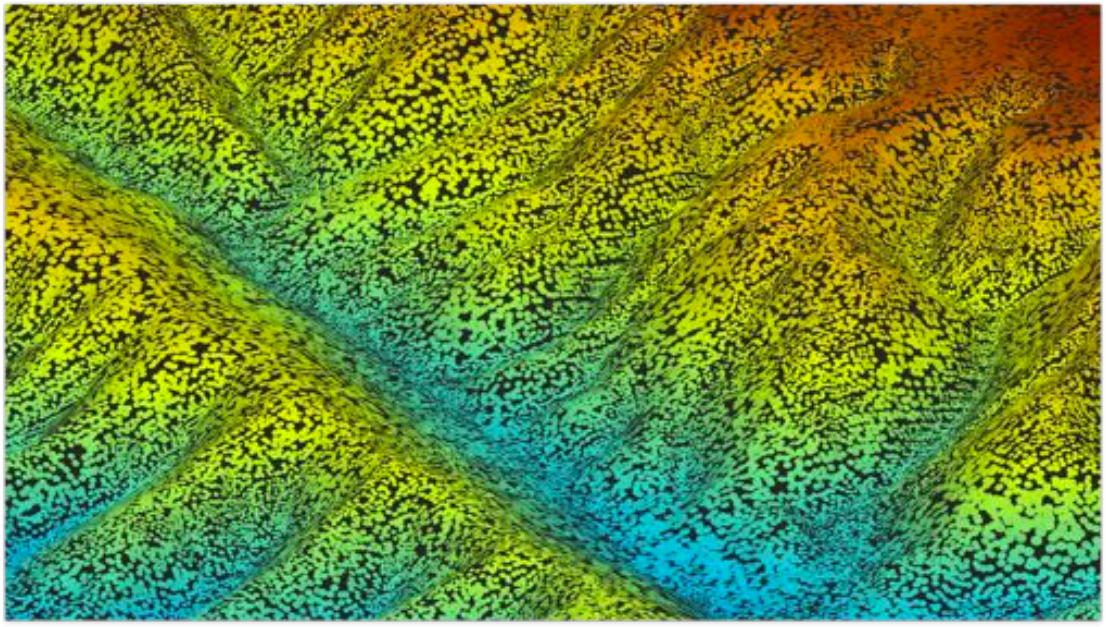


Figure 18: MAT-based simplification and splat-radii. The splat radii in this image are reduced for illustrative purposes.

(a) fixed-sized splats (b) lfs-sized splats

Figure 19: Simplified point cloud ($\epsilon = 0.4$) of mountain dataset.

bution after simplification should be more homogeneous. This may be achieved by enforcing a more grid-like distribution of points during the simplification or by more carefully considering the order in which points are decimated.

While we did not address how to efficiently create and manage different (discrete) levels of simplification for one dataset, we do not foresee any major problems in combining our simplification approach with an octree data-structure (similar to e.g. Kreylos et al. (2008)).

The simple partitioning scheme that we have implemented effectively limits the amount of memory required to complete the computation of massive datasets. The main limitation of this approach is that the number of points in a tile is still bounded by the amount of available memory. Also, the size of a tile should be at least as large as the buffer radius. Thus for extremely dense datasets (several hundreds points per square meter) it may no longer be feasible to process reasonably sized tiles. The point streaming techniques introduced by Isenburg et al. (2006) are a possible solution to this problem, although these techniques require a good spatial coherence of the input point cloud.

7 Conclusions

In this paper we have made three main contributions. First, we have shown that a usable MAT approximation can be obtained from a massive LiDAR point cloud. To make that possible we have extended the ball-shrinking algorithm to approximate a *robust* MAT from a noisy input point cloud. Second we have proposed an out-of-core partitioning scheme to approximate the MAT for massive datasets that do not fit a computer's main memory. And third, we have demonstrated one potential application of the MAT in visualisation of LiDAR point clouds, by using it to perform geometry-aware simplification and splat-radius determination. As a result we have obtained a visualisation in which it is easier to perceive depth and structure in the rendered LiDAR point cloud, while rendering only a fraction of the full point cloud.

Acknowledgements

This research was financially supported by the Dutch Technology Foundation STW, which is part of the Netherlands Organisation for Scientific Research (NWO), and which is partly funded by the Ministry of Economic Affairs (project code: 12217) We also would like to thank the municipality of Rotterdam for making their LiDAR dataset available to us.

References

Amenta N, Bern M, and Kamvysselis M (1998). A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 415–421. ACM, New York, NY, USA.

Amenta N, Choi S, and Kolluri RK (2001). The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, SMA '01, pages 249–266. ACM, New York, NY, USA.

Attali D, Boissonnat JD, and Edelsbrunner H (2009). Stability and computation of medial axes-a state-of-the-art report. *Mathematical foundations of scientific visualization, computer graphics, and massive data exploration*, pages 109–125.

Attali D and Montanvert A (1997). Computing and Simplifying 2D and 3D Continuous Skeletons. *Computer Vision and Image Understanding*, 67(3):261 – 273.

Botsch M and Kobbelt L (2003). High-quality point-based rendering on modern GPUs. In *Computer Graphics and Applications, 2003. Proceedings. 11th Pacific Conference on*, pages 335–343. IEEE.

Chaussard J, Couprie M, and Talbot H (2011). Robust skeletonization using the discrete λ-medial axis. *Pattern Recognition Letters*, 32(9):1384 – 1394.

Dey TK, Giesen J, and Hudson J (2001). Decimating samples for mesh simplification. In *Proc. 13th Canadian Conf. Comput. Geom*, pages 85–88.

Dey TK and Zhao W (2004). Approximate medial axis as a voronoi subcomplex. *Computer-Aided Design*, 36(2):195–202.

Dykes J, MacEachren A, and Kraak M (2005). Beyond Tools: Visual Support for the Entire Process of GIScience. *Exploring Geovisualization*, page 83.

Elseberg J, Borrmann D, and Nüchter A (2013). One billion points in the cloud – an octree for efficient processing of 3D laser scans. *ISPRS Journal of Photogrammetry and Remote Sensing*, 76(0):76 – 88. ISSN 0924-2716. Terrestrial 3D modelling.

Fewtrell TJ, Duncan A, Sampson CC, Neal JC, and Bates PD (2011). Benchmarking urban flood models of varying complexity and scale using high resolution terrestrial lidar data. *Physics and Chemistry of the Earth, Parts A/B/C*, 36(7–8):281 – 291. ISSN 1474-7065. Recent Advances in Mapping and Modelling Flood Processes in Lowland Areas.

Foskey M, Lin MC, and Manocha D (2003). Efficient computation of a simplified medial axis. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, SM '03, pages 96–107. ACM, New York, NY, USA.

Garland M and Heckbert PS (1995). Fast polygonal approximation of terrain and height fields. Technical Report CMU-CS-95-181, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA.

Gross M and Pfister H (2011). *Point-based graphics*. Morgan Kaufmann.

Gschwandtner M, Kwitt R, Uhl A, and Pree W (2011). BlenSor: Blender Sensor Simulation Toolbox Advances in Visual Computing. volume 6939 of *Lecture Notes in Computer Science*, chapter 20, pages 199–208. Springer Berlin / Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-24030-0. doi:10.1007/978-3-642-24031-7_20.

Haala N and Rothermel M (2012). Dense multi-stereo matching for high quality digital elevation models. *Photogrammetrie, Fernerkundung, Geoinformation (PFG)*, 2012(4):331–343.

Hesselink W, Visser M, and Roerdink J (2005). Euclidean skeletons of 3d data sets in linear time by the integer medial axis transform. In

Ronse C, Najman L, and Decencière E, editors, *Mathematical Morphology: 40 Years On*, volume 30 of *Computational Imaging and Vision*, pages 259–268. Springer Netherlands.

Isenburg M, Liu Y, Shewchuk JR, and Snoeyink J (2006). Streaming computation of Delaunay triangulations. *ACM Transactions on Graphics*, 25(3):1049–1056.

Jalba AC, Kustra J, and Telea AC (2012). Surface and curve skeletonization of large 3D models on the GPU. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(6):1495–1508.

Koenig K, Höfle B, Hämmerle M, Jarmer T, Siegmann B, and Lilienthal H (2015). Comparative classification analysis of post-harvest growth detection from terrestrial lidar point clouds in precision agriculture. *ISPRS Journal of Photogrammetry and Remote Sensing*, 104:112–125.

Kovač B and Žalik B (2010). Visualization of LIDAR datasets using point-based rendering technique. *Computers & Geosciences*, 36(11):1443 – 1450. ISSN 0098-3004.

Kraus K and Pfeifer N (1998). Determination of terrain models in wooded areas with airborne laser scanner data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 53(4):193–203.

Kreylos O, Bawden G, and Kellogg L (2008). Immersive visualization and analysis of LiDAR data. *Advances in Visual Computing*, pages 846–855.

Krüger T and Meinel G (2008). Using raster DTM for dike modelling. In van Oosterom P, Zlatanova S, Penninga F, and Fendel E, editors, *Advances in 3D Geoinformation Systems*, Lectures Notes in Geoinformation and Cartography, chapter 6, pages 101–113. Springer Berlin Heidelberg.

Kumler MP (1994). An intensive comparison of triangulated irregular networks (TINs) and digital elevation models (DEMs). *Cartographica*, 31(2).

Lee J (1989). *A drop heuristic conversion method for extracting irregular network for digital elevation models*. ASPRS/ACSM.

Li Z, Zhu Q, and Gold CM (2005). *Digital Terrain Modeling—Principles and Methodology*. CRC Press.

Ma J, Bae SW, and Choi S (2012). 3D medial axis point approximation using nearest neighbors and the normal field. *The Visual Computer*, 28(1):7–19.

Mallet C and Bretar F (2009). Full-waveform topographic lidar: State-of-the-art. *ISPRS Journal of Photogrammetry and Remote Sensing*, 64(1):1–16.

National Science Foundation (2005). B4 Project - Southern San Andreas and San Jacinto Faults. http://dx.doi.org/10.5069/G97P8W9T.

Pauly M, Gross M, and Kobbelt L (2002). Efficient simplification of point-sampled surfaces. In *Visualization, 2002. VIS 2002. IEEE*, pages 163–170. IEEE.

Peters R (2015). masbpy. https://github.com/tudelft3d/masbpy.

Richter R and Döllner J (2010). Out-of-core real-time visualization of massive 3D point clouds. In *Proceedings of the 7th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, pages 121–128. ACM.

Richter R and Döllner J (2014). Concepts and techniques for integration, analysis and visualization of massive 3D point clouds. *Computers, Environment and Urban Systems*, 45:114–124.

Rottensteiner F (2003). Automatic generation of high-quality building models from lidar data. *IEEE Computer Graphics and Applications*, 23(6):42–50.

Scheiblauer C (2014). *Interactions with Gigantic Point Clouds*. Ph.D. thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria.

Siddiqi K and Pizer SM (2008). *Medial representations: mathematics, algorithms and applications*, volume 37. Springer.

Snyder GI (2013). The benefits of improved national elevation data. *Photogrammetric Engineering and Remote Sensing*, 79(2).

Sobiecki A, Yasan HC, Jalba AC, and Telea AC (2013). Qualitative comparison of contraction-based curve skeletonization methods. In *Mathematical Morphology and Its Applications to Signal and Image Processing*, pages 425–439. Springer.

Ujang U, Anton F, and Abdul Rahman A (2013). Unified data model of urban air pollution dispersion and 3D spatial city model: Groundwork assessment towards sustainable urban development for Malaysia. *Journal of Environmental Protection*, 4(7):701–712.

van Leeuwen M and Nieuwenhuis M (2010). Retrieval of forest structural parameters using lidar remote sensing. *European Journal of Forest Research*, 129(4):749–770. ISSN 1612-4669. doi:10.1007/s10342-010-0381-4.

Wand M, Berner A, Bokeloh M, Jenke P, Fleck A, Hoffmann M, Maier B, Staneker D, Schilling A, and Seidel HP (2008). Processing and interactive editing of huge point clouds from 3D scanners. *Computers & Graphics*, 32(2):204 – 220. ISSN 0097-8493.

Wimmer M and Scheiblauer C (2006). Instant points: Fast rendering of unprocessed point clouds. In *SPBG*, pages 129–136. Citeseer.