Eurographics Workshop on Urban Data Modelling and Visualisation (2015)
F. Biljecki and V. Tourre (Editors)

# Visibility analysis in a point cloud based on the medial axis transform

R. Peters, H. Ledoux and F. Biljecki

3D geoinformation group, Delft University of Technology, the Netherlands

**Abstract**

*Visibility analysis is an important application of 3D GIS data. Current approaches require 3D city models that are often derived from detailed aerial point clouds. We present an approach to visibility analysis that does not require a city model but works directly on the point cloud. Our approach is based on the medial axis transform, which models the urban environment as a union of balls, which we then use to construct a depthmap that is used for point visibility queries. As we demonstrate through our experiments on a real-world aerial LiDAR point cloud, the main benefits of our approach are 1) it is robust to noise, irregular sampling and holes of typical aerial LiDAR datasets, 2) it gives visibility results that are significantly more accurate than the often highly generalised city models, and 3) it is a simple algorithm that is easy to parallelise.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Picture/Image Generation—Visible line/surface algorithms I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—

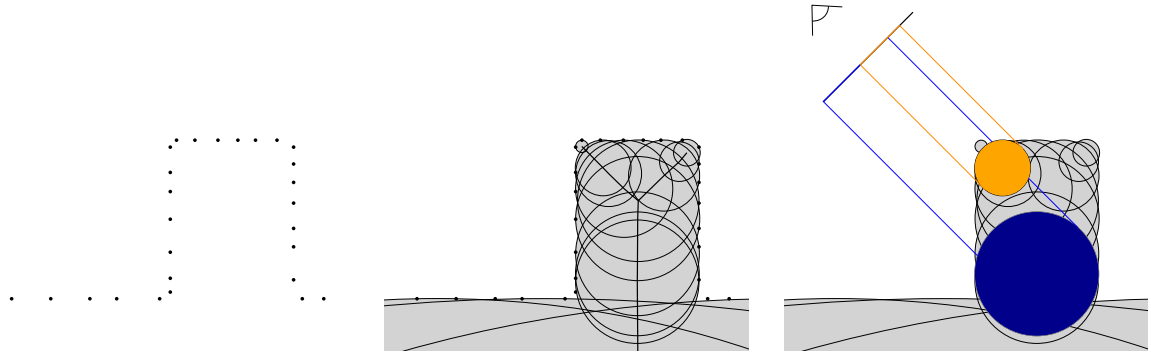
## 1. Introduction

Visibility analysis is one of the prominent use cases of 3D GIS data, since this provides information about spatial relations and potential obstacles in the line of sight between two points in space. For instance, such analyses have been done in estimating the visibility of a landmark [BRKM10], and in finding the optimal location to place a surveillance camera [YYKK15]. An important variant of the visibility analysis is the estimation of shadows, since the position of the sun is variable and it is located at a practically infinite distance [BLS16]. Shadow analysis has gained importance in several disciplines. For instance, shadows are important to account for the loss of the photovoltaic potential [NP12, EMDN15], for determining solar envelopes [Kno03], for assessing the value of real estate [HJMH13], and for estimating the thermal comfort [HLM11, YS94]. Shadows are also critical in geovisualisation [Lov03].

Visibility analysis is usually performed on a 3D city model, ie a geometric model that was reconstructed from elevation information, such as airborne LiDAR point clouds, and 2D datasets (eg building footprints from a topographic map). The analysis involves testing if a line of sight (ray) intersect a face in the dataset, usually with algorithms developed in the computer graphics domain, eg [MT97]. However, the creation and maintenance of 3D city models often involves manual labour and typically results in a generalised version of the city that only contains the terrain and the buildings [ASVJT09, Rot03, BLSZ14], and sometimes other man-made objects such as roads, overpasses, and trees [OE10]. Despite the fact that many 3D city models are constructed from very dense points clouds which practically contain all urban features, many of these details are lost in the final city model. One cause is that many, especially automatically generated city models are in fact 2.5D, which means it is not possible to model truly 3D features such as balconies and trees. And even though a number of algorithms for 3D surface reconstruction have been proposed and implemented (see for instance [ACK01], [KSO04] and [DG06]), these have several assumptions on the input point cloud, which usually come from close range laser-scanners and are therefore not suitable for eg airborne LiDAR point clouds that are sampled sparsely, have irregular sampling density and often contain significant noise and holes. Hence, despite the availability of highly detailed airborne point clouds, visibility analysis on a derived city model typically deviates significantly from reality. In this research we attempt to skip the generation of a 3D city model, and conduct shadow analysis directly on the point cloud. Apart from not having to first generate and store a city model, it yields



DOI: 10.2312/udmv.20151342



(a) Approximating MAT from point cloud (b) Computing depthmap (c) Point visibility querying

Figure 1: Our approach to visibility analysis in three steps.

a more realistic visibility analysis that includes all scanned objects in their true 3D appearance. As we explain in Section 2, and demonstrate in Section 4, our approach is based on the Medial Axis Transform (MAT), which is an alternative skeleton-like shape-descriptor that models an object as a union of balls. We obtain these balls using a robust adaptation of algorithm of [MBC12] and use them to construct a view-dependent depthmap, which is then used to efficiently perform point visibility queries (see also Figure 1).

Because the visibility computations are performed in raster-space our algorithm can run highly parallelised on GPU hardware. We therefore believe that our approach could be a simple, fast, robust and cost-effective way of performing visibility analysis.

## 2. Related work

[KTB07] introduced the hidden point removal operator to determine the visible points in a point cloud as viewed from a given viewpoint by first performing a spherical flipping on the point cloud and then a convex hull computation. The algorithm does not require point normals, and is shown to be useful for shadow mapping and view-dependent surface reconstruction. However, unlike the algorithm that we present in this paper, the hidden point removal operator can only determine the visibility of points that are part of the point cloud itself, which limits its potential applications. [MTSM10] extend the algorithm from [KTB07] for handling noisy point clouds.

[PZVBG00], [SP04] and [KB04] compute visibility for well-sampled and oriented point clouds as part of a point-based rendering pipeline. They render points as splats; disks that are aligned with the point normals. Using these splats a depthmap is computed for the viewport to determine point visibility. However, when the sampling density of the point cloud is low and non-uniform it becomes non-trivial to choose optimal splat radii. Another significant difference with the approach we present in this paper is that we compute a volumetric representation of the sampled surface, whereas a splatting approach can represent only the boundary of the sampled surface. Holes are therefore handled quite differently (see also Figure 7).

[WS05] perform ray-tracing in a point cloud based on an implicit surface representation. It works well for high quality point clouds that are densely sampled.

Finally, [JKT12] implement a rendering pipeline that performs on-screen surface reconstruction by directly rendering interior medial balls. This is somewhat comparable to our approach, but we focus specifically on performing efficient visibility queries for arbitrary query points.

## 3. MAT-based visibiblity computation

Our algorithm involves first constructing the Medial Axis Transform (MAT) of the point cloud, a skeleton-like structure where the volume of each object is represented as a union of balls. We use these medial balls to 'block' lines-of-sight from a user-defined viewport to a given set of query points. Whether the line-of-sight to a query point is blocked or not is determined by the use of a depthmap that encodes the distances from the viewport to all visible medial balls.

For the sake of simplicity we assume an orthographic projection and consider only point visibility queries. However, it is a straightforward task to extend our algorithm to work with a perspective projection and more complex query objects such as triangular meshes.

### 3.1. Approximating the MAT of a point cloud

The Medial Axis Transform (MAT) is formally defined as the set of maximal balls tangent to the surface of shape at two or more points. The centers of these balls, commonly referred to as *medial balls*, form a medial skeletal structure. Here we are primarily interested in the union of medial

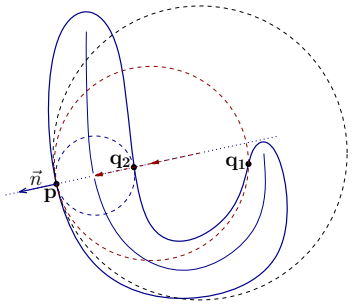



Figure 2: The shrinking ball algorithm.

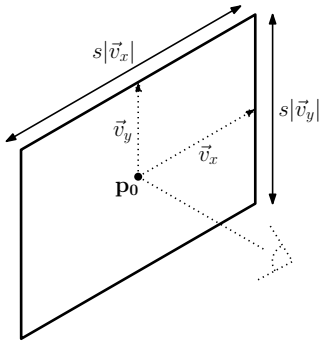



Figure 3: Parameters that define the viewport

balls, which corresponds to the volume of the shape (see Figure 1a).

We use an adapted version of the shrinking ball algorithm introduced by [MBC12] to approximate a point approximation of the MAT from an oriented input point cloud. The shrinking ball algorithm is illustrated in Figure 2. For each point $\mathbf{p}$ a medial ball is found by iteratively shrinking a very large ball that is centered along the point's normal $\vec{n}$. At each iteration a point $\mathbf{q}$ is found that is nearest to the ball's center and the ball is shrunk such that it touches both $\mathbf{p}$ and $\mathbf{q}$ and remains centered along $\vec{n}$. The iteration converges when the ball's interior is empty and there are no closer points to its center than $\mathbf{p}$ and $\mathbf{q}$, which is in effect how a medial ball is defined.

When the normals point outward, such as in Figure 2, the *interior* MAT is obtained. With flipped normals that point inward, the *exterior* MAT is obtained that occupies the complement of the space that is occupied by the interior MAT. For this paper we are only interested in the interior MAT.

To improve the performance of the shrinking ball algorithm for typical LiDAR point clouds that contain significant noise (unlike the pristine point clouds used by [MBC12] and [JKT12]), we extended the algorithm with a number of heuristics that will stop the shrinking of a ball prematurely based on the progression of the separation angle, ie the angle $\angle\mathbf{pcq}$ where $\mathbf{c}$ denotes the ball's center. The ball shrinking of a given point $\mathbf{p}$ is halted if either the separation angle of the initial ball is below a threshold $t_a$ or if the separation angle of a succeeding ball drops below a second threshold $t_b < t_a$. We choose the last ball that does not violate both thresholds as the approximate medial ball for $\mathbf{p}$ (see [PL16] for more details).

Ultimately, the extended shrinking ball algorithm is simple, fast, robust to noise and easy to parallelize (see also [MBC12] and [JKT12]) which makes it a good choice for approximating the MAT of large LiDAR point clouds.

### 3.2. Depthmap computation

Prior to performing the visibility queries we must generate a depthmap of the the interior medial balls. The depthmap is computed for a viewport that is described by a point $\mathbf{p_0}$ to fix its position, two vectors $\vec{v}_x$ and $\vec{v}_y$ to fix its orientation and size in model space and a scalar $s$ that scales model units to the pixels on the screen (see Figure 3).

Computing the depthmap is a fairly straightforward process that involves first projecting each medial ball center, rasterising the ball to the viewport and finally performing a depth test for each pixel of the rasterised ball. Figure 1b illustrates this process.

Algorithm 1 which updates the depthmap for one medial ball first projects the ball center to the viewport. Then for each pixel in the ball's projected image it computes the depth, and performs a depthtest. When a depth test succeeds (ie the depth of the ball is smaller than the current pixel depth), the pixel in the depthmap is updated.

**Algorithm 1** WRITEBALL

**Input:** a ball with center $\mathbf{c}$ and radius $r$,the viewport parameter $s$ and depthmap $D$
**Output:** $D$ is updated with the depths of ball $(\mathbf{c}, r)$

```
c_s ← PROJECTPOINT (c)
for integer x from −rs to +rs do
    for integer y from −rs to +rs do
        h ← √(x² + y²)
        if h smaller than rs then
            d′ ← c_s.z − (rs − h)
            d ← D[c_s.x + x, c_s.y + y]
            if d′ smaller than d then
                D[c_s.x + x, c_s.y + y] ← d′
            end if
        end if
    end for
end for
```

Prior to calling WRITEBALL for each medial ball, the depthmap is initialised with an infinite depth for each pixel.

### 3.3. Point visibility queries

After the depthmap has been computed, we use Algorithm 2 to perform the point visibility queries, it involves projecting the query point to the viewport, and then comparing its depth to the corresponding depth in the depthmap (similar to [Wil78]). As illustrated in figure 1c the query point is visible only if its depth test succeeds. The depth test will fail for query points that are behind any medial ball as seen from the viewport.

**Algorithm 2** QUERYPOINT

**Input:** a querypoint $\mathbf{q_m}$ in model coordinates, depthmap $D$
**Output:** whether $\mathbf{q_m}$ is visible or not

1: $\mathbf{q_s} \leftarrow$ PROJECTPOINT ($\mathbf{q_m}$)
2: $d \leftarrow D[\mathbf{q_s}.x, \mathbf{q_s}.y]$
3: **if** $\mathbf{q_s}.z$ smaller than $d$ **then**
4: $q_m$ is visible
5: **else**
6: $q_m$ is not visible
7: **end if**

## 4. Implementation and Experiments

We have built a prototype implementation of the algorithms we propose in this paper. Our prototype utilises OpenCL for parallel execution of the algorithms listed in Section 3.2 and 3.3.

We ran our experiments on two different datasets:

1. A simple artificially generated point cloud with its points and normals derived from a triangular mesh (2 690 points), and
2. an airborne LiDAR dataset of a housing block in Zagreb, Croatia (24 647 points).

For the latter dataset the normals were approximated using principal component analysis of the 6 nearest neighbours of each point. For a good separation of the interior and exterior MAT it is important that the normals are properly oriented. This can be achieved by flipping the normals with respect to the scanner position at the time a point is acquired. However, because this information is not present in our LiDAR dataset we used a city model to properly orient the point normals.

For the visibility queries we randomly generated 1 million query points that are uniformly distributed inside the bounding box of the respective dataset. For both datasets the total computation time (from raw point cloud to point visibility queries) is in the order of a few seconds, when all computations are performed on a quadcore 2.9 GHz Intel Core i5 CPU. From our pseudocodes it can easily be seen that the computation of the depthmap (it must happen once for every viewport) is the most expensive ($O((rs)^2N)$ time, with $N$ the number of medial balls, $r$ the ball radius and $s$ the number of pixels per model unit). However, once the depthmap is computed point visibility queries are extremely fast, since they run in constant time (thus independent of depthmap resolution or size of the dataset).

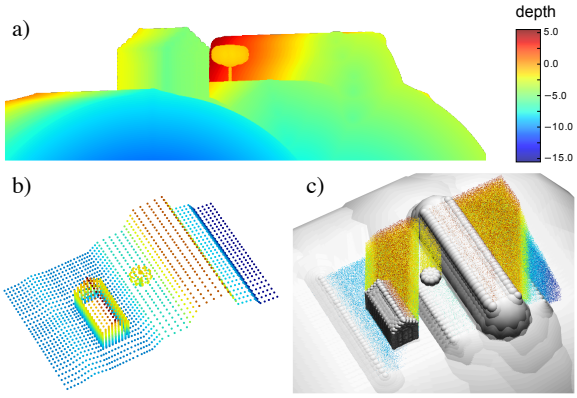



Figure 4: Artificially generated dataset. (a) Depthmap for viewport, (b) top-down view of pointcloud and (c) point visibility from viewport with medial balls

Figure 4 shows the results for the artificial dataset. From the depthmap (4a) it is clear that three-dimensional features in the point cloud (4b) such as the tree in the center are accurately modeled by the medial balls, given a sufficiently dense and complete sampling. The invisible or 'shadowed' points (inside the bounding box of the point cloud) as seen from the viewport (4a) are depicted in Figure 4c.

Figures 5, 6 and 7 illustrate the results for the LiDAR dataset. First, note the difference in sampling density between horizontal and the vertical surfaces in the dataset (5a,c). Despite the low number of samples on the vertical segments we are still able to model the building facades without holes that would distort our visibility analysis (5d,e). Also note how various sparsely sampled details such as dormer windows and chimneys are modelled by the medial balls, and how that affects the visibility analysis (5b and 6). In case of a complete lack of samples for surfaces such as the right side of the roofs in Figure 7 the dimensions of an object may be wrongly represented due to protruding medial balls. Whether this leads to realistic results in the visibility analysis depends on the actual (unknown) shape of the roof. We do believe that this behaviour is preferable to being able to see through the right side of the roof to the backside of the left side of the roof. Finally, from 7 it can be seen that trees can also be handled by our approach. We must notice however that especially in the case of trees the orientation of point normals becomes rather ambiguous, which leads to a fuzzy definition of what is an interior or an exterior medial ball, which can affects the visibility queries.

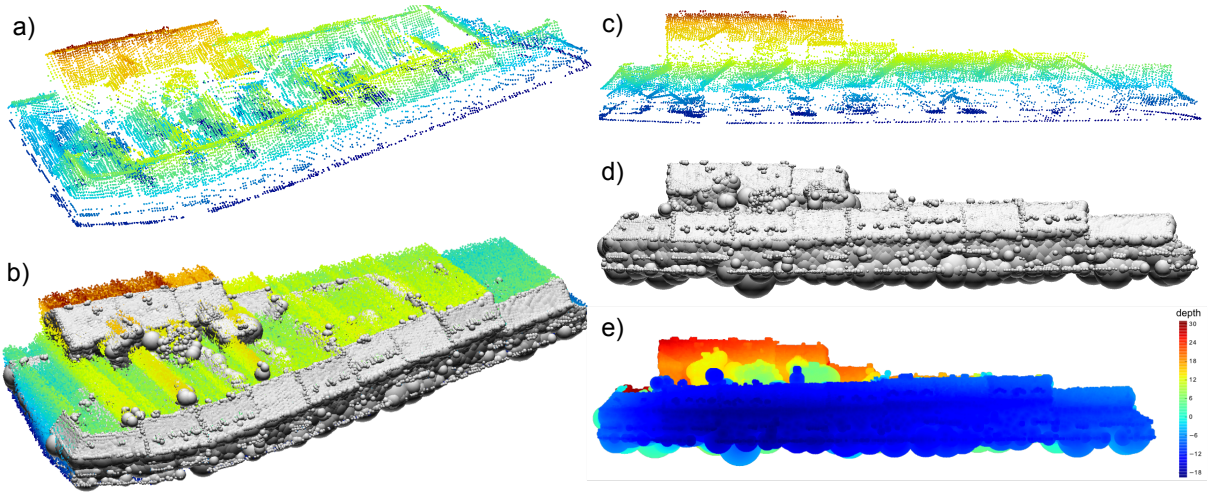



Figure 5: Aerial LiDAR point cloud dataset. Top-down view of point cloud (a) and point visibility with medial balls (b). Viewport view with point cloud (c), medial balls (d) and depthmap (e).

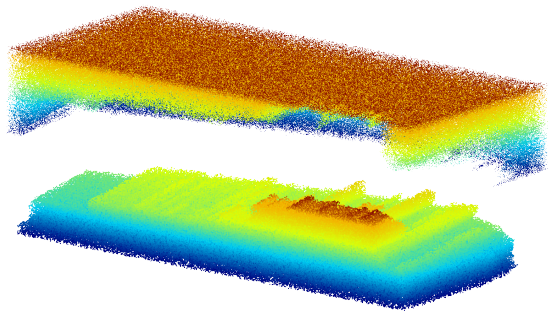


Figure 6: Visible (top) and invisible (bottom) points for viewport and LiDAR dataset of Figure 5

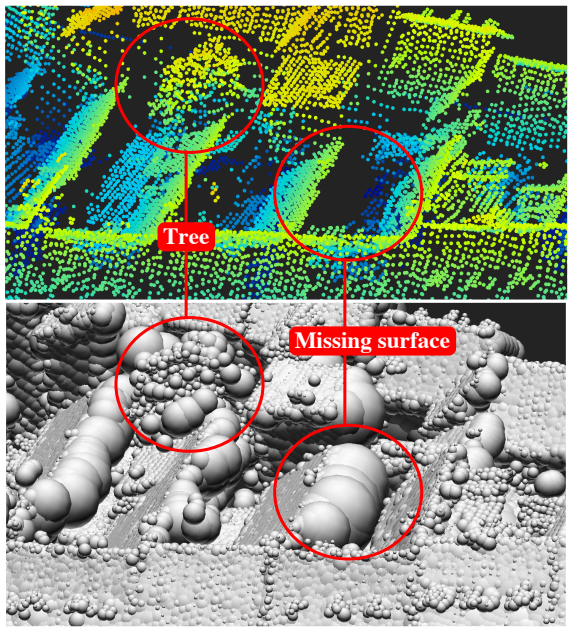



Figure 7: Detail view of LiDAR dataset for point cloud (top) and medial balls (bottom)

## 5. Conclusion and future work

We have introduced, implemented and demonstrated a new approach to do visibility analysis in urban scenes directly on a point cloud, thus without the need of an overly simplified intermediate 3D city model. Because the visibility analysis is performed in the raster domain, our algorithm can be implemented to exploit the computing power of parallel computing devices such as GPUs.

Our experiments show that our approach can be successfully applied to a typical airborne LiDAR dataset with holes, non-homogeneous point density and noise. We notice that the MAT on which our algorithm is based depends on properly oriented point normals, which may be ambiguous to define for vegetation. However, this did not lead to abnormalities in our experimental results. We also observe that our algorithm deals rather successfully with missing walls and roof sides in the LiDAR point cloud. This is a major advantage over existing point-based visibility approaches such as splatting.

The most obvious extensions to our visibility algorithm are 1) perspective projection (see for instance [MM12]), 2) handling more complex query object such as triangular meshes.

**Acknowledgments**

This research is 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 codes: 11300 and 12217)

The authors gratefully acknowledge the test dataset shared by the company Geofoto (Zagreb, Croatia).

## References

[ACK01] AMENTA N., CHOI S., KOLLURI R. K.: The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications* (New York, NY, USA, 2001), SMA '01, ACM, pp. 249–266. 1

[ASVJT09] ALEXANDER C., SMITH-VOYSEY S., JARVIS C., TANSEY K.: Integrating building footprints and LiDAR elevation data to classify roof structures and visualise buildings. *Computers, Environment and Urban Systems 33*, 4 (2009), 285–292. 1

[BLS16] BILJECKI F., LEDOUX H., STOTER J.: Does a finer level of detail of a 3D city model bring an improvement for estimating shadows? In *Advances in 3D Geoinformation*. 2016, pp. 1–15. 1

[BLSZ14] BILJECKI F., LEDOUX H., STOTER J., ZHAO J.: Formalisation of the level of detail in 3D city modelling. *Computers, Environment and Urban Systems 48* (Nov. 2014), 1–15. 1

[BRKM10] BARTIE P., REITSMA F., KINGHAM S., MILLS S.: Advancing visibility modelling algorithms for urban environments. *Computers, Environment and Urban Systems 34*, 6 (Nov. 2010), 518–531. 1

[DG06] DEY T. K., GOSWAMI S.: Provable surface reconstruction from noisy samples. *Computational Geometry 35*, 1–2 (2006), 124–141. 1

[EMDN15] EICKER U., MONIEN D., DUMINIL E., NOUVEL R.: Energy performance assessment in urban planning competitions. *Applied Energy 155* (Oct. 2015), 323–333. 1

[HJMH13] HELBICH M., JOCHEM A., MÜCKE W., HÖFLE B.: Boosting the predictive accuracy of urban hedonic house price models through airborne laser scanning. *Computers, Environment and Urban Systems 39*, C (May 2013), 81–92. 1

[HLM11] HWANG R.-L., LIN T.-P., MATZARAKIS A.: Seasonal effects of urban street shading on long-term outdoor thermal comfort. *Building and Environment 46*, 4 (Apr. 2011), 863–870. 1

[JKT12] JALBA A. C., KUSTRA J., TELEA A. C.: Surface and curve skeletonization of large 3D models on the GPU. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 35*, 6 (June 2012), 1495–1508. 2, 3

[KB04] KOBBELT L., BOTSCH M.: A Survey of Point-based Techniques in Computer Graphics. *Computer Graphics 28*, 6 (Dec. 2004), 801–814. 2

[Kno03] KNOWLES R. L.: The solar envelope: its meaning for energy and buildings. *Energy and Buildings 35*, 1 (Jan. 2003), 15–25. 1

[KSO04] KOLLURI R., SHEWCHUK J. R., O'BRIEN J. F.: Spectral surface reconstruction from noisy point clouds. In *Proceedings Symposium on Geometry Processing* (Nice, France, 2004), pp. 11–21. 1

[KTB07] KATZ S., TAL A., BASRI R.: Direct visibility of point sets. *ACM Transactions on Graphics 26*, 3 (2007). 2

[Lov03] LOVETT A.: GIS-based visualisation of rural landscapes: defining 'sufficient' realism for environmental decision-making. *Landscape and Urban Planning 65*, 3 (Oct. 2003), 117–131. 1

[MBC12] MA J., BAE S. W., CHOI S.: 3D medial axis point approximation using nearest neighbors and the normal field. *The Visual Computer 28*, 1 (2012), 7–19. 2, 3

[MM12] MARA M., MCGUIRE M.: 2d polyhedral bounds of a clipped, perspective-projected 3d sphere. *JCGT. in submission* (2012). 5

[MT97] MÖLLER T., TRUMBORE B.: Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools 2*, 1 (1997), 21–28. 1

[MTSM10] MEHRA R., TRIPATHI P., SHEFFER A., MITRA N. J.: Visibility of noisy point cloud data. *Computers & Graphics 34*, 3 (2010), 219–230. 2

[NP12] NGUYEN H. T., PEARCE J. M.: Incorporating shading losses in solar photovoltaic potential assessment at the municipal scale. *Solar Energy 86*, 5 (May 2012), 1245–1260. 1

[OE10] OUDE ELBERINK S.: *Acquisition of 3D topgraphy: automated 3D road and building reconstruction using airborne laser scanner data and topographic map*. PhD thesis, University of Twente Faculty of Geo-Information and Earth Observation (ITC), 2010. 1

[PL16] PETERS R., LEDOUX H.: *Visualisation of massive and noisy point clouds based on the medial axis transform*. In submission: Computers & Geosciences, 2016. 3

[PZVBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 335–342. 2

[Rot03] ROTTENSTEINER F.: Automatic generation of high-quality building models from lidar data. *IEEE Computer Graphics and Applications 23*, 6 (2003), 42–50. 1

[SP04] SAINZ M., PAJAROLA R.: Point-based rendering techniques. *Computers & Graphics 28*, 6 (2004), 869 – 879. 2

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1978), SIGGRAPH '78, ACM, pp. 270–274. 4

[WS05] WALD I., SEIDEL H.-P.: Interactive ray tracing of point-based models. In *Proceedings of the Second Eurographics / IEEE VGTC Conference on Point-Based Graphics* (Aire-la-Ville, Switzerland, Switzerland, 2005), SPBG'05, Eurographics Association, pp. 9–16. 2

[YS94] YEZIORO A., SHAVIV E.: Shading: A design tool for analyzing mutual shading between buildings. *Solar Energy 52*, 1 (Jan. 1994), 27–37. 1

[YYKK15] YAAGOUBI R., YARMANI M., KAMEL A., KHEMIRI W.: HybVOR: A Voronoi-Based 3D GIS Approach for Camera Surveillance Network Placement. *ISPRS International Journal of Geo-Information 4*, 2 (May 2015), 754–782. 1