Assessing the sensibility of intervisibility on the quality of 3D geometry

Darshan Venkatarayappa, Teng Wu, Bruno Vallet

Univ Gustave Eiffel, Géodata Paris, IGN, LASTIG, F-77454 Marne-la-Vallée, France

Keywords: Intervisibility, Ray-tracing, Depth rendering, 3D modeling, digital twin.

Abstract

This work explores a new evaluation framework for 3D Model Quality Assessment using 3D intervisibility, a critical concept in 3D spatial analysis. In this work we will consider a high-quality LiDAR ground-truth 3D model and lower quality (dense matching and decimated) versions of it. Then we run the same intervisibility analysis on all of them and compare the results. This will allow us to evaluate the impact of geometric quality on intervisibility analysis. This analysis is useful for anyone using 3D data for simulations, as it indicates what data quality they actually need to purchase or produce for their specific use case. Ultimately, the goal of this work is to see how much the quality of the 3D model affects intervisibility results.

1. Introduction

The digital twin is an increasingly popular paradigm for describing the interactions between real objects and their digital counterparts. In the context of geographical information, a digital twin of a real scene—at local, regional, or even country-scale—consists of a geometrical (3D) description of the scene and its constituent objects, alongside simulations that model the behavior of various processes occurring within it. The results of these simulations are instrumental in making informed decisions and acting upon the physical scene. However, the implementation of digital twins faces a considerable quality challenge: a digital twin is never a perfectly faithful clone of its real counterpart, and approximations, if not errors, may exist that cause uncertainty; and 1) this uncertainty is often not provided by the data producer or is not trustworthy and 2) the data consumer (simulation operator) might not know how uncertainty on the input geometry impacts the results of his simulation. In order to address this fundamental issue, we propose to study the impact of the quality of 3D scene representation on a very simple form of simulation: intervisibility.

In the realm of 3D spatial analysis, intervisibility refers to the computational assessment of whether two points or regions within a three-dimensional scene environment can maintain a direct line of sight (LOS) without obstruction by intervening geometric elements, such as terrain, buildings, or other surfaces. This concept, rooted in computational geometry and visibility determination algorithms, involves tracing parametric line segments between viewpoints and testing for intersections with 3D models such as meshes, multi-view stereo models (MVS), or point clouds (Bittner and Wonka, 2003). Unlike 2D visibility, which often simplifies to planar graphs, 3D intervisibility accounts for volumetric occlusions, elevation variations, and complex topologies, making it essential to simulate real-world perceptual dynamics (De Floriani and Magillo, 1999). Techniques range from basic ray-triangle intersections to advanced spatial partitioning methods, such as bounding volume hierarchies (BVH) or octrees, which optimize efficiency in large-scale datasets by minimizing unnecessary checks (Wald et al., 2014). The precision of these computations is highly dependent on the quality of the underlying 3D data, where degradations such as noise, incomplete reconstructions, or low resolution can introduce errors in occlusion detection. As digital representations of environments become increasingly prevalent in fields like geographic information systems (GIS) and computer graphics, understanding intervisibility enables precise modeling of spatial relationships, fostering innovations in environmental simulation and decision-making processes .

The applications of 3D intervisibility span diverse domains, highlighting its versatility in addressing practical challenges. In urban planning, it facilitates the evaluation of visual impacts in built environments, such as analyzing sightlines from buildings to assess aesthetic quality, exposure to sunlight, or obstruction of the skyline in multilevel structures such as atriums (Teller and Séquin, 1991). For example, planners use visibility graphs to optimize green space placement or mitigate visual clutter in dense cities, ensuring equitable access to views and natural light (Batty, 2001). In robotics and autonomous systems, intervisibility is crucial for path planning and Simultaneous Localization and Mapping (SLAM), where robots must navigate cluttered 3D spaces by dynamically assessing obstacle-free routes, as demonstrated by drone operations or self-driving vehicles that rely on point cloud data for real-time occlusion handling (Oleynikova et al., 2017). Computer graphics leverages it for rendering optimizations, such as culling hidden surfaces to enhance performance in virtual reality simulations (Cohen-Or et al., 2003). Furthermore, in military and defense contexts, intervisibility informs strategic positioning, such as determining optimal surveillance points or coverage gaps in terrain models (Franklin and Ray, 1994). These applications underscore how intervisibility not only improves operational efficiency, but also supports safety-critical decisions in dynamic environments.

Despite its broad utility, challenges in 3D intervisibility arise from variations in data quality, which can compromise result reliability in critical applications (Katz and Katz, 2019). High-fidelity models, such as detailed CityGML representations, yield precise results, but real-world datasets often suffer from degradations such as sensor inaccuracies or incomplete geometries, leading to inconsistent visibility assessments (Biljecki et al., 2015). This variability requires robust evaluation frameworks to quantify reliability based on model type and quality. Traditional approaches focus on algorithmic efficiency or specific use cases, yet there is a gap in standardized testing methodologies that correlate data degradation with out-

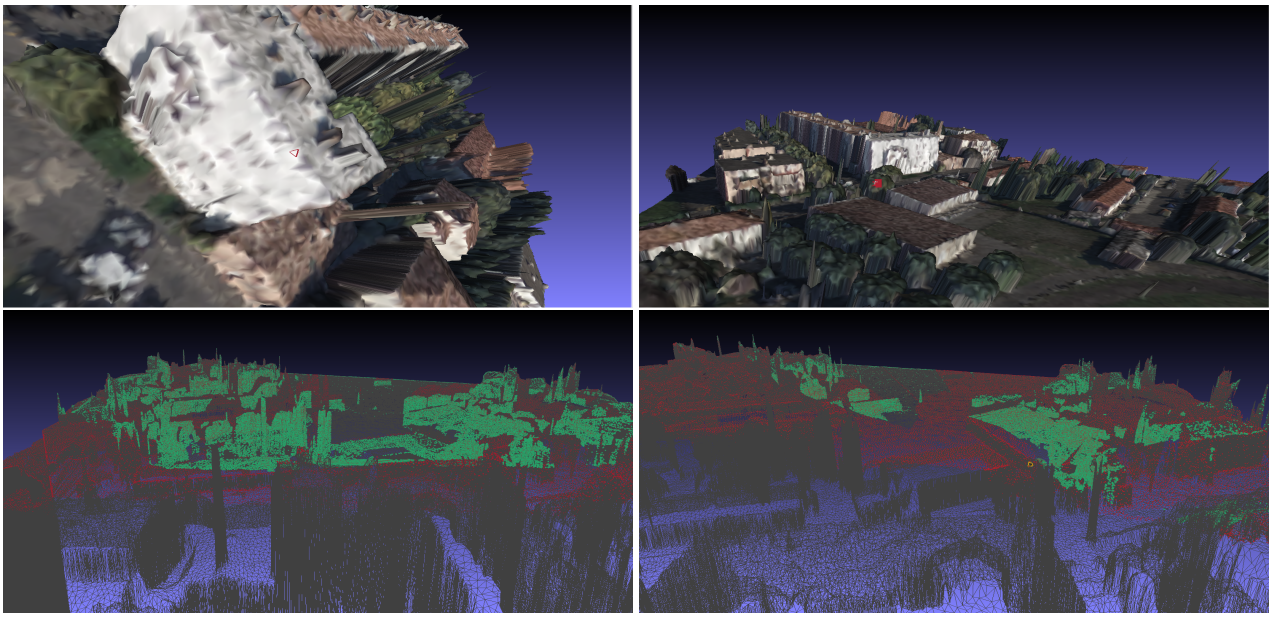


Figure 1. Visualization of visible and non-visible vertices from a given triangle’s barycenter. (Top) The original triangles and its barycenter. (Bottom) The resulting visibility classification across the mesh (Green: Visible, Red: Non-visible).

put accuracy (Nagy and Zhang, 2006). This paper addresses this by proposing a novel evaluation framework that utilizes a ground-truth 3D model and systematically degraded variants to benchmark intervisibility Analyses. The intervisibility computation requires a continuous 3D scene representation, such as Multi-view stereo (MVS) meshes and CityGML models. Because MVS and CityGML can easily be converted to triangle meshes, our benchmark will only address mesh representations. The framework provides guidelines on required data quality for desired reliability levels, empowering users in fields like security and military, urban planning, and robotics to make informed decisions about data acquisition without over-investing in unnecessary precision. In our work, we consider the highest quality data (mesh) LiDAR HD as ground truth, and evaluate the loss in quality: 1) when using a photogrammetric mesh instead of a lidargrammetric one and 2) when applying various levels of decimation to a reference mesh (LiDAR HD).

This paper is structured as follows: Section 2 details our implementation of the ray-tracing and depth-map rendering methodologies for intervisibility analysis. Section 3 then presents our framework for mesh quality validation using these intervisibility techniques, describing the datasets and evaluation metrics. Finally, Section 4 offers concluding remarks.

2. Intervisibility

2.1 Ray-Tracing

When the scene is represented as a triangle 3D mesh—a collection of vertices, edges, and triangle faces that define the surface of objects—the intervisibility problem becomes one of determining if the line segment between the two points intersects any of the mesh’s triangles. This is a classic application of ray-tracing, a family of algorithms designed precisely for this purpose. At its core, the intervisibility problem reduces to a series of ray-scene intersection tests, where a ray cast from one point to another must be checked for intersections with any obstructing geometry.

The ray-tracing paradigm, as formalized for complex scenes, involves casting a ray from the eye through each pixel and finding the closest object blocking the path of that ray. For intervisibility, the ”eye” is one point, the pixel is the second point, and the algorithm needs to only determine if any intersection occurs along the ray segment, not necessarily the closest one, making it a boolean visibility query. The computational cost of this operation is directly proportional to the number of faces in the mesh, necessitating the use of sophisticated acceleration data structures to avoid an exhaustive $O(n)$ check against every polygon for every query.

The Computational Geometry Algorithms Library (CGAL) provides a robust, efficient, and mathematically rigorous framework for implementing such ray-mesh intersection queries, which are central to solving the intervisibility problem. CGAL’s primary strength lies in its adherence to exact geometric predicates, which avoids the robustness issues caused by floating-point arithmetic in naive implementations. A core component for this task is the Axis-Aligned Bounding Box Tree (AABB_tree) data structure, which is specifically designed for efficient ray-shooting and intersection detection against a set of geometric primitives, such as the facets of a polyhedral mesh. AABB_tree provides efficient methods for intersection queries against a set of 3D geometric primitives. It accelerates the intersection queries by building a hierarchy of Axis-Aligned Bounding Boxes (AABBs) around the primitives (CGAL., 2023).

By traversing this hierarchical tree, the algorithm can cull large portions of the mesh that the ray cannot possibly intersect, reducing the average-case complexity from $O(n)$ to $O(\log n)$. The intersection query itself is typically performed using a function such as `AABB_tree::do_intersect()` or `AABB_tree::any_intersection()`, which returns a Boolean result indicating whether an intersection exists. This function is the direct computational tool for an intervisibility check. Additionally, we only consider 3D points whose normals are in the direction of the ray.

The primary drawbacks of using ray-tracing with CGAL for in-

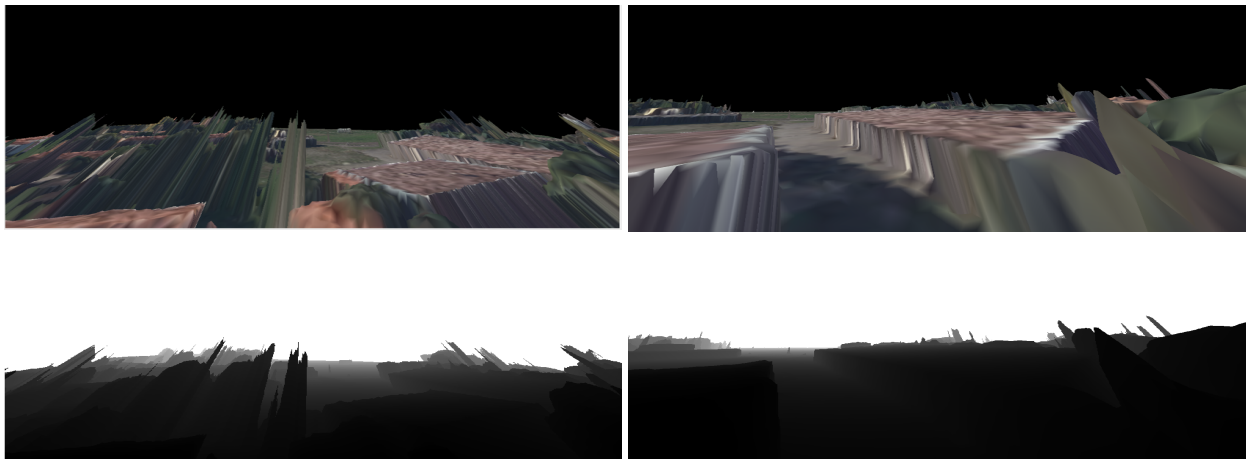


Figure 2. Rendered color image (top) and corresponding depth map (bottom) from a camera positioned at a triangle’s barycenter.

tervisibility analysis are that it is computationally expensive for analyses involving millions of points and the memory overhead of its acceleration data structures. The method’s accuracy is also limited by the quality of the input mesh, where discretization artifacts or flawed geometry can lead to incorrect visibility results. This work analyses visibility from a triangular face on a 3D mesh. For a given triangle, rays are cast from its barycenter to every mesh vertex. A vertex is classified as invisible (red) if its corresponding ray intersects another part of the mesh before reaching it; otherwise it is visible (green). Figure 1 illustrates this, with the top row showing the triangle and barycenter, and the bottom row color-coding the visibility results for the vertices.

2.2 DepthMap Rendering

Intervisibility analysis, the process of determining mutual visibility between points on a 3D surface, can be efficiently approximated using rasterization-based techniques from computer graphics, notably through the generation and comparison of depth maps. Unlike the above computational geometry approaches that rely on explicit ray-scene intersection calculations, this method leverages the highly optimized rendering pipeline of graphics Application Programming Interfaces (APIs) like Open Graphics Library(OpenGL) to answer visibility queries for a large number of points simultaneously. The core principle involves treating the problem as one of shadow mapping, where a point is visible from a given viewpoint if it is not in shadow. The depth map, or Z-buffer, generated by OpenGL provides a discrete, view-dependent representation of the scene’s geometry from a specific camera position, storing the depth value of the closest surface for each pixel. This depth image serves as the ground-truth for visibility from that vantage point, enabling efficient per-pixel comparisons to determine if other points in the scene are occluded.

This methodology analyses visibility from a specific triangular face on a 3D mesh. An OpenGL camera is configured with its position at the triangle’s barycenter and its view direction aligned with the triangle’s face normal. A chosen Field Of View (FOV) defines the frustum for rendering. The mesh is then rasterized from this perspective to generate a depth map. The resulting depth buffer values are subsequently converted into linear depth within world space, yielding a set of 3D points that were visible to the camera. Figure 2 visualizes the rasterization output from the OpenGL camera positioned at the barycenter of the triangle with a 120° FOV. The top row shows the rendered color image, while the bottom row displays the corresponding depth map used for the visibility analysis.

Conceptually, this process establishes a set of rays originating from the camera center to these reconstructed 3D world points. In this framework, the visible points correspond to finite-length rays that successfully intersected the mesh surface or vertices during rasterization. Conversely, any point not captured in the depth map can be considered the result of an effectively infinite ray that did not intersect the mesh within the camera’s frustum and is therefore classified as invisible from the source triangle. Figure 3 illustrates the visible points in yellow.

The primary limitation remains the inherent discretization error and its management. The computational speed is undeniable, but the accuracy is bounded by the resolution of the depth buffer and the projective transformation. Therefore, the depth-map approach is best suited for applications where an approximate solution is acceptable.

3. Mesh quality evaluation for intervisibility

Our intervisibility analysis takes a 3D surface mesh as input. The geometric quality of this mesh, which directly impacts visibility accuracy, depends on both the source data from aerial platforms (drones, airplanes) or LiDAR HD and the mesh reconstruction techniques, including dense matching, DEM-to-mesh conversion, and mesh decimation (Lindstrom and Turk, 1998, Lindstrom and Turk, 1999). To evaluate how 3D mesh quality affects intervisibility results, we compare a highly accurate LiDAR HD mesh (ground truth Figure 4(b)) against a lower-quality mesh from dense matching (Figure 4(a)) and a decimated versions of the LiDAR HD mesh (Figure 4(d)). Figure 4 presents the input meshes for our intervisibility analysis: (a) MVS and (b) a high-fidelity LiDAR HD mesh. (c) shows an overlay of these two models, confirming that they represent the same geographical area and enabling a direct comparison of how their differing geometric qualities affect visibility results. Finally, (d) shows the decimated LiDAR HD mesh and (e) illustrates LiDAR HD mesh (black) overlapped on decimated LiDAR HD mesh (red).

3.1 Evaluation methodology

For both ray-tracing and depth map rendering methods, we compare a lower quality / evaluated mesh E (Figure 4(a), (d)) with a ground-truth mesh GT (Figure 4(b)).

https://doi.org/10.5194/isprs-archives-XLIX-B2-2026-577-2026

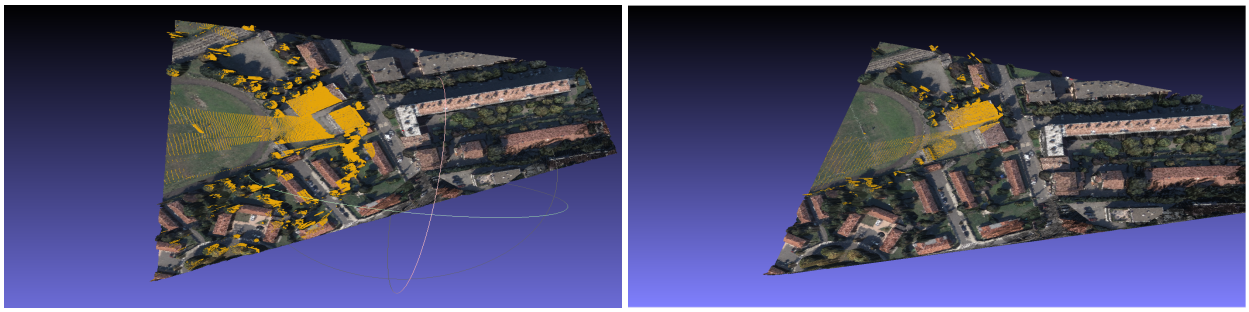
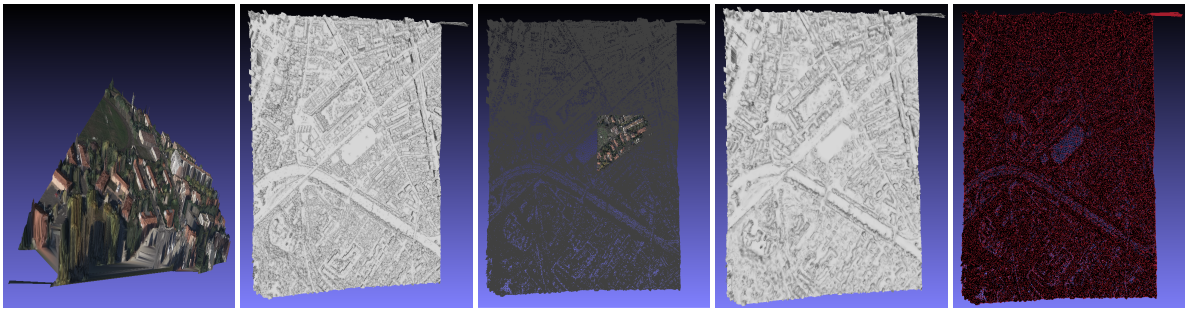


Figure 3. Visible points(Yellow) from camera center 1 and camera center 2



(a) MVS (b) LiDAR HD (c) MVS overlapped on LiDAR HD (d) LiDAR HD 90% (e) LiDAR HD(Black) overlapped on LiDAR HD(Red) 90%

Figure 4. Different types of dataset used in our experiments.

3.1.1 Ray-tracing In the case of ray-tracing, we consider a triangle face on the GT mesh and trace rays from its barycenter O to all the GT vertices V_{GT}, and intersects these rays with the E mesh to propose the following metrics.

- True Positives (TP): number of rays $[OV_{GT}]$ that intersects E at V_E such that the euclidean distance $||V_{GT}V_E|| < 2$ meters. We consider that under this threshold, the visibility information is correct. Note that we take the first intersection with a triangle with normal in the opposite direction to the ray to count only intersections from empty to full space.
- Finite false negatives (FN): number of rays $[OV_{GT}]$ that intersects E at V_E such that $||V_{GT}V_E|| \geq 2$ meters. In this case we consider that the visibility information is incorrect: a part of E occluded the correct visible triangle.
- Infinite false negatives (FN^{∞}): number of rays with no intersection with E: E predicts that only the sky is seen in this direction while the GT predicts it is a triangle of the scene.
- Because the 2 meters threshold is somewhat arbitrary, we also provide a histogram of $||V_{GT}V_E||$ for rays intersecting the E mesh.

All metrics aggregate rays issued from triangles 1 and 2.

Figure 5 shows the infinite rays (yellow) from the barycenter of triangles 1 and 2 to all vertices of GT. Table 1 (left) displays the quantitative results for ray-tracing method. Figure 6 (a) and (b) show the error histograms in the cases where both meshes are intersected (TP and FN).

3.1.2 Depthmap Rendering In depthmap rendering, the triangle barycenter O serves as an opengl virtual camera position with a 120° FOV, from which we render both depth maps independently. Each pixel corresponds to a ray, so we can use the same metrics as above. However, for rendering, ground-truth rays can also be infinite so we add two more metrics:

- True Negatives (TN): number of rays that intersect neither GT nor E: both predict that only sky is visible in this direction
- False Positives (FP): number of rays that intersect E but not GT: GT predicts that only sky is visible in this direction but not E

To compute these metrics, for each valid pixel (with finite depth for both meshes), we convert the depth map coordinates to normalized device coordinates and compute the corresponding ray direction originating from the camera center. We then calculate the orientation angle between this ray and the principal viewing axis to account for perspective effects. Using the depth values from both meshes, we reconstruct the corresponding 3D points along each ray. Table 1 (right) presents the quantitative results for the depth rendering method. Figure 6 (c-f) shows the error histograms in the cases where both meshes are intersected (TP and FN) for decimations keeping 10% and 90% of the triangles.

3.2 Analysis

The metrics show a relatively similar behavior for ray-tracing and rendering, which is a very positive result as it shows that our conclusions do not depend on the intervisibility method used. We see that if the error is acceptable for low levels of decimation (less than 1% of FN^{∞} and FN), this error rapidly

GT / Result	RAY-TRACING			RENDERING				
				FINITE GT			∞ GT	
	TP	FN	FN^{∞}	TP	FN	FN^{∞}	TN	FP
MVS	0	369881	0	255199	55581	1364	736432	705213
10%	367733	2072	76	366577	1216	6	680777	680775
50%	346808	22606	467	356603	11063	36	680874	680775
75%	361040	8652	189	364177	3585	15	680799	680775
90%	276595	91094	2192	321643	45423	72	681438	680775

Table 1. Quantitative evaluation metrics for MVS mesh and decimated LiDAR HD. Higher TP/TN and lower FP/FN indicate better quality.

increases with the amount of decimation. This means that only moderate mesh decimation should be applied when intervisibility analysis is involved. Given how fast intervisibility results degrade with mesh decimation, we can even wonder if our high quality LiDAR mesh is itself sufficiently good for intervisibility calculation. Concerning photogrammetric (MVS) mesh, the results are quite disappointing, with over 20% of FN for rendering. This is even worse for 90% decimation of the LiDAR mesh. This seems too significant, in particular for security applications where human lives are at stake, such that image derived DSMs seem improper for intervisibility at 20cm resolution. Further investigation is required to evaluate at which resolution MVS might be applicable for intervisibility.

The quantitative metrics presented in Table 1 establish the geometric reliability of both approaches; however, the operational feasibility of these methods is largely determined by their computational overhead. To evaluate this, benchmarks were conducted on a mobile workstation equipped with an Intel® Core™ i7-10750H CPU (2.60 GHz, 12 cores), 16.0 GiB of RAM, and an Intel® UHD Graphics (CML GT2) GPU running Ubuntu 24.04 LTS. The experimental results highlight a significant disparity in execution time between the two approaches:

- **Ray-tracing:** The explicit geometric intersection test, processed via the CPU-based CGAL AABB tree, required 363.40 seconds to compute intervisibility from a single barycenter to 371,136 mesh vertices.
- **Depthmap Rendering:** In contrast, the OpenGL-based depth-buffer approach completed the visibility query for 1048576 vertices (1024×1024 pixel grid) in only 0.034 seconds.

These results demonstrate that while the ray-tracing method provides high geometric precision, the rendering-based paradigm offers a massive speedup—approximately four orders of magnitude.

4. Conclusion and Future work

This paper has presented two distinct methods for 3D mesh intervisibility calculation: a precise but computationally intensive ray-tracing method and an efficient, approximate depth-rendering technique. These techniques have been used to evaluate the fitness of 3D meshes of varying quality to calculate intervisibility. In particular, we have compared photogrammetric meshes with LiDAR meshes, and evaluated the impact of mesh decimation on the quality of intervisibility calculation. The analysis shows that intervisibility is very sensitive to mesh quality, with only rather limited decimation being acceptable, and photogrammetric mesh seeming unsuitable for intervisibility analysis.

Future work will involve more systematic experiments to comprehensively map the trade-offs between computational cost, accuracy, and input data characteristics. In particular, we plan to address LiDAR and image acquisitions with different characteristics (specifically, varying ground-sample distance for images and point density for LiDAR) to refine the current analysis. A key objective will also be to bridge these technical metrics with their implications for practical real-world applications.

Acknowledgements

This work was supported by the S3D2 research project.

References

Batty, M., 2001. Exploring isovist fields: Space and shape in architectural and urban morphology. *Environment and Planning B: Planning and Design*, 28(1), 123–150.

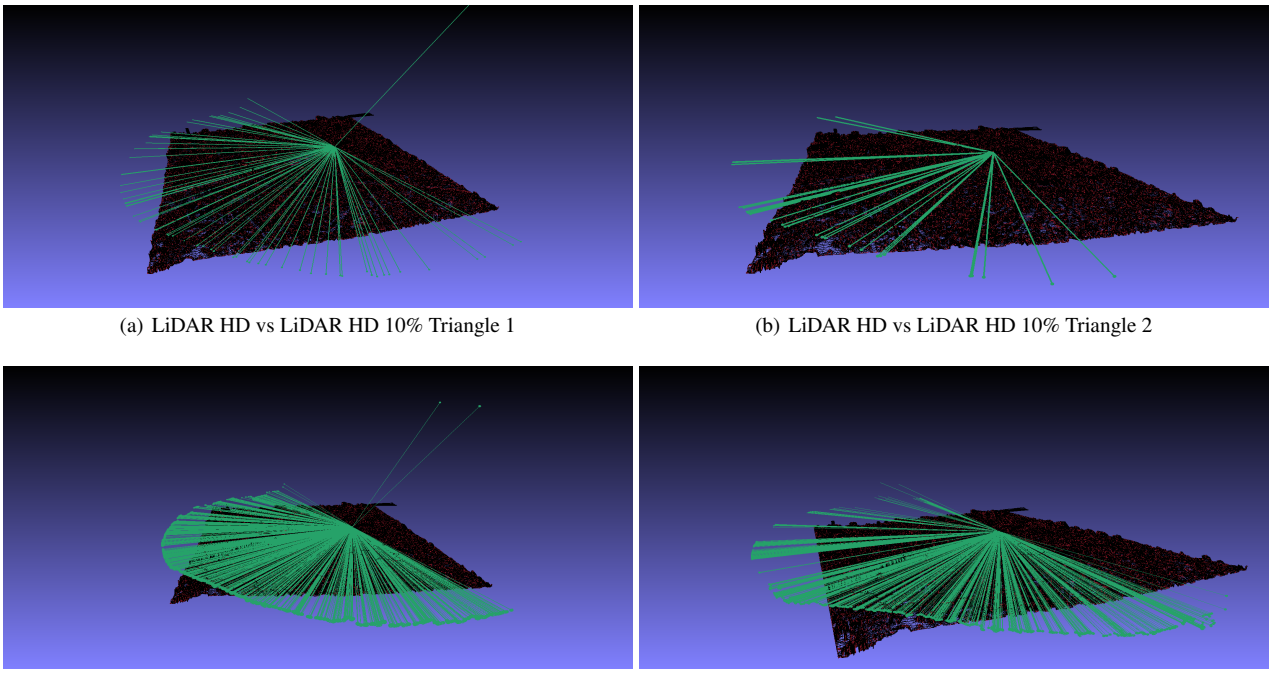
Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., Çöltekin, A., 2015. Applications of 3D city models: State of the art review. *ISPRS International Journal of Geo-Information*, 4(4), 2842–2868.

Bittner, J., Wonka, P., 2003. Visibility in computer graphics. *Environment and Planning B: Planning and Design*, 30(5), 729–755.

CGAL., 2023. CGAL Documentation: AABB Tree. . The CGAL Project. Retrieved from https://doc.cgal.org/latest/AABB_tree/.

Cohen-Or, D., Chrysanthou, Y., Silva, C. T., Durand, F., 2003. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, 9(3), 412–431.

De Floriani, L., Magillo, P., 1999. Intervisibility on terrains. *Geographical Analysis*, 31(2), 104–121.



(a) LiDAR HD vs LiDAR HD 10% Triangle 1

(b) LiDAR HD vs LiDAR HD 10% Triangle 2

(c) LiDAR HD vs LiDAR HD 90% Triangle 1

(d) LiDAR HD vs LiDAR HD 90% Triangle 2

Figure 5. Infinite rays comparison between LiDAR HD vs and decimated versions. In the figure LiDAR HD mesh in black is overlapped with Decimated LiDAR HD in red.

Franklin, W. R., Ray, C. K., 1994. Higher isn’t necessarily better: Visibility algorithms and experiments. *Advances in GIS Research*, 6(1), 751–770.

Katz, D., Katz, Y., 2019. Visibility analysis in the presence of occlusions. *Computer-Aided Design*, 112, 1–10.

Lindstrom, P., Turk, G., 1998. Fast and memory efficient polygonal simplification. *IEEE Visualization*, 279–286.

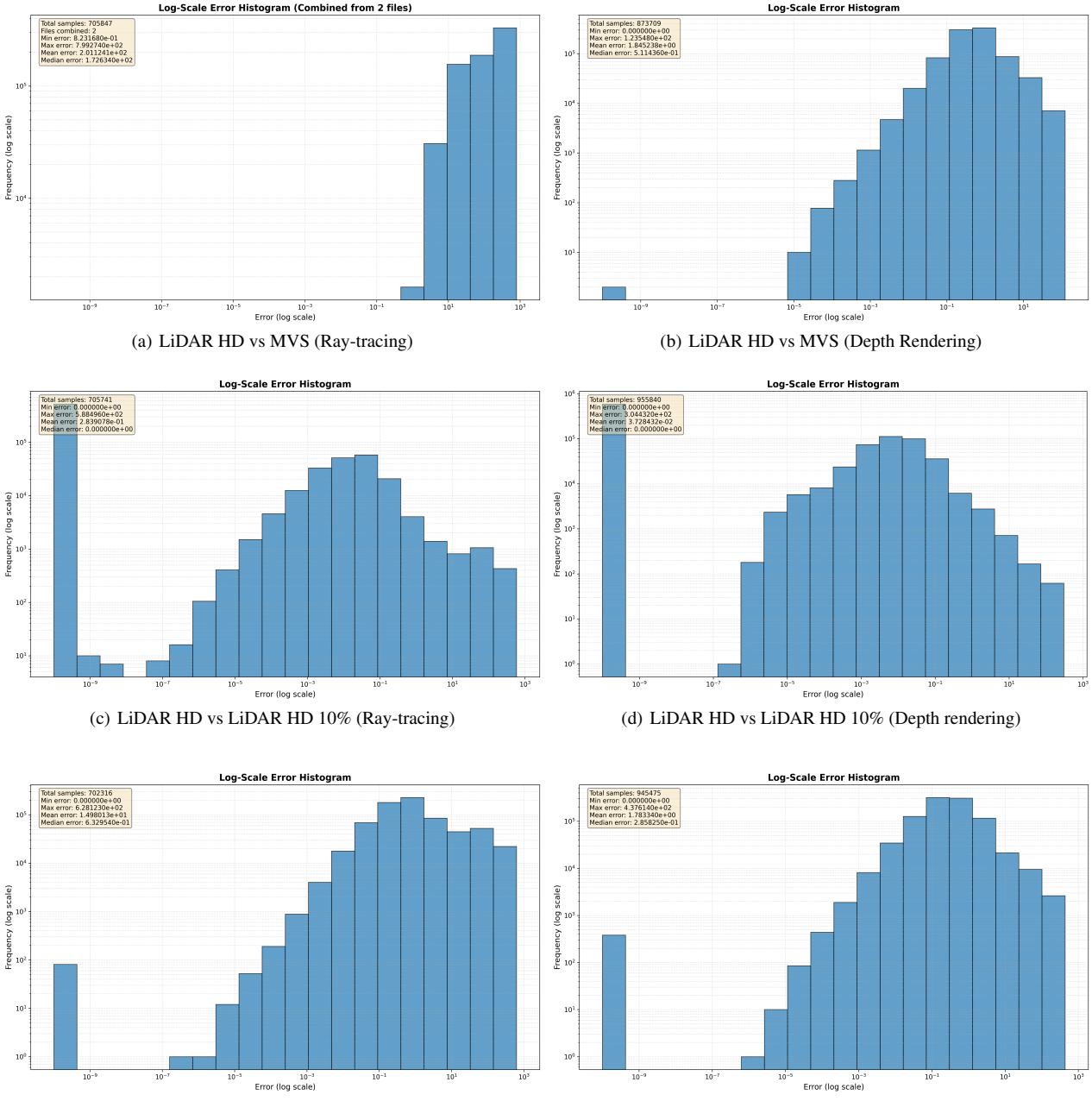
Lindstrom, P., Turk, G., 1999. Evaluation of memoryless simplification. *IEEE Transactions on Visualization and Computer Graphics, 5(2)*, 98–115.

Nagy, G., Zhang, J., 2006. Error analysis of viewshed algorithms. *International Journal of Geographical Information Science*, 20(9), 1005–1022.

Oleynikova, H., Taylor, Z., Fehr, M., Siegwart, R., Nieto, J., 2017. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1366–1373.

Teller, S. J., Séquin, C. H., 1991. Visibility preprocessing for interactive walkthroughs. *ACM SIGGRAPH Computer Graphics*, 25(4), 61–69.

Wald, I., Woop, S., Benthin, C., Johnson, G. S., Ernst, M., 2014. Embree: A kernel framework for efficient CPU ray tracing. *ACM Transactions on Graphics*, 33(4), 1–8.



(a) LiDAR HD vs MVS (Ray-tracing)

(b) LiDAR HD vs MVS (Depth Rendering)

(c) LiDAR HD vs LiDAR HD 10% (Ray-tracing)

(d) LiDAR HD vs LiDAR HD 10% (Depth rendering)

(e) LiDAR HD vs LiDAR HD 90% (Ray-tracing)

(f) LiDAR HD vs LiDAR HD 90% (Depth rendering)

Figure 6. Log Histogram of Depth error for combined rays from the center of Triangle 1 and Triangle 2. X-axis indicated error in meters, Y-axis indicates number of rays/pixels

This contribution has been peer-reviewed.
https://doi.org/10.5194/isprs-archives-XLIX-B2-2026-577-2026