Editing and Visualization of Large Point Clouds for Urban Planning Applications

Gorazd Gorup*
Supervised by: Ciril Bohak,† and Žiga Lesar‡

Faculty of Computer and Information Science
University of Ljubljana
Ljubljana / Slovenia

Abstract

Urban planning has become increasingly complex in recent years, necessitating more detailed and accurate representations of real-world environments to meet design objectives. While mesh and parametric geometry are dominant in current workflows, they often fail to capture essential contextual information. In contrast, point clouds have untapped potential for improving urban design processes. This paper addresses the challenge of interactively visualizing and editing large point cloud datasets from LiDAR scans, proposing methods that we implemented as a set of tools and utilities to enhance urban planning workflows. We extended the open-source 3D modelling software Blender with features for loading and visualizing large point clouds, and implementing levels-of-detail and classification mechanisms. We do so by adapting Blender's internal representations and functions to accommodate point geometry. We created tools for targeted editing operations and leverage Blender's Geometry Nodes to facilitate non-destructive, node-based editing for operations such as point copying, painting, and flattening – basic tasks in urban design. We tested the rendering functionality of Blender's rendering engines to visualize different point attributes, such as colour, elevation, and classification. Finally, we evaluated our solution against Rhinoceros 8 and its Grasshopper extension, commonly used in urban planning, and identified areas where Blender's point cloud manipulation and visualization capabilities could be improved.

Keywords: Blender, Computer Graphics, Editing, Geometry Nodes, Point Cloud, Rendering, Urban Planning, Visualization

1 Introduction

The increasing availability of high-resolution point cloud datasets has greatly enhanced urban planning, offering detailed spatial data for analysis and manipulation. Advances in light detection and ranging (LiDAR) and photogrammetry have enabled the creation of large-scale datasets, many of which are publicly accessible, for example Toronto-3D [10] for autonomous driving and urban mapping, Paris-Lille-3D [8], which was acquired to develop automatic segmentation of urban infrastructure, and SemanticKITTI [1] and TUM-MLS-2016 [19] for urban environment modelling. National airborne LiDAR datasets, such as those from Slovenia [2, 6] and the Netherlands [12], provide extensive coverage, although at a lower resolution and with a limited field of view.

These datasets support machine learning applications and enhance urban planning workflows, promoting data-driven decision-making [4, 3, 15]. With a growing demand for detailed urban modelling, there is an increasing need for advanced visualization and editing tools. Existing point cloud tools primarily offer basic operations like cropping, trimming, and basic spacial transformations. However, they lack direct editing features similar to those in common 3D modelling software, which support various geometry types (e.g., polygon meshes, Bézier curves, NURBS surfaces, metaballs, and constructive solid geometry). Consequently, users often convert point clouds to other formats for editing, leading to errors and data loss.

To address this, we propose a non-destructive point cloud editing approach within Blender, an open-source 3D modelling application. We present an efficient handling of large point cloud data via level-of-detail (LoD) functionality, a non-destructive editing approach through Blender's *Geometry Nodes*, and advanced rendering of point clouds utilizing attributes like colour, position, and classification.

In Section 2, we review prior work and existing tools, in Section 3, we detail our approach, in Section 4, we evaluate results of our approach and comparisons with a popular Computer Aided Design (CAD) application Rhinoceros, and in Section 5, we present findings, limitations, and future research aims.

*gorazd@lgm.fri.uni-lj.si
†ciril.bohak@fri.uni-lj.si
‡ziga.lesar@fri.uni-lj.si

2 Background

The convergence of procedural modelling, point cloud technologies, and urban planning has significantly advanced urban design and visualization. Recent studies emphasize the integration of artistic and technical approaches to enhance planning workflows. Yang [16] presents a model for transforming 2D sketches into 3D CAD representations, streamlining design for architects. Ihle and Wichmann [5] explore the fusion of scientific data and artistic representation to deepen landscape visualization. Additionally, Urech *et al.* [11] highlight point clouds as a crucial link between design and planning, facilitating accurate urban modelling.

2.1 Procedural Methods in Urban Modelling

Procedural modelling addresses the inefficiencies of traditional urban design methods. Yang and Delparte [17] demonstrate a framework using CityEngine[1] to generate 3D models from Geographic Information System (GIS) datasets, while Mustafà *et al.* [7] integrate environmental factors, such as flood sensitivity, into procedural workflows. Open-source tools, including Ladybug[2] and Dragonfly[3], further promote sustainable urban design.

2.2 Point Cloud Modelling in Urban Planning

Advancements in LiDAR and mobile laser scanning have expanded point cloud applications. Wang *et al.* [14] provide an overview of mobile scanning for high-resolution urban modelling, while You *et al.* [18] focus on airborne laser scanning for green space management. Schmohl *et al.* [9] integrate Artificial Intelligence (AI) with point cloud data for improved urban forestry planning. Vijaywargiya and Ramiya [13] highlight point clouds' role in sustainable governance, supporting applications such as energy management and infrastructure monitoring.

2.3 Existing Point Cloud Processing Tools

Several software solutions support point cloud visualization and editing to varying degrees.

Rhinoceros 3D and Grasshopper. Rhinoceros[4] is a CAD software widely used in architecture and engineering. Its extension Grasshopper[5] provides a node-based approach for geometry manipulation. For point clouds, the Volvox[6] extension is widely used, and supports basic point selection, deletion, and merging. With Rhinoceros 8, native point cloud support improves workflow integration, though processing with Grasshopper remains computationally intensive due to data being copied with every node operation.

Autodesk Solutions. Autodesk offers Revit[7], AutoCAD[8], and ReCap PRO[9]. ReCap PRO focuses on 3D scanning, but primarily supports visualization rather than direct editing. Most Autodesk tools treat point clouds as static references rather than editable datasets.

Other Tools. Several other solutions provide varying levels of point cloud functionality:

- **Pix4D**[10] specializes in photogrammetry-based point cloud processing, offering noise removal and classification features.
- **ESRI ArcGIS**[11] supports geospatial analysis but lacks direct editing capabilities, focusing instead on classification and visualization.
- **DJI Modify**[12] integrates with drone-based mapping, enabling noise removal and point flattening.
- **Tcp Point Cloud Editor**[13] offers classification, slicing, and simultaneous localization and mapping (SLAM) visualization.
- **Vega**[14] extends AutoCAD with contour generation and texture projection.
- **3D Survey**[15] includes selection, transformation, and X-Ray visualization for urban scanning.
- **CloudCompare**[16] is an open-source tool that supports segmentation, classification, and format conversion.

Despite all advancements and dedicated software, existing tools have limitations. Many procedural modelling software, such as CityEngine, lack direct point cloud editing capabilities, necessitating conversion to alternative geometries, which introduces errors and reduces traceability. While proprietary solutions offer powerful tools, they are rigid and costly, and prioritize visualization over direct editing, whereas open-source alternatives remain fragmented. To address these gaps, we propose a non-destructive point cloud editing approach using Blender's

[1] https://www.esri.com/en-us/arcgis/products/arcgis-cityengine
[2] https://github.com/ladybug-tools/ladybug-blender
[3] https://www.ladybug.tools/dragonfly.html
[4] rhino3d.com/
[5] https://www.grasshopper3d.com
[6] https://www.grasshopper3d.com/group/volvox
[7] https://www.autodesk.com/products/revit
[8] https://www.autodesk.com/products/autocad
[9] https://www.autodesk.com/products/recap
[10] https://www.pix4d.com
[11] https://www.arcgis.com/
[12] https://enterprise.dji.com/modify
[13] https://www.aplitop.com/products/tcp-pointcloud-editor
[14] https://www.vegaapp.co.il
[15] https://3dsurvey.si
[16] https://cloudcompare.org

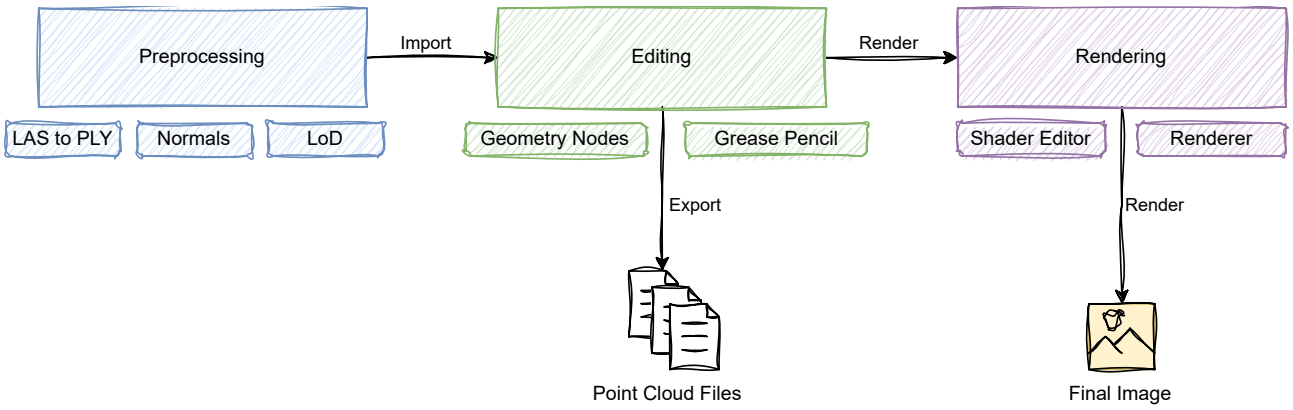


Figure 1: The diagram illustrates our workflow for point cloud manipulation in Blender.

procedural capabilities. Our method enables interactive manipulation, enhanced rendering, and improved workflow integration, bridging procedural urban modelling and detailed point cloud transformations.

3 Methods

We implemented an extension for Blender to import the LAS point cloud files and to provide some pre-made Geometry Nodes constructions for operations mentioned below. Python was used to tie logic to the Blender UI, for some simple computations, and for instantiating Geometry Node templates and helper objects. For more expensive operations (reading from LAS files, normal calculation, ...) Rust was used and attached to the Python code as a module.

Our workflow for point cloud manipulation consists of three main stages: (1) preprocessing and import, (2) manipulation via node trees, and (3) rendering of the resulting geometry, as seen in Figure 1. Initially, we preprocess the point cloud by converting the LAS file to PLY format [17], considering LoD and classification. Normals can be optionally calculated, though this step may be deferred to the editing stage based on user preference. During the editing phase, Geometry Nodes and Grease Pencil are the primary tools for transforming the point cloud. After editing, the modified point clouds are either exported or proceed to the rendering stage, where materials are applied to the points in the Shader Editor, and rendering settings are adjusted for either the Cycles or EEVEE engine, which are the default rendering engines in Blender.

3.1 Preprocessing

Blender and Rhinoceros offer limited native support for point cloud data. Both can handle PLY files, while Rhinoceros supports additional formats like ASTM E57. Since LiDAR point clouds are often stored in LAS format, we used CloudCompare to convert LAS files to ASTM E57 for Rhinoceros. For Blender, we implemented our custom LAS importer, with which we also recentred the point cloud to mitigate precision issues.

Blender organizes data into objects, with point clouds represented as unconnected vertices that require conversion into renderable instances via Geometry Nodes.

3.1.1 LoD and Classifications

Blender imposes an approximately 4.2 GB point buffer limit per object. At the same time, due to the huge amount of data, we needed LoD to optimize rendering and interaction. To satisfy both restrictions, we implemented LoD as separate objects containing subsets of points.

LiDAR datasets classify points into 256 categories, such as ground, buildings, and vegetation. We explored two approaches for toggling classification visibility:

1. Storing classified points in separate objects for efficient visibility toggling.
2. Storing class IDs as point attributes and filtering via Geometry Nodes.

While the first approach improved performance, we adopted the second due to better integration with Geometry Nodes: we would have to apply the Geometry Nodes modifier on a limited set of objects built from LoD instead of many more built from LoD and classifications. [18] However, the second approach introduced computational overhead at the import stage, since the classification visibility toggling is implemented with Geometry Nodes and the building of node tree takes a significant amount of time.

3.1.2 Normal Computation

Normals are computed externally using a PCA-based method and stored into the existing PLY files, before reimporting the point cloud into Blender. We decided to offer

[17] PLY is a file format for 3D data storage, which can contain different entities – vertices, edges, faces, etc. – and any number of attributes they may have.

[18] In Conclusions, we discuss why Blender modifiers pose a challenge for editing with Geometry Nodes.

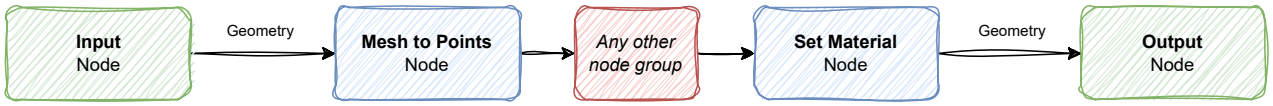


Figure 2: Here, our preferred node tree construction is shown. *Mesh to Points* node could also be placed after other nodes but before *Set Material* node.

this as a separate functionality from importing, to allow the user to work on the point cloud faster if computed normals are not the priority in the process.

3.1.3 Viewport Display

Blender requires a Geometry Nodes modifier to convert unconnected vertices into point instances. The coloured points are available with the Material Preview shading, but only if the correct material is applied to the point cloud (see *Set Material* node in Figure 2). In EEVEE, disabling shadow calculations improved performance.

3.2 Editing

We tested Blender's Geometry Nodes (seen in Figure 3) on common urban planning operations. Geometry Nodes are applied to each object via a modifier. An object can have multiple Geometry Nodes modifiers. Geometry Node trees can be shared between objects, but the modifiers and modifier stacks themselves are unique for each object. Some parameter adjustments were controlled through auxiliary objects to allow for interactive transformations. We also implemented respective functionality in Grasshopper whenever possible.

Point Copying Points can be duplicated based on conditions such as location, colour, or classification. In our case, we defined a region using a Cube primitive, identified points within that region using *Object Info* and *Raycast* nodes, and separated the selected geometry from the rest. We then joined a transformed copy of the selection with the entire cloud. The transformation of the copied region was controlled by an empty object.

Point Removal Similarly to point copying, points were selected based on a user-defined predicate and then removed via the *Delete Geometry* node. In our case, we used a plane: through Geometry Nodes, we added and extended a cube in the direction of the plane's normal to reach all possible points that could be projected onto the plane, and checked if the points were contained inside the cube.

Drawing Points Blender's Grease Pencil allows strokes to be converted into mesh geometry, from which points are sampled onto mesh faces. This method is useful for filling voids in point clouds and creating irregular structures.

Painting on Points Grease Pencil strokes enable point attribute modifications based on proximity, allowing for localized adjustments in colour and classification. Strokes can be converted into meshes and points within them can be identified via *Raycast* node and modified accordingly. For example, we were able to modify the colour attribute of the points within stroke meshes, essentially enabling painting colour into point clouds.

Erasing Points The same principle as with painting on points can also be applied to point removal. The selected points do not have their attributes modified, but are instead deleted.

Flattening Points Flattening removes structures and shapes in the point cloud while filling gaps. Our first approach involved removing points as described in Point Removal and sampling points on a defined plane, ensuring continuity by sampling colours from surrounding points. An alternative method modified existing point coordinates instead of deleting and resampling, maintaining texture consistency but introducing some overlapping points. We use the second method due to the mentioned texture consistency providing more visually appealing results.

Mesh Tools Blender's mesh and curve modelling tools allow geometric structures to be integrated into point clouds by sampling points from mesh surfaces. The *Store Named Attribute* node enables precise control over point properties. Converted points retain the attributes from vertices they originate from, ensuring consistency in assigned properties.

3.3 Rendering

Blender's Shader Editor enables the application of materials to point clouds for flexible visualization. Different point attributes can be converted to colour, combined, and passed to shaders. Both Cycles and EEVEE renderers are supported, with Cycles offering path-traced realism and EEVEE providing faster, although less physically accurate renders.

To better evaluate our methods, we compared our implementation of the above-mentioned operations with similar functionality in Rhinoceros 8 and Grasshopper.

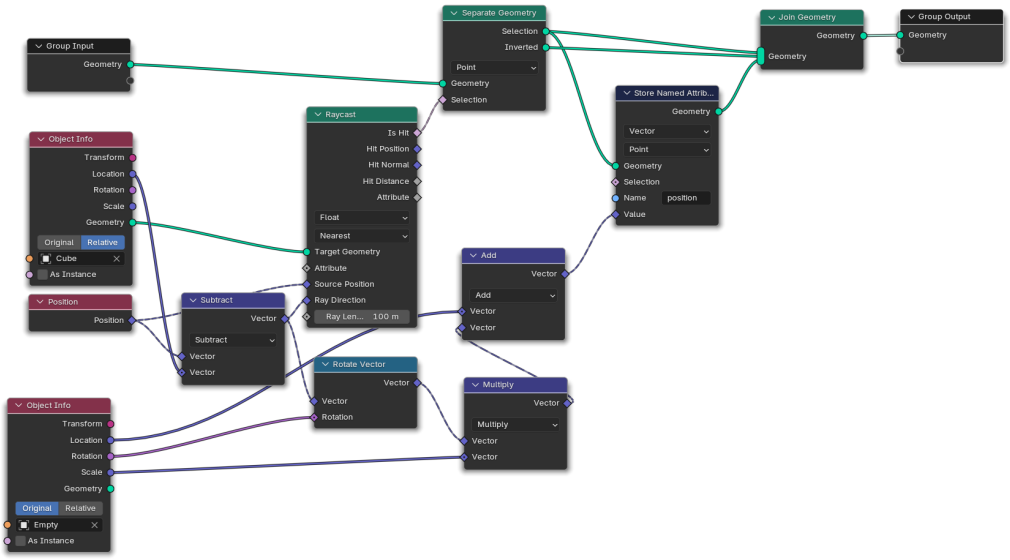


Figure 3: An example of Blender Geometry Nodes configuration. This one is used for point copying operation. Notice the lack of *Mesh to Points* node or *Set Material* node – these are present in a different node tree on a Geometry Nodes modifier right before and after this one.

4 Results

We present the experimental setup, performance analysis, and visual outputs of our comparison of Blender and Rhinoceros 8 functionality. Rhinoceros was chosen because it is also used in urban planning, it offers point cloud manipulation and visualization, and because of its extension Grasshopper, which employs node-based non-destructive editing. The latter allowed us to compare node trees and performances on point clouds to highlight advantages and disadvantages in each program.

4.1 Experimental Setup

Experiments were conducted in Blender and Grasshopper, using internal debugging tools for timing measurements. Blender accumulates execution time up to a node, whereas Grasshopper measures each node separately. These values are approximate, useful for optimization but not precise benchmarking.

Certain Grasshopper operations — like point copying — were limited to smaller datasets due to crashes with larger ones. The manual LoD was not tested.

Experiments were run on a workstation with dual Intel Xeon Gold 6140 CPUs (2.30 GHz, 18 cores each), 256 GB RAM, an NVIDIA RTX A4000 GPU, and Windows Server 2022 operating system. Note that the workstation was accessed remotely and while processing unrelated workload of various intensity throughout our experiments, although we made sure that we were given enough resources for that to not impact the performance of the software. The dataset was a 6.4-million-point subset of Tokyo LiDAR data[19].

4.2 Performance Evaluation

Execution times were measured for Point Copying, Point Removal, and Flattening Points in both applications (Tables 1 and 2), with four additional measurements for Blender where the same operations were not possible in Rhinoceros. We noticed that Blender performed operations about a thousand times faster on roughly the same number of points. Maths nodes and comparison related nodes cost almost no time, while nodes operating on geometry execute the longest (*Join Geometry*, *Delete Geometry*, *Store Named Attribute*). A *Raycast* node, which was used in some operations for segmenting points into regions to be transformed, also contributed to the final time considerably, although it was never the bottleneck.

The node trees in Blender generally contained fewer nodes than in Grasshopper, resulting in more maintainable node trees and fewer operations on the point cloud per tree. However, Blender experienced performance drops when using paint tools on points due to attribute recalculations. Grasshopper offered more efficient transformations, but was more prone to crashes with large datasets. We attributed this to the fact that each node in Grasshopper copies the data it operates on, greatly increasing memory consumption.

Blender supported advanced point cloud attributes, enabling better customization. Grasshopper's support was limited to position, colour, and normals.

[19] `https://3dview.tokyo-digitaltwin.metro.tokyo.lg.jp/`, section *Shinjuku Ward*, figure `09LD1638`

Table 1: Execution times for operations in Blender. In the Bottleneck column, we note the nodes which contribute the most to the computation time, and their execution time.

Operation	Nodes	Time	Points	Bottleneck
Point Copying	10	0.29 s	(~340k points)	*Separate Geometry* node (0.15 s), *Join Geometry* node (0.12 s)
Point Removal	9	0.11 s	(~460k points)	*Delete Geometry* node (0.11 s)
Drawing Points	4	0.21 s	(~18k points added)	*Distribute Points on Faces* node (0.21 s)
Erasing Points	8	0.26 s	(~200k points)	*Delete Geometry* node (0.26 s)
Colouring Points	7 + 7*	0.50 s	(~200k points coloured)	*Store Named Attribute* node (0.50 s)
Flattening Points	13	0.11 s	(~880k points)	*Store Named Attribute* node (0.11 s)
Mesh to Points	11	0.02 s	(~104k points)	*Points on Faces* node (0.02 s)

* There are two node groups, one applied to the point cloud object and one applied to the Grease Pencil object.

Table 2: Execution times for operations in Grasshopper.

Operation	Nodes	Time	Points	Bottleneck
Point Copying	16	91 s	(~340k points)	Point in Brep (46 s), Construct Point Cloud (44 s)
Point Removal	9	90 s	(~460k points)	Point in Brep (45 s), Construct Point Cloud (44 s)
Flattening Points	19	65 s	(~880k points)	Construct Point Cloud (41 s)

Table 3: Rendering times for Tokyo dataset. With Cycles and EEVEE, we applied two different materials on points.

Renderer	Shader	Time
Blender EEVEE	Principled BSDF	3.9 s
	Emission	2.8 s
Blender Cycles	Principled BSDF	40 s
	Emission	19 s
Rhino Renderer	/	<1 s *

* Excludes post-processing (~3 s).

4.3 Rendering Comparison

To highlight the differences, we provide four images, rendered in both tools (Figure 5). Blender's Attribute node allowed custom materials, enhancing visualization. Rendering performance (Table 3) showed Blender Cycles was slower but allowed material flexibility and more realism. The difference in rendering times is also seen from the perspective of the shader used for shading points. The Emission shader is not influenced by its surroundings and therefore requires less computation. Rhinoceros rendered quickly, but additional post-processing, such as gamma correction and tone-mapping, generated a significant delay (notably up to 3 seconds). Finally, the visual comparison of Cycles and EEVEE is shown in Figure 4. In the Cycles render, the transition between points close together is smoother, which might be due to physically correct shading or as a result of denoising. There is also a slight difference in shadows, where EEVEE produces lighter shadows.

5 Conclusions

Blender provided a more stable and responsive node-based editing experience than Rhinoceros, particularly with optimized Geometry Nodes. However, our limited experience with Rhinoceros means that certain Grasshopper configurations may not have been fully optimized. Preprocessing techniques, such as point cloud reduction, could improve performance, but reversing reductions during rendering would likely negate the benefits.

Blender's lack of shared modifier stacks makes editing multiple levels of detail and classification clusters inefficient without custom scripting. Scripted operators could automate tasks like applying Geometry Nodes across objects and batch-processing transformations. Additionally, Grease-Pencil-based point erasure struggles with large datasets, introducing severe lag due to simultaneous geometry updates. A solution could involve temporarily disabling Geometry Nodes during drawing, working with a lower-resolution representation, or applying Geometry Nodes only to selected objects directly interacting with Grease Pencil strokes.

Both Blender and Rhinoceros support scripting extensions, but their effectiveness for high-performance point cloud processing is limited. Both offer a Python scripting interface, with Rhinoceros supporting C# as well, which does introduce some flexibility for point cloud editing, although API structure and transfer of data between the software and the script introduces a lot of overhead, resulting is slow data processing.

Our study explored procedural point cloud and mesh editing for urban planning using Blender, comparing it to Rhinoceros Grasshopper. Blender enabled non-destructive manipulation via Geometry Nodes and interactive modi-

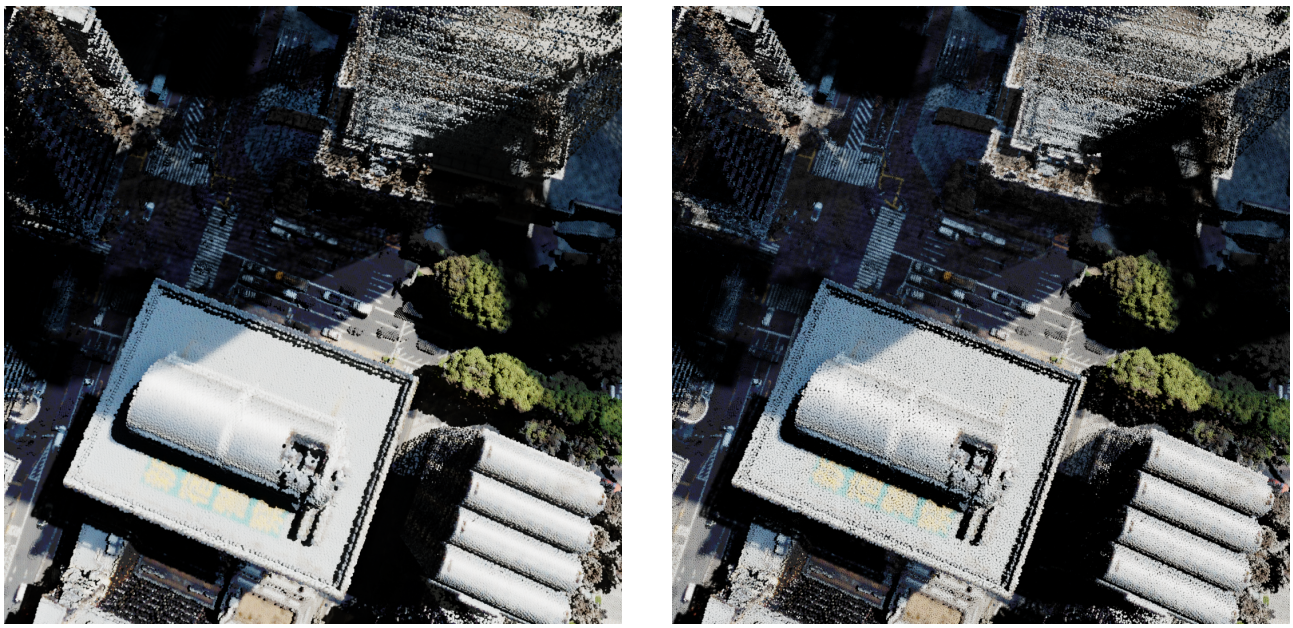


Figure 4: A comparison of rendering in Cycles (left) and EEVEE (right) rendering engines. Most notably, the borders between points are more pronounced in EEVEE render and the darker spots are brighter than in Cycles.

fication using Grease Pencil. While Rhinoceros handled basic transformations more efficiently, Blender excelled in complex node-based operations, attribute manipulation, and classification-based filtering. Both Blender's renderers were slower than Rhinoceros's default renderer (ignoring the post-processing lag), but they enabled more flexible and appealing visualizations.

Blender's procedural point cloud editing capabilities show promise, but certain challenges remain, including modifier stack limitations and slow Grease Pencil processing. With further development and community-driven improvements, Blender could become a powerful tool for large-scale urban modelling and design workflows.

References

[1] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. Semantickitti: a dataset for semantic scene understanding of lidar sequences. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[2] Geodetic Institute of Slovenia. izvedba laserskega skeniranja slovenije. blok 35 – tehnično poročilo o izdelavi izdelkov. Technical report, Geodetic institute of Slovenia, 2015-2023.

[3] Y. Guo, H. Wang, Q. Hu, H. Líu, L. Liu, and M. Bennamoun. Deep learning for 3D point clouds: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43:4338–4364, 2021.

[4] Q. Hu, B. Yang, S. Khalid, W. Xiao, N. Trigoni, and A. Markham. Towards semantic segmentation of urban-scale 3D point clouds: A dataset, benchmarks and challenges, 2021.

[5] M. E. Ihle and V. Wichmann. Blurring boundaries between scientific and artistic representation of landscapes. *Journal of Digital Landscape Architecture*, 2024.

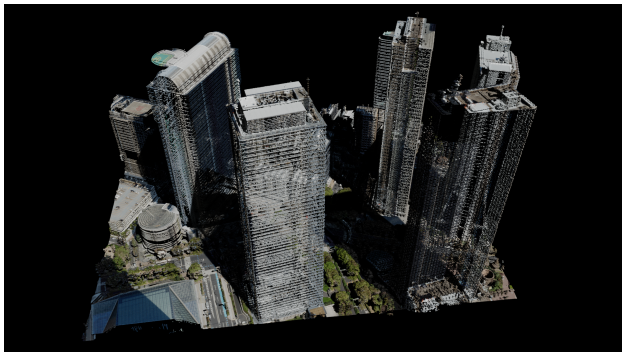
[6] D. Mongus, N. Lukač, and B. Žalik. Ground and building extraction from lidar data based on differential morphological profiles and locally fitted surfaces. *ISPRS Journal of Photogrammetry and Remote Sensing*, 93:145 – 156, 2014.

[7] A. Mustafà, X. W. Zhang, D. G. Aliaga, M. Bruwier, G. Nishida, B. Dewals, S. Erpicum, P. Archambeau, M. Pirotton, and J. Teller. Procedural generation of flood-sensitive urban layouts. *Environment and Planning B Urban Analytics and City Science*, 2018.

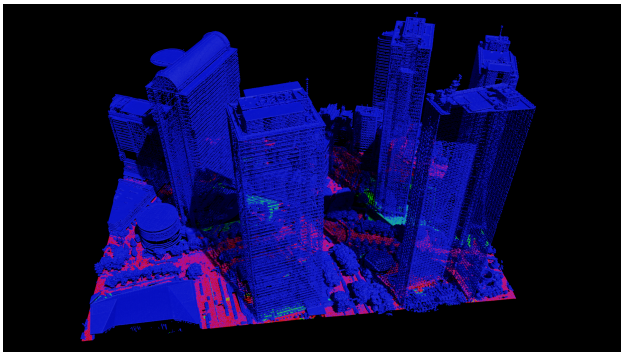
[8] X. Roynard, J. Deschaud, and F. Goulette. Paris-lille-3d: a large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification. *The International Journal of Robotics Research*, 37:545–557, 2018.

[9] S. Schmohl, A. Vallejo, and U. Soergel. Individual tree detection in urban als point clouds with 3D convolutional networks. *Remote Sensing*, 14:1317, 2022.

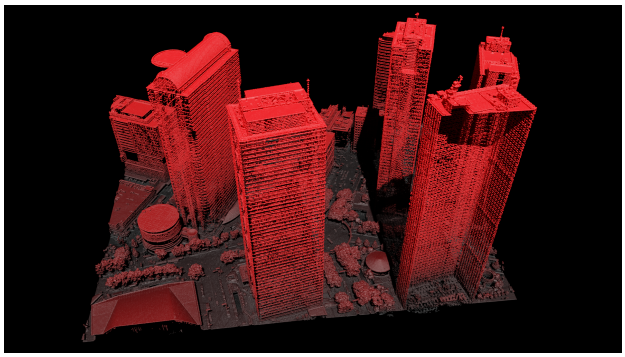
[10] W. Tan, N. Qin, L. Ma, Y. Li, J. Du, G. Cai, K. Yang, and J. Li. Toronto-3D: A large-scale mobile lidar dataset for semantic segmentation of urban roadways. In *2020 IEEE/CVF Conference*



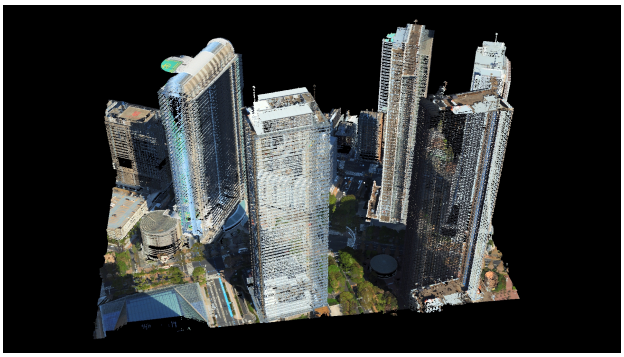
(a) Tokyo dataset rendered in Blender Cycles. Principled BSDF material is used, the color is sampled from each point's color attribute.



(b) Classification render in Blender Cycles. Principled BSDF material is used, the color is sampled from each point's classification attribute.



(c) Elevation render in Blender Cycles. Principled BSDF material is used, the color is sampled from the Z coordinate and passed as hue angle in HSV color space. The Z coordinate is normalized beforehand to the interval $[0,1]$ where 0 maps to the Z coordinate value of the lowest point and 1 maps to the Z coordinate value of the highest point.



(d) Tokyo dataset render in Rhinoceros with the default renderer. Some post-processing gamma-correction was applied.

Figure 5: Renders of the Tokyo dataset in Blender and Rhinoceros. The Cycles renders show more flexibility in data representation (colour, classification, elevation) and showcase features that emphasize depth in the image (shadows and point shading). Shadows that are shown in the Rhinoceros render, are baked into the point colour information, captured at the scanning stage, while in Cycles renders the shadows are produced with ray-tracing on the point cloud.

on Computer Vision and Pattern Recognition Workshops (CVPRW), page 797–806. IEEE, 2020.

[11] P. R. W. Urech, M. A. Dissegna, C. Girot, and A. Grêt-Regamey. Point cloud modeling as a bridge between landscape design and planning. *Landscape and Urban Planning*, 203:103903, 2020.

[12] A. van Natijne. Geotiles: readymade geodata with a focus on the Netherlands. Technical report, Delft University of Technology, 2023.

[13] J. Vijaywargiya and A. Ramiya. Metamorphism of als point data for multitude application. *Isprs Annals of the Photogrammetry Remote Sensing and Spatial Information Sciences*, X-1/W1-2023:25–31, 2023.

[14] Y. Wang, Q. Chen, Q. Zhu, L. Liu, C. Li, and D. Zheng. A survey of mobile laser scanning applications and key techniques over urban areas. *Remote Sensing*, 11:1540, 2019.

[15] W. Xiao, H. Cao, M. Tang, Z. Zhang, and N. Chen. 3D urban object change detection from aerial and terrestrial point clouds: A review. *International Journal of Applied Earth Observation and Geoinformation*, 118:103258, 2023.

[16] H. Yang. Sketch2CAD: 3D CAD model reconstruction from 2D sketch using visual transformer, 2025.

[17] X. Yang and D. Delparte. A procedural modeling approach for ecosystem services and geodesign visualization in old town pocatello, idaho. *Land*, 2022.

[18] H. You, S. Li, Y. Xu, Z. He, and D. Wang. Tree extraction from airborne laser scanning data in urban areas. *Remote Sensing*, 13:3428, 2021.

[19] J. Zhu, J. Gehrung, R. Huang, B. Borgmann, Z. Sun, L. Hoegner, M. Hebel, Y. Xu, and U. Stilla. Tum-mls-2016: an annotated mobile lidar dataset of the tum city campus for semantic point cloud interpretation in urban areas. *Remote Sensing*, 12:1875, 2020.