From Points to Prints

Monthly Meeting (4)

Alexandre Bry

May 18, 2026

TU Delft

IGN

Context

Roofprint vs. footprint

	Roofprint	Footprint
Defined by	Roof	Façades
Comes from	Aerial data (images, ALS)	Terrain measurements (TLS, MLS)
Used for	3D modelling, solar potential	Tax estimation, energy consumption

Table 1: Comparison between roofprints and footprints

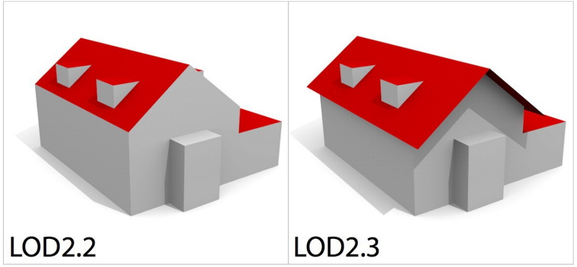


Figure 1: Part of the visual definition of buildings Level of Detail (LoD) containg LoD 2.2 and 2.3 (Biljecki et al., 2016) .

- Roofprints and footprints can serve different purposes, and knowing the difference between them is important when the roof overhangs are significant.
- Since this difference is usually not available, it is unclear what the use cases could be, but we can think about:
  - More accurate visualisation and texturing of 3D models
  - More accurate simulations (wind, luminosity)

Strengths and weaknesses of BD TOPO

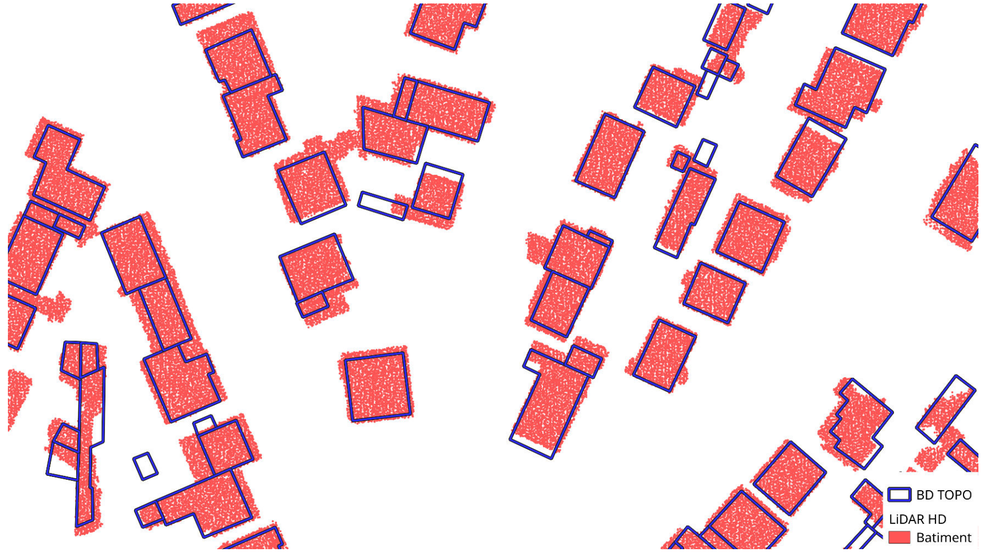


Figure 2: BD TOPO buildings are most of the time footprints, and often shifted.

- Strengths:
  - The BD TOPO contains most of the buildings in France
  - Buildings were manually delineated by humans, so their shape is generally of good quality
- Weaknesses:
  - It is a mix of footprints and roofprints, coming from different sources
  - Georeferencing is not perfect because it was not crucial for the cadastre initially

Objective

> *Generate for each building a **roofprint** and a **footprint** coherent with that roofprint, and by doing so estimate the **roof overhangs**.*

In practice our goal is to identify the roof overhangs in 3D, which is equivalent to finding the horizontal gap between the roof edge and the façade if the roof plane is known. Therefore, generating a roofprint and a footprint is enough, and we assume that the footprint is most of the time coherent with the roofprint, meaning that its edges will be parallel to the roof edges.

Overview of the methodology

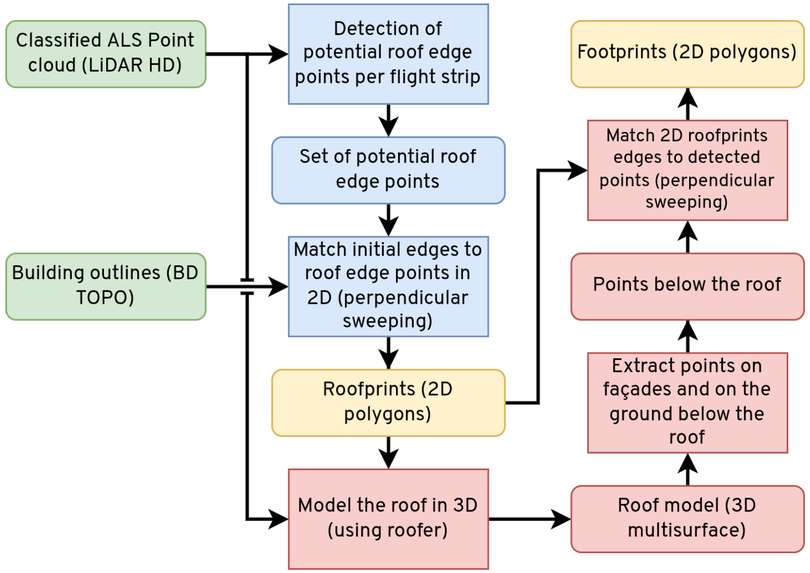


Figure 3: Overview of the pipeline.

- We start with the roofprint because aerial LiDAR data gives a much higher density of points on the roof than on the façades, making the roof much easier to identify.
- Then, once the roofprint is accurately registered on the point cloud, we can use it to generate an accurate 3D roof model that allows to select all the points below the roof (hopefully façade and ground points), which are all the points that provide information about the façades and the roof overhangs.

Roof edge points

Point cloud topology

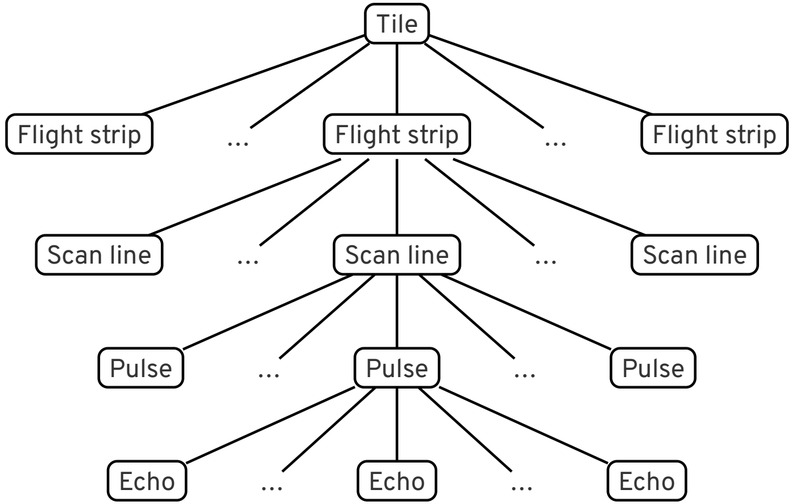
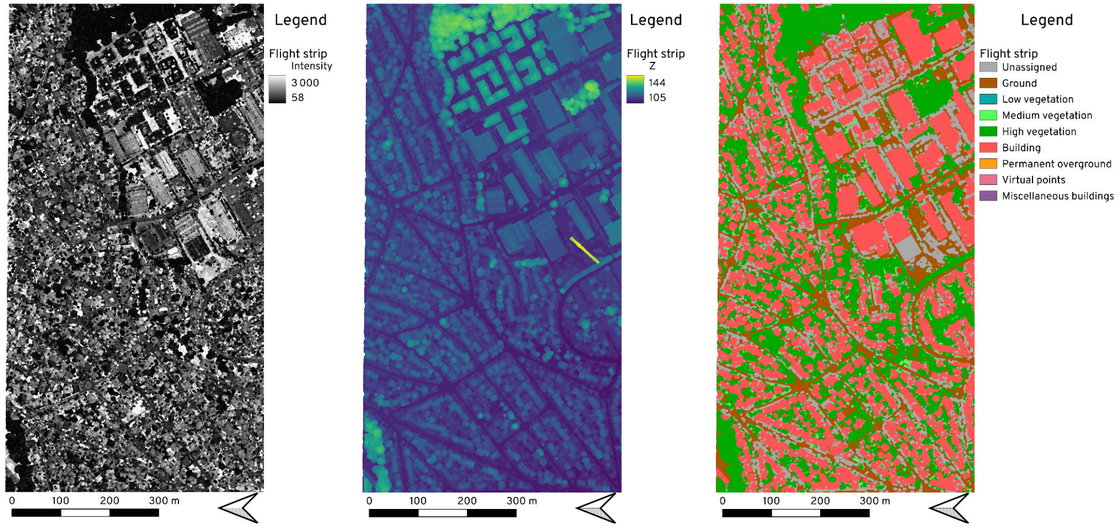


Figure 4: Illustration of the structure of an ALS point cloud.

- This creates a **hierarchy** in the point cloud, with points belonging to pulses, pulses belonging to scan lines, and scan lines belonging to flight strips.
- With multiple flight strips in the same area, this creates a sort of **irregular 3D structure**:
  - 1st dimension: the flight strip
  - 2nd dimension: the scan line
  - 3rd dimension: the pulse

  It is then possible to **navigate** in this structure along any of these dimensions.

Point cloud topology

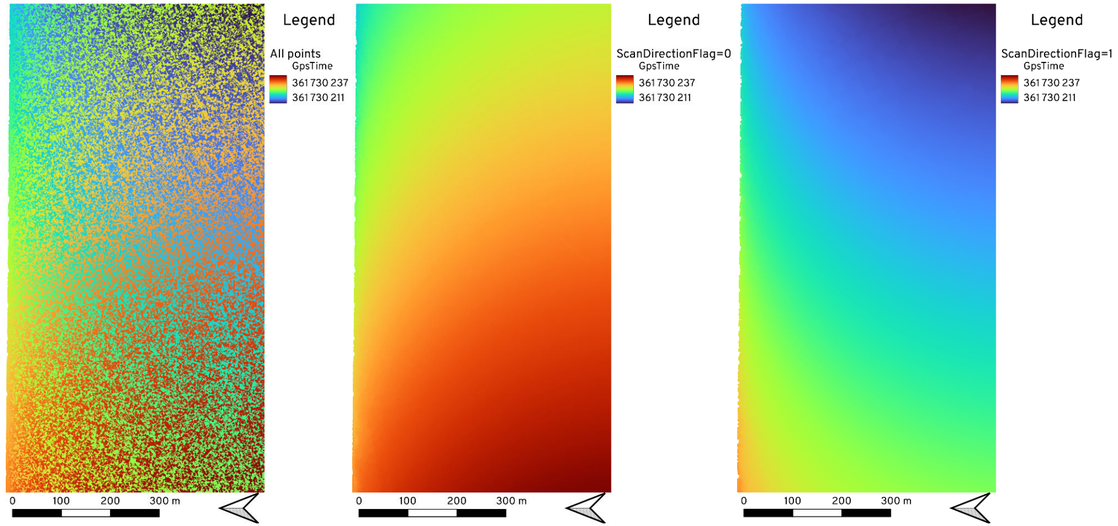


(a) Points coloured by intensity. (b) Points coloured by height. (c) Points coloured by classification.

Figure 5: Different visualizations of one flight strip.

Different visualizations of a part of a flight strip, cropped to fit in a given LiDAR HD tile.

Point cloud topology



(a) All points of one flight strip. (b) Front scan line. (c) Back scan line.

Figure 6: Visualization of the topology of the point cloud, with points coloured by their GPS Time.

Due to the ellipsoidal scanning pattern of the LiDAR scanner, the GPS Time values follow a **curved pattern**, and are mixed up when looking at the whole flight strip. However, using the **Scan Direction Flag** to separate front and back scan lines shows the distribution.

Edge points detection

We use the **height differences between consecutive points** on the same scan line to identify points that are likely to be on the edges of roofs.

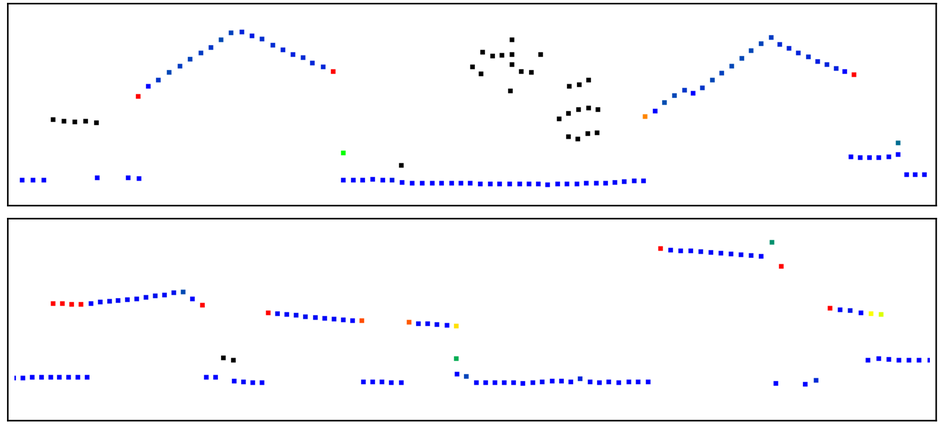


Figure 7: Two examples of the height differences obtained on a scan line. Black points are vegetation points, and for the other points the color represents the value of Δh, with blue-green-yellow-red from low to high values.

- Points at the edges of what looks like building roofs have **high values of** Δh
- Other points get **low values** of Δh
- Points classified as **vegetation** would have gotten high values if not excluded
- These scan lines are actually **curved** when looked at from above, but this is not a problem as the **angle is very small** between consecutive pulses

Creation of the roofprints

Constraints over the movements

We modify the polygons by applying the following rules:

- **Never rotate an edge.**
- **Never flip an edge.**
- **Move overlapping edges together.**

This ensures that we **preserve some topology** while keeping a **lot of freedom** for the edges to move.

However it is not perfect as it can **separate two points** that were initially at the same position.

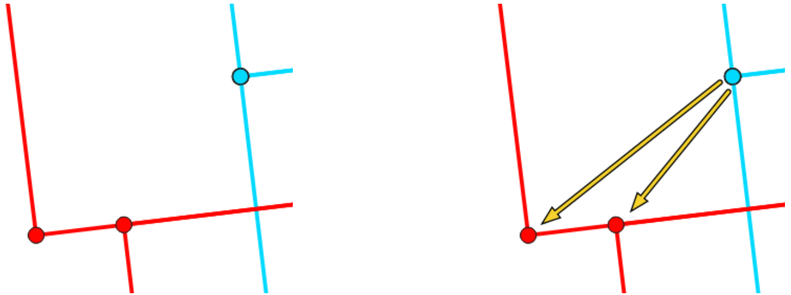


Figure 8: Two vertices at the exact same position become two distinct vertices at the boundary between two buildings.

General idea

We focus on **edges one by one**, except for overlapping edges which are treated together. Edges are treated as **directed lines**, which can be translated in any direction, but not rotated.

Therefore, to satisfy the constraints described previously, there is only one property $\mathcal{P}(l)$ that we need to preserve for each line l, regarding its previous line l^- and next line l^+:

> ***The intersection between l and l^- should occur before the intersection between l and l^+.***

This property is **equivalent to not flipping the edge l**.

Moreover, we know that if we shift l, l^- and l^+ by the **same extent in the same direction**, this property will still hold for l.

The ideas behind this slide is actually quite simple:

- We want to **move edges** without rotating them, so it is easier to simply consider them as **lines** that we shift
- By considering lines, the definition of points depends on the **intersection with the previous and next lines**, therefore not flipping edges becomes a simple property about 3 lines.
- Translating the whole polygon in one direction preserves the topology completely, so we know that there will be a **solution for any shift applied to the focus edge.**

Our simple approach

Simple algorithm to find a correct configuration given a line l_0 to move by an extent δ:

1. Compute the shift direction as the normal $\vec{n}$ of l_0.
2. Move l_0 by $\delta\vec{n}$.
3. If $\mathcal{P}(l_0)$ is not satisfied, move l_0^- and l_0^+ by $\delta\vec{n}$ as well. This ensures that $\mathcal{P}(l_0)$ is satisfied.
4. Set $l_p = l_0^-$. While either of $\mathcal{P}(l_p)$, $\mathcal{P}(l_p^-)$ or $\mathcal{P}(l_p^+)$ is not satisfied, move l_p by $\delta\vec{n}$ (if not already moved) and set $l_p = l_p^-$.
5. Set $l_n = l_0^+$. While either of $\mathcal{P}(l_n)$, $\mathcal{P}(l_n^-)$ or $\mathcal{P}(l_n^+)$ is not satisfied, move l_n by $\delta\vec{n}$ (if not already moved) and set $l_n = l_n^+$.

The only guarantees of this algorithm is that it will **terminate** and that the resulting configuration will **satisfy the constraints**.

However, it will **not find an optimal solution**, and in particular, it is **not continuous** with respect to δ.

Illustrations of the polygon deformation algorithm

Illustrations can be viewed by following this link: **https://alexandre-bry.github.io/MSc_Thesis-Report/slides_pages/2026_05_18-monthly.html**

Total energy to minimize

The energy that we try to minimize for each group of roofprints is the following:

$$E = \underbrace{-\sum_{i \in \mathcal{P}} w_i \max_{j \in \mathcal{L}} \{\text{score}(p_i, l_j)\}}_{\text{proximity to the points}} + \alpha \underbrace{\sum_{j \in \mathcal{L}} \left(|l_j| - |l_j^0|\right)^2}_{\text{similarity to the initial edges}}$$

Where:

- $\mathcal{P}$ is the set of points, and $\mathcal{L}$ is the set of edges (lines). $\mathcal{L}$ contains all the edges moved in any configuration and their neighbours.
- w_i is the weight of point p_i, which is independent of the lines.
- $\text{score}(p_i, l_j)$ is the score of point p_i for edge l_j, which is defined in the next slide.
- $|l_j|$ is the length of edge l_j in the current configuration, and $|l_j^0|$ is its initial length.
- α is a parameter to adjust between the two terms.

- The energy is divided in two terms:
  - The first term is a **proximity term** that both counts the number of points close to the edges while prioritizing points with higher weights.
  - The second term is a **regularization term** that entices the edges to keep their initial length, to prioritize a shape closer to the initial one.
- Many regularization terms are possible, but we chose this one because it makes sure that by default, the edges will **keep their initial length** and otherwise **share the change equally** between them. This is desired if we assume that the scaling that we need to perform is the same on both sides for a given direction, no matter the size of the edges, because the roof overhang is the same.

Total energy to minimize

The score of a point p_i for an edge l_j is defined as follows:

- The score is 0 if any of the following conditions is true:
  - The orthogonal projection $p_{i\perp j}$ of p_i on the supporting line of l_j is outside of the segment l_j.
  - The distance from p_i to $p_{i\perp j}$ is greater than a certain threshold ε.
  - The dot product of the inward vector v_i of p_i with the normal n_j of l_j oriented towards the inside of the building is negative.
- Otherwise, the score is:

$$\text{score}(p_i, l_j) = \underbrace{(v_i \cdot n_j)}_{\substack{\text{alignment of} \\ \text{point and edge} \\ \text{'normals'}}} \times \underbrace{\left(1 - \frac{|p_i - p_{i\perp j}|}{\varepsilon}\right)}_{\text{proximity to the edge}} \geq 0$$

We use $\varepsilon = 30$ centimetres.

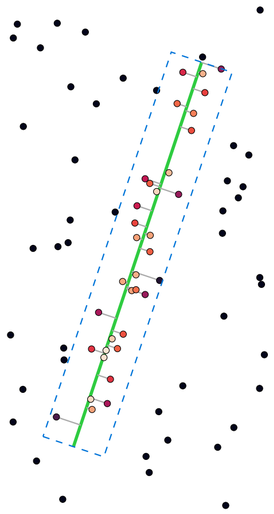


Figure 9: Illustration of the proximity score for an isolated edge.

- Any point **outside the rectangle** defined by extruding the edge along its normal by ε has a score of 0. This geometric ensures that only points that are close enough to the edge will count.
- Then, the dot product between the inward vector and the normal of the edge ensures that points that are part of a parallel but opposite edge will not count, **preventing matching to the neighbour building.**

Definition of the inward direction

The goal of the **inward direction** is to be as close as possible to the 2D normal of the roof edge, pointing towards the inside of the building. To compute it for a given point p_i, we use the following method:

1. Identify all the points p_j that are less than a certain distance δ from p_i.
2. For each point p_j, compute the vector from p_i to p_j, and normalize it into a unit vector $u_{i \to j}$.
3. Compute the inward vector v_i as the average of the vectors $u_{i \to j}$:

$$v_i = \frac{1}{|\mathcal{B}(p_i, \delta)|} \sum_{p_j \in \mathcal{B}(p_i, \delta)} \frac{p_j - p_i}{|p_j - p_i|}$$

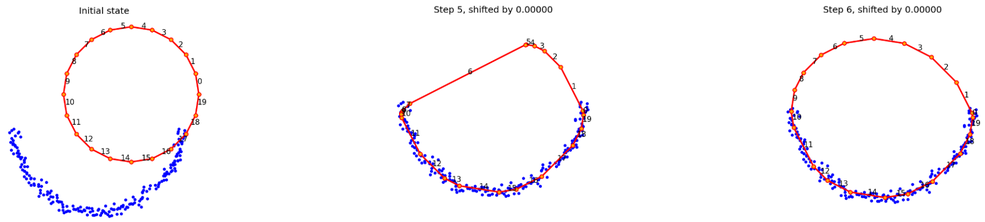
We use $\delta = 2$ metres.

- The idea behind this method is that points on the edge of the roof should have **many points towards the inside** of the building, and **few points towards the outside**.
- Using unit vectors and averaging them allows for all points in the neighbourhood to contribute equally, meaning that we are somewhat counting the **density of points** in each direction. Therefore, the **magnitude** of the inward vector is an indication of the confidence of the direction.

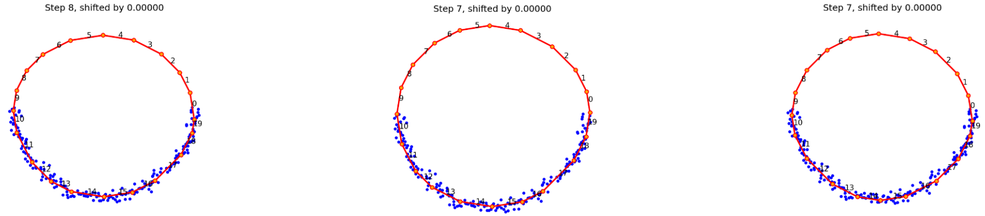
Illustrations of matching the edges to the points

Illustrations can be viewed by following this link: **https://alexandre-bry.github.io/MSc_Thesis-Report/slides_pages/2026_05_18-monthly.html**

Illustrations of matching the edges to the points



(a) Initial state (b) $\alpha = 0.00$ (no regularization) (c) $\alpha = 0.05$

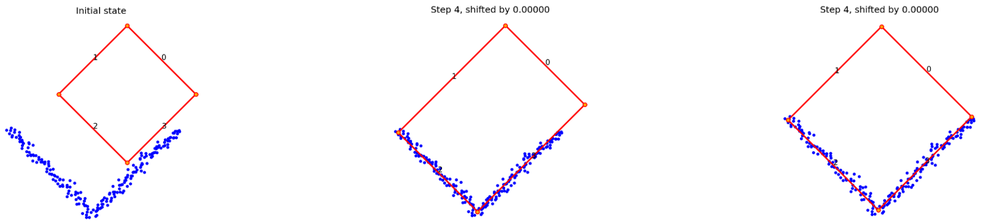


(d) $\alpha = 0.20$ (e) $\alpha = 0.50$ (f) $\alpha = 1.00$

Figure 10: Matching a circle to a half-circle of points with different values of the regularization parameter α.

We can see how regularization helps to get a shape more similar to the target even with half of the points missing.

Illustrations of matching the edges to the points



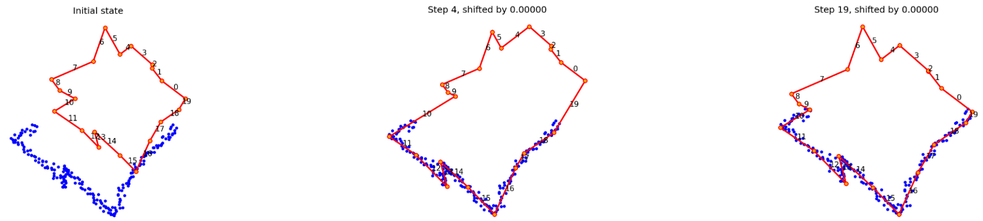
(a) Initial state (b) $\alpha = 0.00$ (no regularization) (c) $\alpha = 0.05$

(d) $\alpha = 0.20$ (e) $\alpha = 0.50$ (f) $\alpha = 1.00$

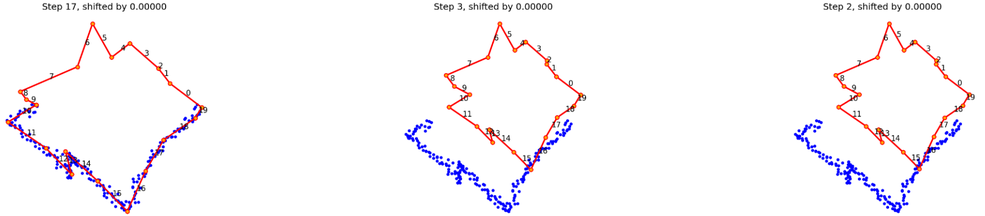
Figure 11: Matching a square to points along two sides of a larger square with different values of the regularization parameter α.

$\alpha > 0$ encourages the shape to be closer to the initial one, which is better than $\alpha = 0$, until a certain point where the regularization is too strong and prevents the shape from extending to capture all the points.

Illustrations of matching the edges to the points



(a) Initial state (b) $\alpha = 0.00$ (no regularization) (c) $\alpha = 0.05$

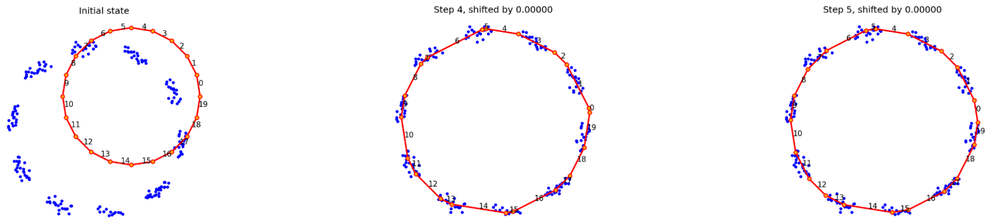


(d) $\alpha = 0.20$ (e) $\alpha = 0.50$ (f) $\alpha = 1.00$

Figure 12: Matching a polygon to half of its shape with points with different values of the regularization parameter α.

- $\alpha > 0$ forces the shape to be closer to the initial one, but it takes many more steps to converge, because the more edges that don't have points take longer to balance the regularization term between each other.
- If α is too high, it prevents the shape from matching the points, because the proximity score is not good enough to balance the regularization term, which is the case here for $\alpha \geq 0.50$.

Illustrations of matching the edges to the points



(a) Initial state (b) $\alpha = 0.00$ (no regularization) (c) $\alpha = 0.05$

(d) $\alpha = 0.20$ (e) $\alpha = 0.50$ (f) $\alpha = 1.00$

Figure 13: Matching a circle to points along half of its boundary with different values of the regularization parameter α.

At the cost of more iterations, **a larger α outputs a more regular shape**, closer in proportions to the initial one.

Illustration with a real building

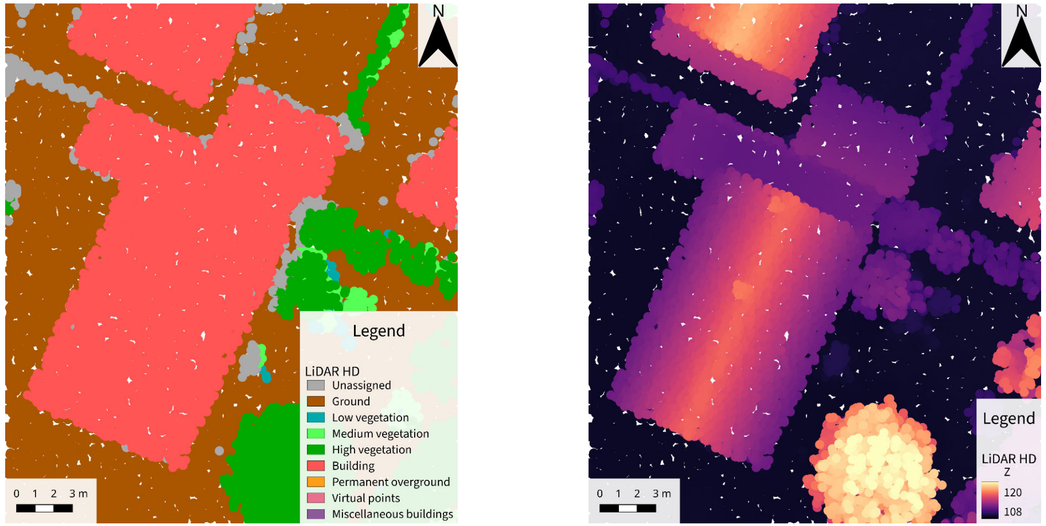


Figure 14: The LiDAR HD data.

- We can see **three different building parts** that are connected, one with a pitched roof and the two others with flat roofs.
- The points are well classified overall, but there are a number of points **classified as Unassigned (grey)**, especially at the boundaries between the building and the vegetation.

Illustration with a real building

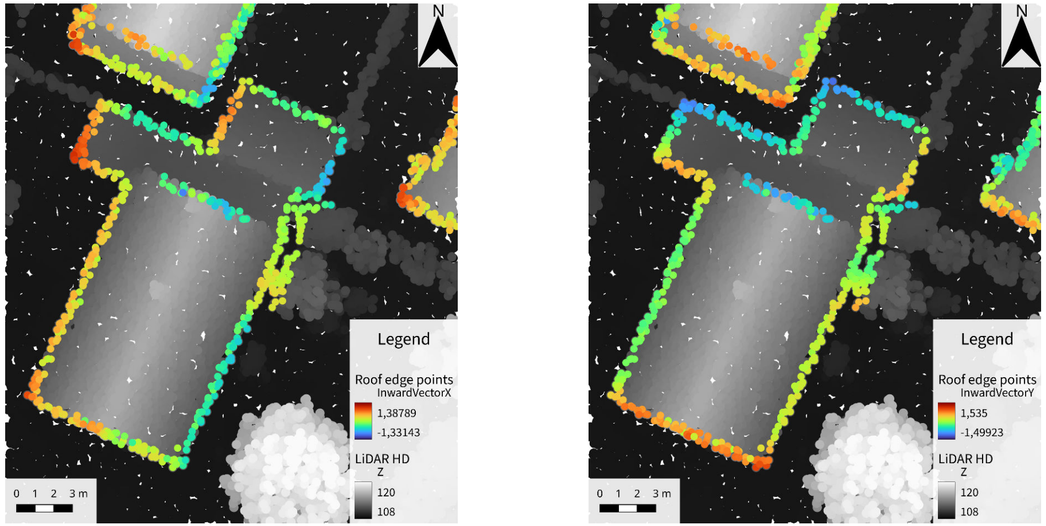


Figure 15: The detected roof edge points coloured by their inward vector.

- The detected roof edge points are shown here, coloured by their **inward vector** (the direction in which the building is located from the edge).
- We can see that the inward vectors are **generally well oriented** towards the building, except in areas where the building is **very close to vegetation** at the same height.

Illustration with a real building

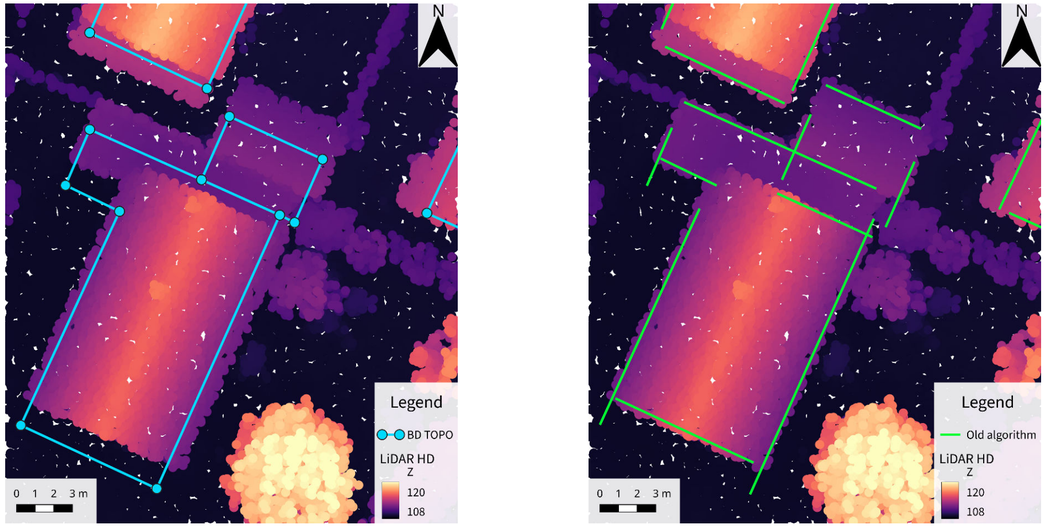


Figure 16: Comparison of BD TOPO outlines and roofprints computed with the old algorithm.

- Interestingly, this building unit is actually divided into **two parts in the BD TOPO**, even though we could expect 1 or 3 as well.
- In the old algorithm, we processed edges individually, resulting in assessing the segments that are displayed on the right.
- The results give a better alignment, but there is still an issue to fix: the bottom left edge of the small top right building was **matched incorrectly**. This could be fixed by **matching the whole initial line once** instead of keeping it split between buildings.

Illustration with a real building

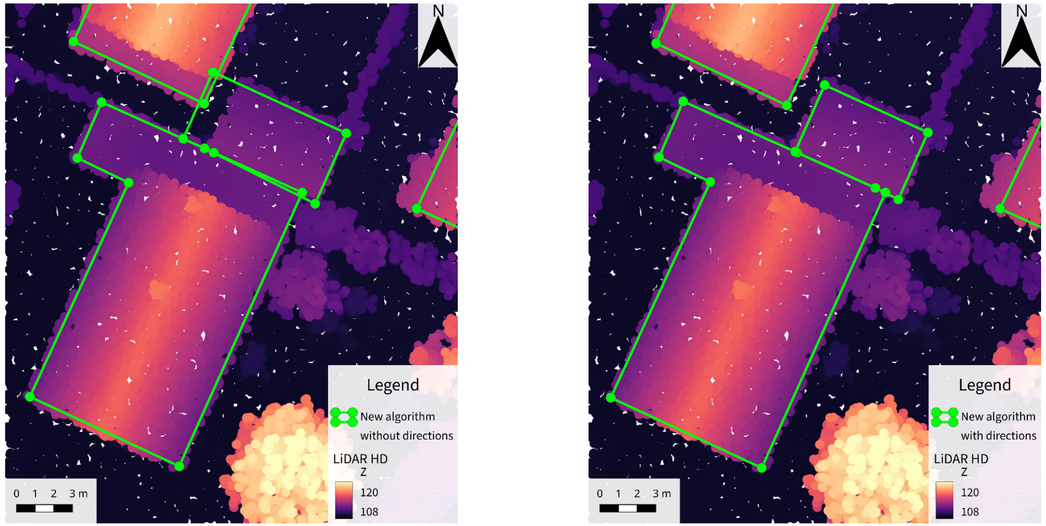


Figure 17: Comparison of the roofprints computed with the new algorithm without and with inward vectors.

- The new algorithm allows **fix the issue** for the bottom left edge of the small top right building, thanks to its alignment with the almost collinear edge of the big building.
- However, for the same building without using the inward vectors, one edge ends up **matching the neighbouring building** instead. This is fixed with the inward vectors, which prevent the points from the other building from counting in the score of the edge.

Footprint points

Footprint points

1. Build the roof in 3D using a 3D roof constructor (such as **roofer**) with the roofprints as input
2. Select all the points **under** the 3D roof with a small horizontal buffer (for the scoring function) and a small vertical buffer (for roof points slightly below the roof)

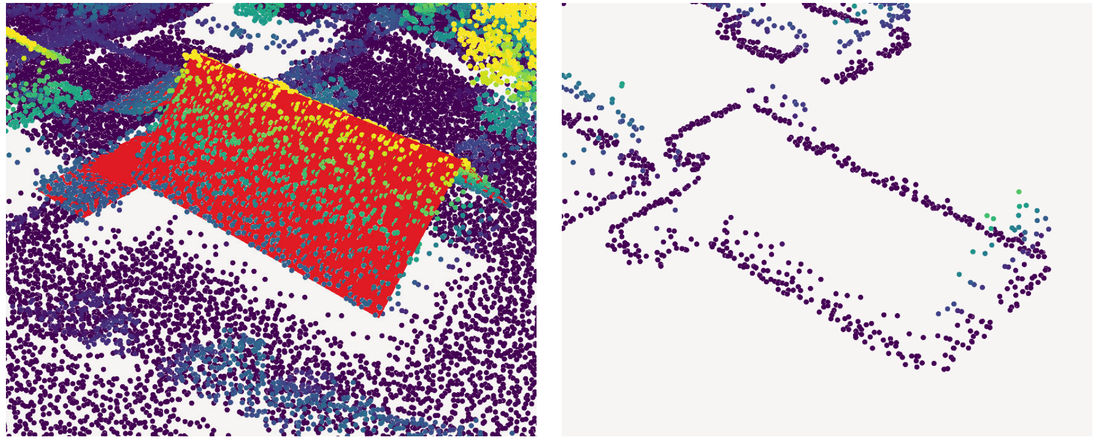
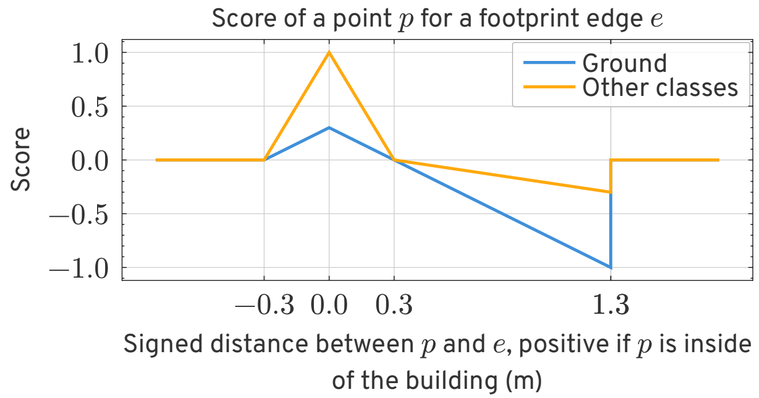


Figure 18: Illustration of the 3D roof structure and the selected points for the footprint computation.

Creation of the footprints

Creation of the footprints

- **Sweep each roof edge horizontally towards the inside** up to 1.5 metres and select the best scoring position
- The score has two components:
  1. A **proximity term** that counts the number of points close to the edge
  2. A **penalty term** that penalizes points that are behind the edge (inside the building)



- The ideas behind the two terms of the score are simple:
  - The proximity encourages finding a position with a concentration of points (likely to be the façade in 3D)
  - The penalty term discourages positions that have points behind them (likely to be points on the façade or on the ground outside the building)
- The difference between ground points and other classes of points comes from the ground points being very good indicators for the penalty term but much less for the proximity, while the others are assumed to be potential facade points, which are good indicators for the proximity but not necessarily for the penalty term.

Creation of the footprints

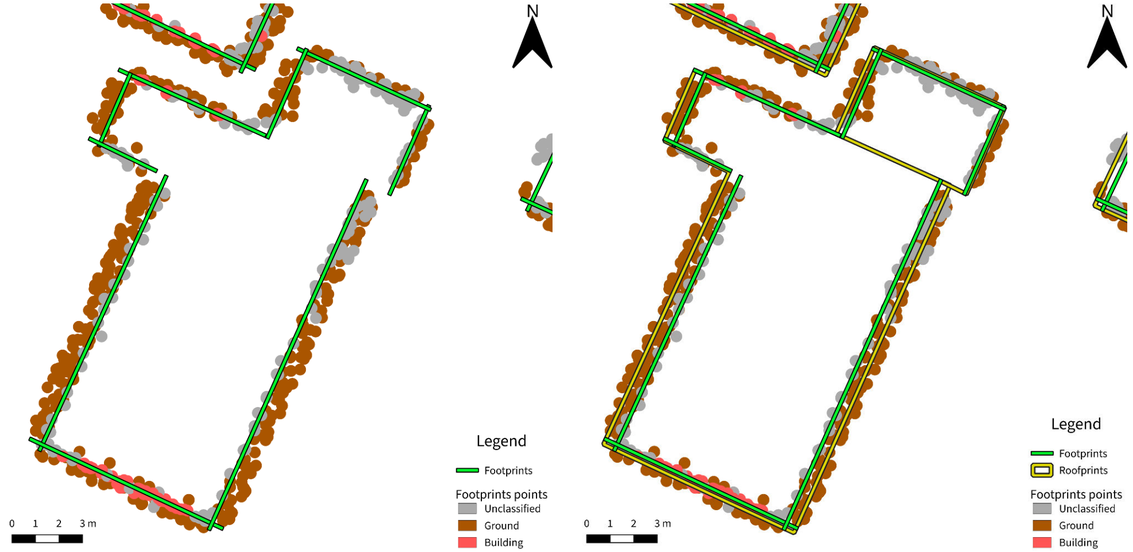


Figure 20: Illustration of the computed footprints.

Creation of the footprints

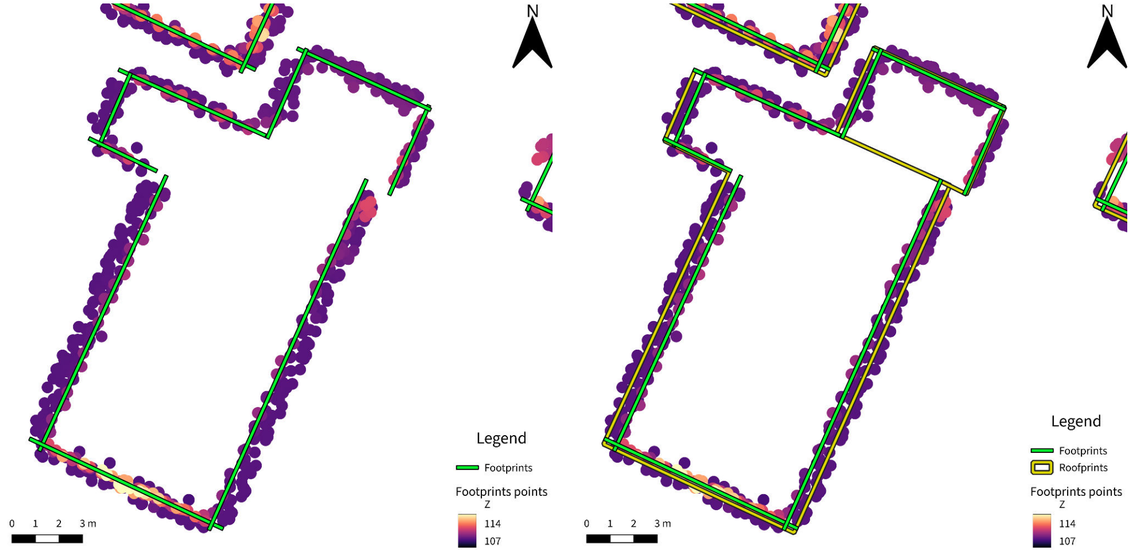


Figure 21: Illustration of the computed footprints.

Thank you for your attention!

Link to the slides:

https://alexandre-bry.github.io/MSc_Thesis-Report/slides_pages/2026_05_18-monthly.html

alexandre.bry@ign.fr, abry@tudelft.nl

The end

—

References

Biljecki, F., Ledoux, H., & Stoter, J. (2016). An improved LOD specification for 3D building models. *Computers, Environment and Urban Systems*, *59*, 25–37. https://doi.org/10.1016/j.compenvurbsys.2016.04.005

The end

—