From Points to Prints

Monthly Presentation (2)

Alexandre Bry

March 16, 2026

Title

—

Outline

Contents

Context 1
Work done until now 2
Preliminary results 14
Next objectives 22

Outline

—

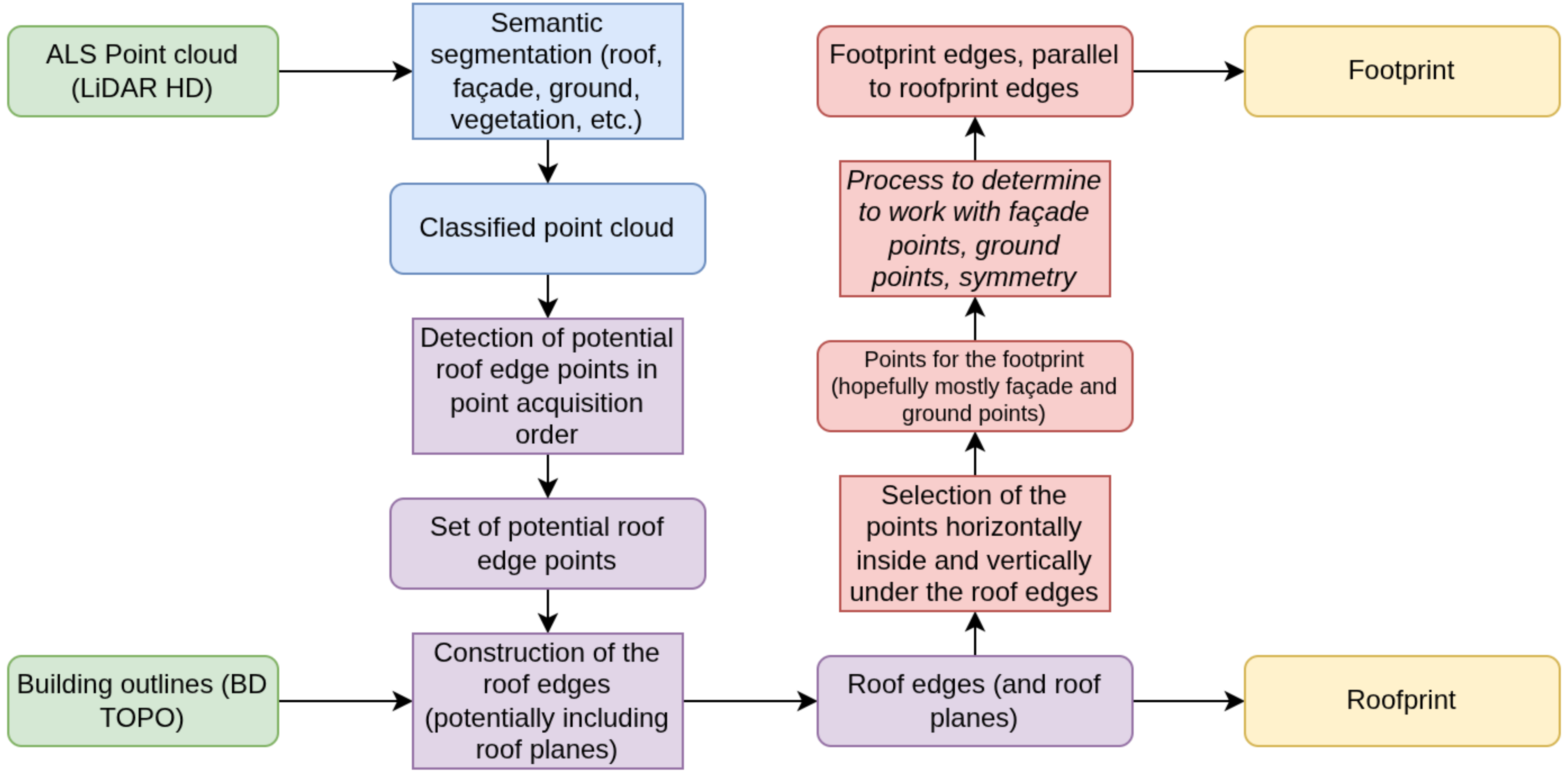
Context

Context 1
Work done until now 2
Preliminary results 14
Next objectives 22

Context

—

Planned pipeline



Context
– Planned pipeline

Work done until now

Context 1
Work done until now 2
Preliminary results 14
Next objectives 22

Point cloud topology

Pulse

A single emission of the LiDAR sensor, which may result in **zero, one or more echoes** (points) depending on the number of surfaces the pulse hits.

Scan line

A **set of pulses** emitted during one rotation of the LiDAR scanner.

Flight strip

A **set of scan lines** collected along one pass of the aircraft over the ground.

- 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.

Extraction of the topology

- LiDAR HD: **cloud-optimized tiles** with **spatially ordered points**
- Topology must be extracted in multiple steps:
  1. Flight strips: using the **Point Source ID** field
  2. Scan lines: using the **GPS Time** field and the **Scan Direction Flag** field
  3. Pulses: using the **GPS Time** field

Work done until now – Extraction of the topology

LiDAR HD is distributed as a set of **cloud-optimized tiles** (1000m x 1000m) in the COPC format. This means that **flight strips are mixed together** in the same tile, and points are **spatially ordered** instead of being ordered by acquisition time.

Therefore, to extract the topology, we use:

- For flight strips: the **Point Source ID** field, which identifies the flight strip the point belongs to
- For scan lines:
  - The **GPS Time** field, which is a timestamp of the acquisition of the point, and can be used to sort points in acquisition order
  - The **Scan Direction Flag** field, which indicates the direction of the scan and alternates between 0 and 1 for consecutive scan lines
- For pulses: the **GPS Time** field, which is the same for all points of a pulse

The **Number of Returns** field is not reliable for pulses in the LiDAR HD dataset, as it is not updated when points are filtered out during the processing of the raw data.

Extraction of the topology

- LiDAR HD: **cloud-optimized tiles** with **spatially ordered points**
- Topology must be extracted in multiple steps:
  1. Flight strips: using the **Point Source ID** field
  2. Scan lines: using the **GPS Time** field and the **Scan Direction Flag** field
  3. Pulses: using the **GPS Time** field

> The **Number of Returns** field is not reliable for pulses in the LiDAR HD dataset, as it is not updated when points are filtered out during the processing of the raw data.

Work done until now – Extraction of the topology

LiDAR HD is distributed as a set of **cloud-optimized tiles** (1000m x 1000m) in the COPC format. This means that **flight strips are mixed together** in the same tile, and points are **spatially ordered** instead of being ordered by acquisition time.

Therefore, to extract the topology, we use:

- For flight strips: the **Point Source ID** field, which identifies the flight strip the point belongs to
- For scan lines:
  - The **GPS Time** field, which is a timestamp of the acquisition of the point, and can be used to sort points in acquisition order
  - The **Scan Direction Flag** field, which indicates the direction of the scan and alternates between 0 and 1 for consecutive scan lines
- For pulses: the **GPS Time** field, which is the same for all points of a pulse

The **Number of Returns** field is not reliable for pulses in the LiDAR HD dataset, as it is not updated when points are filtered out during the processing of the raw data.

Extraction of the topology

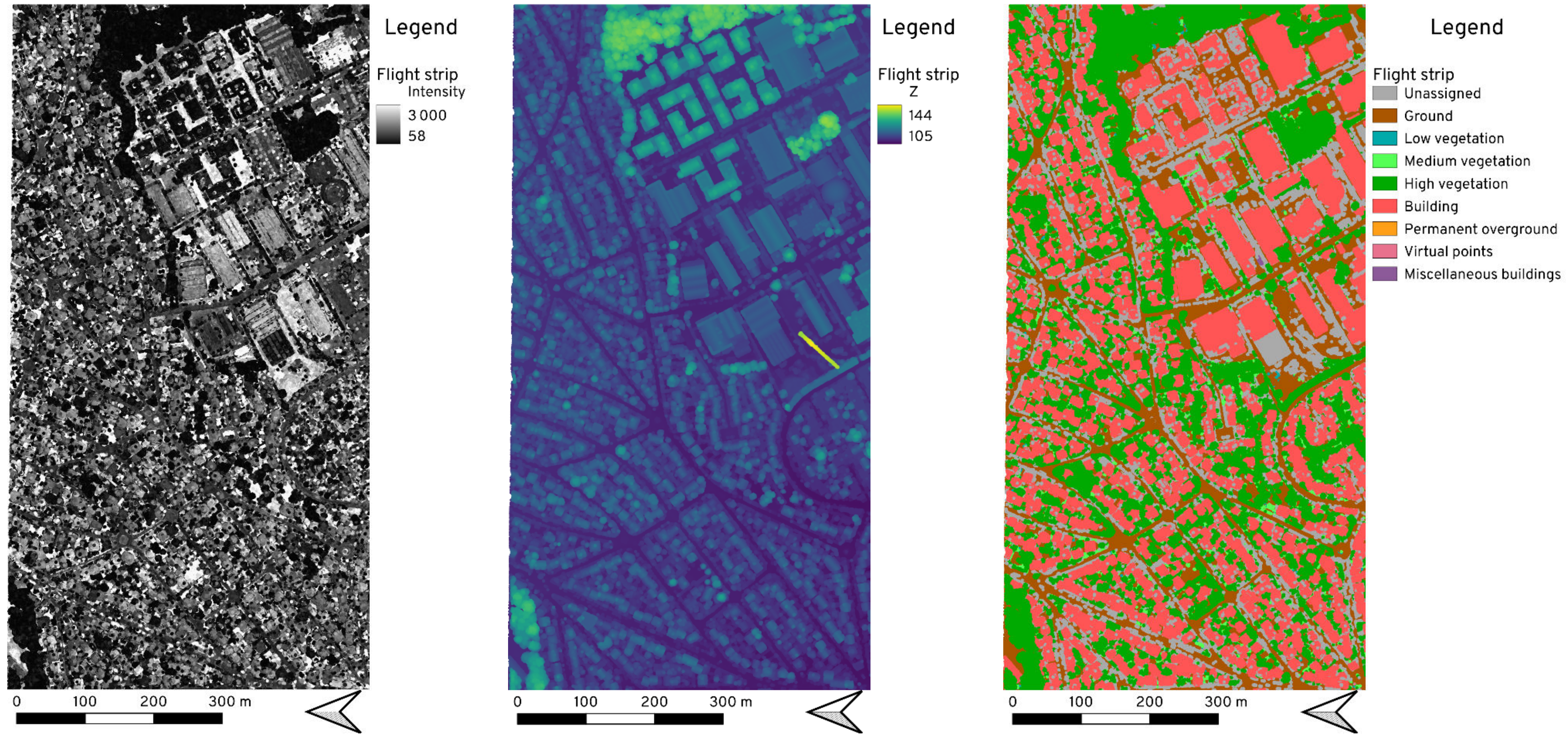


Figure 1: Visualization of the topology of the point cloud.

Work done until now
– Extraction of the topology

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

Extraction of the topology

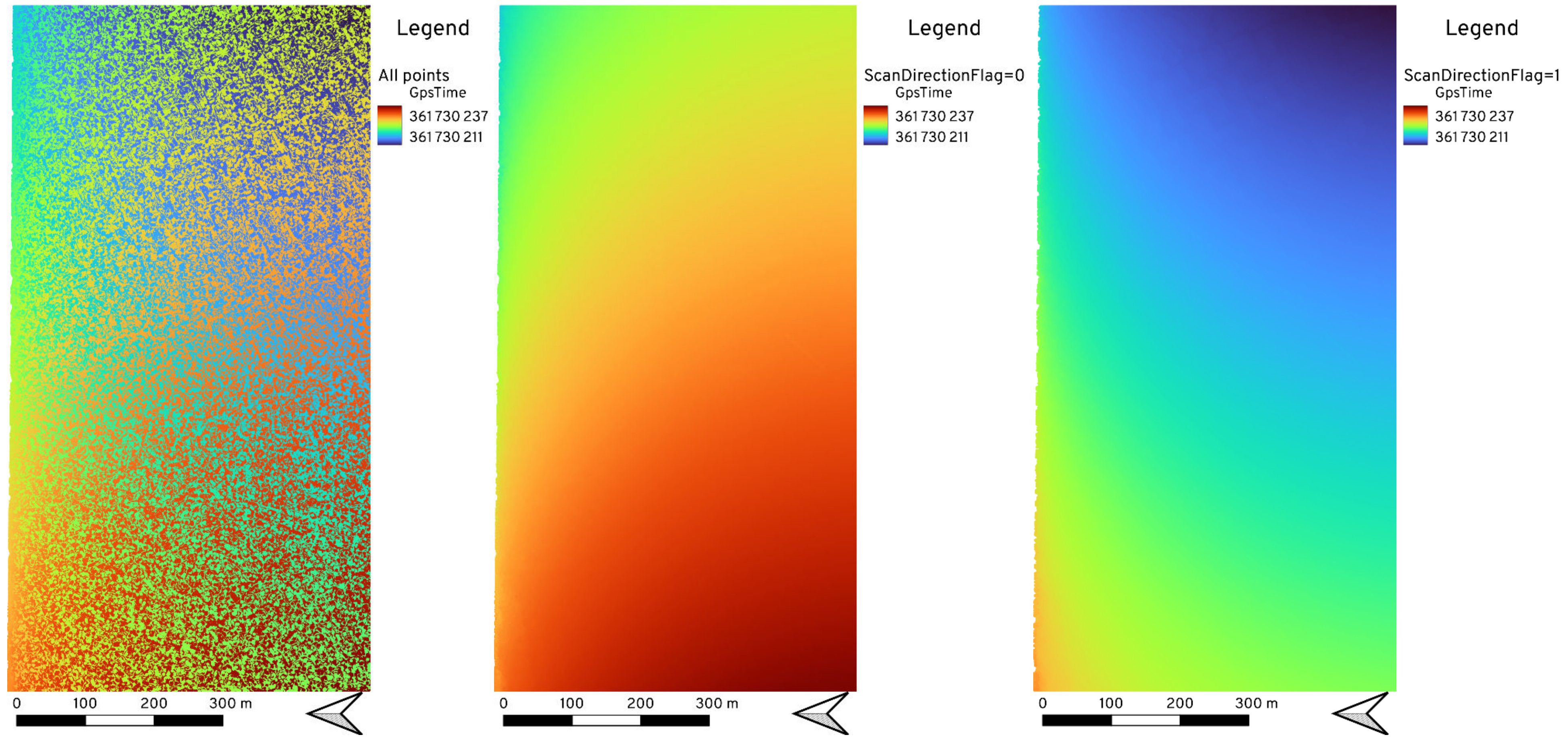


Figure 2: Visualization of the topology of the point cloud.

Work done until now
– Extraction of the topology

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.

Flight strips trajectories

- Trajectories computed using **multi-echo pulses** (code written by Wu Teng)
- Necessity to **reconcatenate the different parts of each flight strip** for better precision

- I use code written by Wu Teng to **retrieve the trajectory** of the scanning vehicle, using **multi-echo pulses**
- The precision of the method increases with the number of multi-echo pulses, meaning that it is necessary to **reconcatenate the different parts of each flight strip** (scattered between tiles) in order to get a good estimation of the trajectory.
- This method allows to use the trajectory **without relying on its availability**, and therefore without adding new requirements on the input data.

Flight strips trajectories

- Trajectories computed using **multi-echo pulses** (code written by Wu Teng)
- Necessity to **reconcatenate the different parts of each flight strip** for better precision

> This method allows to use the trajectory **without relying on its availability**.

- I use code written by Wu Teng to **retrieve the trajectory** of the scanning vehicle, using **multi-echo pulses**
- The precision of the method increases with the number of multi-echo pulses, meaning that it is necessary to **reconcatenate the different parts of each flight strip** (scattered between tiles) in order to get a good estimation of the trajectory.
- This method allows to use the trajectory **without relying on its availability**, and therefore without adding new requirements on the input data.

Identification of edge-indicator points

Multi-echo pulse

1. Find the **lowest point** in the same pulse
2. Compute the **height difference** (Δh) between the point and this lowest point
3. Set the point as a **edge-indicator point** if $\Delta h > (\Delta h)_{\text{min}} = 2\ m$

Work done until now
– Identification of edge-indicator points

Each points is compared to the **lowest point in the neighbourhood**:

- The lowest point in the same pulse in case of a multi-echo pulse
- The lowest point in the previous and next pulses in case of a single-echo pulse

To account for the variation of angular difference between pulses in ellipsoidal LiDAR scanners, the **height difference** (Δh) between the points is divided by the **temporal difference** (Δt) between the pulses when comparing to other pulses:

$$v = \frac{\Delta h}{\Delta t}$$

The temporal difference Δt is assumed to be a **good approximation of the angular difference**, as the scanning speed is constant.

Finally, this value is compared to a threshold v_{min}, which is set to $v_{\text{min}} = 2.10^6$ with $(\Delta h)_{\text{min}} = 2$ in metres and $(\Delta t)_{\text{min}} = 10^{-6}$ in seconds. For multi-echo pulses, since $\Delta t = 0$, the height difference is directly compared to the threshold: $\Delta h > (\Delta h)_{\text{min}}$.

Identification of edge-indicator points

Multi-echo pulse

1. Find the **lowest point** in the same pulse
2. Compute the **height difference** (Δh) between the point and this lowest point
3. Set the point as a **edge-indicator point** if $\Delta h > (\Delta h)_{\min} = 2\ m$

Single-echo pulse

1. Find the **lowest point** in the previous and next pulses
2. Compute the **height difference** (Δh) between the point and these lowest points and the **temporal difference** (Δt) between the pulses
3. Set the point as a **edge-indicator point** if

$$\frac{\Delta h}{\Delta t} > \frac{(\Delta h)_{\min}}{(\Delta t)_{\min}} = \frac{2}{10^{-6}} = 2.10^6\ ms^{-1}$$

for any of the two comparisons.

Work done until now
– Identification of edge-indicator points

Each points is compared to the **lowest point in the neighbourhood**:

- The lowest point in the same pulse in case of a multi-echo pulse
- The lowest point in the previous and next pulses in case of a single-echo pulse

To account for the variation of angular difference between pulses in ellipsoidal LiDAR scanners, the **height difference** (Δh) between the points is divided by the **temporal difference** (Δt) between the pulses when comparing to other pulses:

$$v = \frac{\Delta h}{\Delta t}$$

The temporal difference Δt is assumed to be a **good approximation of the angular difference**, as the scanning speed is constant.

Finally, this value is compared to a threshold $v_{\min}$, which is set to $v_{\min} = 2.10^6$ with $(\Delta h)_{\min} = 2$ in metres and $(\Delta t)_{\min} = 10^{-6}$ in seconds. For multi-echo pulses, since $\Delta t = 0$, the height difference is directly compared to the threshold: $\Delta h > (\Delta h)_{\min}$.

Identification of edge-indicator points

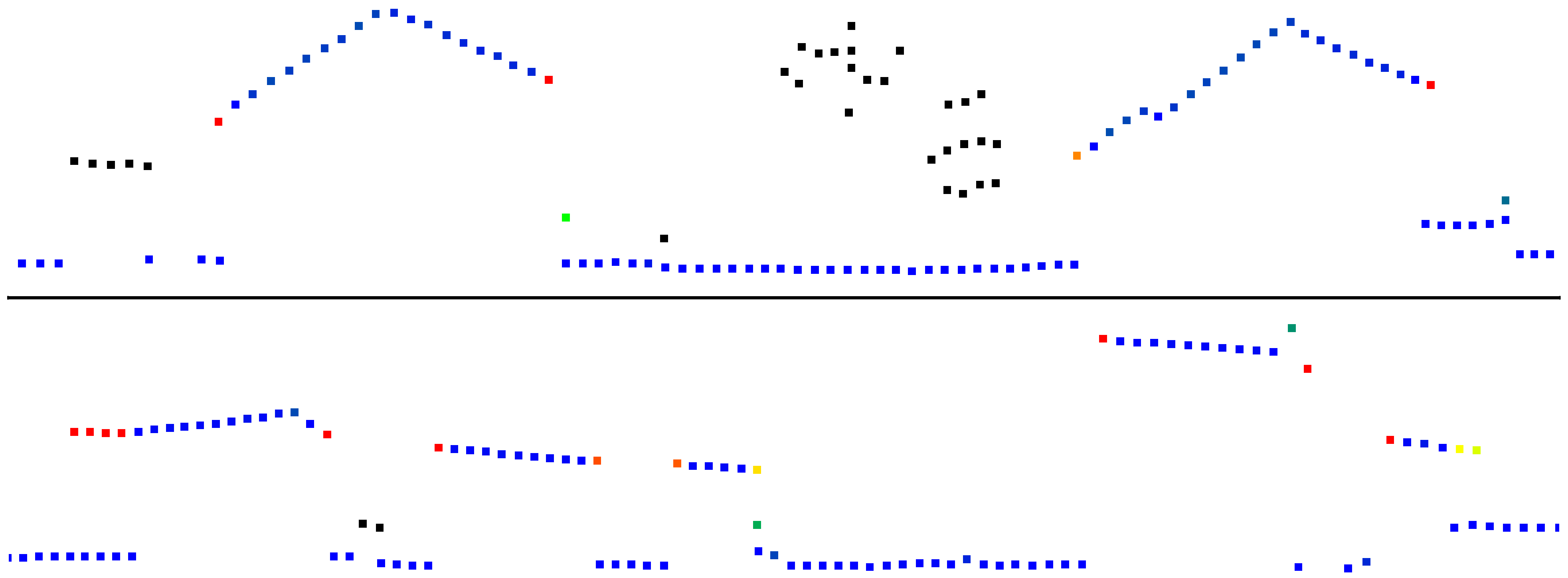


Figure 3: Two examples of the height differences obtained on a scan line. Black points are vegetation points, and for the other points the color represents the value of Δh, with blue-green-yellow-red from low to high values.

Work done until now
– Identification of edge-indicator points

- 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

Multi-echo pulse

The edge-indicator point directly gives the position of the **edge point**.

- In multi-echo pulses, the pulse is assumed to have hit both the roof and the façade/ground, which means that the edge-indicator point is assumed to be **well positioned** on the roof edge.

Placement of edge points

Single-echo pulse

The edge-indicator point **needs to be refined** to get a better estimation of the position of the edge point.

Work done until now
– Placement of edge points

In single-echo pulses, the edge-indicator point is assumed to be the last point to have hit the roof, which means that the actual edge point is assumed to be **somewhere between this point and the lowest point in the neighbourhood.**

1. Aligned points are assumed to be a **good approximation of the roof slope**, and the **half-way approximation** is the best guess for the position of the edge point.
2. Non-aligned points cannot be used to estimate the roof slope, so we fall back to using the position of the scanner to simply **rotate the point around the scanner**, as we do not know the actual distance. For a pitched roof, the edge point we obtain with this method will be **too high**, but its **horizontal position** is what matters for computing roofprints in 2D.

Placement of edge points

Single-echo pulse

The edge-indicator point **needs to be refined** to get a better estimation of the position of the edge point.

Case 1: Aligned points

If the 3 previous points (p_1, p_2, p_3) in the same scan line (in the direction of the roof) are aligned with the edge-indicator point (p_0), then the edge point p_e is generated by **translating** p_0 with

$$p_e = p_0 + \frac{p_1 - p_0}{2}$$

Work done until now – Placement of edge points

In single-echo pulses, the edge-indicator point is assumed to be the last point to have hit the roof, which means that the actual edge point is assumed to be **somewhere between this point and the lowest point in the neighbourhood.**

1. Aligned points are assumed to be a **good approximation of the roof slope**, and the **half-way approximation** is the best guess for the position of the edge point.
2. Non-aligned points cannot be used to estimate the roof slope, so we fall back to using the position of the scanner to simply **rotate the point around the scanner**, as we do not know the actual distance. For a pitched roof, the edge point we obtain with this method will be **too high**, but its **horizontal position** is what matters for computing roofprints in 2D.

Placement of edge points

Single-echo pulse

The edge-indicator point **needs to be refined** to get a better estimation of the position of the edge point.

Case 1: Aligned points

If the 3 previous points (p_1, p_2, p_3) in the same scan line (in the direction of the roof) are aligned with the edge-indicator point (p_0), then the edge point p_e is generated by **translating** p_0 with

$$p_e = p_0 + \frac{p_1 - p_0}{2}$$

Case 2: Non-aligned points

Otherwise we use the **position of the scanner** p_s at the time of the pulse to generate the half-way pulse direction between p_0 and the next point p_g (supposed to be on the ground or on the façade) with

$$v_e = \frac{\mathrm{normalize}(p_0 - p_s) + \mathrm{normalize}(p_g - p_s)}{2}$$

Then we compute the edge point with

$$p_e = p_s + \mathrm{norm}(p_0 - p_s) v_e$$

Work done until now – Placement of edge points

In single-echo pulses, the edge-indicator point is assumed to be the last point to have hit the roof, which means that the actual edge point is assumed to be **somewhere between this point and the lowest point in the neighbourhood**.

1. Aligned points are assumed to be a **good approximation of the roof slope**, and the **half-way approximation** is the best guess for the position of the edge point.
2. Non-aligned points cannot be used to estimate the roof slope, so we fall back to using the position of the scanner to simply **rotate the point around the scanner**, as we do not know the actual distance. For a pitched roof, the edge point we obtain with this method will be **too high**, but its **horizontal position** is what matters for computing roofprints in 2D.

Weighing of edge points

Edge points are weighted using:

1. Their **height** (normalized over the area of the building) to a factor $w_h \in [1.0, 3.0]$

1. The **height** is a **simple but effective** metric to give more weight to points on a roof edge than to the points on the façade below it.
2. The **origin** gives more weight to real points that are **less likely to be imprecisely positioned.**
3. The **classification** gives more weight to points classified as building, which are **less likely to actually be something else such as vegetation.** This is important because many points are unclassified.

Weighing of edge points

Edge points are weighted using:

1. Their **height** (normalized over the area of the building) to a factor $w_h \in [1.0, 3.0]$
2. Their **origin** to a factor $w_o \in \{0.1, 0.5, 1.0\}$:
   - Multi-echo: 1.0
   - Single echo with good estimation of position: 0.5
   - Single echo with less precise estimation of position: 0.1

Work done until now
– Weighing of edge points

1. The **height** is a **simple but effective** metric to give more weight to points on a roof edge than to the points on the façade below it.
2. The **origin** gives more weight to real points that are **less likely to be imprecisely positioned.**
3. The **classification** gives more weight to points classified as building, which are **less likely to actually be something else such as vegetation.** This is important because many points are unclassified.

Weighing of edge points

Edge points are weighted using:

1. Their **height** (normalized over the area of the building) to a factor $w_h \in [1.0, 3.0]$
2. Their **origin** to a factor $w_o \in \{0.1, 0.5, 1.0\}$:
   - Multi-echo: 1.0
   - Single echo with good estimation of position: 0.5
   - Single echo with less precise estimation of position: 0.1
3. Their **classification** to a factor $w_c \in \{1.0, 2.0\}$:
   - Building: 2.0
   - Other: 1.0

Work done until now
– Weighing of edge points

1. The **height** is a **simple but effective** metric to give more weight to points on a roof edge than to the points on the façade below it.
2. The **origin** gives more weight to real points that are **less likely to be imprecisely positioned.**
3. The **classification** gives more weight to points classified as building, which are **less likely to actually be something else such as vegetation**. This is important because many points are unclassified.

Weighing of edge points

Edge points are weighted using:

1. Their **height** (normalized over the area of the building) to a factor $w_h \in [1.0, 3.0]$
2. Their **origin** to a factor $w_o \in \{0.1, 0.5, 1.0\}$:
   - Multi-echo: 1.0
   - Single echo with good estimation of position: 0.5
   - Single echo with less precise estimation of position: 0.1
3. Their **classification** to a factor $w_c \in \{1.0, 2.0\}$:
   - Building: 2.0
   - Other: 1.0

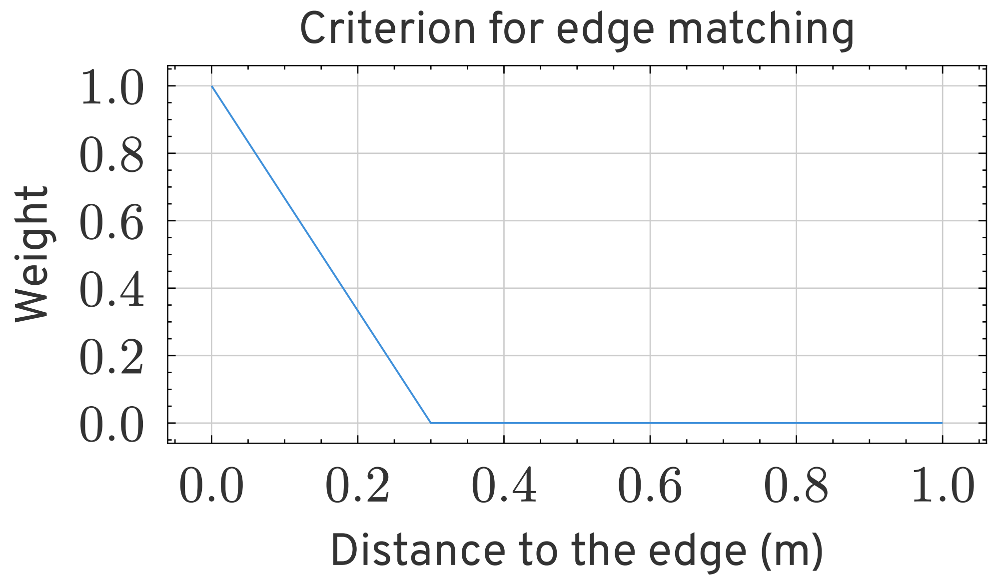
These values were picked arbitrarily and would need to be optimized. The final weight w is given by:

$$w = w_h \times w_o \times w_c$$

1. The **height** is a **simple but effective** metric to give more weight to points on a roof edge than to the points on the façade below it.
2. The **origin** gives more weight to real points that are **less likely to be imprecisely positioned.**
3. The **classification** gives more weight to points classified as building, which are **less likely to actually be something else such as vegetation.** This is important because many points are unclassified.

Displacement of BD TOPO edges

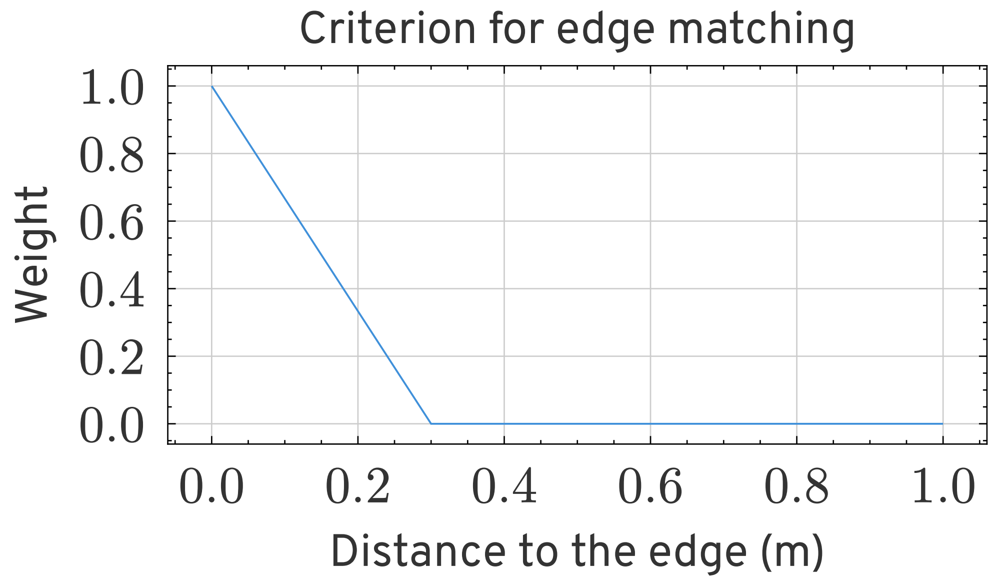
The value of a point is not only determined by its weight, but also by its **distance to the edge**. Distances are computed **in 2D** after getting rid of the vertical component of the position.



Work done until now
– Displacement of BD TOPO edges

- The decreasing weight based on distance to the edge allows to find the **best position** for the edge.
- The current threshold of **30 cm** was **picked arbitrarily** and would need to be optimized.
- Discarding points that **do not project on the segment** prevents matching to a longer edge that would be close but not actually the right one.

The value of a point is not only determined by its weight, but also by its **distance to the edge.** Distances are computed **in 2D** after getting rid of the vertical component of the position.



This distance is evaluated by projecting the point onto the edge, keeping only the points that project on the segment.

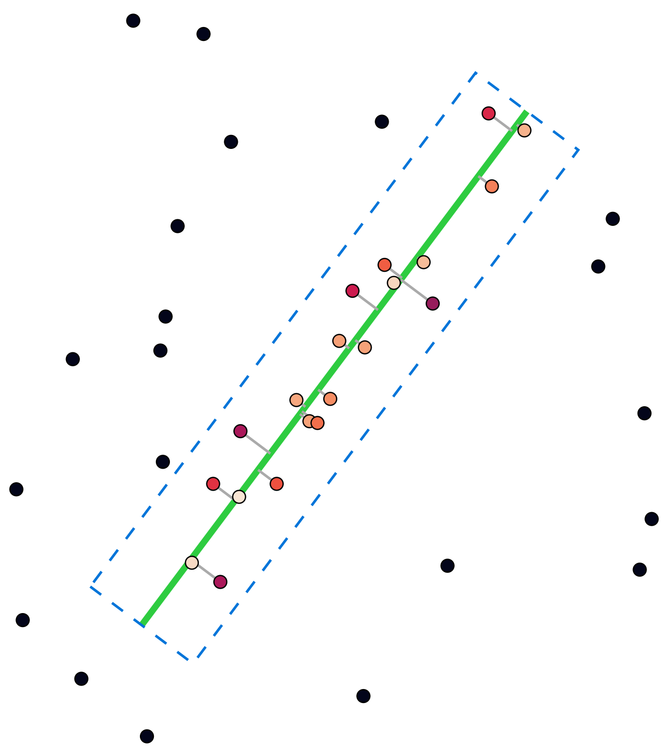
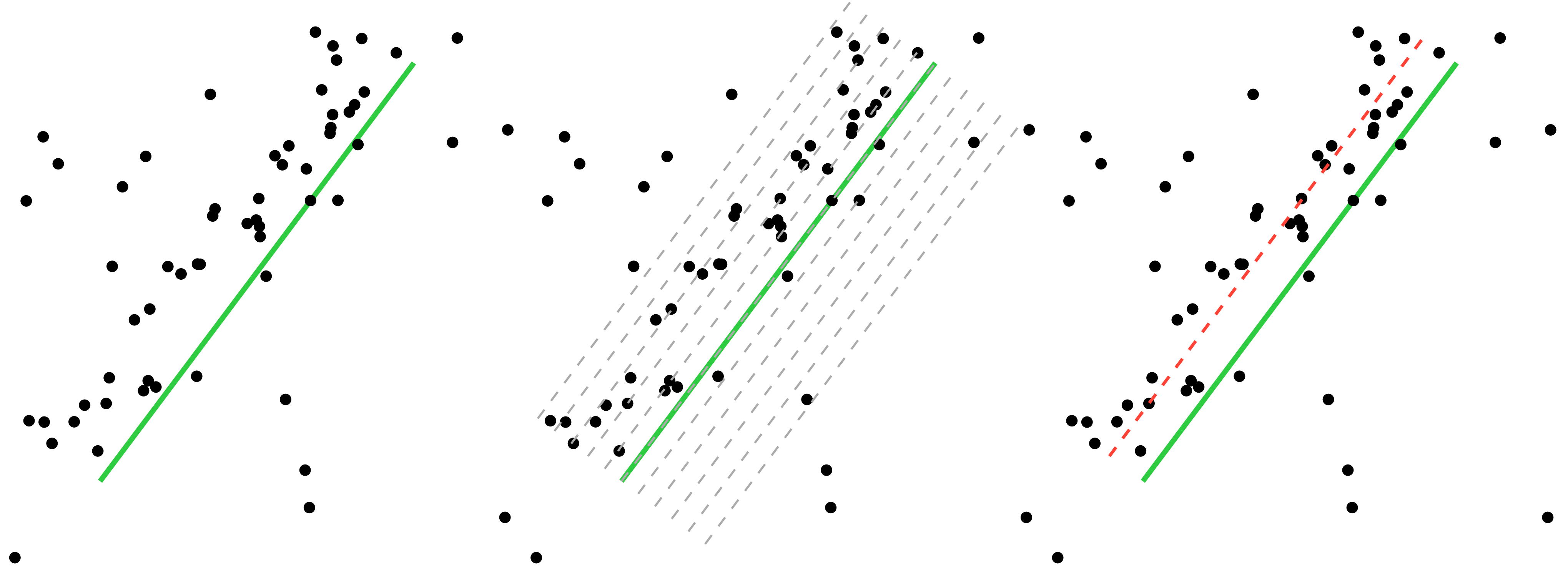


Figure 5: Illustration of the criterion that scores edges.

- The decreasing weight based on distance to the edge allows to find the **best position** for the edge.
- The current threshold of **30 cm** was **picked arbitrarily** and would need to be optimized.
- Discarding points that **do not project on the segment** prevents matching to a longer edge that would be close but not actually the right one.

Displacement of BD TOPO edges



(a) Step 0 (b) Step 1 (c) Step 2

Figure 6: Process of finding a better edge position.

Work done until now
– Displacement of BD TOPO edges

The BD TOPO edges are translated perpendicularly to themselves, and the criterion is evaluated for each translation.

Preliminary results

Context 1
Work done until now 2
Preliminary results 14
Next objectives 22

Identification of edge points

Figure 7: Edge points in blue and LiDAR HD data in gray scale from low (black) to high (white) intensity.

Preliminary results
– Identification of edge points

Identification of edge points

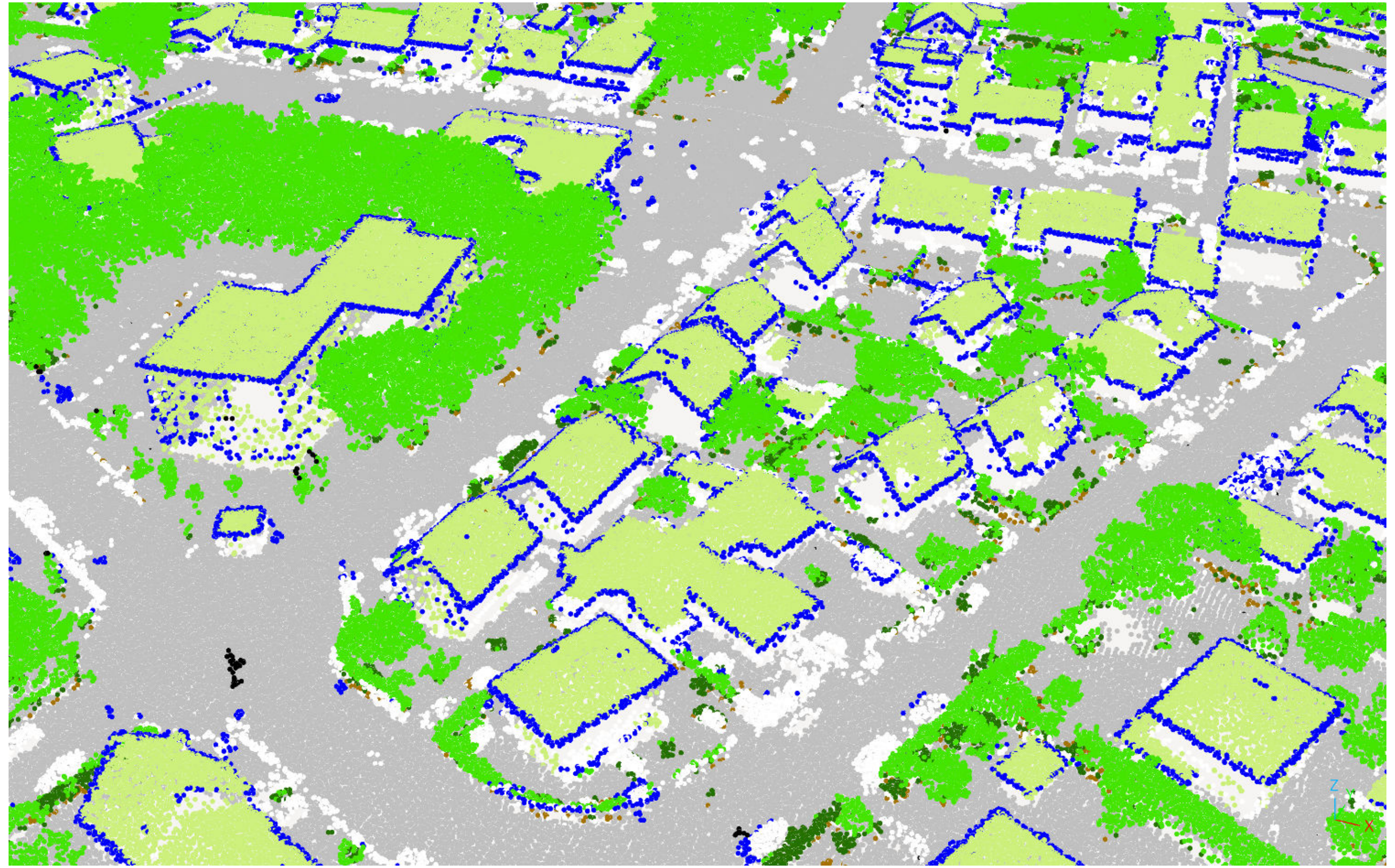


Figure 8: Edge points in blue and LiDAR HD data coloured by classification.

Preliminary results
– Identification of edge points

Identification of edge points

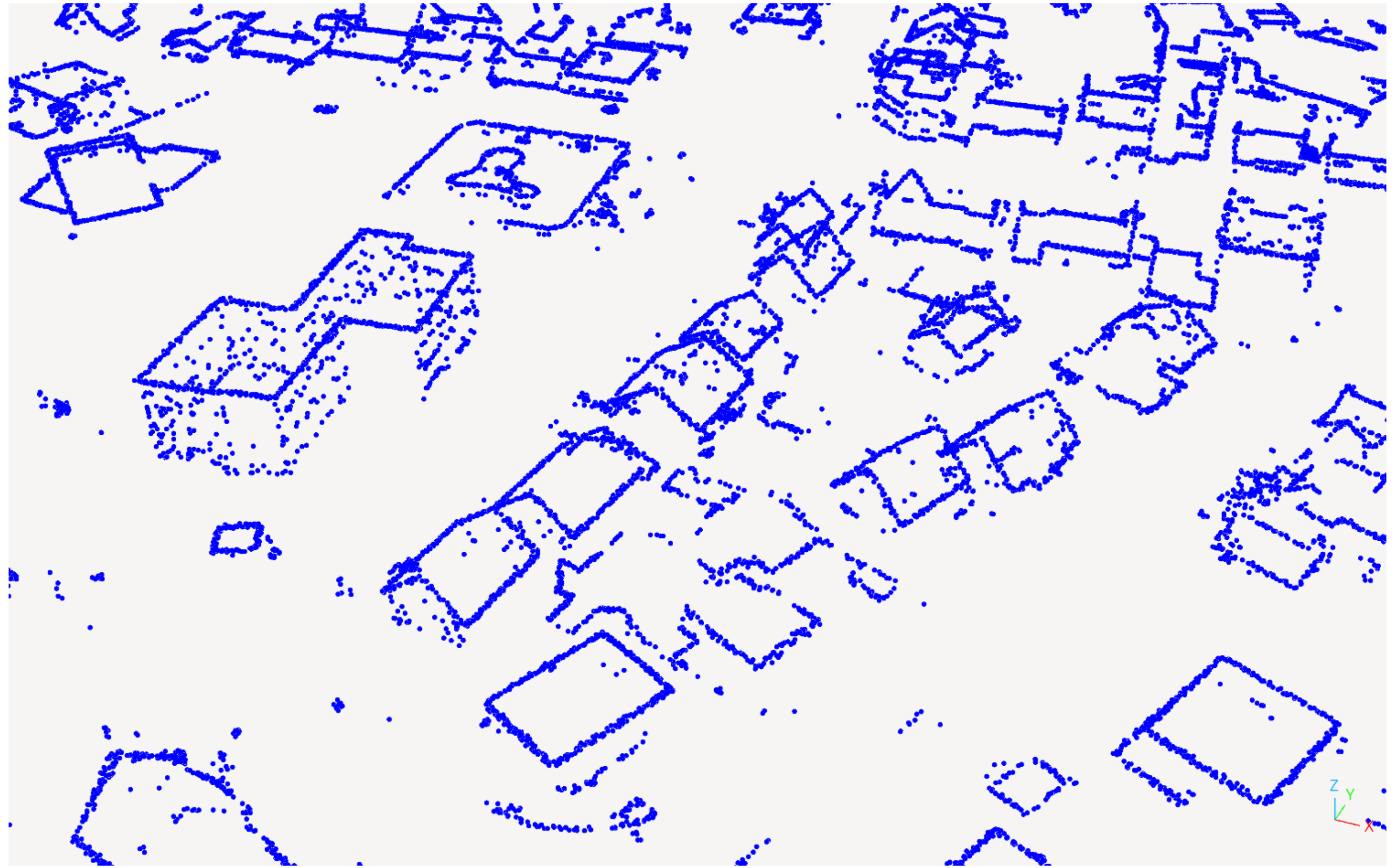


Figure 9: Edge points.

Preliminary results
– Identification of edge points

Identification of edge points

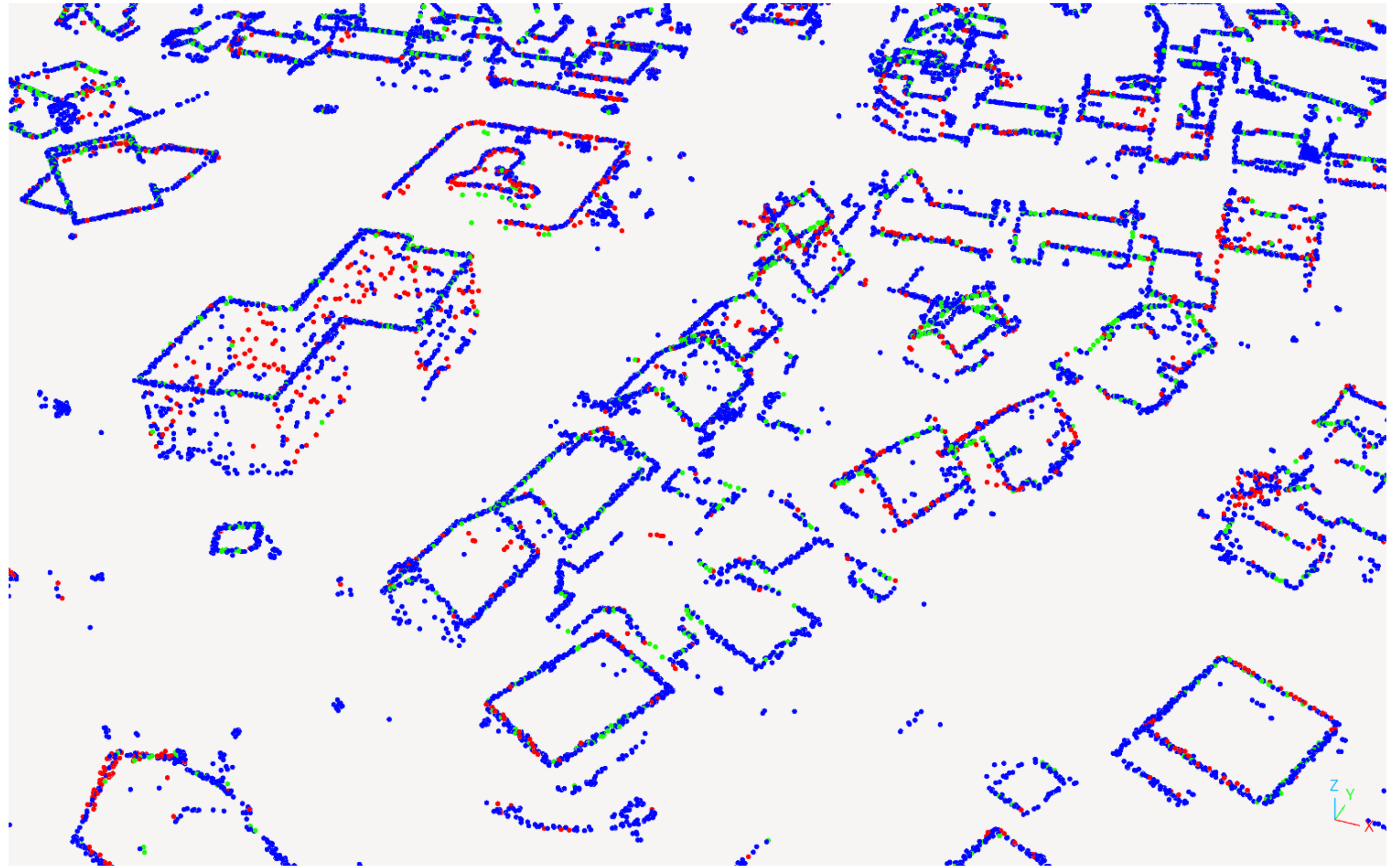


Figure 10: Edge points coloured per type (blue: real point, green: generated by translation, red: generated with trajectory).

Preliminary results
– Identification of edge points

Computation of roofprints from BD TOPO

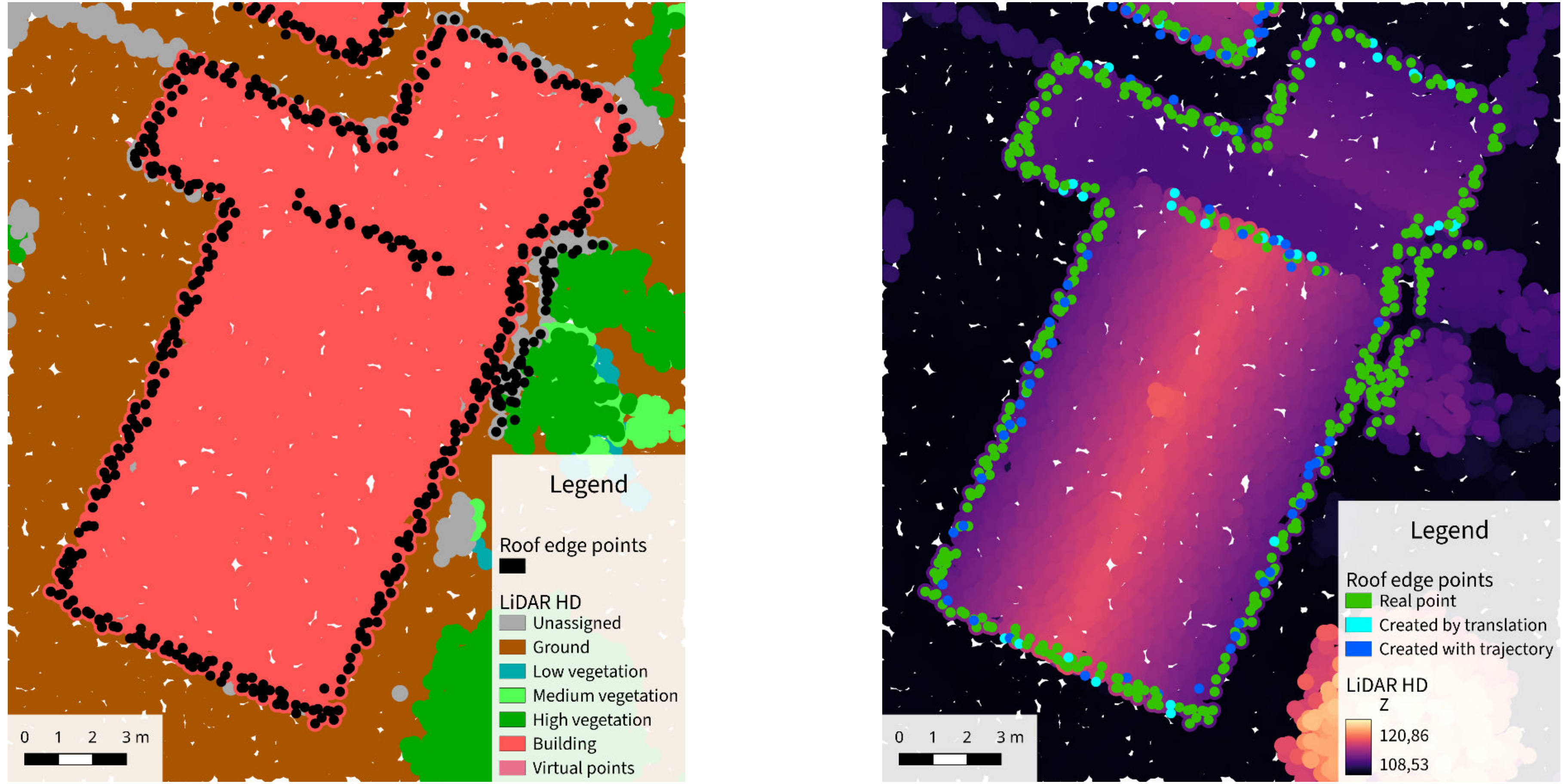


Figure 11: The potential roof edge points.

- We can see **points almost everywhere on the outer edges** of the roofs, except on the part of the taller building that has a small height difference with the other building.
- Most of the points are real points (in green) and the generated points are mostly well positioned (in light and dark blue).
- However **many incorrect points** are found on the part of the tree that is close to the building and **not classified as vegetation.**

Computation of roofprints from BD TOPO

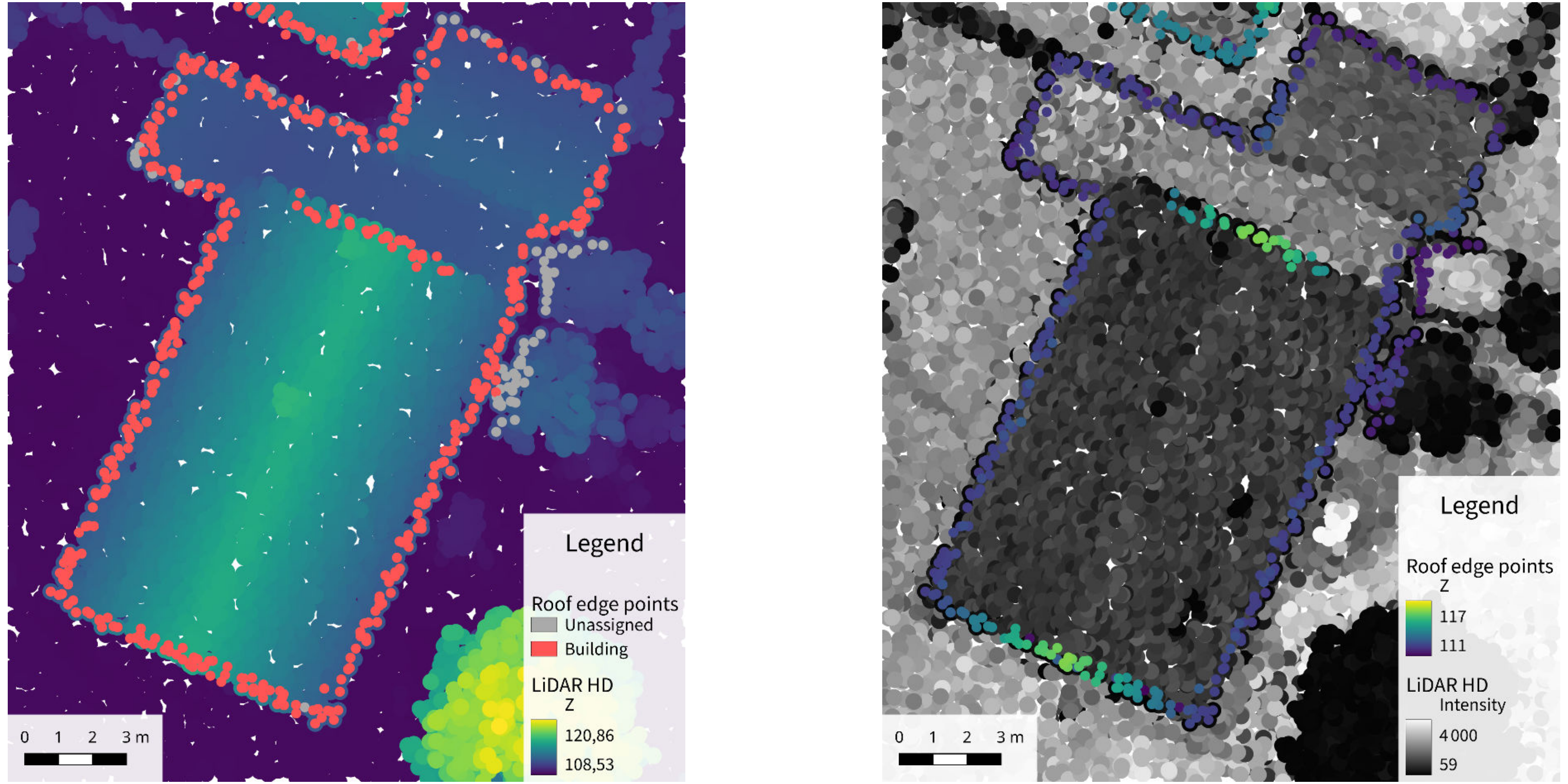


Figure 12: The potential roof edge points.

Preliminary results
– Computation of roofprints from BD TOPO

- The incorrect points are visible here again in grey, showing why giving a **lower weight to points that are not classified as building** is important.
- The second picture shows the height of the points, which shows that the points indeed **follow the 3D shape of the roofs.**

Computation of roofprints from BD TOPO

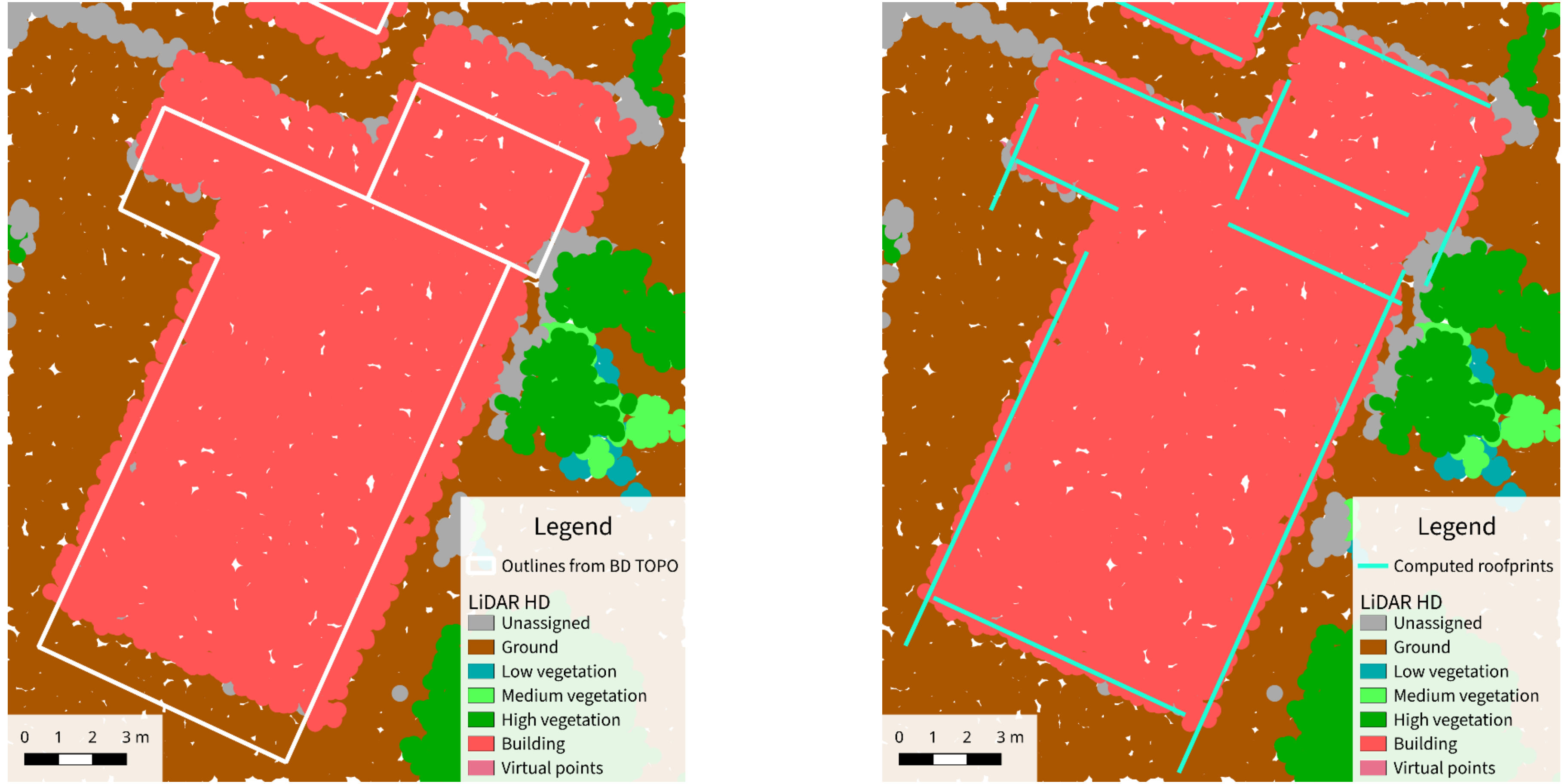


Figure 13: Comparison of BD TOPO outlines and computed roofprints.

Preliminary results
– Computation of roofprints from BD TOPO

- The computed roofprint is shown with the translated segments for easier understanding of the process leading to these edges.
- Result shows **better positioning** of the outlines which now correspond to the edges of the building.
- 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.

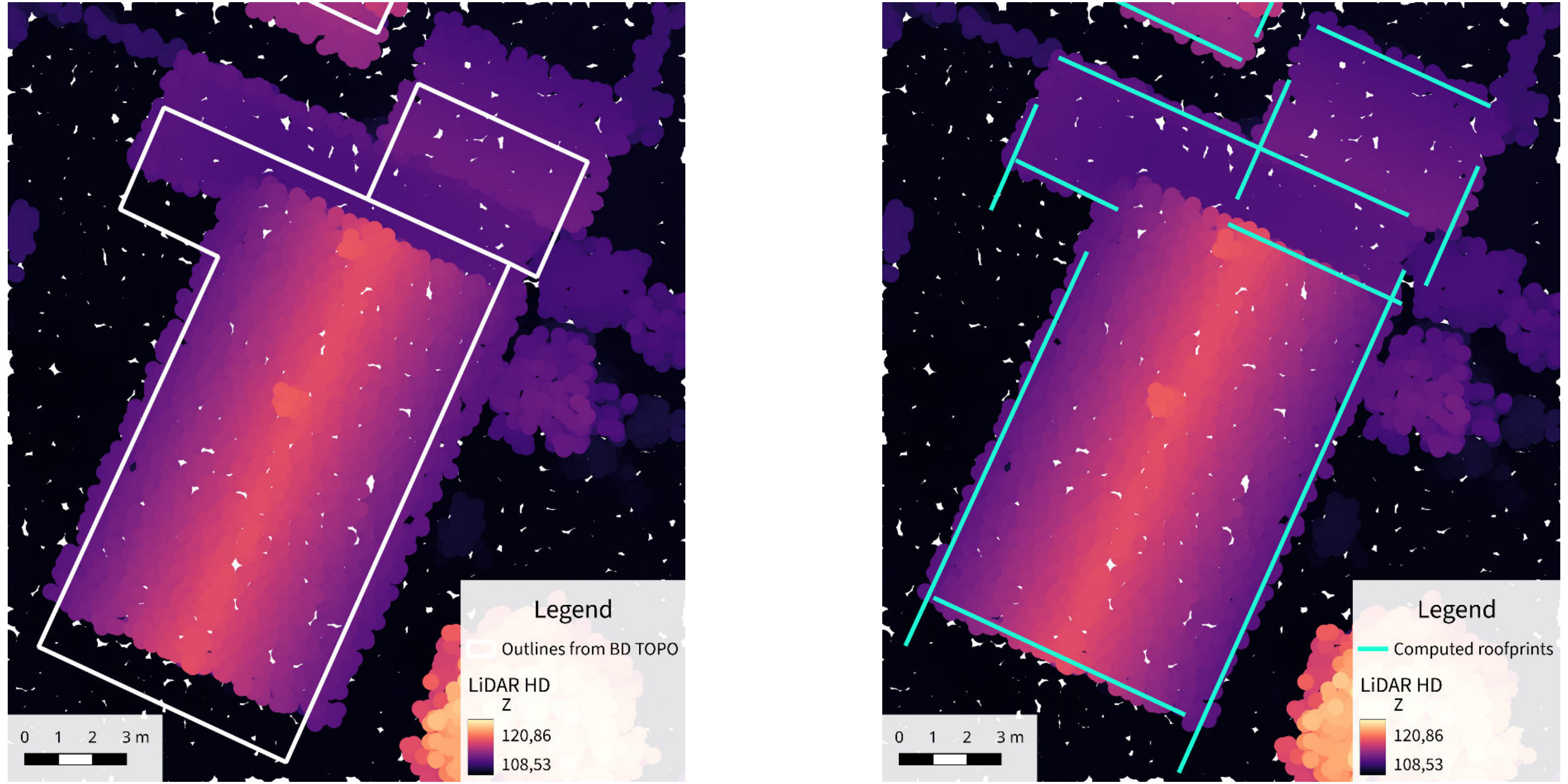


Figure 14: Comparison of BD TOPO outlines and computed roofprints.

- Other visualization of the same result, with points coloured by height instead of classification.
- Reason for the misclassification of the bottom left edge of the small top right building: the edge points are **too low** compared to the other edge points of the building, which makes them get a **low weight** and therefore get matched to the wrong edge.

Next objectives

Context 1
Work done until now 2
Preliminary results 14
Next objectives 22

Edge points

- Identify cases with a **dense set of points on a façade**. Possibility to compute normals to identify these cases.
- Consider **neighbours in other directions** than scan order (such as perpendicularly to it).
- Identify **lines in the point cloud topology** with Wu Teng's method:
  - Could be used to estimate the **normals locally** in cases where segments are found to estimate if points are on façades or on roofs.
  - Could be used to identify the breaks of roof planes (out of scope for now).

Next objectives
– Edge points

Edge matching - Improvements

For each edge

- **Group edges** with angles **close to 180°** (instead of currently merging them).
- Match each edge by also moving and computing the **score of its two (or more) neighbours**.
- Use the **repartition of points** on the edge in their score.
- Try (maybe) to use the **distance to the current edge** as an indication as well.

Next objectives
– Edge matching - Improvements

- Matching with the two neighbours could improve results for **tiny edges** and prevent matching edges from **other buildings.**
- Two ideas to use the repartition of points on the edge:
  - Divide the score by the **length** of the edge.
  - **Reward uniform distribution** of points along the edge with a histogram-like metric.
- Not sure about using the distance to the current edge as an indication, because the assumption that a closer edge is better is often wrong due to the shift in BD TOPO. If we still decide to try it, a simple solution would be to add a penalty proportional to the distance, but this would need to be tested.

Edge matching - Improvements

For the whole building

- Matching **longer edges first**.
- Run the matching **algorithm multiple times in a row**, or until it converges.
- Use **combinatorial optimization** to pick the best set of edges with multiple solutions for difficult edges.

Next objectives
– Edge matching - Improvements

- Matching longer edges first may help matching the **smaller edges** properly, if we shift the starting point of the neighbouring edges.
- Running the matching algorithm multiple times in a row may help for **small edges**, where a translation along the direction of the edge is problematic. It could also allow for **more iterations with smaller frames**.
- Combinatorial optimization would be a more complex solution, but it would allow to **globally optimize** the matching of all edges together, to avoid local mistakes.

Edge matching - Other ideas

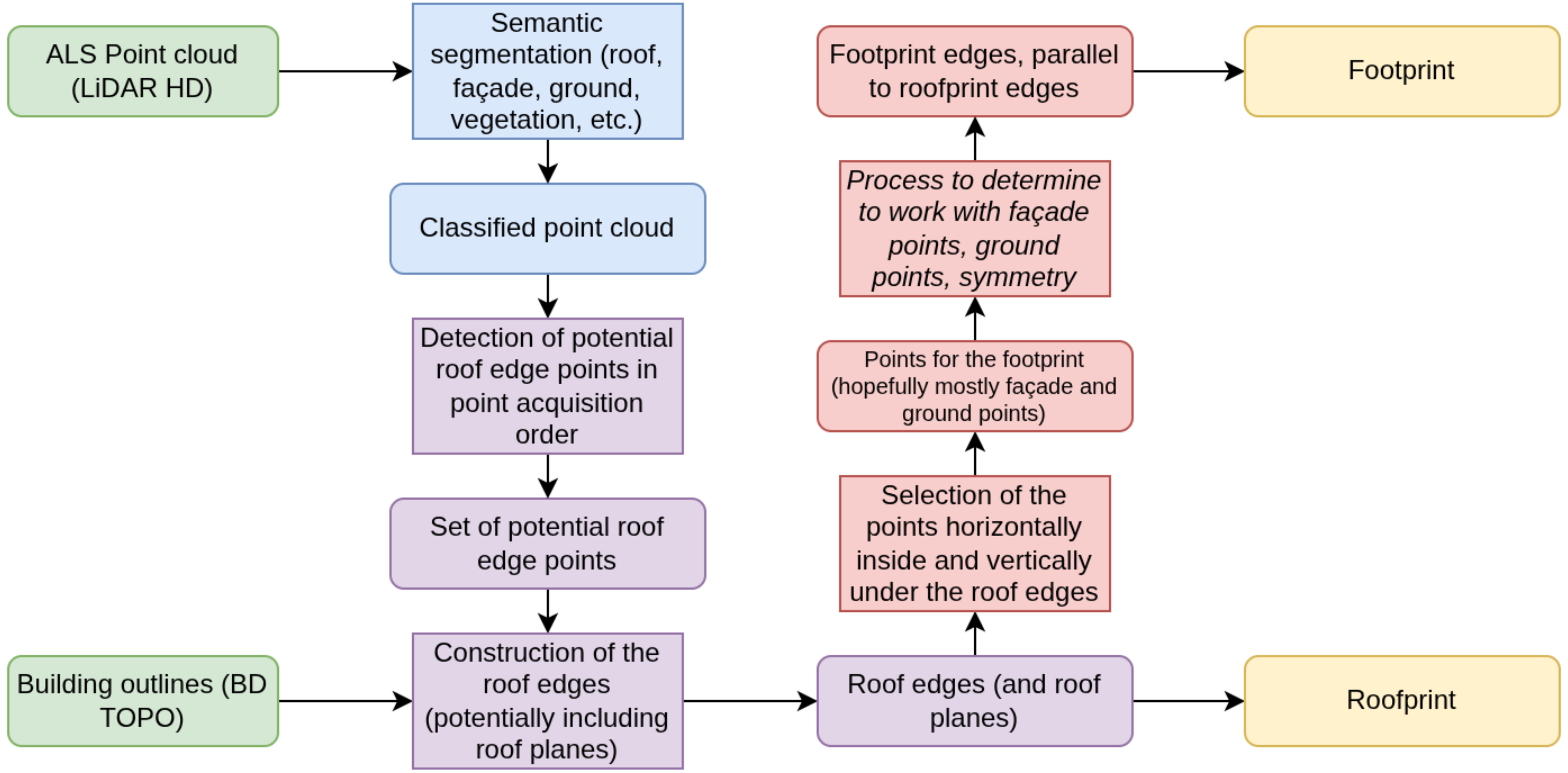
- Use **regularization** before or after.
- Use the **classification** of the points in combination with the outlines to crop the point clouds better.
- Try to make better edges:
  - With a **RANSAC-like** approach (potentially using the BD TOPO to guide the process) in **2D or 3D**.
  - By other means in 3D?
  - By **recomputing the edge** after identifying the inlier points with the current process.
  - Try to estimate **where the edge starts and stops** from the point cloud.

Next objectives

— Edge matching - Other ideas

- **Regularization before** could help removing small artefacts in the edges due to touching buildings and ensure right angles.
- **Regularization after** could be necessary if computing edges not parallel to the initial edges.
- Using the **classification** of the points to crop could prevent matching edges from other buildings.

Rest of the pipeline



Next objectives
— Rest of the pipeline

Thank you for your attention!

Any questions?

alexandre.bry@ign.fr

The end

—