From Points to Prints

Monthly Presentation (3)

Alexandre Bry

April 20, 2026

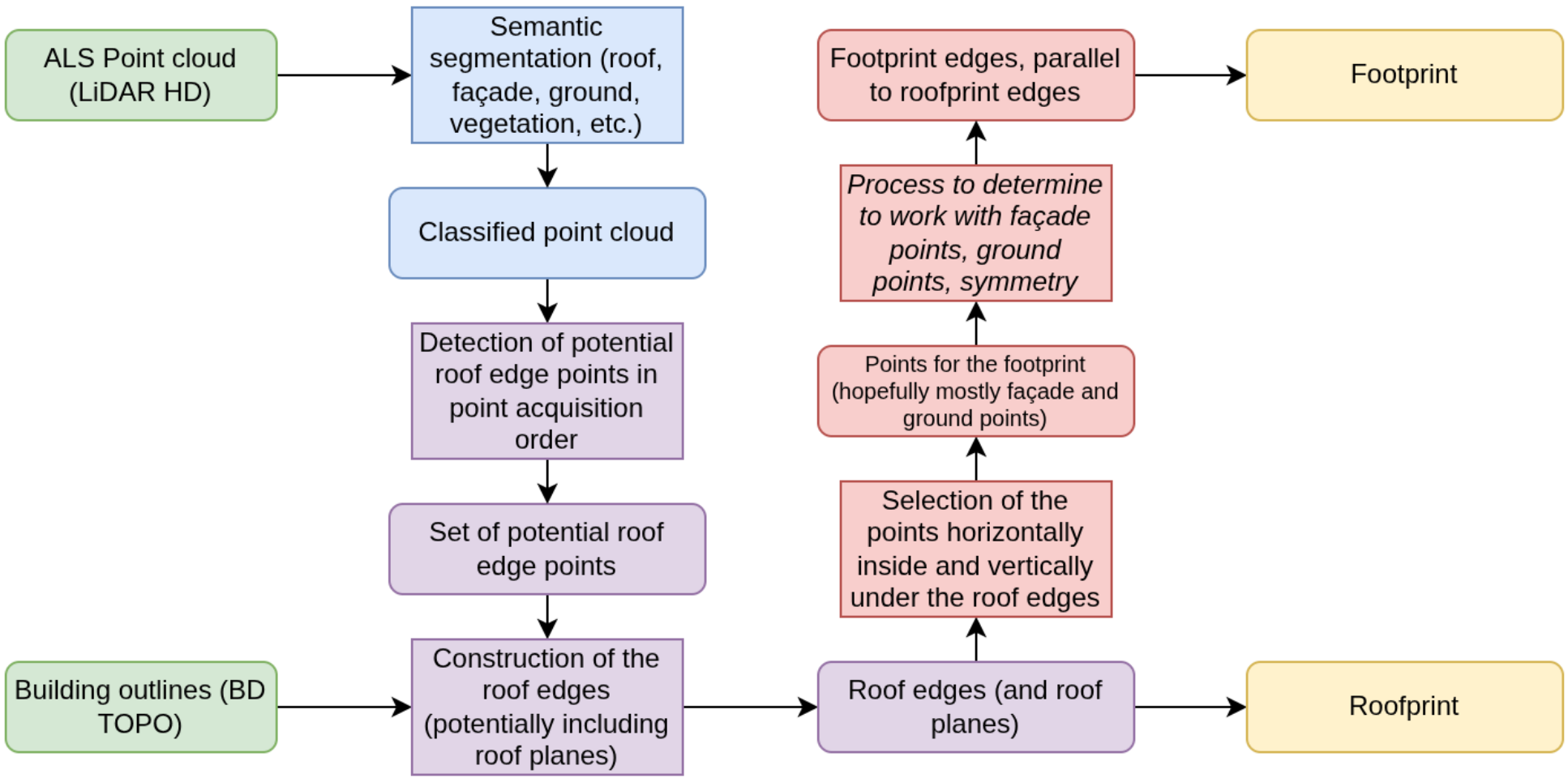
TU Delft

IGN

Outline

Context

Planned pipeline



General structure

General structure of the method:

- **Identifications of points** to deform the polygons on.
- **Creation of a set of candidate configurations**:
  - By translating each edge perpendicularly to itself.
  - The other edges are translated if necessary to keep the same topology.
- **Definition of a criterion to evaluate the quality of each configuration**, based on:
  - The position of the points.
  - Weights computed for the points.
  - The initial configuration of the edges.

Point cloud topology

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.

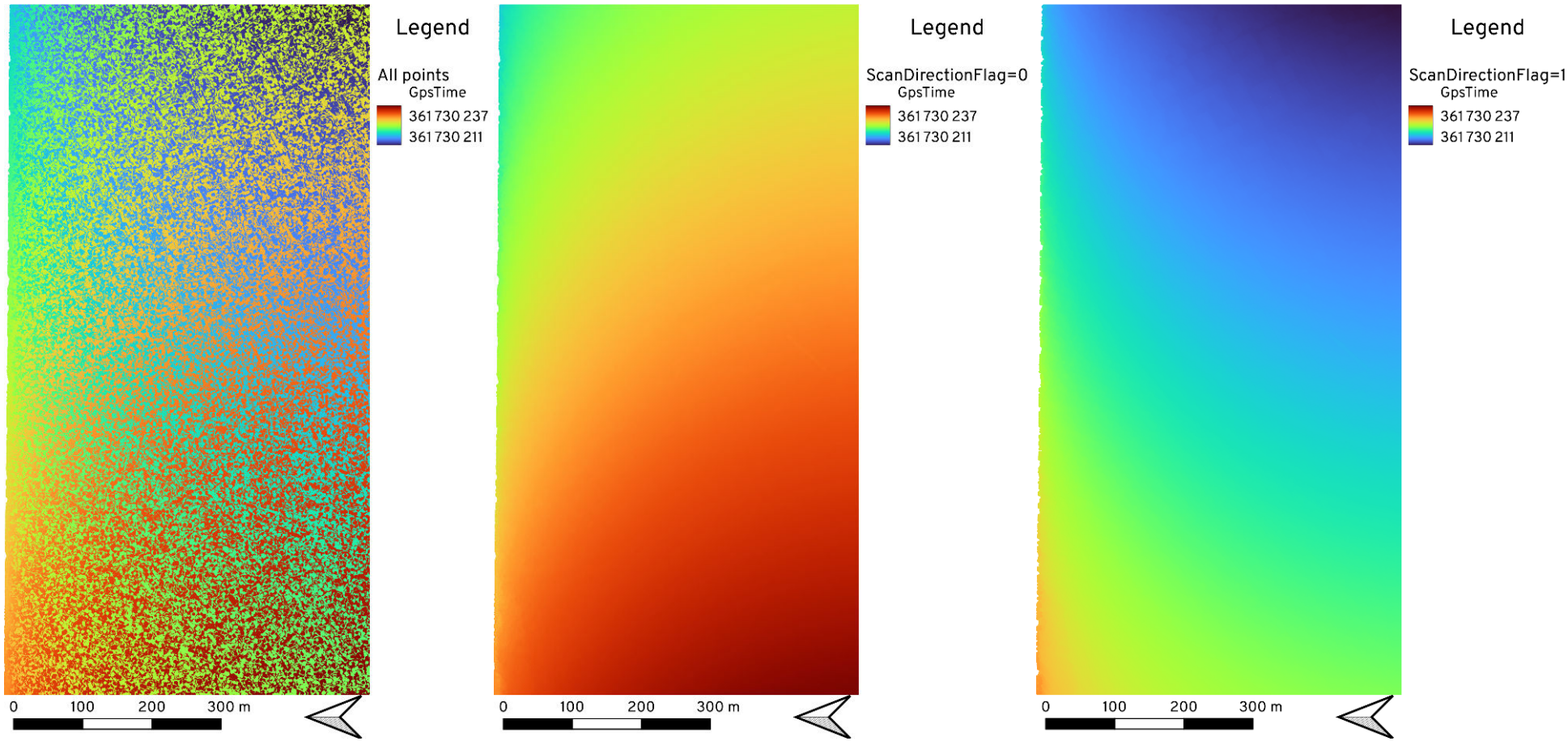
Scan line

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.

Point cloud topology



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

Figure 1: Visualization of the topology of the point cloud, with points coloured by their GPS Time.

Flight strips trajectories

- Trajectories are computed using **multi-echo pulses** (code written by **Wu Teng**)
- Necessity to **reconcatenate the different parts of each flight strip** for better precision (not done yet)

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

Edge points

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 (more details in the slides of the previous presentation).

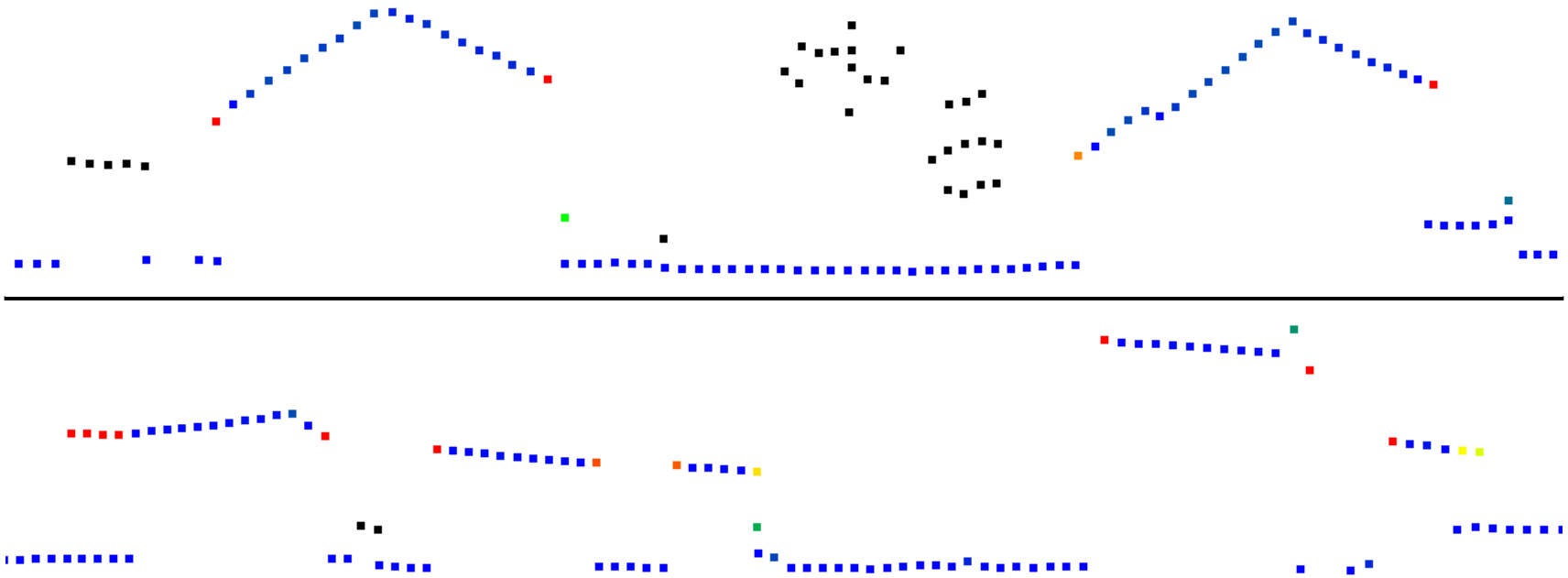


Figure 2: Two examples of the height differences obtained on a scan line. Black points are vegetation points, and for the other points the color represents the value of Δh, with blue-green-yellow-red from low to high values.

Candidate roofprint configurations

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.**

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, this **does not ensure that we preserve the same topology**, as can be seen in the examples in the next slides.

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, this **does not ensure that we preserve the same topology**, as can be seen in the examples in the next slides.

In practice, we only ensure that:

- At the level of one polygon, for each edge:
  - The edge is never flipped.
  - Therefore its two neighbours do not intersect.
- At the level of multiple buildings:
  - Two initially overlapping edges will remain collinear (not necessarily overlapping).

Constraints over the movements

One point becomes two

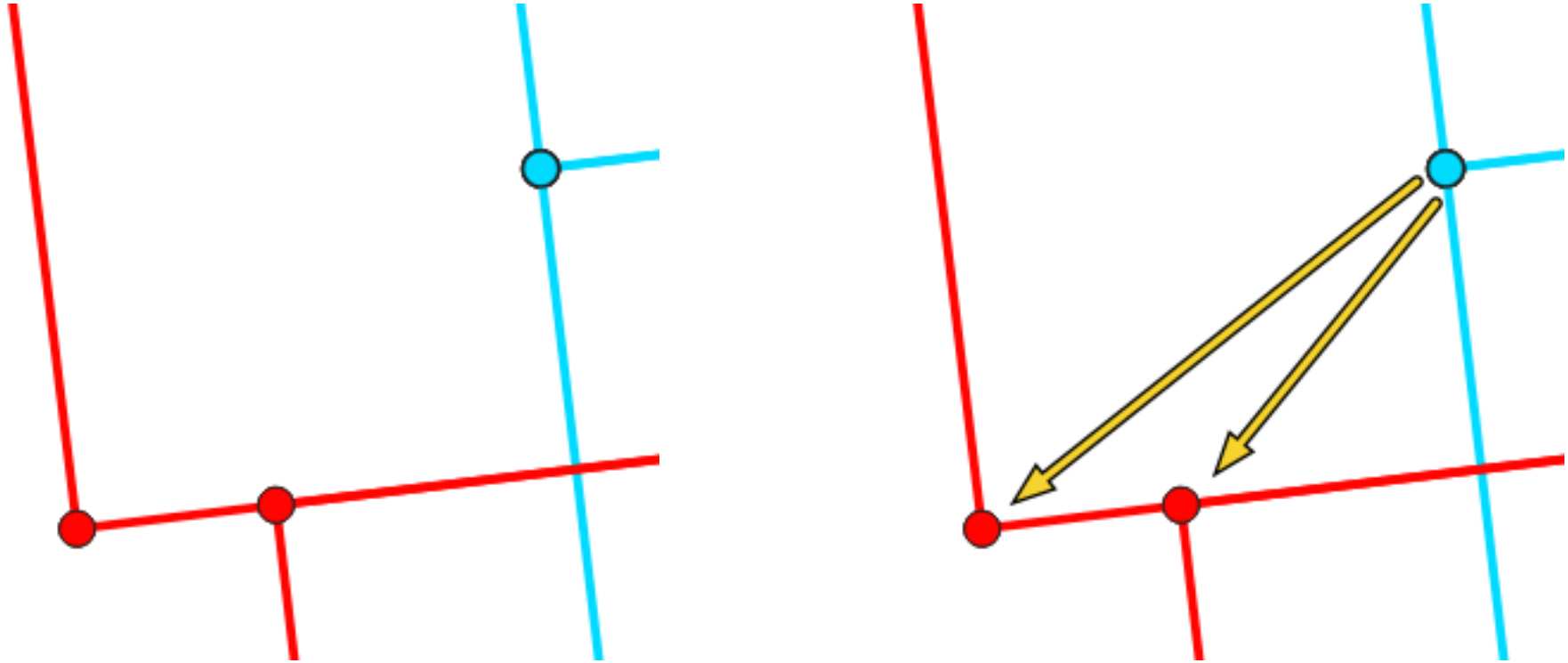


Figure 3: Two vertices at the exact same position become two distinct vertices at the boundary between two buildings.

Constraints over the movements

One point becomes two

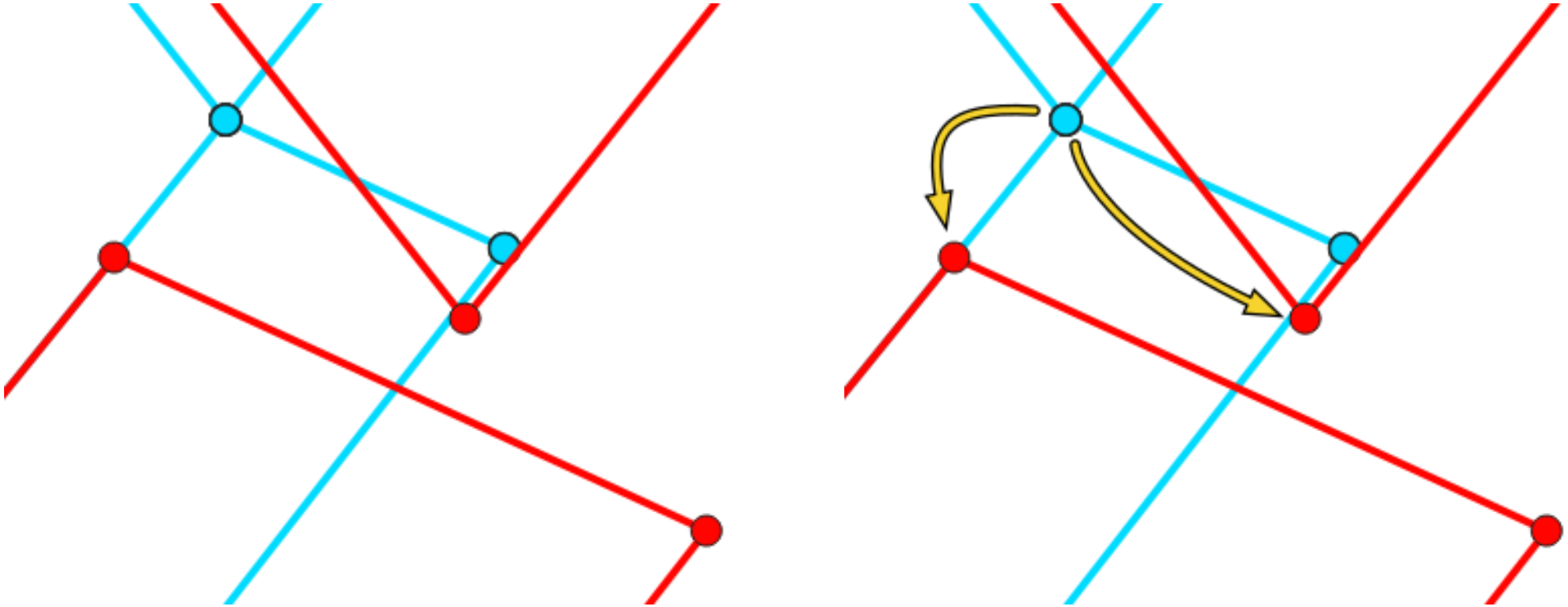
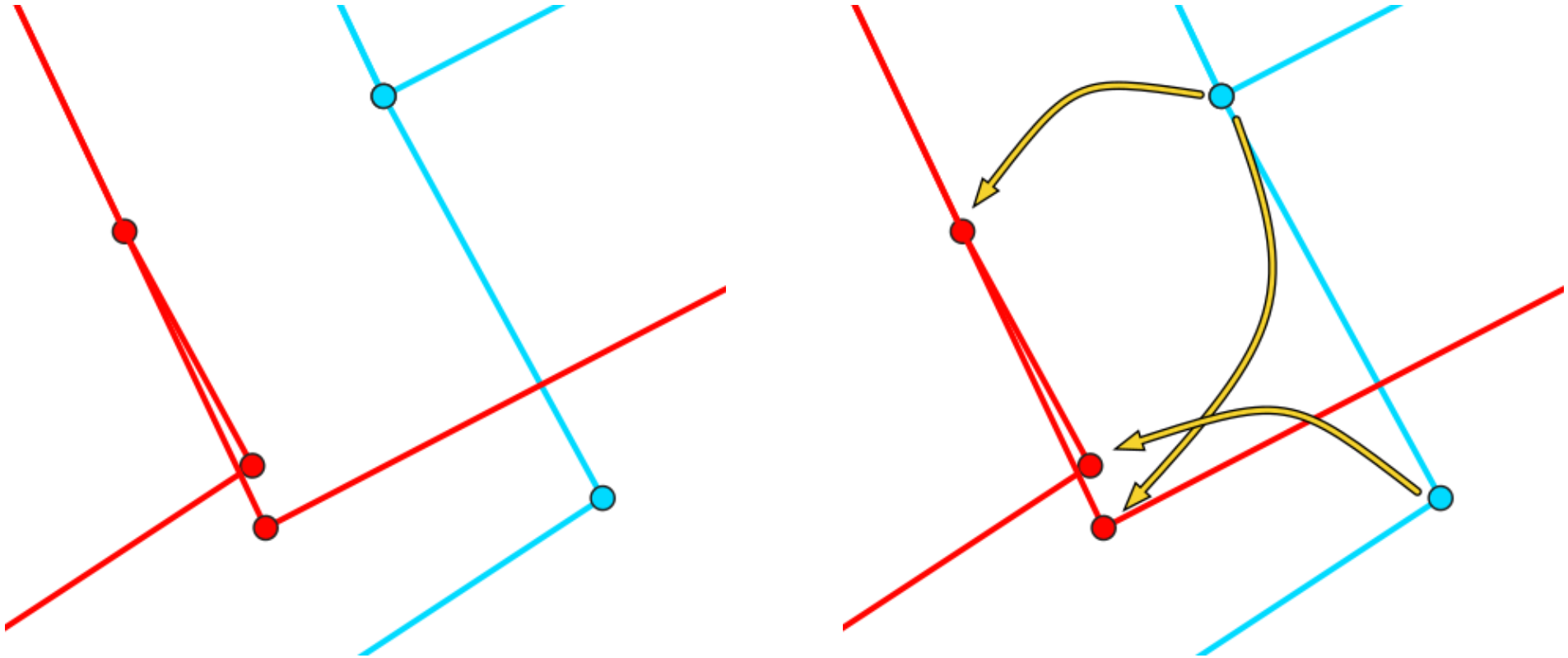


Figure 4: Two vertices at the exact same position become two distinct vertices at the boundary between two buildings.

Constraints over the movements

One point becomes two and Two buildings intersect



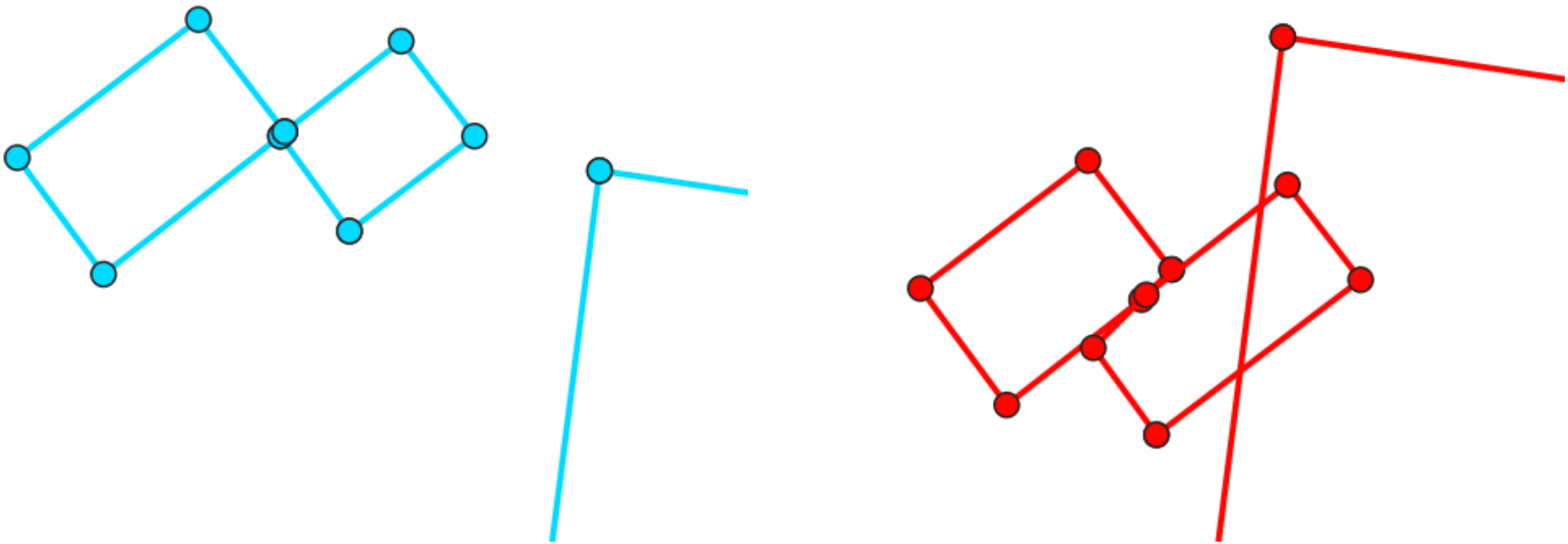
(a) Before

(b) After

Figure 5: Two buildings that were initially separate end up intersecting after the deformation.

Constraints over the movements

Two buildings intersect

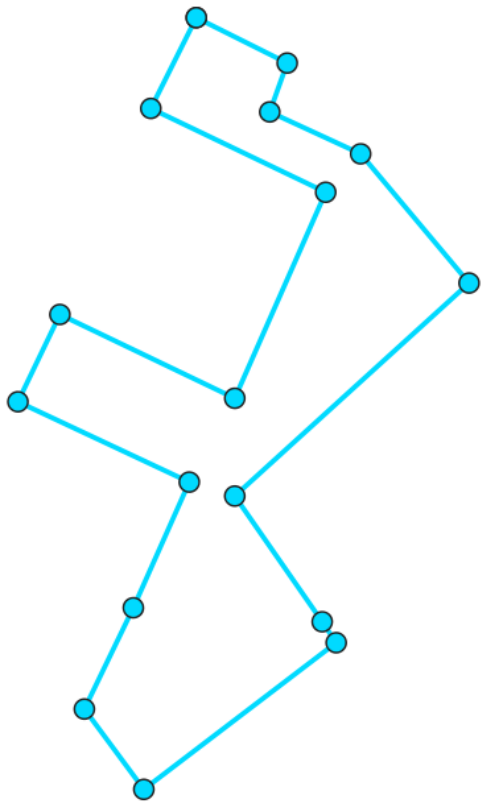


(a) Before (b) After

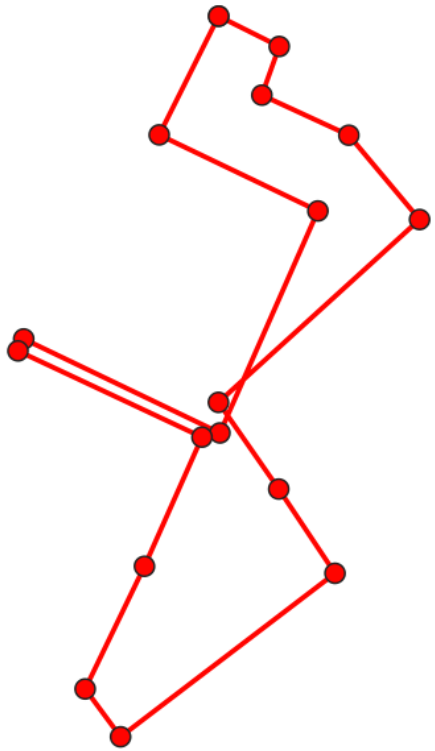
Figure 6: Two buildings that were initially separate end up intersecting after the deformation.

Constraints over the movements

One building self-intersects



(a) Before



(b) After

Figure 7: Two opposite sides of the same building that were initially close end up intersecting after the deformation.

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.

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.**

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.

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 $l_n = l_n^+$.

Our simple approach

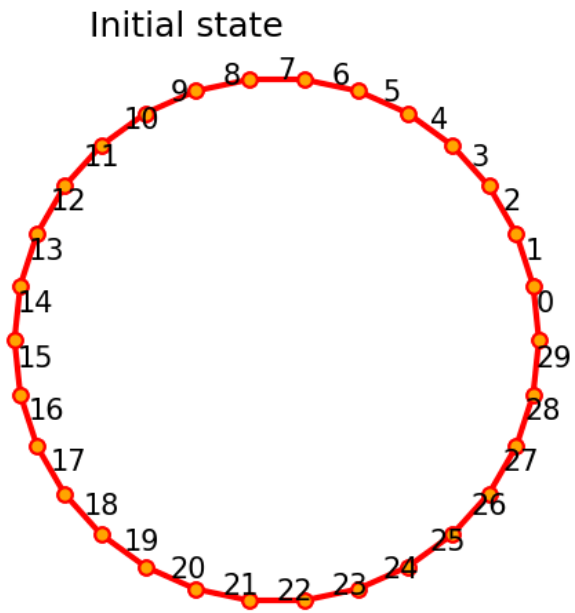
Simple algorithm to find a correct configuration given a line l_0 to move by an extent δ:

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



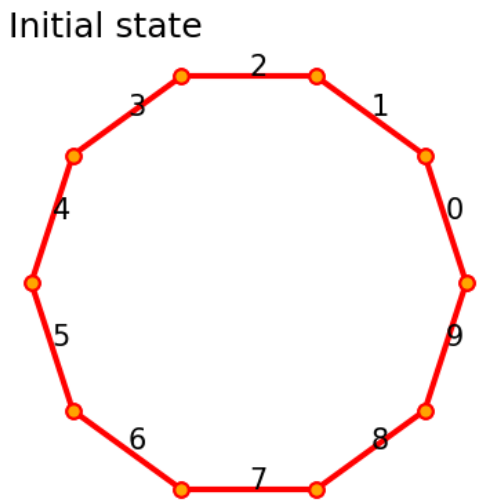


Figure 8: Simple illustrations of the polygon deformation algorithm (click to view the animated versions).

Illustrations

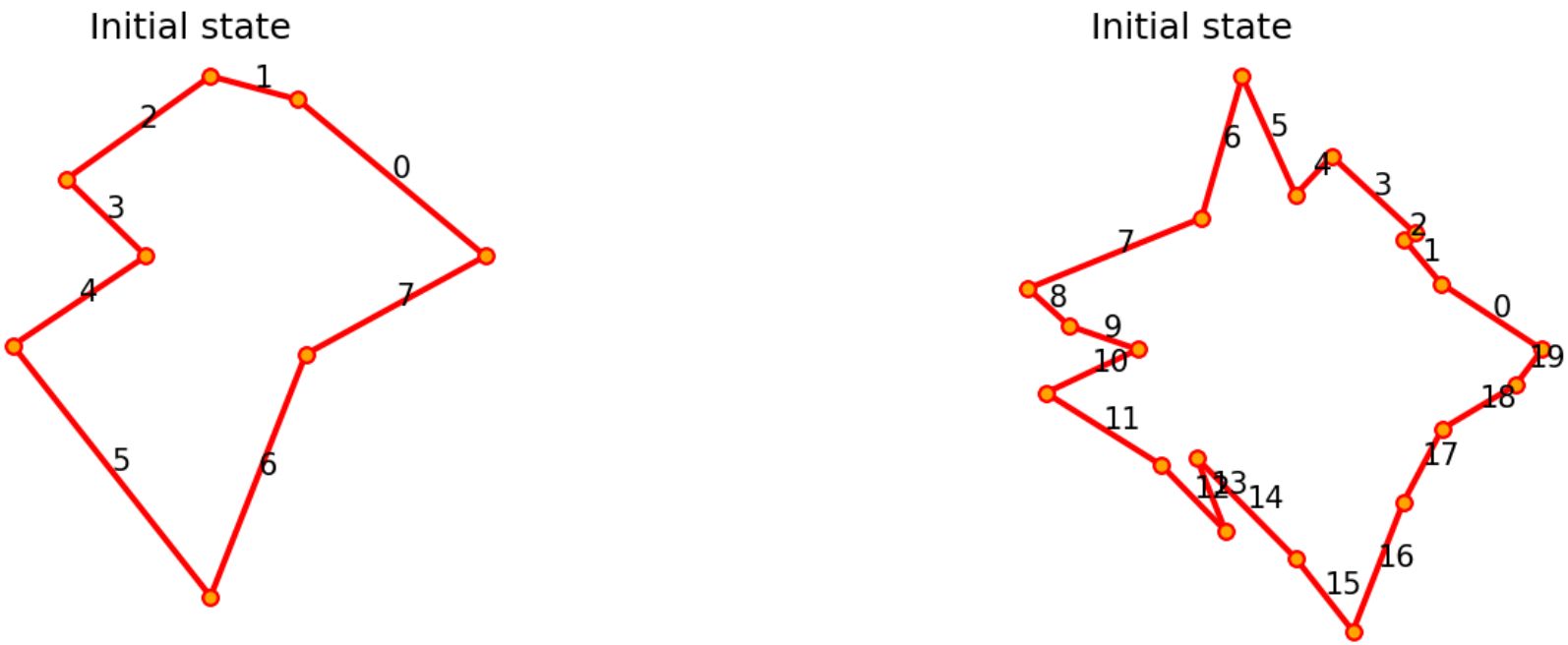


Figure 9: More complex illustrations of the polygon deformation algorithm (click to view the animated versions).

Illustrations

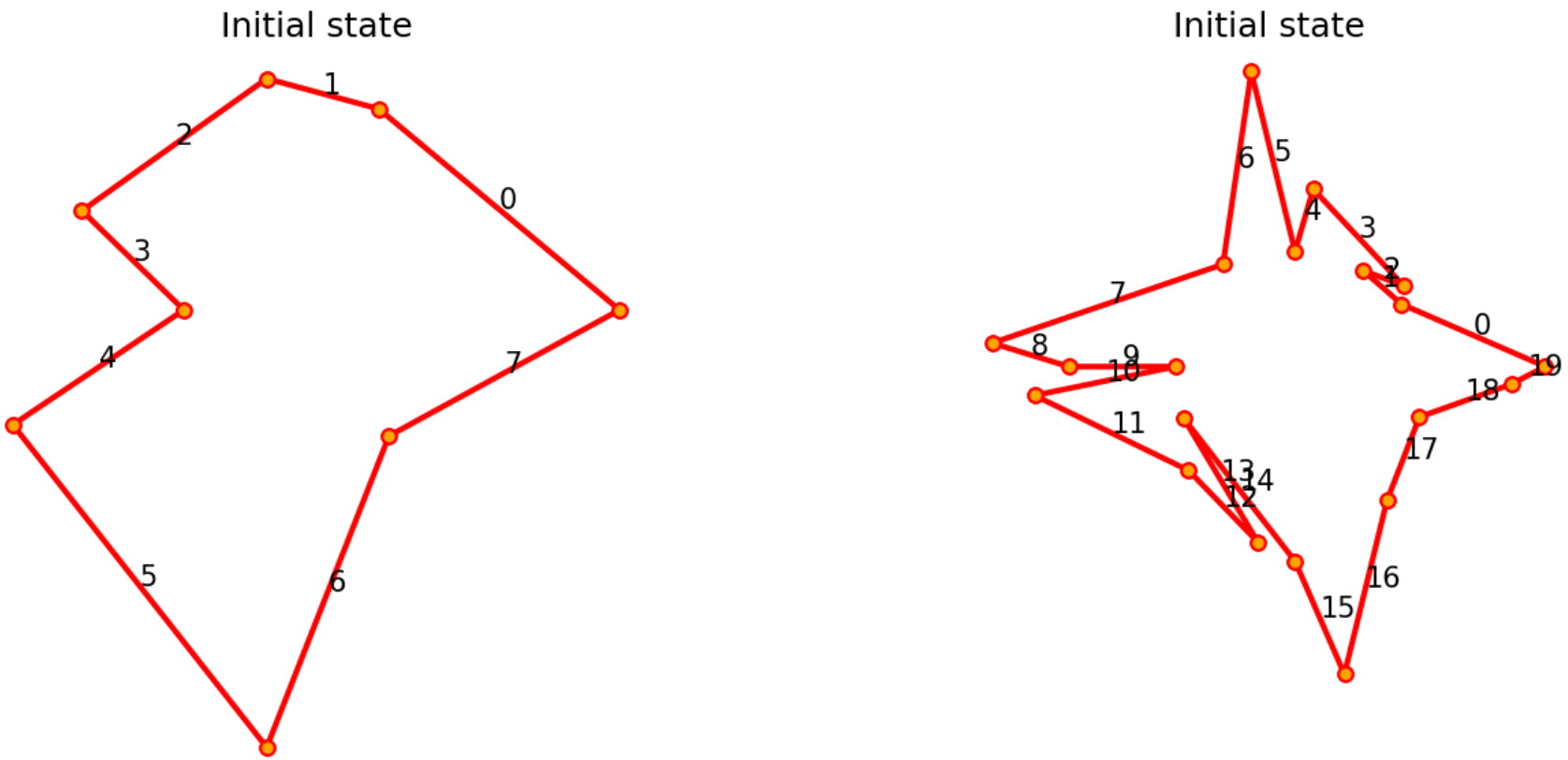


Figure 10: Illustrations of self-intersections with the polygon deformation algorithm (click to view the animated versions).

Criterion

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.

Definition of the proximity score

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 line supporting 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.

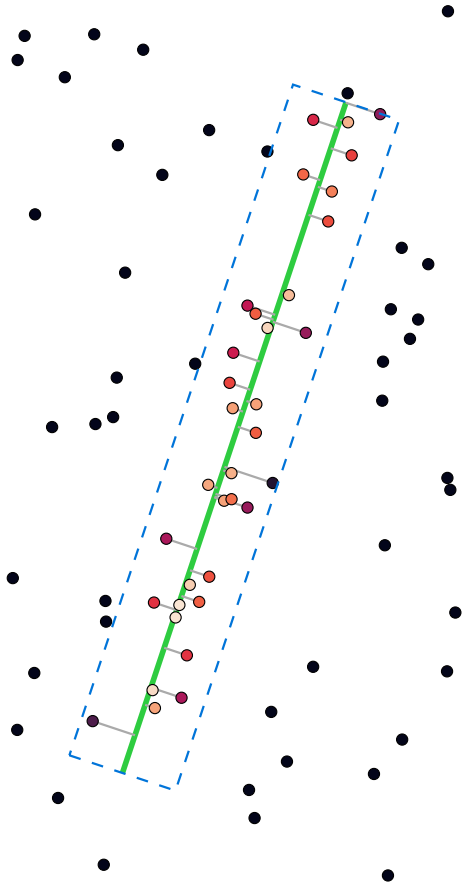


Figure 11: Illustration of the proximity score for an isolated edge.

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.

Illustrations

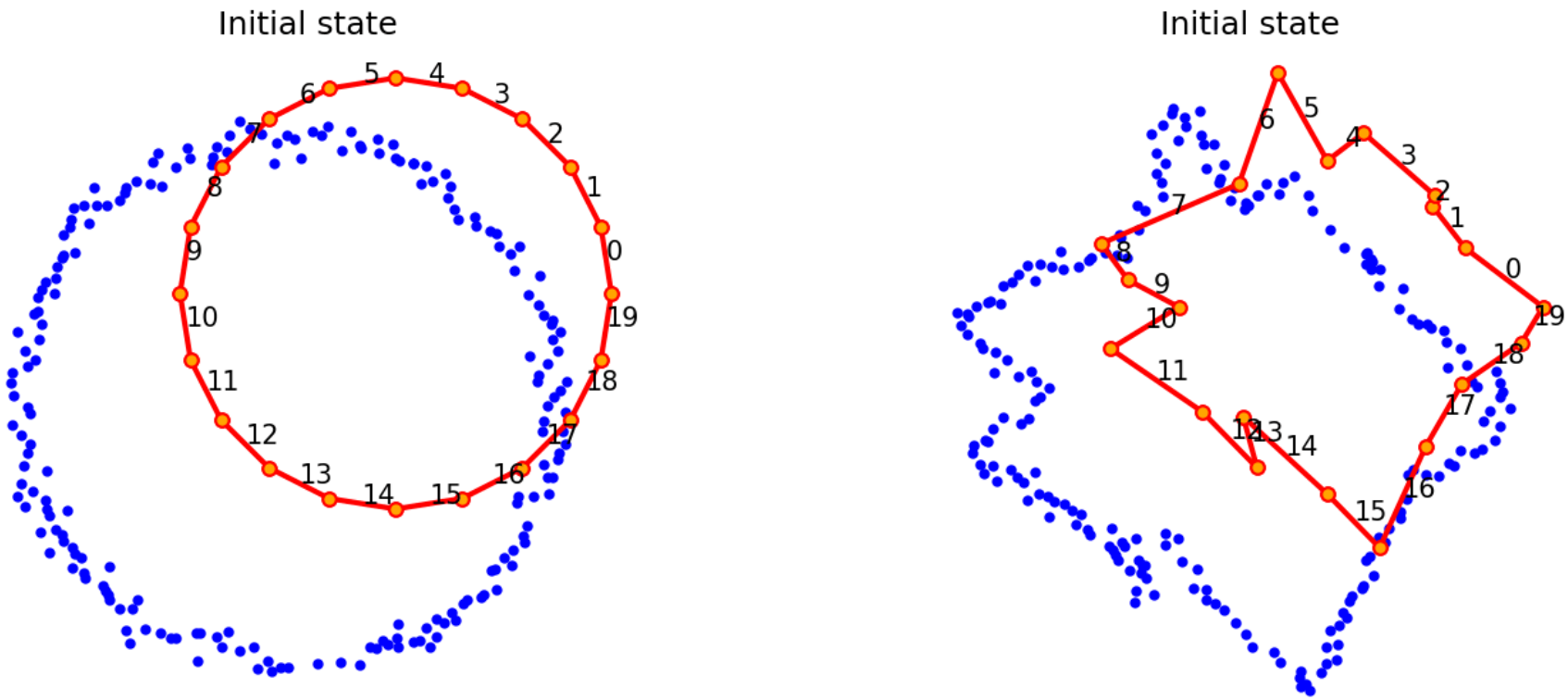
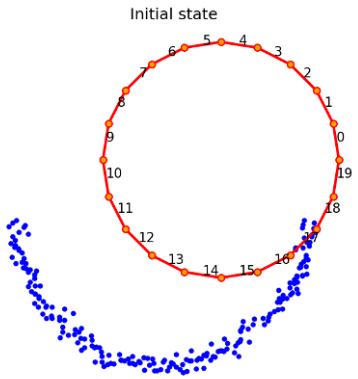
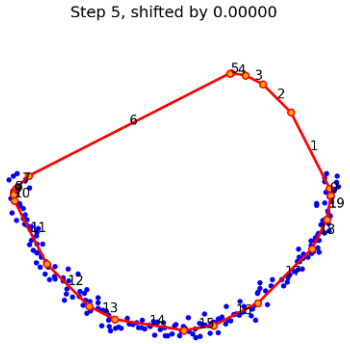


Figure 12: Simple illustrations of the full polygon deformation algorithm (click to view the animated versions).

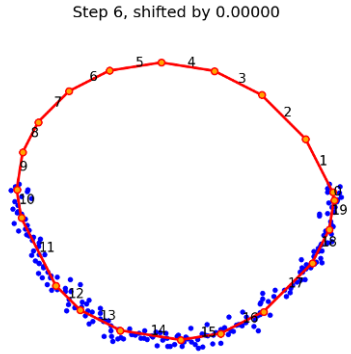
Illustrations



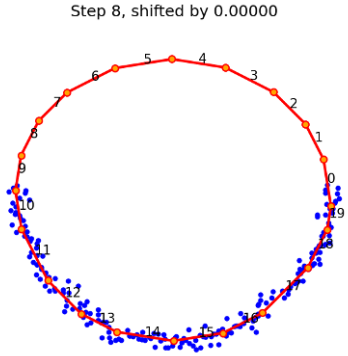
(a) Initial state



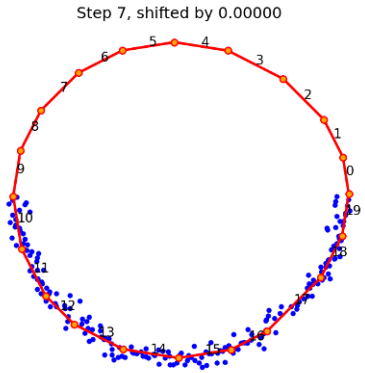
(b) $\alpha = 0.00$ (no regularization)



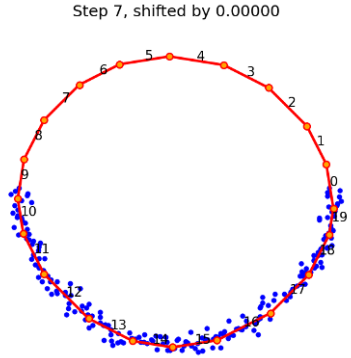
(c) $\alpha = 0.05$



(d) $\alpha = 0.20$



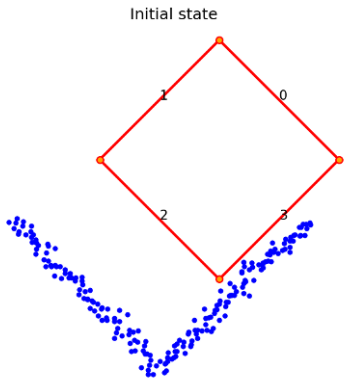
(e) $\alpha = 0.50$



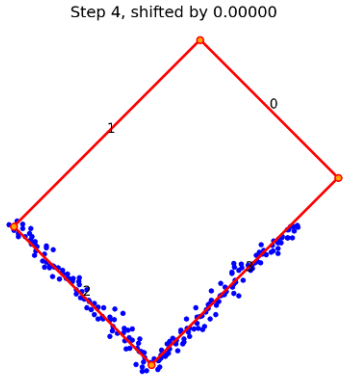
(f) $\alpha = 1.00$

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

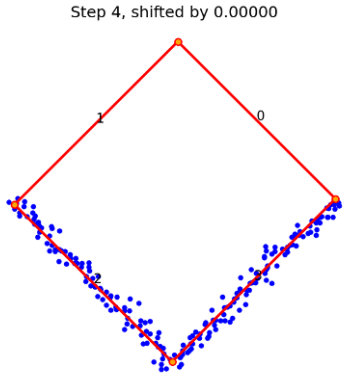
Illustrations



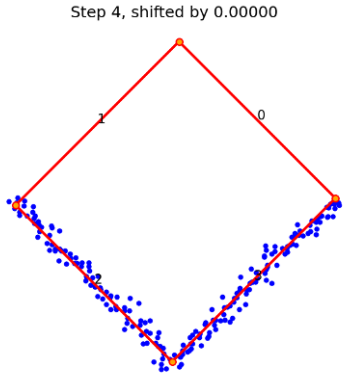
(a) Initial state



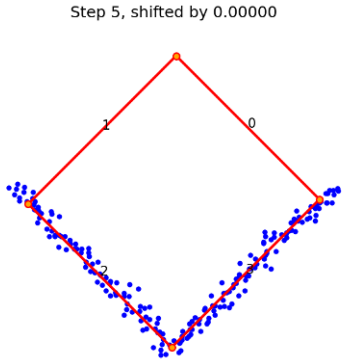
(b) $\alpha = 0.00$ (no regularization)



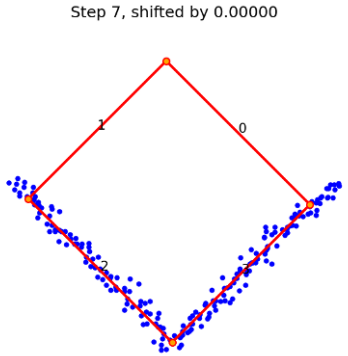
(c) $\alpha = 0.05$



(d) $\alpha = 0.20$



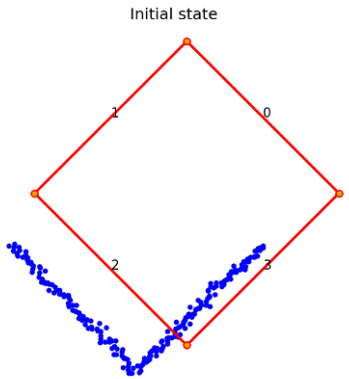
(e) $\alpha = 0.50$



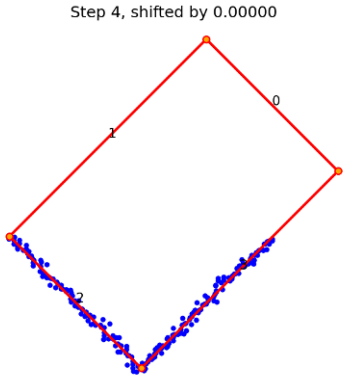
(f) $\alpha = 1.00$

Figure 14: Matching a square to points along two sides of a larger square with different values of the regularization parameter α.

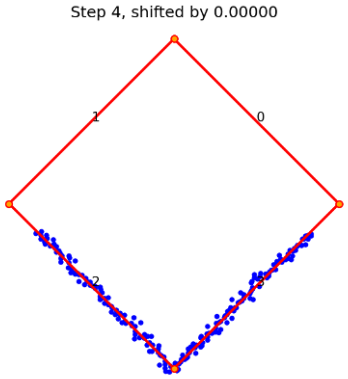
Illustrations



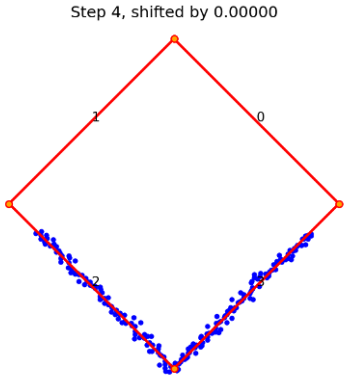
(a) Initial state



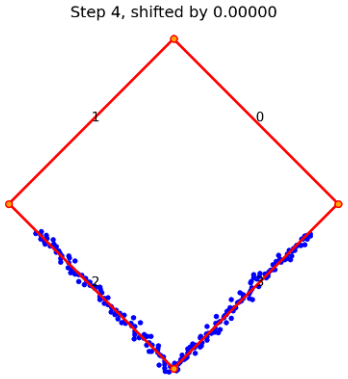
(b) $\alpha = 0.00$ (no regularization)



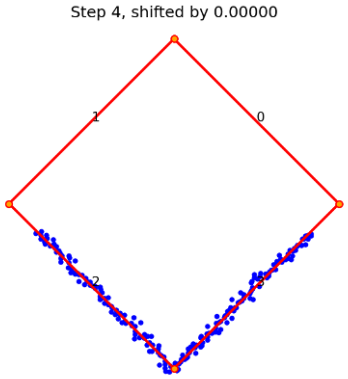
(c) $\alpha = 0.05$



(d) $\alpha = 0.20$



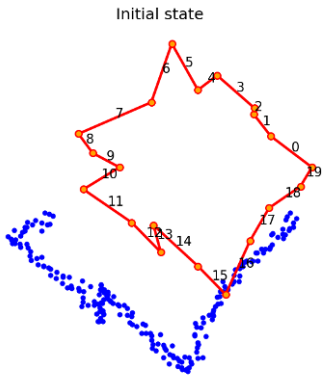
(e) $\alpha = 0.50$



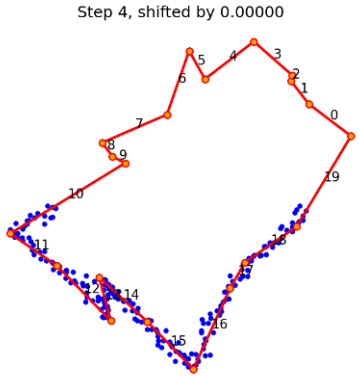
(f) $\alpha = 1.00$

Figure 15: Matching a square to points along two sides of a smaller square with different values of the regularization parameter α.

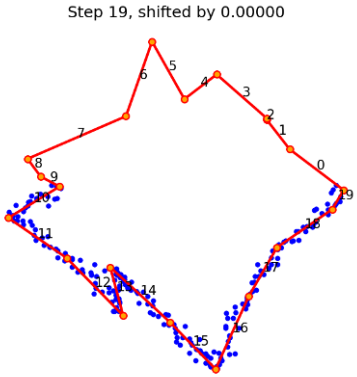
Illustrations



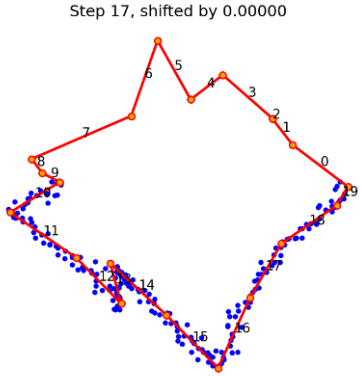
(a) Initial state



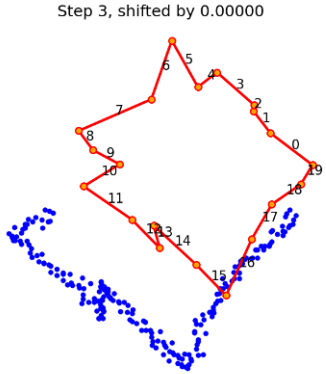
(b) $\alpha = 0.00$ (no regularization)



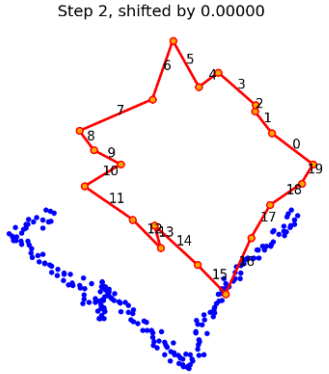
(c) $\alpha = 0.05$



(d) $\alpha = 0.20$



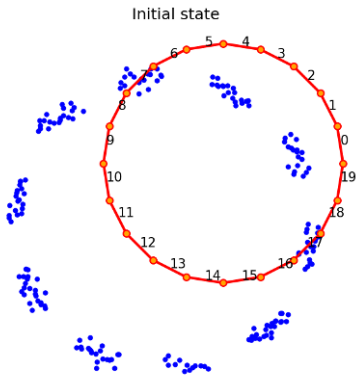
(e) $\alpha = 0.50$



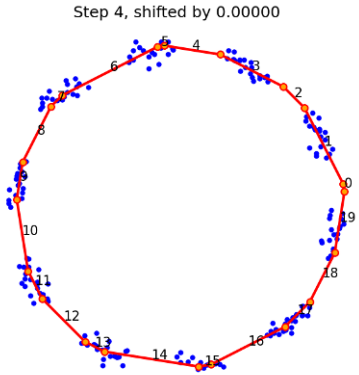
(f) $\alpha = 1.00$

Figure 16: Matching a polygon to half of its shape with points with different values of the regularization parameter α.

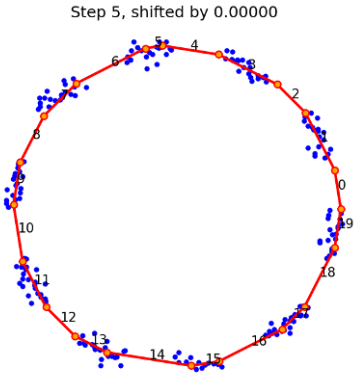
Illustrations



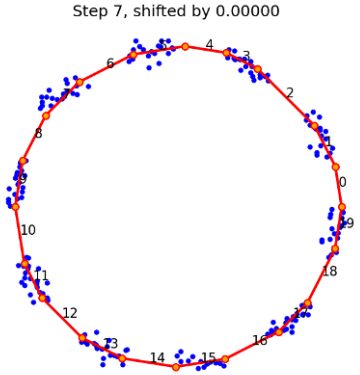
(a) Initial state



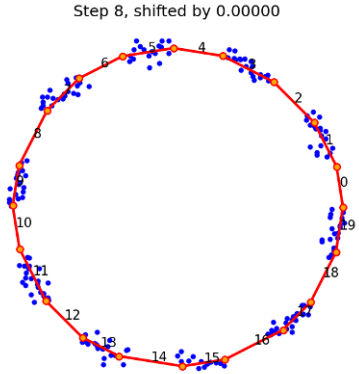
(b) $\alpha = 0.00$ (no regularization)



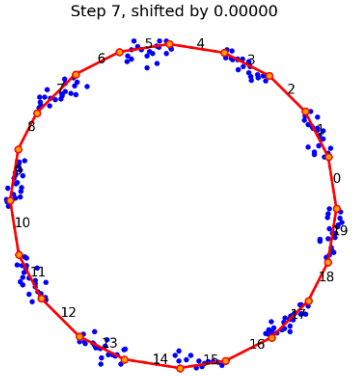
(c) $\alpha = 0.05$



(d) $\alpha = 0.20$



(e) $\alpha = 0.50$



(f) $\alpha = 1.00$

Figure 17: Matching a circle to points along half of its boundary with different values of the regularization parameter α.

Last results

Computation of roofprints from BD TOPO

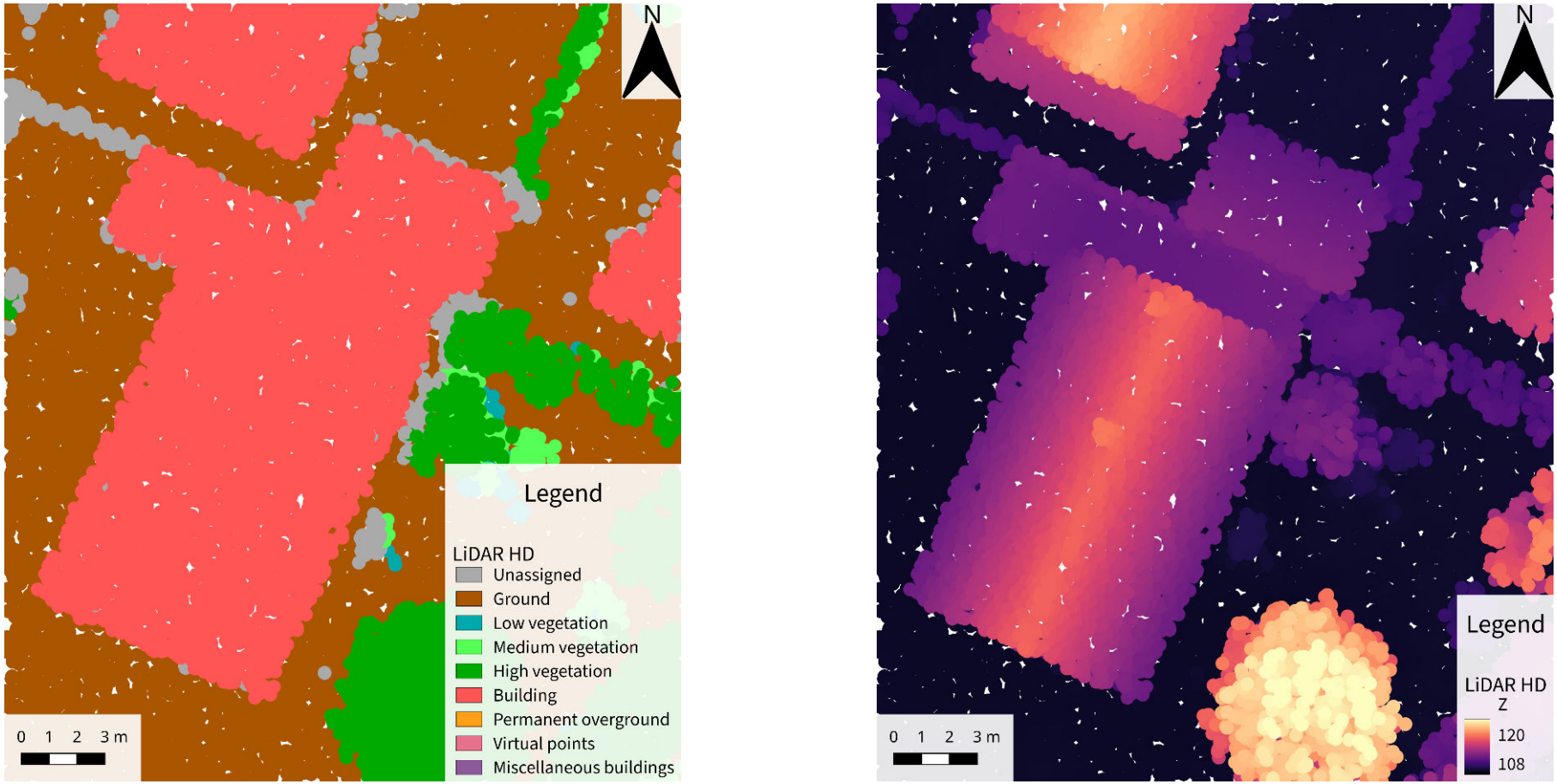
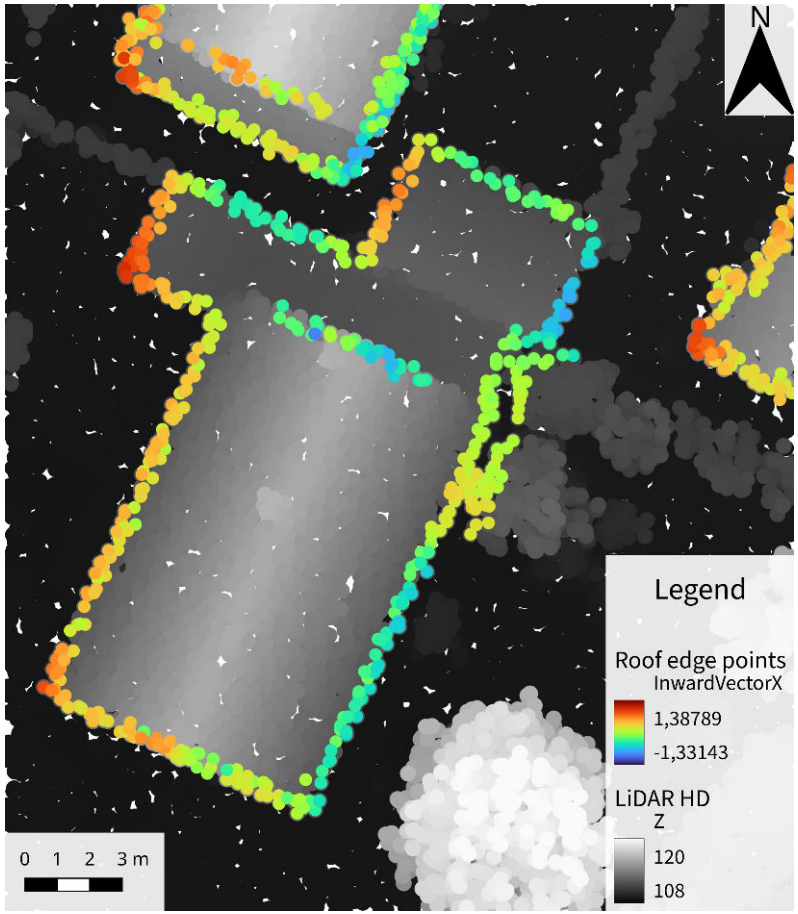


Figure 18: The LiDAR HD data.

Computation of roofprints from BD TOPO



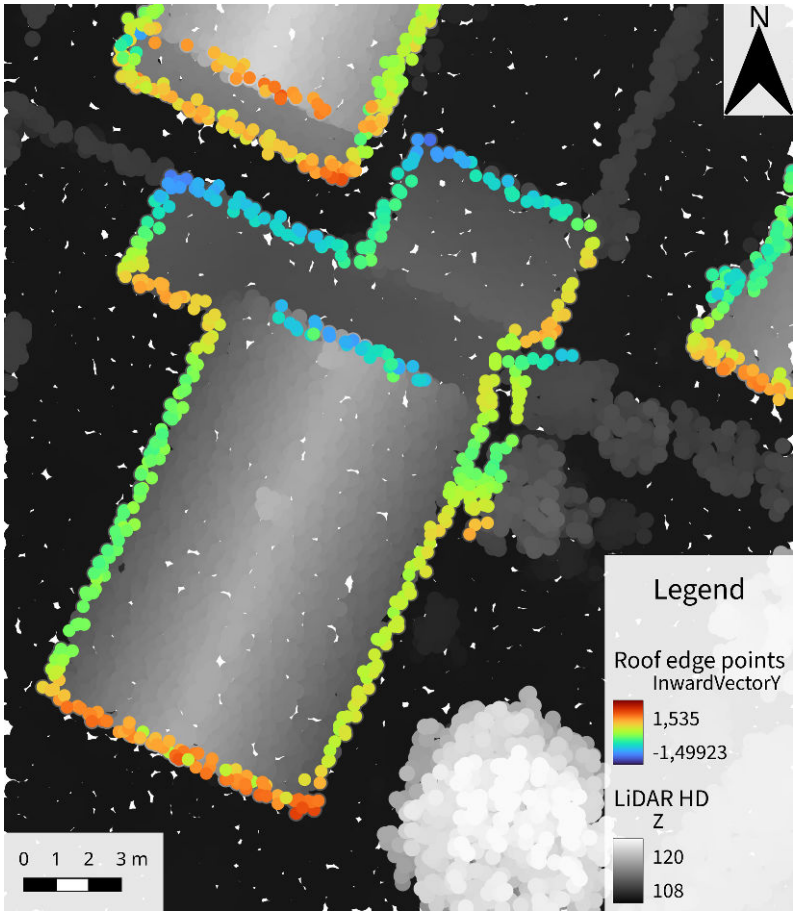


Figure 19: The detected roof edge points coloured by their inward vector.

Computation of roofprints from BD TOPO

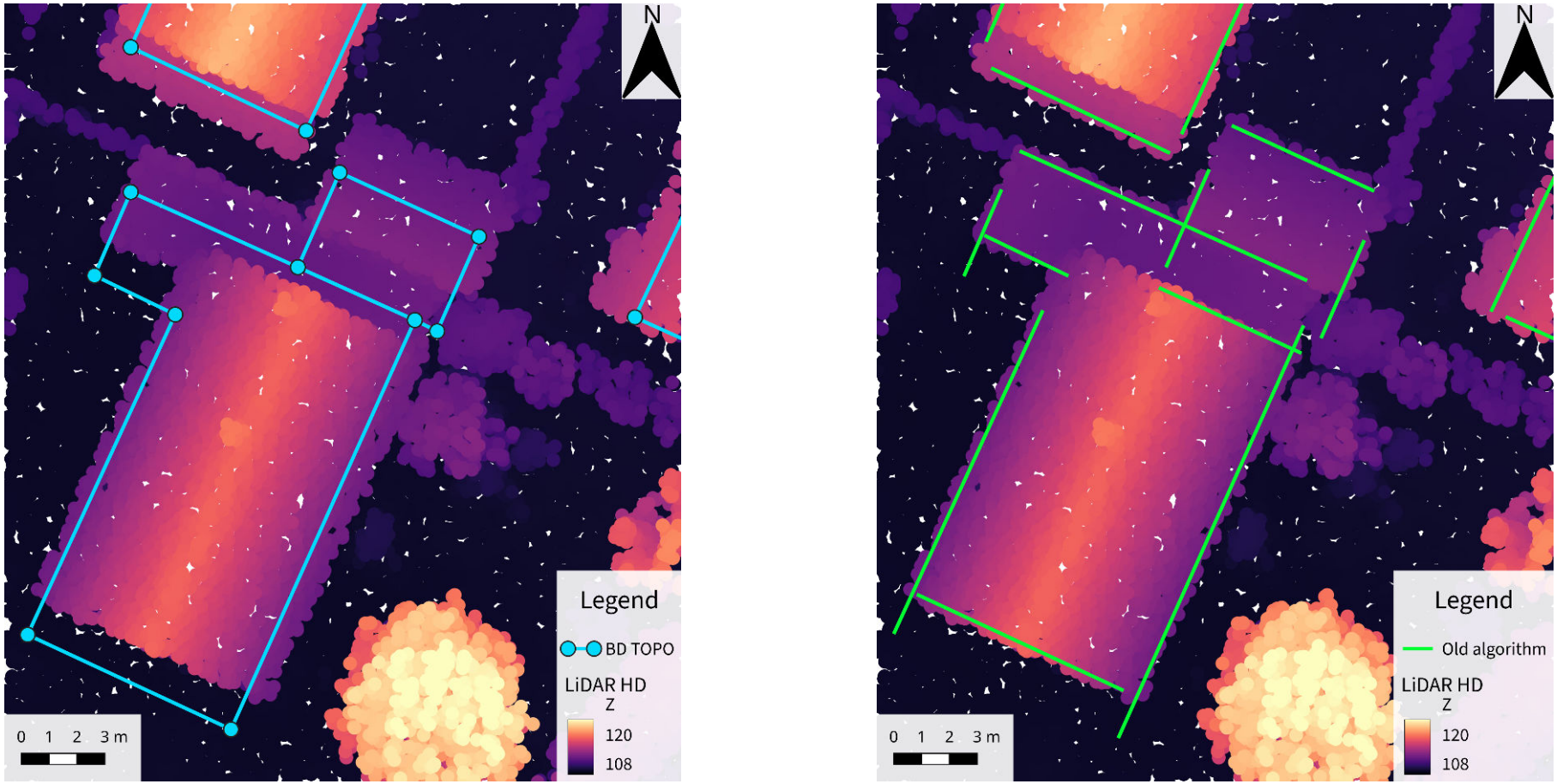


Figure 20: Comparison of BD TOPO outlines and roofprints computed with the old algorithm.

Computation of roofprints from BD TOPO

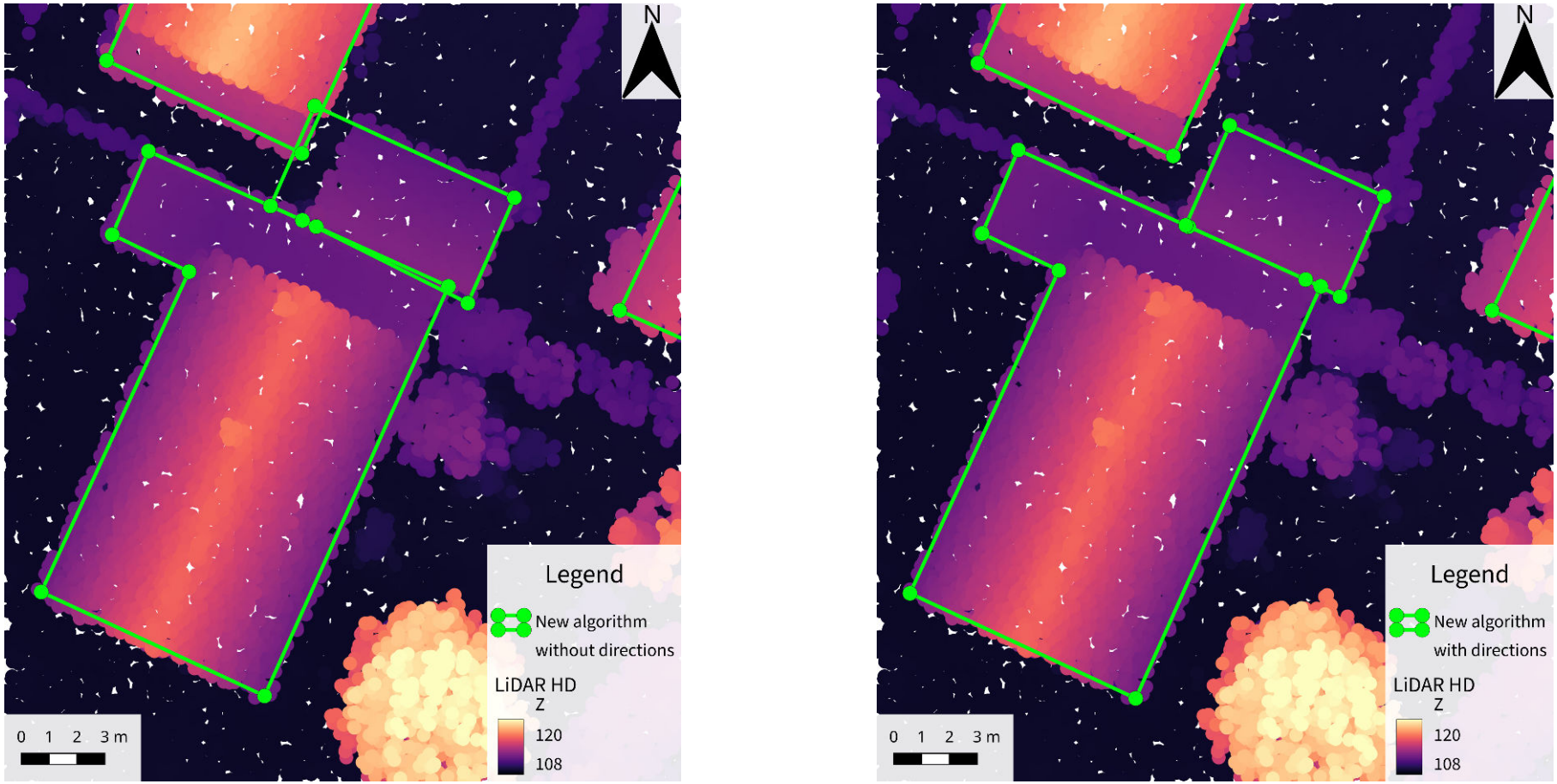


Figure 21: Comparison of the roofprints computed with the new algorithm without and with inward vectors.

Further improvements

Point cloud semantic segmentation

Planned

- Look into using the **height** of points and normals to tell apart points on façades and roof edges.
- Look into training a deep learning model to **classify points** into more specific classes (such as roof and façade)

Not planned

- Consider **neighbours in other directions** than scan order (such as perpendicularly).
- 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).

Roofprint generation

Planned

- Experiment with the order of edge processing. For now it is **longer edges first**, but we could try random order.
- Start with a single 2D translation for all edges, to see if it can improve the results.
- (Maybe) run the **whole pipeline multiple times** with different conditions (different orders, different initial shifts, etc.) and pick the best result.

Not planned

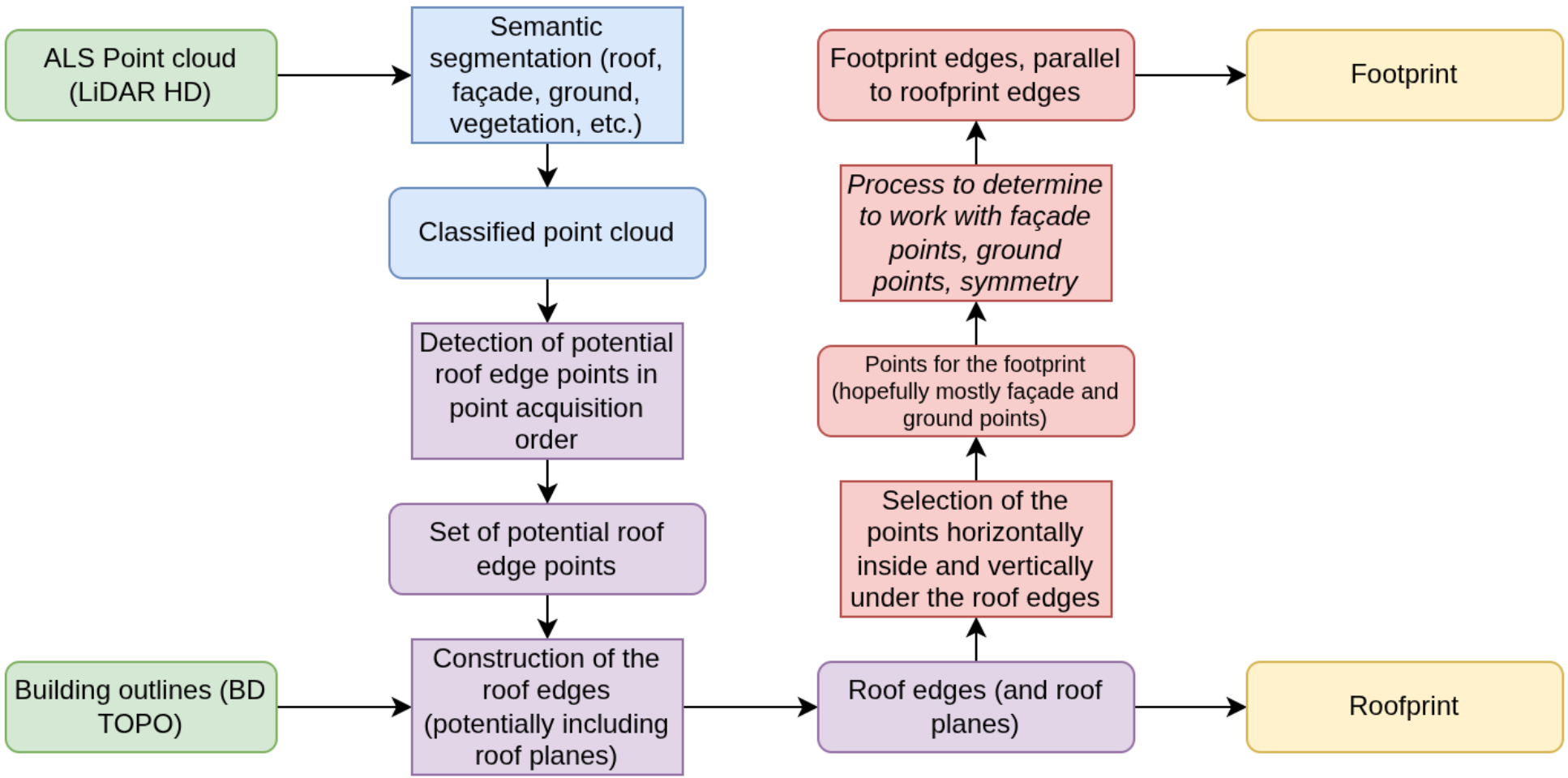
- Use the **repartition of points** on the edge in their score to prioritize a less points with a uniform repartition over a lot of points clustered on one side of the edge.
- Use **combinatorial optimization** to pick the best set of edges with multiple solutions for difficult edges.
- Allow for small **rotations** of the edges.
- Look at edges in **3D** instead of 2D.

Footprint generation

Planned

- Find criteria to differentiate between roof edges and façade points (including training a deep learning model if needed).
- Use the **height variations in acquisition order** with different thresholds.
- Use **rays directions** to count points on potential façades **positively** and **negatively**.
- **Adapt the matching criterion** to the façades:
  - Not easy to compute the **inward vector** (potentially with the ground points).
  - Care even more about **distribution of points along the Z axis** to avoid matching to roof edges or vegetation.

Rest of the pipeline



Thank you for your attention!

Any questions?

alexandre.bry@ign.fr