Supplementary Material: Towards Globally Optimal Normal Orientations for Large Point Clouds

Nico Schertler, Bogdan Savchynskyy, and Stefan Gumhold

TU Dresden, Germany

Abstract

This supplementary material gives more information about Xie's flip criterion and the Signed Union Find data structure. Additionally the two segmentation criteria are defined formally.

1. Xie's flip criterion

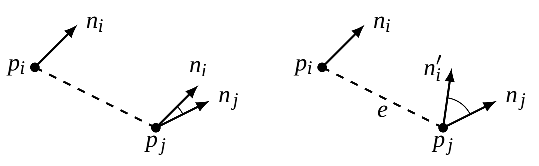
Xie's flip criterion can handle sharp creases and close surface sheets by reflecting one normal along the direction vector between the two points (see figure 1 for details). The criterion can be expressed as:

$$\begin{aligned} \phi_{Xie}(i,j) &= \langle n_i', n_j \rangle \\ n_i' &= n_i - 2e\langle e, n_i \rangle \\ e &= \frac{p_i - p_j}{\|p_i - p_j\|} \end{aligned} \tag{1}$$

2. Signed Union Find for MST Solutions

In our paper, we use the Signed Union Find data structure to find the Maximum Spanning Tree solution for an orientation problem. In this section, we explain the details of this data structure.

Hoppe's original algorithm [HDD*92] first calculates the spanning tree and in a second step performs the propagation. This requires traversing the graph twice (once completely and once the spanning tree) as well as storing the spanning tree explicitly. Our improved version merges both steps into a single traversal, calculating the spanning tree on-the-fly without explicitly storing it.



(a) Hoppe's flip criterion (b) Xie's flip criterion

Figure 1: Visualization of Hoppe's and Xie's flip criterion.

The basis of this improvement is Kruskal's minimum spanning tree algorithm [Kru56], which is usually implemented using a Union-Find data structure [Knu69]. A possible application of normal orientation using Kruskal's algorithm can be found in [XD11].

The Union-Find data structure is a forest of rooted trees where each entry corresponds to a node in the graph. Each node maintains a pointer to its parent node. The entire structure is implemented with a list of indices. Initially, each node is a separate tree. A connected component's representative (operation *find*) can be found by following the path of parent pointers up to the root. Two entries belong two the same connected component iff they share the same representative. Merging two connected components (operation *union*) is achieved by updating the parent pointer of one component's root to point to the other component's root. Due to some acceleration techniques like path compression, both operations can be executed in effectively constant time (more precisely, it grows very slowly in order of the inverse Ackermann function).

We augment this structure with a sign bit s_i for each node ($0 \equiv +$, $1 \equiv -$) and refer to it as *Signed Union-Find*. The idea is to enable quick sign flips for entire connected components, which is achieved by using the sign bits on the path from a node to its root as XOR summands of the node's actual sign:

$$sign(i) = \bigoplus_{j \in \text{path from i to root}} s_j \tag{2}$$

Path compression will ensure that this chained XOR is kept short, enabling effectively constant time complexity.

If the sign of an entire connected component must be changed (method *flipConnectedComponentSign* in algorithm 1), it is sufficient to flip the sign bit of the component's root (also effectively constant time). Since this bit is included in the sign formulas for every child node, this essentially flips the signs of all nodes within this connected component.

Care must be taken when uniting two root nodes. Without loss of generality we assume that node i will be the new root for node j. Then, in order to preserve the sign in the connected component of node j, the sign bit has to be updated with $s_j \leftarrow s_i \oplus s_j$. This works because $\oplus$ is its own inverse operation.

Path compression has to adapt the sign bits in a similar way. If node i with path to its root consisting of nodes $j, \ldots, q, r$ (node r being the root node) is re-linked to be a direct child of its root r, the sign calculation of node i changes from $s_i \oplus s_j \oplus \ldots \oplus s_q \oplus s_r$ to $s_i \oplus s_r$. It is obvious that the update must be $s_i \leftarrow s_i \oplus s_j \oplus \ldots \oplus s_q$. An entire path can be compressed (i.e. re-linking all nodes to the root) in $\mathcal{O}(k)$ time, where k denotes the number of nodes on that path.

This Signed Union-Find data structure enables on-the-fly computation of the spanning tree. Algorithm 1 shows its application to solving the orientation problem.

Algorithm 1 MST Solver Using Signed Union-Find

```
1: function SOLVE($\mathcal{P}, \mathcal{E}$)        ▷ Calculates optimal labeling
2:     $uf \leftarrow$ init Signed Union-Find with $|\mathcal{P}|$ nodes
3:     sort edges in descending order with respect to $|\phi|$
4:     for $\{i, j\} \in \mathcal{E}$ do
5:         $r_i \leftarrow uf.find(i)$
6:         $r_j \leftarrow uf.find(j)$
7:         if $r_i \neq r_j$ then
8:             $diffSigns \leftarrow uf.getSign(i) \neq uf.getSign(j)$
9:             if $diffSigns \oplus (\phi(i, j) < 0)$ then
10:                $uf.flipConnectedComponentSign(i)$
11:            $uf.merge(i, j)$
12:    return signs of $uf$
```

Although this algorithm does not directly propagate orientations along edges, the general idea is the same as before. One difference though is the absence of a starting point with known orientation. If desired, the orientation for a single point (or more non-contradicting points) can be forced after the execution of the algorithm by checking the according sign and flipping the component if necessary.

Figure 2 compares the performance of the MRF solver with the Signed Union Find data structure and with our implementation of the traditional one. Both data structures

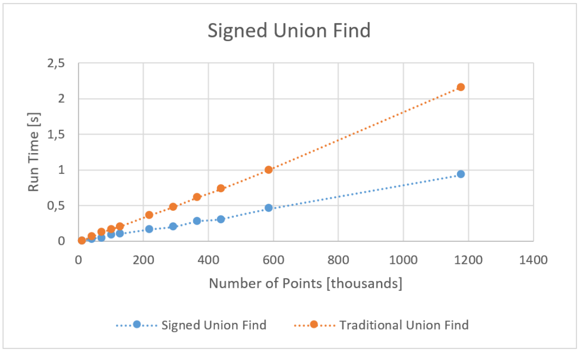


Figure 2: Run time comparison of the MST solver with the traditional Union Find data structure and our Signed Union Find data structure for different sizes of the dragon data set.

scale nearly linearly, while the Signed Union Find is in average 2.2 times as fast as the traditional data structure because it avoids the second traversal.

3. Segmentation Criteria

The segmentation criteria are used to evaluate if a segment can be assigned to a point. Given the neighbor vote of two neighboring points $vote(p_1, p_2)$, we derive the segment vote of a point p with respect to segment s as

$$vote(p, s) := \sum_{q \in s} vote(p, q), \tag{3}$$

where $vote(p, q)$ is zero for non-neighboring points. Similarly, the vote between two segments s_1 and s_2 is

$$vote(s_1, s_2) := \sum_{p \in s_1} \sum_{q \in s_2} vote(p, q) \tag{4}$$

The intra-segment criterion is defined as follows: The assignment of segment s to point p is valid iff, given the set of considered neighbor points $\mathcal{N}^{pt}(p, s)$ of point p from segment s, i.e. all neighbors that are already assigned to this segment:

$$\left(\sum_{n \in \mathcal{N}^{pt}(p,s)} vote_{>0}(p, n) \quad \leq \theta_S \quad \vee \right. \\ \left. - \sum_{n \in \mathcal{N}^{pt}(p,s)} vote_{<0}(p, n) \leq \theta_S \right) \quad \wedge \\ |vote(p, s)| \geq \theta_{acc} \tag{5}$$

The symbols $vote_{>0}(p_1, p_2)$ and $vote_{<0}(p_1, p_2)$ denote the positive and negative part of the scalar neighbor vote between two neighboring points p_1 and p_2. I.e.

$$vote_{>0}(p_1, p_2) = \begin{cases} vote(p_1, p_2) & \text{if } vote(p_1, p_2) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The inter-segment criterion specifies that a segment is valid iff

$$\forall t \in \mathcal{N}^{seg}(p) \setminus s : \\ (sgn(vote(s,t)) = sgn(vote(p,t)) \cdot flip) \vee |vote(p,t)| < \theta_S, \tag{7}$$

where $\mathcal{N}^{seg}(p)$ denotes the set of neighbor segments of point p and $flip$ is the flip decision $= sgn(vote(p,s))$.

References

[HDD*92] Hoppe H., DeRose T., Duchamp T., et al.: Surface reconstruction from unorganized points. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1992), SIGGRAPH '92, ACM, pp. 71–78. doi:10.1145/133994.134011. 1

[Knu69] Knuth D. E.: *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. Reading, Mass.: Addison-Wesley, 1969. 1

[Kru56] Kruskal J. B.: On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society 7*, 1 (1956), 48–50. 1

[XD11] Xi Y., Duan Y.: A new integrated depth fusion algorithm for multi-view stereo. *Computer Graphics International* (2011). 1