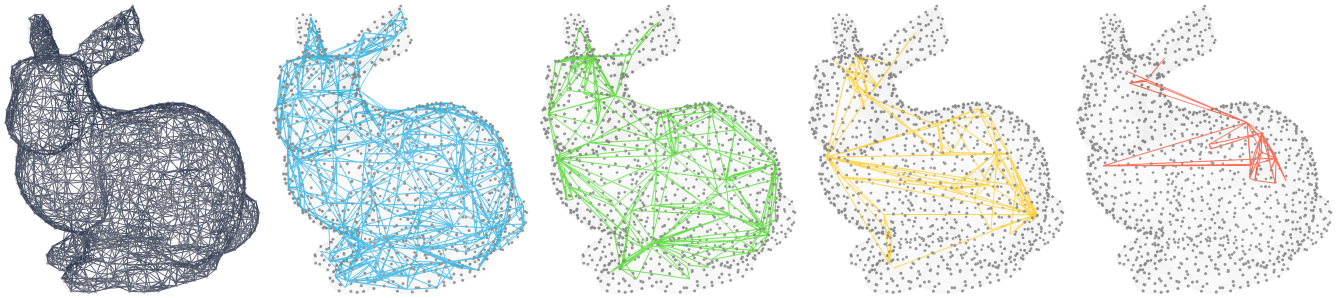
DOI: 10.1111/cgf.13797
Eurographics Symposium on Geometry Processing 2019
D. Bommes and H. Huang
(Guest Editors)

Volume 38 (2019), Number 5

Parallel Globally Consistent Normal Orientation of Raw Unorganized Point Clouds

J. Jakob, C. Buchenau, M. Guthe

University of Bayreuth, Visual Computing, Germany



(a) (b) (c) (d) (e)

Figure 1: *Graphical illustration of our developed graph collapse algorithm. 1a shows the base kNN-Graph for a neighborhood size of eight, which was built from the input point cloud. 1b-1e visualize successive steps of our graph collapse method, that represents more compact information of the underlying point cloud. Each collapsed graph-point represents an oriented patch (set of points in the cloud) with a relation to other already oriented patches (graph edges).*

Abstract

A mandatory component for many point set algorithms is the availability of consistently oriented vertex-normals (e.g. for surface reconstruction, feature detection, visualization). Previous orientation methods on meshes or raw point clouds do not consider a global context, are often based on unrealistic assumptions, or have extremely long computation times, making them unusable on real-world data. We present a novel massively parallelized method to compute globally consistent oriented point normals for raw and unsorted point clouds. Built on the idea of graph-based energy optimization, we create a complete kNN-graph over the entire point cloud. A new weighted similarity criterion encodes the graph-energy. To orient normals in a globally consistent way we perform a highly parallel greedy edge collapse, which merges similar parts of the graph and orients them consistently. We compare our method to current state-of-the-art approaches and achieve speedups of up to two orders of magnitude. The achieved quality of normal orientation is on par or better than existing solutions, especially for real-world noisy 3D scanned data.

CCS Concepts

• ***Computing methodologies*** → *Shape analysis;* • ***Theory of computation*** → *Computational geometry; Massively parallel algorithms;*

1. Introduction

Nowadays, point clouds captured by geometry acquisition techniques such as structure from motion, multi-view geometry, or laser scanners contain not only spatial data but also additional attributes such as colors or estimated normals. The latter are of particular importance for algorithms such as registration, computing signed distance fields, feature-detection, segmentation, surface reconstruction, or shaded rendering. Most acquisition techniques have the restriction of either providing only locally oriented normals or none at all. However, oriented point normals are mandatory to allow inside-outside determination, since this information is a fundamental building block for higher-level geometry processing algorithms. In the case of surface reconstruction, their absence can lead to poor-quality meshes or artifacts, as shown in Figure 2. To overcome this limitation, a globally consistent orientation of normals across the entire data set is essential. This is a challeng-

ing field of research because real-world data contain noise, non-uniformities, and thin sharp features.

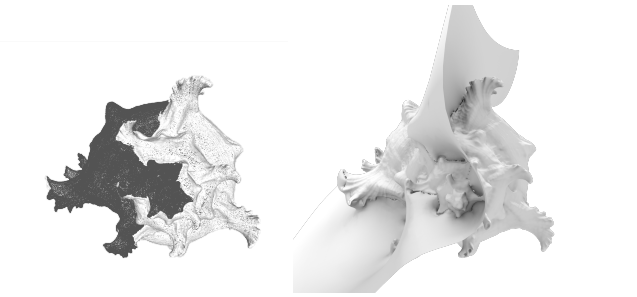


Figure 2: *Visualization of a defective poisson reconstruction caused by inconsistently oriented point normals. Left: inconsistently oriented source point cloud as a result of a minimum spanning tree propagation method using the flip criterion of Hoppe et al. [HDD*92], right: the screened poisson reconstruction of the cloud.*

Due to higher resolution geometry acquisition methods the resulting point clouds increase in size, which also increases the processing time for estimating and computing globally consistent normals. To address this issue, we present a fast parallel greedy solver for estimating and computing globally consistent oriented normals using the GPU. The challenging aspect of an optimized algorithm is to ensure maximum parallelism with same results as for sequential processing.

Our main contributions are:

- A parallel greedy solver running on the GPU to estimate globally consistent normals across the entire point cloud.
- A new normal similarity criterion for data sets with sharp creases and thin structures as well as imperfect real world data, containing (non-gaussian) noise and outliers.

Our normal estimation and orientation works well with both synthetic and unstructured, noisy, imperfect real-world point clouds and does not make any assumption about the input data (e.g. watertight, connected, unconnected parts). We achieve speedups of up to two orders of magnitude, while the quality of the global normal orientation is on par or better than state-of-the-art methods, e.g. compared to MST+QPBO-I of Schertler et al. [SSG16].

2. Related Work

Normal orientation algorithms can be classified into propagation-based, volume-based and other approaches. In this section we provide a short overview of existing normal orientation algorithms.

2.1. Propagation-based methods

Hoppe et al. [HDD*92] presented one of the first propagation-based methods. They compute tangent planes using a principle component analysis to define a signed distance function for a Riemannian Graph. Starting from a seed point on this graph, the orientation is propagated over the local neighborhood of the k-nearest points. The algorithm constructs a minimum spanning tree from these edges and flips the normals along the edges traversing the tree. It determines if the orientation is consistent for each edge, and performs a flip if necessary. Their approach is optimized for smooth surfaces but has limitations if the input-data contains sharp edges and creases. Based on this method, Xie et al. [XWH*03] proposed an improved flip criterion for handling sharp edges and creases. Compared to Hoppe et al., they used multiple seed points and only propagate the orientation along suitable edges. Because the algorithm excludes regions of high curvature while propagating the orientation, several oriented patches are constructed. Finally, these patches are oriented with respect to each other using a different criterion. A drawback, however, is that the algorithm requires the specification of a feature size for correct results. In addition, Guennebaud and Gross [GG07] developed another flip criterion to support thin features, that was further improved by König and Gumhold [KG09].

A slightly different approach was introduced by Huang et al. [HLZ*09], that first reduces the noise of the input data by using a weighted locally optimization projection (WLOP). Similar to Hoppe et al., they used a modified priority weight to propagate the local orientation in order to handle close-by surfaces. A modified PCA is applied to iteratively calculate normal orientations based on previous orientations. This method first detects thin sharp features using a simple conservative approach before performing a test operation to determine the orientation of close-by surface patches. However, it still fails at concave sharp features, creases, or in undersampled regions. Another drawback is the usage of a down-sampled point set instead of the original data.

The method for propagating the normal orientation by Seversky et al. [SBY11] used the gradients of harmonic functions to find an optimal orientation. Similar to Hoppe et al., a spanning tree with a global harmonic function was used for the edge weights of neighboring points.

Analogously to [XWH*03], Liu et al. [LCL*14] used a spanning tree with multiple seed points, which are distributed optimally based on a visibility criterion. Despite this local and fast method, they cannot achieve equal quality results compared to the aforementioned approaches.

All propagation-based methods share the common drawback of neglecting all edges that are not part of the final spanning tree. This inherently prevents finding a global optimum for the underlying flip criterion.

2.2. Volume-based methods

Volume-based methods can be split up into grid-based methods or approaches using a (signed) distance function to orient the point set. Grid-based methods apply algorithms on a volumetric discretization to estimate an optimal orientation.

2.2.1. Regular Volumetric Grid-based

One of the first volumetric grid methods is the approach by Zhao et al. [ZOF01], which first calculates a rough initial surface approximation using a minimal surface approach and then used this information to solve a variational model. The point normal and isosurface are determined in a single step using the surface energy.

Their approach is capable of handling noisy, non-uniform as well as multi-resolution data, but requires a large amount of computation time.

Mello et al. [MVT03] flagged each grid vertex as inside or outside based on an initial local fitting algorithm and compute a global orientation afterwards using simulated annealing.

Jalba et al. [JR09] and Chen et al. [CCLN10] used an octree and determine a global orientation by tagging nodes or corners as inside or outside. This tagging is based on membrane potentials or a visibility function, which is evaluated by rendering the point cloud from predefined view points. This restricts the approach to complete point clouds, i.e. processing partial scans is not possible.

In summary, volumetric approaches are well suited for noisy data sets that may contain outliers. The main drawbacks, however, are the high computational costs and the assumption that the point cloud represents a closed manifold without boundaries.

2.2.2. Signed Distance Function based

Instead of orienting the points, it is also possible to use a robust unsigned distance function. The global orientation is then found by reconstructing the sign on the computed field function.

Mullen et al. [MDGD*10] computed an unsigned level-set on the initial point cloud via an octree and performed a coarse delaunay triangulation from these octree points afterwards. The sign is reconstructed by shooting rays across the entire domain. This method however, struggles with structured outliers and piecewise smooth surfaces.

Similar to Mullen et al, Mehra et al. [MTSM10] used an unsigned distance function, which is signed by stochastic ray tracing and global smoothing. Their method is stable in the presence of noise and outliers but is complex and requires many parameters.

Gong et al. [GPS12] used a connected component analysis to separate inside and outside instead of ray tracing. The distances to the constructed band regions are used to reconstruct the sign of the field.

It is also possible to reformulate the normal orientation as an energy minimization task on a regular grid [GCSA13]. First, a robust unsigned distance function is built, which is then signed afterwards with a graph labeling approach. The proposed method tries to find the best out of many orientation hypotheses using global optimization. The downside of this approach is the combinatorial search space, resulting in a non-scalable algorithm.

In summary, signed distance function approaches work best on point sets that represent a closed manifold without boundaries and fail quickly in the presence of larger holes or under-sampled regions.

2.3. Other methods

Based on a Voronoi diagram of the point cloud, Alliez et al. [ACSTD07] developed an algorithm to directly reconstruct the surface from the unoriented points. The basic idea is to compute a tensor field and calculate the orientation using a generalized eigendecomposition of the local tensor afterwards.

Liu and Wang [LW10] proposed to first generate a triangle mesh from the unoriented points using an adaptive spherical cover. After topology cleaning and consistently orienting the mesh, a nearest triangle search is used to orient each point of the input point cloud. Structural noise and outliers, however, cause incorrect results.

Cao et al. [CHL*11] used global shrinking of the point cloud before visibility voting to improve the results. Further, a confidence is assigned to each point and Laplacian smoothing is applied to propagate the orientation of highly confident neighbors to points with low confidence.

Wang et al. [WYC12] constructed a constrained linear system and solve it using quadratic programming. While this is robust and effective, it needs fine tuning of parameters if the point cloud contains sharp features. Their approach cannot handle large point clouds or fails if the cloud contains noise or outliers.

Ochmann et al. [OK19] presented a method which was developed especially for building scans. First, the input cloud is simplified by finding planar surfaces. Then they classify correct orientations via a voting scheme based on path tracing and transfer them to the original points.

Schertler et al. [SSG16] reformulated the task as a graph-based energy optimization problem, similar to the propagation-based approaches. Instead of spanning trees, they use global solvers to minimize the total energy. The achieved results are superior to the spanning tree approaches, but the computational complexity is higher. For applying their models to large data sets they introduced a streamed version, that orients the cloud in a hierarchical manner over local patches. Compared to previous methods, Schertler et al. do not make any assumptions on the point cloud and are not restricted by the data-set size.

3. Algorithm Overview

Our normal orientation algorithm can be split up into three phases: *parallel normal estimation*, *knn-graph construction* and *parallel globally consistent normal orientation*. In the first phase (see Section 3.1), we estimate unoriented normals using the GPU and reuse the neighborhood dependency to create a full kNN-Graph in the second phase (see Section 3.2). Normals are then oriented globally consistent across the entire point cloud using a greedy edge collapse algorithm on the constructed graph in the third phase (see Section 3.3).

3.1. First Phase: Parallel Normal Estimation

Given a set of points, point normals can be estimated by applying a local low-order surface fitting. A common approach is a local tangent plane estimation using a Principal Component Analysis (PCA) [Pea01] on the k-nearest points of the point of interest, which is called the k-neighborhood [HDD*92].

Parallel k-Nearest Neighbor To fit a normal to every point in the data-set, all k-nearest neighbors (kNN) must be found for each point. To speed up the kNN search we use a combination of a fast bounding volume hierarchy (BVH) construction and a heap-based traversal until all kNN are found.

This is done by sorting and organizing the point cloud by assigning a 30 bit (10 bit for each dimension) morton code to each point. Then we perform a bottom-up construction of a simple radix tree as proposed by Karras et al. [Kar12]. Subsequently, we apply a heap-based traversal for each vertex in the cloud to find all kNNs and store them in a sorted list. Because the points are sorted, our implementation achieves a high data locality (because of the usage of the z-curve) and coherence resulting in very fast execution times even for a large set of nearest neighbors.

Parallel Normal Estimation As shown in earlier works, the eigenanalysis of the covariance matrix of a local neighborhood can be used to efficiently estimate local first-order surface properties [PGK02, HDD*92, Pea01]. We apply this covariance analysis to all found kNNs

$$C[p] = \frac{1}{k-1}\sum_{i=1}^{k}(p_i-\bar{p})(p_i-\bar{p})^T, \quad \text{with} \quad \bar{p} = \frac{1}{k}\sum_{i=1}^{k} p_i$$

and solve the eigenvector problem $C \cdot v_l = \lambda_l \cdot v_l$, with $l \in 1,2,3$ for this matrix.

The surface normal n_p of point p is approximated by the smallest eigenvector v_1 and a plane through $\bar{p}$ can be defined as:

$$T(x) : (x-\bar{p}) \cdot v_1 = 0$$

In the case of nearly collinear points (two eigenvalues nearly vanish), we remove the current point from the normal computation.

3.2. Second Phase: Parallel kNN-Graph Construction

To compute globally consistent oriented normals we propagate the correct orientation across our data-set using a full kNN-Graph. This graph can be easily constructed by reusing the neighborhood information gathered during the normal estimation phase. For each found neighbor an edge is created in a fully parallel manner. Duplicate edges are then removed via a parallel sort and compaction method.

After building the graph, for each edge a potential $E_{i,j}$ and a weight $W_{i,j}$ is assigned:

$$E_{i,j} = \psi(i,j) \cdot \omega(p_i, p_j)$$
$$W_{i,j} = |E_{i,j}|,$$

where $\omega(p_i, p_j)$ is a distance weighting based on the *support radius* r:

$$\omega(p_i, p_j) = e^{-\frac{d(p_i,p_j)^2}{\max(r_i,r_j)^2}}$$

and $\psi(i,j)$ is the used flip-criterium that specifies the per-vertex normal relation between two connected neighbors. The *support radius* is defined as the maximum of all point to point distances within each neighborhood of the two points an edge is formed with.

The assigned edge energies and weights are a key component of our method since they allow us to decide whether point-normals across an edge are aligned in the same direction or if one of them has to be inverted. The decision, if a normal must be flipped is considerably easier the more precise the used flip criterion is.

Flip Criterion A Function $\psi : (i,j) \mapsto \mathbb{R}$ can be used as a flip criterion, if it represents a similarity correlation of n_i and n_j. Common flip criteria are those of Hoppe et al. [HDD*92]:

$$\psi_{Hoppe}(i,j) = \langle \vec{n}_i, \vec{n}_j \rangle$$

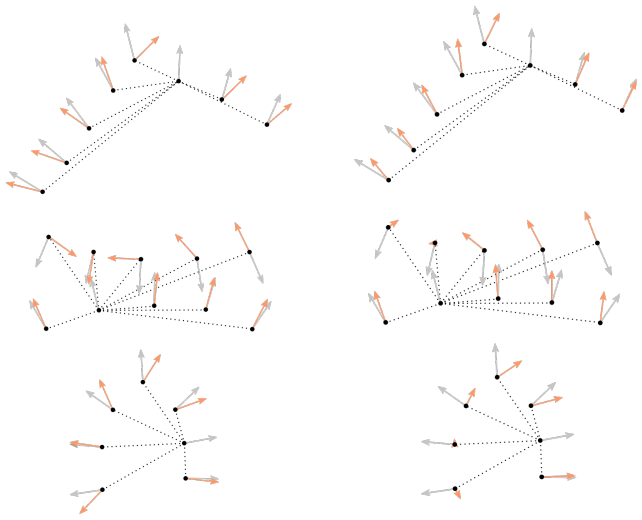
and the more elaborate one of Xie et al. [XWH*03]. The latter one reflects the normal along the direction vector of p_i and p_j:

$$\psi_{Xie}(i,j) = \langle \vec{n}_i', \vec{n}_j \rangle \quad with$$
$$\vec{n}_i' = \vec{n}_i - 2e\langle e, \vec{n}_i \rangle$$
$$e = \frac{p_i - p_j}{||p_i - p_j||}.$$

As both are limited and do not perform best on noisy real-world data sets containing thin and sharp structures, we have developed a new flip criterion.

Our idea is based on a simple observation: If an edge is drawn between two points in the point cloud, then this edge ideally lies in the approximated tangent area, which is exclusively described by the local environment. The smaller the deviation of both normals with respect to the virtual surface normals, the higher the confidence (see Figure 3). Therefore, our approach differs from that of Xie et al. in that we perform a *projection* onto the plane bisecting the edge, instead of a *reflection* across the plane bisecting the edge:

$$\psi_{Our}(i,j) = \langle \vec{n}_i', \vec{n}_j \rangle \quad with$$
$$\vec{n}_i' = \vec{n}_i - e\langle e, \vec{n}_i \rangle$$
$$e = \frac{p_i - p_j}{||p_i - p_j||}.$$



(a) Xie's criterion (b) Our criterion

Figure 3: *Comparison of our new flip criterion to Xie. Gray colored vectors represent the local point normals, orange vectors are the $\vec{n}_i'$ created by the respective energy criterion. Usually, the difference is very subtle, but with sharp creases and thin structures our criterion shows much lower confidence rates compared to Xie if an edge exists between two parts that should not propagate any information between each other.*

3.3. Third Phase: Parallel Globally Consistent Normal Orientation

A consistent orientation of all point normals can be obtained by maximizing the sum of all edge energies of the kNN graph:

$$\max \sum_{(i,j)\in\Omega} E_{i,j}$$

with $i, j \in \Omega$ being all valid pairs of edges between all vertices in the point set Ω. Figure 4 shows an example of a resulting kNN-graph. Once this graph is constructed and all edges received an energy value, maximizing the sum of edge energies is equivalent to finding the optimal sign for each vertex, which reduces to a binary quadratic programming problem. This is known to be NP hard.

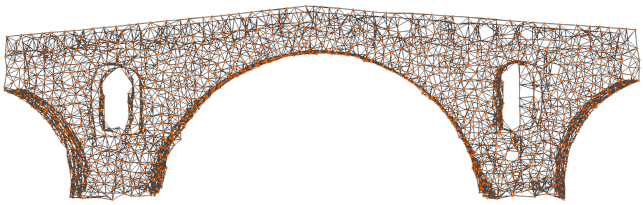


Figure 4: *Example kNN-Graph of a downsampled 3D scan of a bridge with neighborhood size 8.*

We adopt a greedy approach to solve this optimization problem. Previous greedy methods propagated the orientation of one or multiple seed normals across a minimal spanning tree (MST). Therefore, they consider only the energy of the edges within the MST, which facilitates finding the optimal solution with respect to the formerly build MST. The downside of this strategy is the neglect of potentially important edges, leading to incorrect decisions and thus inconsistently oriented point clouds as shown in Figure 2.

3.3.1. Concept

Instead of accelerating a combinatorial solver which finds the maximum energy for that graph, we use a completely different approach, which is inspired by the Quadric Error Simplification [GH97].

The basic idea is to take the kNN-graph and merge successive points into patches to simplify the graph as shown in Figure 5.

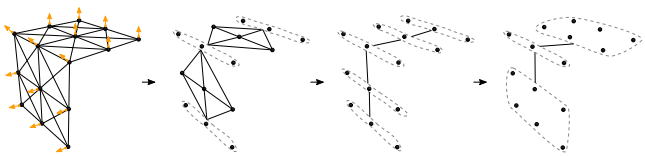


Figure 5: *Exemplary graph collapse. Edge collapses result in patches (shown by dashed lines) that contain a locally consistent orientation. Edges between formed patches store accumulated correspondences about orientation, encoded as the graph energy.*

Two points or patches are merged if the edge weight of the connecting edge is very high. If the orientation of the normal flips along an edge, we correct and store this information. The resulting patch then represents the relationship of the underlying point set (and its orientation) to all other patches connected by edges. The collapsing edges are merged together and receive new weights and energies based on the collapse. An example is depicted in Figure 6.

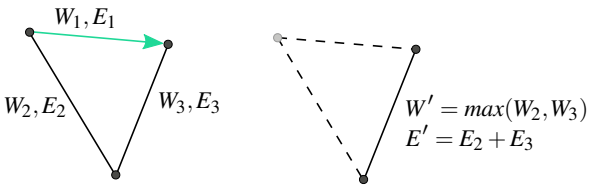


Figure 6: *Due to the collapse of e_1, e_2 and e_3 coincide. Since both edges had different relationships to the now merged patches, these are now accumulated in the remaining edge accordingly.*

This is repeated until no edge is left. The final orientation of each point in the point cloud can then be corrected by traversing the collapse hierarchy and accumulating each stored flip.

Since we use the weight to decide which edge is collapsed next, we use the same ordering as an MST, but consider the total energy as we accumulate it over the collapsed edges. The potential of this approach is shown by the fact that a patch describes a locally oriented set of points. If two patches with different orientations are collapsed together, only the orientation of the whole patch has to be inverted (and saved).

3.3.2. Graph Collapse Criteria

The outlined concept can be achieved by using an iterative sequential approach: simply collapsing the edge with the highest confidence (edge-weight) in each step.

To get the most efficient algorithm we formulate a parallel edge-clustering algorithm running entirely on the GPU on top of the full kNN-graph. Furthermore, we derive the following criteria to ensure identical results as with a sequential approach, while maximizing performance:

1. The edge of a patch with maximum weight is its collapse edge.
2. An edge is allowed to collapse if it is the collapse edge of both patches.

These conditions guarantee that no edge is considered in a different collapse than in sequential processing. An example is shown in Figure 7 with the collapse edges marked in green and red, depending on whether they are allowed to collapse or not.

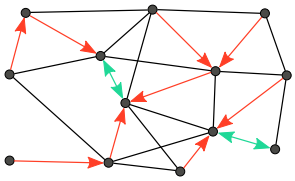


Figure 7: *Example collapse iteration. All green edges are allowed to collapse simultaneously, red edges are not allowed and black edges do not collapse. The arrow origin indicates which node selected this edge as collapse edge (due to maximum weight of all adjacent edges) and onto which node it wants to collapse.*

Therefore, similar to minimal spanning tree methods, our algorithm is a greedy approach. In each collapse the overall-orientation

of one of the two patches can be completely flipped. Thus, the relative orientation within a patch is fixed. But in contrast to MST methods, we use all graph edges instead of only a subset, leading to a much better solution.

Criteria Extension However, tests have shown that collapsing only edges identified this way, results in a graph consisting of only a few vertices of high valence, as shown in Figure 8.

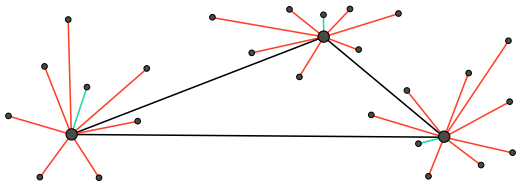


Figure 8: *Example star-shaped kNN-Graph after several iterations, using only collapse criteria 1 and 2.*

The resulting star-shaped patches are interconnected by only one edge, which results in a nearly sequential processing of the remaining edges and a significant performance drop in a SIMD execution model. It can be easily seen that patches which are only connected to the remaining graph via a single residual edge, can be collapsed at the same time without changing the collapse sequence and thus the final result (energy), unless they are connected by a non-collapse edge. The same applies to all edges forming a collapse chain that collapses into the same energy maximum. This leads to the following additional collapse-criteria:

3. If a non-collapse edge connects two outgoing collapse edges, that fall into the same weight maximum, the one with the lower weight cannot collapse
4. All remaining collapse edges can collapse

Thus, neither edge-energies nor edge-weights receive different values compared to a sequential processing, since these are updated as shown in Figure 6. An example of this extension is shown in Figure 9. Compared to the previous collapse (Figure 7), the single

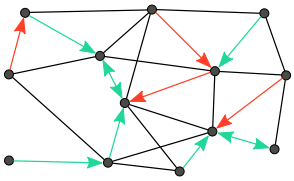


Figure 9: *Example collapse iteration with extension. All green edges are allowed to collapse simultaneously, red edges are not allowed and black edges do not collapse. Arrows also depict the direction of current energy maximum.*

patch on the bottom left, for example, is now allowed to be collapsed at the same time. All adjacent green edges that would be collapsed into the same patch and are not interconnected with each other (only chain-wise) are now also allowed to be collapsed.

3.3.3. Parallel Greedy Solver

In the following we discuss the parallel implementation of this method in more detail. The developed parallel clustering algorithm can be roughly split up into steps 2-7 shown in Algorithm 1.

Algorithm 1 GreedyEdgeClusteringSolver()

```
1: while edges left in G do
2:     c ← FIND_COLLAPSIBLE_EDGES(G)
3:     c ← FIND_COLLAPSE_CHAINS(G, c)
4:     HANDLE_ORIENTATION_FLIPS(G, c)
5:     COLLAPSE_EDGES(G, c)
6:     ADJUST_EDGE_ENERGIES_WEIGHTS(G, c)
7:     REMOVE_COLLAPSED_EDGES(G, c)
```

The `GreedyEdgeClusteringSolver` iteratively selects as many simultaneously collapsible edges as possible, then collapses them, and adjusts the graph energy and topology accordingly. These steps are repeated until no more edges are available. In each iteration, we flag the point normal of a respective patch if it has to be inverted. Finally, the normal orientation is determined by traversing the collapse hierarchy.

The required data structures for the solver is given in Table 1.

Group	Elements	Size	Type
Edges	*edges*	e	`int2`
	edge_energies	e	`float`
	edge_weights	e	`float`
	collapse_flag	e	`int`
Vertices	*max_edge_energy_per_vertex*	n	`float`
	edge_weight_edge	n	`int`
	edge_index_per_vertex	n	`int`
	collapse_target	n	`int`

Table 1: *Elements and size of data-structures required by our solver, where n is the number of vertices and e the number of edges.*

Find Collapsible Edges In the first stage, all edges that are collapse edges for both patches (see criterion 2) are identified. To determine all collapse edges, first the maximum edge weight of all incident edges is computed and stored with each vertex (see lines 1-4 in Algorithm 2). Via criterion 1, a collapse edge is discovered by comparing its weight to the weight of its vertices. If the weight is equal to one of both, the edge is identified as a collapse edge and its ID is stored with the respective vertex into *max_weight_edge*, as shown in lines 5-10 in Algorithm 2. To prevent race conditions of edges with equal weights, only the one with the maximum ID is stored. All valid collapse edges are then found by comparing if both incident vertices refer to it, see lines 12-15 in Algorithm 2. To store a collapse, the collapse target for each vertex is recorded. E.g. if an edge $e = (v_s, v_e)$ with $(v_s < v_e)$ is marked collapsible, v_s is stored as the collapse target for v_e.

To find and mark collapse chains, we also flag the remaining collapse-edges (lines 16-18 in Algorithm 2) which are the only

Algorithm 2 FindCollapsibleEdges()

```
1:  for all edges e in parallel do
2:      W ← edge_weights[e]
3:      ATOMICMAX(max_edge_energy_per_vertex[e.v1], W)
4:      ATOMICMAX(max_edge_energy_per_vertex[e.v2], W)
5:  for all edges e in parallel do
6:      W ← edge_weights[e]
7:      if (max_edge_energy_per_vertex[e.v1] == W) then
8:          ATOMICMAX(max_weight_edge[e.v1], e)
9:      if (max_edge_energy_per_vertex[e.v2] == W) then
10:         ATOMICMAX(max_weight_edge[e.v2], e)
11: for all edges e in parallel do
12:     ind1 ← max_weight_edge[e.v1]
13:     ind2 ← max_weight_edge[e.v2]
14:     if ((ind1 = ind2) & (ind2 = eID)) then
15:         collapse_target[e.v2] ← e.v1
16:     else if ((ind1 = eID) ⊕ (ind2 = eID)) then
17:         if (ind1 = eID) then collapse_target[e.v1] ← e.v2
18:         if (ind2 = eID) then collapse_target[e.v2] ← e.v1
```

maximum weight edge of one patch and remove the edges that violate criterion 3, see Algorithm 3. This is done by iterating over all non-collapse edges and checking if the outgoing collapse edges of their vertices fall into the same weight maximum. In this case, the one with the lower weight must not collapse and is removed (see lines 18/19 in Algorithm 3). Finally, the remaining edges are collapsed (see criterion 4).

Algorithm 3 FindCollapseChains()

```
1:  for all edges e in parallel do
2:      ind1 ← max_weight_edge[e.v1]
3:      ind2 ← max_weight_edge[e.v2]
4:      col1 ← collapse_target[e.v1]
5:      col2 ← collapse_target[e.v2]
6:      w1   ← edge_weight[ind1]
7:      w2   ← edge_weight[ind2]
8:      flag1 ← ((ind1 ≠ eID) & (ind2 ≠ eID))
9:      flag2 ← ((col1 ≠ {}) & (col2 ≠ {}))
10:     if (flag1 & flag2) then
11:         while (collapse_target[col1] ≠ {}) do
12:             col1 ← collapse_target[col1]
13:         while (collapse_target[col2] ≠ {}) do
14:             col2 ← collapse_target[col2]
15:         if (col1 ≠ col2) then
16:             RETURN
17:         else
18:             if (w1 < w2) then collapse_target[e.v1] ← {}
19:             else              collapse_target[e.v2] ← {}
```

Handle Orientation Flips To correct the local orientation between two collapsing patches, the stored edge-energy is checked. In the case of being negative, which implies differently oriented patches (or vertex normals if it is the first iteration), the stored edge-energy is negated.

Collapse Patches If all collapsible edges are found and the corresponding edge energies corrected due to differing orientations of two collapsing patches, the collapse can be executed and the topology adapted. This process is shown in Algorithm 4. For each edge, it is checked whether and onto which vertex (=patch) the adjacent patches of this edge were collapsed. Since we explained that in the previous section collapse chains are also possible, this chain must be traversed to find the very last target patch, that patch was collapsed on (see lines 4-11). To be able to sort all edges in ascending order based on their beginning and end vertex, the aforementioned edge invariant $(v_s < v_e)$ is restored last.

Algorithm 4 CollapseEdges()

```
1:  for all edges e in parallel do
2:      new1 ← {}
3:      new2 ← {}
4:      tmp ← collapse_target[e.v1]
5:      while (tmp ≠ {}) do
6:          new1 ← tmp
7:          tmp ← collapse_target[tmp]
8:      tmp ← collapse_target[e.v2]
9:      while (tmp ≠ {}) do
10:         new2 ← tmp
11:         tmp ← collapse_target[tmp]
12:     if (new1 ≠ {}) then e.v1 ← new1
13:     if (new2 ≠ {}) then e.v2 ← new2
14:     if (e.v1 > e.v2) then SWAP(e.v1, e.v2)
```

Adjust Edge Energies and Weights As mentioned before, a globally consistent orientation of all point normals can be obtained by maximizing the sum of all edge-energies in the kNN graph. The edge weights are used to order the collapse operations. One of the most important steps is to adjust the weight and energy of each edge after a collapse.

For this, the collapse operations are applied (as described in the previous paragraph) to all current edges and the resulting edges are sorted in ascending order with respect to their vertex IDs. Collapsed edges can thus be detected by checking their begin and end vertex IDs. For each edge with duplicates (edges that collapse together), the weight and energy is adjusted as follows:

$$W' = \max_{e \in \Delta} W(e)$$
$$E' = \sum_{e \in \Delta} E(e),$$

with Δ containing all duplicates of the current edge.

Remove Collapsed Edges After performing all collapse operations and edge weight/energy adjustments, invalid edges and duplicates are removed. Steps 2-6 of Algorithm 1 are then repeated until no edges are left.

Final Normal Orientation The globally consistent orientation is obtained by traversing the collapse hierarchy of each vertex and counting the number of flip-flags. Since each edge collapse describes a merge of two nodes and because we have already defined a collapse direction, the ID of the target vertex on which it collapses is stored for each vertex. This collapse chain is then traced backwards for each vertex and the stored orientation flips are accumulated as shown in Algorithm 5.

Algorithm 5 FollowCollapseChains()

```
1: for all vertices v in parallel do
2:     target ← collapseTarget[v]
3:     flipsign ← +1
4:     while (target ≠ {}) do
5:         flipsign ← flipsign · flipflag[target]
6:         target   ← collapseTarget[target]
7:     normal[v] ← flipsign · normal[v]
```

4. Results and Discussions

We compare our proposed algorithm to different existing orientation techniques in terms of runtime and quality. All algorithms were tested with commonly used point clouds of increasing size, different characteristics and surface complexity (we used real 3D scans containing noise and outliers as well as synthetically obtained data sets from sampled polygon meshes). For point clouds without normals, we used our (non-oriented) normal estimate as input for all other algorithms to ensure identical conditions. All vertex normal orientations were randomized beforehand.

We used the MST-approach of Hoppe et a. [HDD*92], as well as the streamed and non-streamed version of the method of Schertler et al. [SSG16]. For all algorithms a neighborhood of size 16 was used except for the streamed variant by Schertler et al., where we used half the neighbor size to generate a similar graph (because only the previous neighbors are considered in the streamed variant). Additionally, Schertler et al. was configured to use a search radius such that for 95% of all vertices all 16 neighbours are found to create an almost identical kNN-Graph. All remaining options of the orientation method of Schertler et al. were configured as outlined in their paper.

All measurements were performend on an Intel® Core™ i7-6700k CPU @ 4.00 GHz, 16 GB RAM and a NVIDIA GeForce™ GTX 1080, running under Linux 5.0.1 with CUDA Driver version 418.43.

In the following, we evaluate our introduced energy-criterion, as well as the influence of different neighborhood sizes in the presence of noise and our performance in terms of orientation-quality and runtime by comparing other methods.

4.1. Orientation Quality Comparison

In this section we compare the orientation quality of our parallel greedy approach to the state-of-the-art method of Schertler et al. [SSG16] using our energy criterion as well as those of Hoppe et al. [HDD*92] and Xie et al. [XWH*03]. For comparison, we used different point clouds with distinct characteristics, to obtain objective results. The results are shown in Table 3. The first two models represent clean and densely sampled polygon meshes, the last three are 3D scans produced by a table-top laser-scanner with noise, containing holes and poorly sampled areas. Our greedy solver computes results at least on par with Schertler's MST+QPBO-I method. Especially for synthetic or clean and dense data sets both algorithms produce correct results with all three energy criteria. However, with incomplete, noisy and non uniformly sampled data sets the advantage of our energy criterion becomes apparent, which can be seen in the last three columns. This is where Xie's criterion fails for both solvers. Using Hoppe's provides much better results, but with a noticeable amount of wrongly oriented points. Both solvers provide nearly perfect oriented results in such noisy data-sets, if our proposed energy criterion is used. The few exceptions (see Schertlers result of Herby) can be fixed for *both* solvers by slightly increasing the neighborhood size. Despite the fact that our method is based on a greedy approach, our results are at least on par (using the same energy criteria) with the state-of-art implementation of Schertler et al. [SSG16]

4.2. Noise

In the previous section, we showed that our energy-criterion can also cope with real-world data sets. In the following we analyze the robustness of our solver using this criterion, by adding outliers and noise at different levels to a synthetic model as also on further real world data sets.

We corrupted all input points using a zero mean Gaussian noise with standard deviations $\sigma = 0.5\%$, 1% and 1.5% of the bounding box diagonal length. To simulate outliers, we additionally added random Gaussian distributed points within the bounding box to the point-set of size 0.05%, 0.1% and 0.15% of the input cloud. All ground-truth normals were randomly inverted. Table 2 shows the results we obtained for both solvers using our energy-criterion.

	Noise Level					
	[SSG16]	**Our**	[SSG16]	**Our**	[SSG16]	**Our**
	$\sigma_1 = 0.5\%, \sigma_2 = 0.05\%$		$\sigma_1 = 1.0\%, \sigma_2 = 0.10\%$		$\sigma_1 = 1.5\%, \sigma_2 = 0.15\%$	
WelshDragon						
Armadillo						

Table 2: *Robustness of MST+QPBO-I [SSG16] and our approach to noise. We compared both solver using $k = 16$ and our energy criterion on two synthetic models corrupted with different levels of gaussian noise. σ_1 is the per point added noise level in percentage w.r.t. bounding box diagonal length. σ_2 denotes percentage of added outliers. Incorrectly oriented parts are colored red.*

Both solvers fail to orient the point clouds correctly in the presence

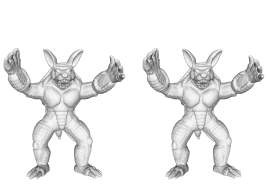
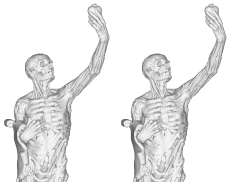
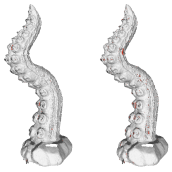
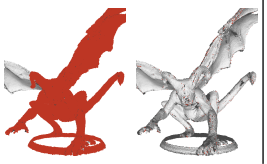
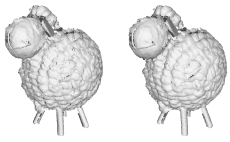
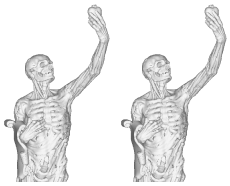
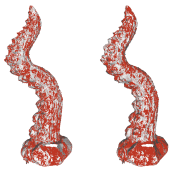
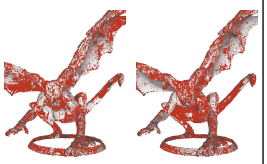
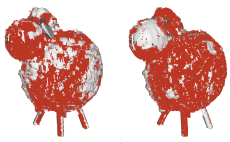
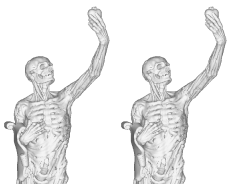
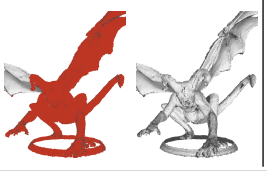
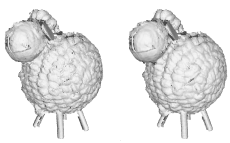
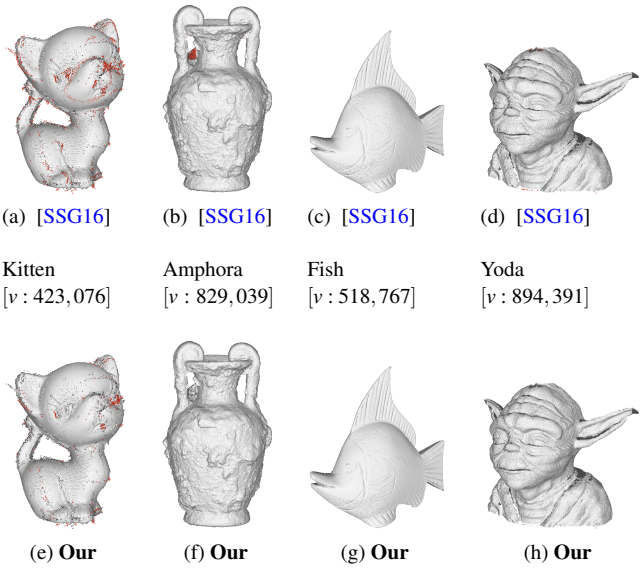
Criterion	Solver									
	[SSG16]	**Our**	[SSG16]	**Our**	[SSG16]	**Our**	[SSG16]	**Our**	[SSG16]	**Our**
	Armadillo		*Skellet*		*Tentacle*		*Herby*		*Sheep*	
Hoppe [HDD*92]										
Xie [XWH*03]										
Our										

Table 3: *Orientation results of MST+QPBO-I solver by Schertler et al. [SSG16] vs our proposed method using the energy criteria of Hoppe et al. [HDD*92], Xie et al. [XWH*03] and our proposed one. Most models contain thin close-by structures and sharp creases. The last three columns are noisy laser scans. Points are phong-shaded based on ther normal. Incorrectly oriented parts are colored red. Using our criterion with our solver leads to less errors compared to Schertler et. al even for real world 3D scanned data containing noise.*

of strong noise. The more details are lost in the noise, the less reliable is the outcome, which can lead e.g. to misaligned ears in the Armadillo model. Figure 10 shows both solvers results on further real world data sets, which were acquired using a laser scanner. In a direct comparison, our approach is slightly more robust against noise and outperforms the approach of Schertler et. al. [SSG16].



(a) [SSG16] (b) [SSG16] (c) [SSG16] (d) [SSG16]

Kitten [v : 423,076] Amphora [v : 829,039] Fish [v : 518,767] Yoda [v : 894,391]

(e) **Our** (f) **Our** (g) **Our** (h) **Our**

Figure 10: *Comparison of orientation results between [SSG16] and our approach for $k = 16$ for laser scan data. Only the first two laser scans (Kitten, Amphora) show erros (colored red).*

4.2.1. Noise vs Neighbohood Size

It is reasonable to assume, that larger neighborhoods are helpful for a better overall global normal orientation, as the neighborhood size directly affects the graph energy and thus individual orientation decisions. In Table 4 we demonstrate the effect of different neighborhood sizes on the global normal orientation in the presence of noise on both solvers using our proposed energy criterion. As it can be seen, a larger neighborhood on a data set with moderate noise delivers less errors on both solvers. In very noisy data sets, however, even a large neighborhood is not sufficient to properly orient the data set. In both cases, a too large kNN size also leads to incorrect results. Generally, however, our approach delivers better results.

4.3. Performance

In order to evaluate our algorithm's performance, we compare the execution time of our algorithm to the MST approach of Hoppe et al. [HDD*92] and the method of Schertler et al. [SSG16]. As the computation time is the only value of interest, all I/O related timings are neglected for all methods. Memory transfer timings via the PCIe bus are included for our approach. All algorithms were tested with a variety of different data-sets with very different characteristics, shown in Figure 11.

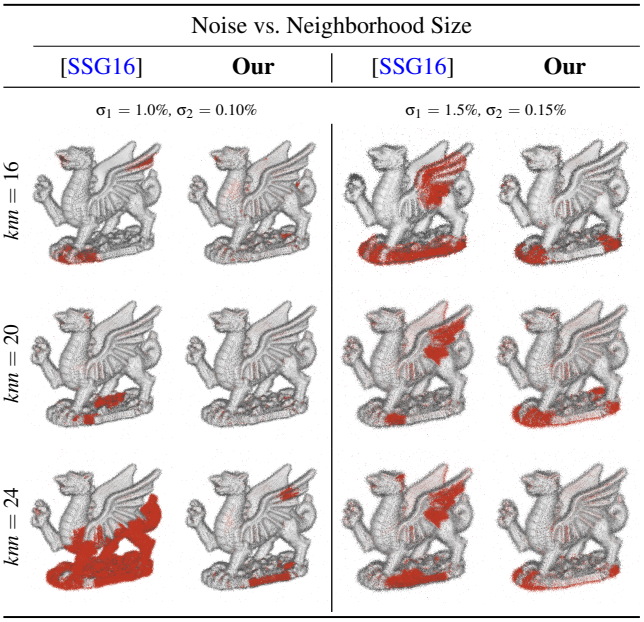


Table 4: *Comparison of achievable orientation results of both solvers (ours and [SSG16]) using our proposed energy criterion at different neighborhood sizes. As input we used the corrupted welsh dragon data from Table 2. Incorrectly oriented parts are colored red.*

Figure 11: *Visualization of all tested point clouds*

Runtime Comparison The runtimes of the algorithms are visualized in Figure 12. Despite the memory transfer timings, our algorithm is one to two orders of magnitude faster.The complexity of the point cloud has a direct influence on the execution time of Schertler et al. and leads to strongly fluctuating execution times, especially in the presence of noise and inhomogeneously sampled data. This is due to the number of non-orientable edges in the preprocessing step, which leads to a longer runtime of the actual QPBO-I solver. Some models could not be oriented by Schertler's approach, so a few measurement points are missing. Our approach is not affected by this and reveals an almost linear behavior with respect to the cloud size for the same kNN neighborhood.

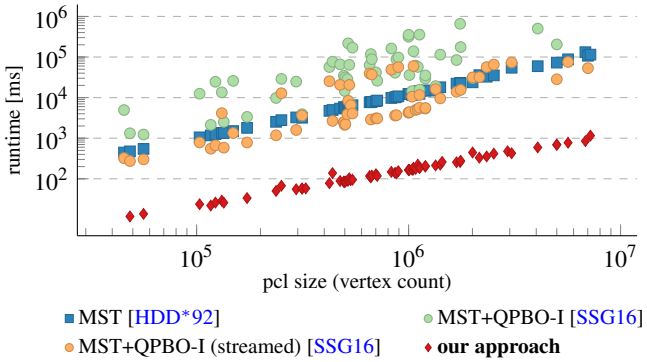


Figure 12: *Comparison of runtimes between MST [HDD*92], MST+QPBO-I in streamed and non streamed fashion [SSG16] and our approach for $k = 16$.*

Runtime Breakdown A breakdown of the runtime for two models and different k-neighborhood sizes is shown in Figure 13. Due to the efficient parallelization, only about 20% of our algorithm runtime is used for transfer, cuda runtime overhead and memory allocation/deallocation.

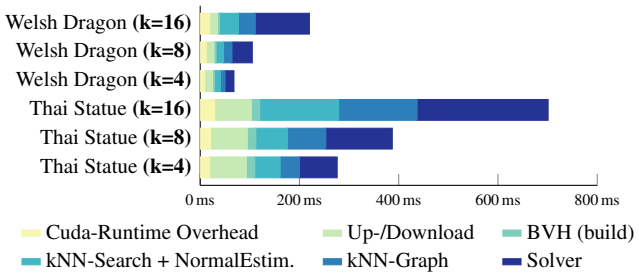


Figure 13: *Breakdown of the execution time of our parallel global normal orientation algorithm for two different models (Welsh Dragon [$v : 1,105,352$] and Thai Statue [$v : 4,999,996$]) using three different neighborhood sizes.*

5. Conclusion

In this paper we presented a novel parallel solver for computing globally consistent normal orientations of raw point clouds. By introducing a new similarity criterion, we can reliably orient both, synthetic and noisy, non-uniformly sampled 3D scan data containing outliers. Even sharp features, creases, and close-by surfaces can be handled. We achieve speedups of up to two orders of magnitude while the quality of the oriented normals is on par or better than current state-of-the-art methods.

However, a major limitation of our method is the GPU memory constraint, which we want to address in future research by developing a streamed approach. We are aware that our normal estimation using a PCA is not state-of-art. Many other algorithms exist that outperform a simple PCA approach. Our current focus was to develop a new energy criterion and a fast solver for the globally

consistent orientation. In the future, we would like to integrate an improved normal estimation as well.

Acknowledgements

We thank the respective authors for providing the used models and point clouds throughout this paper. The Murex Romosus model and Skellet (LeTransitDeReneChalon), are courtesy of Oliver Laric, threedscans.com. The Armadillo model are courtesy of Stanford 3D Scanning Repository. The bridge (Julien Bridge), Kitten, Amphora and Fish models are courtesy of AIM@SHAPE-VISIONAIR Shape Repository, All remaining models were scanned and aligned by ourselves using a portable 3D scanner. For our benchmark-dataset we used the available models provided by AIM@SHAPE-VISIONAIR Shape Repository, Stanford 3D Scanning Repository, Georgia Institute of Technology and Oliver Laric (threedscans.com). The *Streaming Normal Orientation* algorithm of Schertler et al. is licensed under the GNU General Public License v2.0 and is publicly available at https://github.com/NSchertler/StreamingNormalOrientation. We also thank the anonymous reviewers for the valuable contributions they have made to this paper. This research is funded by the Bayerische Forschungsstiftung (For3D).

References

[ACSTD07] ALLIEZ P., COHEN-STEINER D., TONG Y., DESBRUN M.: Voronoi-based variational reconstruction of unoriented point sets. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing* (2007), SGP '07, Eurographics Association, pp. 39–48. 3

[CCLN10] CHEN Y.-L., CHEN B.-Y., LAI S.-H., NISHITA T.: Binary orientation trees for volume and surface reconstruction from unoriented point clouds. *Computer Graphics Forum 29*, 7 (2010), 2011–2019. doi:10.1111/j.1467-8659.2010.01787.x. 3

[CHL*11] CAO J., HE Y., LI Z., LIU X., SU Z.: Short communication to SMI 2011: Orienting raw point sets by global contraction and visibility voting. *Computer & Graphics 35*, 3 (2011), 733–740. doi:10.1016/j.cag.2011.03.026. 3

[GCSA13] GIRAUDOT S., COHEN-STEINER D., ALLIEZ P.: Noise-adaptive shape reconstruction from raw point sets. In *Proceedings of the Eleventh Eurographics/ACMSIGGRAPH Symposium on Geometry Processing* (2013), SGP '13, Eurographics Association, pp. 229–238. doi:10.1111/cgf.12189. 3

[GG07] GUENNEBAUD G., GROSS M.: Algebraic point set surfaces. *ACM Trans. Graph. 26*, 3 (July 2007). doi:10.1145/1276377.1276406. 2

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 209–216. URL: https://doi.org/10.1145/258734.258849, doi:10.1145/258734.258849. 5

[GPS12] GONG Y., PAUL G., SBALZARINI I. F.: Coupled signed-distance functions for implicit surface reconstruction. In *2012 9th IEEE International Symposium on Biomedical Imaging (ISBI)* (2012), pp. 1000–1003. doi:10.1109/ISBI.2012.6235726. 3

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. *SIGGRAPH Comput. Graph. 26*, 2 (July 1992), 71–78. doi:10.1145/142920.134011. 2, 3, 4, 8, 9, 10

[HLZ*09] HUANG H., LI D., ZHANG H., ASCHER U., COHEN-OR D.: Consolidation of unorganized point clouds for surface reconstruction. In *ACM SIGGRAPH Asia 2009 Papers* (2009), SIGGRAPH Asia '09, ACM, pp. 176:1–176:7. doi:10.1145/1661412.1618522. 2

[JR09] JALBA A. C., ROERDINK J. B. T. M.: Efficient surface reconstruction from noisy data using regularized membrane potentials. *IEEE Transactions on Image Processing 18*, 5 (2009), 1119–1134. doi:10.1109/TIP.2009.2016141. 3

[Kar12] KARRAS T.: Maximizing parallelism in the construction of bvhs, octrees, and k-d trees. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics* (Goslar Germany, Germany, 2012), EGGH-HPG'12, Eurographics Association, pp. 33–37. URL: https://doi.org/10.2312/EGGH/HPG12/033-037, doi:10.2312/EGGH/HPG12/033-037. 4

[KG09] KÖNIG S., GUMHOLD S.: Consistent propagation of normal orientations in point clouds. In *VMV* (2009), pp. 83–92. 2

[LCL*14] LIU J., CAO J., LIU X., WANG J., WANG X., SHI X.: Mendable consistent orientation of point clouds. *Comput. Aided Des. 55* (Oct. 2014), 26–36. doi:10.1016/j.cad.2014.05.006. 2

[LW10] LIU S., WANG C. C. L.: Technical section: Orienting unorganized points for surface reconstruction. *Computers & Graphics 34*, 3 (2010), 209–218. doi:10.1016/j.cag.2010.03.003. 3

[MDGD*10] MULLEN P., DE GOES F., DESBRUN M., COHEN-STEINER D., ALLIEZ P.: Signing the unsigned: Robust surface reconstruction from raw pointsets. *Computer Graphics Forum 29*, 5 (2010), 1733–1741. doi:10.1111/j.1467-8659.2010.01782.x. 3

[MTSM10] MEHRA R., TRIPATHI P., SHEFFER A., MITRA N. J.: Technical section: Visibility of noisy point cloud data. *Computers & Graphics 34*, 3 (2010), 219–230. doi:10.1016/j.cag.2010.03.002. 3

[MVT03] MELLO V. C., VELHO L., TAUBIN G.: Estimating the in/out function of a surface represented by points. In *Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications* (2003), SM '03, ACM, pp. 108–114. doi:10.1145/781606.781625. 3

[OK19] OCHMANN S., KLEIN R.: Automatic normal orientation in point clouds of building interiors. *arXiv* (2019). 3

[Pea01] PEARSON K.: On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 2*, 6 (1901), 559–572. doi:10.1080/14786440109462720. 3, 4

[PGK02] PAULY M., GROSS M., KOBBELT L. P.: Efficient simplification of point-sampled surfaces. In *IEEE Visualization, 2002. VIS 2002.* (Nov 2002), pp. 163–170. doi:10.1109/VISUAL.2002.1183771. 4

[SBY11] SEVERSKY L. M., BERGER M. S., YIN L.: Harmonic point cloud orientation. *Computers & Graphics 35*, 3 (2011), 492–499. doi:10.1016/j.cag.2011.03.012. 2

[SSG16] SCHERTLER N., SAVCHYNSKYY B., GUMHOLD S.: Towards globally optimal normal orientations for large point clouds. *Computer Graphics Forum 36*, 1 (2016), 197–208. doi:10.1111/cgf.12795. 2, 3, 8, 9, 10

[WYC12] WANG J., YANG Z., CHEN F.: A variational model for normal computation of point clouds. *Visual Computer 28*, 2 (2012), 163–174. doi:10.1007/s00371-011-0607-6. 3

[XWH*03] XIE H., WANG J., HUA J., QIN H., KAUFMAN A.: Piecewise c1 continuous surface reconstruction of noisy point clouds via local implicit quadric regression. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (2003), VIS '03, IEEE Computer Society, pp. 13–. doi:10.1109/VISUAL.2003.1250359. 2, 4, 8, 9

[ZOF01] ZHAO H.-K., OSHER S., FEDKIW R.: Fast surface reconstruction using the level set method. In *Proceedings IEEE Workshop on Variational and Level Set Methods in Computer Vision* (2001), pp. 194–201. doi:10.1109/VLSM.2001.938900. 2