Field-Aligned Online Surface Reconstruction

NICO SCHERTLER, New York University and TU Dresden
MARCO TARINI, Università dell'Insubria and ISTI - CNR
WENZEL JAKOB, École Polytechnique Fédérale de Lausanne (EPFL)
MISHA KAZHDAN, Johns Hopkins University
STEFAN GUMHOLD, TU Dresden
DANIELE PANOZZO, New York University

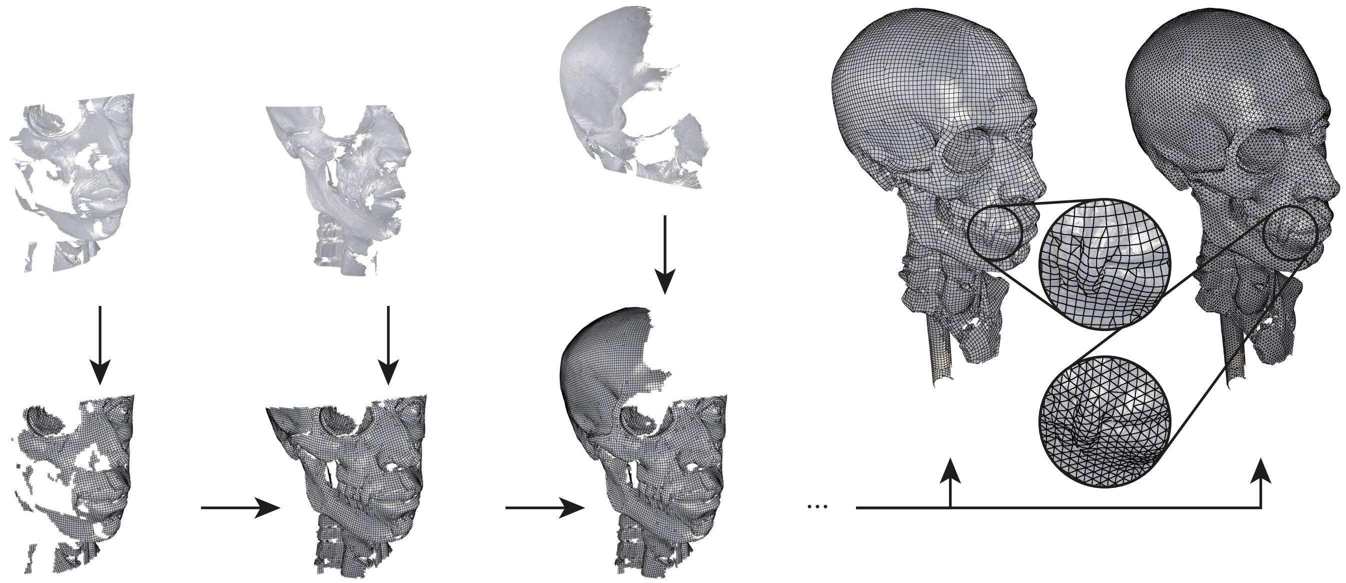


Fig. 1. Multiple successively acquired 3D scans (top row) are interactively integrated into a coarse base mesh (bottom row). The user sees the final reconstructed result at all times, enriched with textures that encode colors and displacement (right-most). The user can decide to reconstruct a triangle or a quad-dominant mesh, which has high isotropy and regularity. [Original Sculpture Courtesy of Michael Defeo, mesh statistics can be found in Table 1]

Today's 3D scanning pipelines can be classified into two overarching categories: offline, high accuracy methods that rely on global optimization to reconstruct complex scenes with hundreds of millions of samples, and online methods that produce real-time but low-quality output, usually from structure-from-motion or depth sensors. The method proposed in this paper is the first to combine the benefits of both approaches, supporting online reconstruction of scenes with hundreds of millions of samples from high-resolution sensing modalities such as structured light or laser scanners. The key property of our algorithm is that it sidesteps the signed-distance computation of classical reconstruction techniques in favor of direct filtering, parametrization, and mesh and texture extraction. All of these steps can be realized using only weak notions of spatial neighborhoods, which allows for an implementation that scales approximately linearly with the size of each dataset that is integrated into a partial reconstruction. Combined, these algorithmic differences enable a drastically more efficient output-driven interactive scanning and reconstruction workflow, where the user is able to see the final quality field-aligned textured mesh during the entirety of the scanning procedure. Holes or parts with registration problems are displayed in real-time to the user and can be easily resolved by adding further localized scans, or by adjusting the input point cloud using our interactive editing tools with immediate visual feedback on the output mesh. We demonstrate the effectiveness of our algorithm in conjunction with a state-of-the-art structured light scanner and optical tracking system and test it on a large variety of challenging models.

CCS Concepts: • **Computing methodologies → Mesh models**;

Additional Key Words and Phrases: Surface Reconstruction, Parameterization

This work was supported by the NSF CAREER award 1652515, NSF award 1422325, MIUR project DSurf, and a fellowship within the FITweltweit program of the German Academic Exchange Service (DAAD).

ACM Reference format:
Nico Schertler, Marco Tarini, Wenzel Jakob, Misha Kazhdan, Stefan Gumhold, and Daniele Panozzo. 2017. Field-Aligned Online Surface Reconstruction. *ACM Trans. Graph.* 36, 4, Article 77 (July 2017), 13 pages.
DOI: http://dx.doi.org/10.1145/3072959.3073635

1 INTRODUCTION

3D scanning and reconstruction are historically independent problems. During a typical scanning session, a coarse preview of the acquired data is shown, and it is only once all scans are available that a reconstruction algorithm consolidates them into a single consistent 3D model.

Different techniques to preview the scanned data exist: collections of range scans can be rendered directly, often after down-sampling, by splatting point primitives [Rusinkiewicz and Levoy 2000] or by superimposing triangulated height fields (e.g. [Cignoni et al. 2011]); other systems update a relatively low-res volumetric distance-field representation and interactively ray-cast it [Izadi et al. 2011; Newcombe et al. 2011]. In either case, the final, unified, high-resolution geometry is extracted after the scanning session, using offline mesh reconstruction algorithms (for example, based on Poisson surface reconstruction [Kazhdan et al. 2006]). Depending on the intended application, this is followed by semiregular remeshing, or simplification, which is then parameterized; finally, the original high frequency details or additional properties like colors are reintroduced as textures. While this workflow has been widely used in the last three decades, it hides a subtle, but major limitation: the preview is not a faithful representation of the final model and thus may not indicate artifacts like the ones due to poor registration, missing data, insufficient sampling density, etc. This leads to a frustratingly time-expensive workflow, where the user has to go back to acquire additional data (or modify the acquired point clouds) and then repeat the full reconstruction step, which can require from minutes to hours depending on the size of the model. The authors' own experience scanning numerous models using this tedious workflow motivated the development of the algorithm proposed in this work.

We propose a novel approach for online mesh reconstruction, which seamlessly integrates high resolution scanning and high quality reconstruction. The final reconstructed, semiregular, and textured model is computed on-the-fly as new geometry is acquired. Our algorithm produces results comparable to state-of-the-art methods that rely on expensive global optimization, while supporting online reconstruction with a cost that is approximately linear in the size of the acquired data. Directly working with the final reconstructed model in lieu of a preview offers major advantages over traditional pipelines:

(1) Artifacts or missing parts are immediately visible and highlighted with visual aids, guiding the user during the scanning process. Such visual feedback dramatically reduces the scanning time, as new scans can be placed only when actually needed
(2) Since our method generates semiregular medium resolution meshes, where hi-frequency details are efficiently stored as textures, it can achieve a faithful reconstruction using a geometry budget that is very small compared to the input point cloud size. Large-scale point clouds that cannot be rendered interactively pose no problems, as they are simply stored in virtual memory and paged when not in use—the only requirement is that the reconstruction *output* fits into GPU memory. This allows our method to remain interactive even when working with extremely large datasets on commodity hardware.
(3) The reconstructed model serves as a proxy for interactive editing tools that modify the original point clouds (e.g. smoothing). After every edit, the final reconstructed model is immediately computed and shown.
(4) The model the user sees at any moment is the final result of the entire scanning and reconstruction pipeline. This follows the WYSIWYG paradigm, which has never been applied to an online 3D scanning pipeline before.

We rely on a two-level representation to ensure that partial reconstructions can be viewed at interactive rates: the visible geometry consists of a relatively coarse polygonal mesh generated using the remeshing approach proposed in the Instant Field-Aligned Meshes technique (henceforth referred to as IM), while detail maps encode an offset surface and color data.

There are two challenges in adapting Instant Meshes (IM) to the context of online reconstruction. First, IM has poor locality: since it relies on global optimization techniques, even a single new data point could affect every location on the mesh, requiring a costly full rebuild of the entire output. Second, the hierarchical solver in IM requires a costly multi-resolution hierarchy which is invalidated when new scans stream in and adjacency relations change.

We address the first problem using an adaptive hierarchical error criterion that allows such large-scale changes to take place while dynamically limiting the amount of computation for localized small-scale changes. The second problem is tackled making the following observation with regards to the Laplace operator: While a "good" definition of the Laplacian requires an accurate notion of neighbors, distances, and angles [Dziuk 1988; Pinkall and Polthier 1993] the difference between a "good" Laplacian and a "bad one" is pronounced at higher frequencies. Having a "good" Laplacian is necessary for many geometry-processing applications. However, in our context, the Laplacian is only needed to solve diffusion systems for low-frequency solutions. This allows us to use an approximate Laplacian defined over an approximation of a k-nn graph — a data-structure that can be easily computed from a point cloud and efficiently updated to incorporate new scan data.

To validate our contribution, we integrate it with a commercial 3D scanner and a commercial optical tracking system, and use it to scan and reconstruct highly detailed and challenging models.

2 RELATED WORK

2.1 Offline Reconstruction

Reconstructing surfaces from scanned points is a well-studied problem in computer graphics [Berger et al. 2014]. In the offline setting, these approaches take as their input a set of points (possibly with normals) and output a manifold surface that interpolates/approximates the input. In general, these methods can be characterized as either *combinatorial* or *implicit*.

Combinatorial Algorithms. These approaches reconstruct the surface by triangulating (a subset of) the point samples. Typically, this is done by tetrahedralizing the points, marking the individual tetrahedra as either internal or external, and then setting the boundary faces to be the triangles of the reconstruction. Although the earliest methods used the local shape of the cells to label the tetrahedra [Amenta et al. 2001; Bernardini et al. 1999; Boissonnat and Oudot 2005; Dey and Goswami 2003; Edelsbrunner and Mücke 1994; Podolak and Rusinkiewicz 2005], labeling techniques using global approaches such as spectral [Kolluri et al. 2004], graph-cut [Hornung and Kobbelt 2006; Labatut et al. 2009], and winding-number [Jacobson et al. 2013] partitioning have also been used.

These combinatorial methods often have provable properties under appropriate sampling. However, they are inherently interpolatory and tend to reproduce the noise often present in scanner data. Our approach also directly uses the point samples, but it is more robust to noise and bad sampling since it relies on an underlying surface parametrization.

Implicit Functions. To be robust to scanner noise, implicit methods reconstruct an approximating surface by using the input points to define a function in 3D and then extracting the appropriate level-set using Marching Cubes [Kazhdan et al. 2007; Lorensen and Cline 1987]. Most often, these methods reconstruct the signed distance transform [Bajaj et al. 1995; Calakli and Taubin 2011; Carr et al. 2001; Curless and Levoy 1996; Hoppe et al. 1992; Mullen et al. 2010] or the indicator function [Kazhdan 2005; Kazhdan and Hoppe 2013; Manson et al. 2008]. They discretize the space of functions using radial basis functions, B-splines, or wavelets, and solve the associated global linear systems using hierarchical approaches like fast-multipole or multigrid.

While the above approaches often require solving a global linear system to compute the surface, a number of methods have been proposed that only require local scan information to define the value of the function [Fuhrmann and Goesele 2014; Ohtake et al. 2003]. These types of approaches tend to give a more efficient implementation at the cost of sacrificing some of the robustness provided by global methods. Our approach achieves a similar quality, but works directly on the point cloud: by sidestepping the use of an implicit function, we reduce memory consumption and computational cost, and we are able to seamlessly handle boundaries. In addition, the meshes produced by our method are highly isotropic and do not require remeshing.

Parameterization. Another representation that is useful for reconstruction is a parameterization defined on the input data, e.g. on multiple range scans [Pietroni et al. 2011]. They define a global parameterization using a common manifold domain with appropriate transition functions between the scans. Our approach also uses a parameterization of the input, but since our parameterization is local, we can achieve significantly higher performance while sacrificing only a small amount of parameterization regularity.

2.2 Online Reconstruction

With the advent of low-cost, high frame-rate, commodity scanners like the Microsoft Kinect sensor [Microsoft 2010], there has been a growing interest in online reconstruction. In this setting, scan data streams into the system as the user moves a scanner around an object (or within a scene) and the system integrates the data into an evolving surface representation that is displayed interactively, guiding the choice of location and orientation for subsequent scans.

To support such interactivity, existing methods [Newcombe et al. 2011; Rusinkiewicz et al. 2002] use a local surface representation, describing the reconstructed surface using a truncated signed distance function [Curless and Levoy 1996]. The local representation allows for parallelizable, space- and time-efficient updates of the surface representation that only require the subset of the model in the vicinity of the new scans to be changed. As a result, the cost of updates in such systems is proportional to the size of the new scan data, *not* the size of the entire reconstruction.

As with these earlier works, our goal is to support integration of new scan data into an evolving mesh interactively. However, in our work we use a hierarchical representation, presenting a new online method that simultaneously preserves the robustness of global approaches and maintains the efficiency of local solutions. Moreover, we directly produce a semi-regular, feature-aligned, and (optionally) quad-dominant mesh, whereas all previous online methods can only offer *irregular* triangle meshes. In the traditional online setup, regularity, feature-alignment, and conversion to quad-dominant meshes, which are recognized as necessary for many downstream applications [Bommes et al. 2012], can only be achieved through post-processing remeshing phases.

3 BRIEF OVERVIEW OF INSTANT MESHES

This section reviews key concepts of field-aligned parametrization methods [Bommes et al. 2012; Vaxman et al. 2016], with a particular focus on the Instant Field-Aligned Meshes approach [Jakob et al. 2015] (IM), upon which our algorithm builds. We restrict the discussion to the quad-dominant case, and we refer to the original paper for its extension to triangular meshes. In Section 4, we discuss specific changes required to adapt IM to the online setting.

The original IM algorithm takes as its input a point cloud or irregular mesh and a target edge length, and outputs a semi-regular quad-dominant mesh with approximately constant element size.

Fields. Field-aligned parametrization methods are based on the observation that assuming such a quadrangulation already exists, one can easily use it to derive a global parametrization whose gradients are aligned with the mesh's edges. This observation is then reversed to compute a field-aligned mesh by first computing the gradients of the parametrization, integrating them, and finally extracting the mesh from the parametrization.

IM implements this idea encoding the gradients with an *orientation (RoSy) field* [Palacios and Zhang 2007] that assigns a frame to each point, aligned with the edges of the containing quad (represented by a unit vector that is unique up to rotation by an integer multiple of $\pi/2$, Figure 2a). Instead of using a globally consistent parametrization, IM uses a *position (PoSy) field* to assign fractional coordinates to each point, giving its position within the containing quad (represented as an element of $\mathbb{R}^3$ that is unique up to translation along the integer lattice oriented with the directional frame and scaled by the target edge length, Figure 2b).

The original IM pipeline is decomposed into three steps: first, an orientation field is computed everywhere on the surface; then, the orientation field is used to define a position field; finally, a

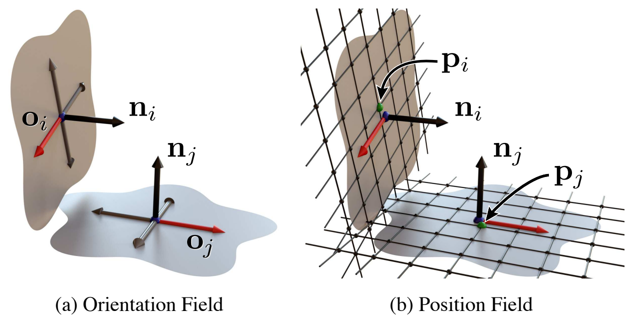


Fig. 2. Illustration of the fields used by the Instant Meshes technique. **(a)** Neighboring point cloud vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ representing surface observations (abstract shapes) with normals $\mathbf{n}_i$ and $\mathbf{n}_j$ store orientations $\mathbf{o}_i$ and $\mathbf{o}_j$ (red arrows) that control the alignment of the output mesh. The orientations satisfy a rotational symmetry (*RoSy*) condition, such that each direction is part of an equivalence class with three other elements (grey arrows); a hierarchical optimization scheme optimizes the field smoothness subject to this symmetry. **(b)** The position field controls the fractional offset (green dot) of a local parameterization (regular grid) that is aligned with the previously computed orientations and used to mesh the point cloud. A similar positional symmetry (*PoSy*) defines translated versions of the offset (black dots at grid intersections) are equivalent. The position step then optimizes the smoothness of the position field subject to this symmetry.

quadrangulation is extracted by clustering points whose positional field values fall into nearby positions to obtain the vertices of the output mesh, and then connecting clusters to obtain the edges.

Smoothing. Obtaining the fields requires solving a diffusion-like system. For the orientation field, if two points are adjacent, their associated frames should be similar. For the position field, if two points are adjacent, the parameter of one point should be close to the parameter of the other *plus* the the coordinates of the difference in positions, expressed in the frame of the orientation field.

IM uses similar approaches to solve these systems, subject to the respective symmetry conditions. The two fields are initially set to random values and iteratively optimized by means of a sequence of local smoothing operations. Each iteration recomputes the field value of a point based on its immediate neighbors analogous to explicit Laplacian smoothing [Taubin 1995]; accounting for the RoSy and PoSy symmetries entails an exhaustive search through the corresponding symmetry spaces. Neighbors are defined in terms of graph adjacency. When the input is a mesh, the graph is the connectivity graph of the mesh. When it is a point cloud, a k-nn graph is used.

Hierarchy. To perform the smoothing efficiently, IM relies on a simple multi-resolution hierarchy: the finest level corresponds to the input points, and each progressively coarser layer contains approximately half the number of points until only a single point is left. The fields are then optimized in a coarse-to-fine manner, and the optimization at each level is warm-started with the projected solution from the previous level. For topological reasons, and to maximize smoothness, both optimized fields may contain singular points where local connectivity deviates from that of a regular grid.

The IM algorithm has several useful properties that we exploit in our method: the most important is that all steps only rely on weak notions of neighborhood that allows them to work with both meshes and unstructured point clouds as input. In addition, the local nature of the individual smoothing operations is well-suited for streaming computation.

4 FIELD-ALIGNED SURFACE RECONSTRUCTION

In the following section, we explain the concepts that allow our reconstruction pipeline to support interactive, online reconstruction. Our pipeline takes as input a streaming set of scans of a 3D object and outputs a textured surface. We represent the surface as the combination of a coarse (semi-regular) quad or triangle mesh with an offset and color map associated with each face.

The core of our pipeline is a multi-resolution hierarchy, which stores the original point and field data as well as coarsened versions thereof, similar to the original IM pipeline. However, since the requirements for surface reconstruction are fundamentally different than those for remeshing, the design of our hierarchy deviates significantly from the IM hierarchy.

Overview. Every action in our system can be described as a modification of the original point data, including addition of new scans, re-registration, and point removal. For every modification, our pipeline executes the following steps to update the final mesh (Figure 3): First, we update the point data at the finest level of the hierarchy according to the type of modification (Section 4.2). Changes made during this update are then propagated to the coarser levels (Section 4.3). This fine-to-coarse pass is followed by a coarse-to-fine pass (Section 4.3), in which the direction and position fields are re-optimized. During this pass, the pipeline adaptively re-optimizes the fields, focusing the computation on regions in the finer level for which non-negligible modifications were observed in the corresponding region on the coarser level. This criterion generally localizes the amount of re-computation associated with new data while allowing large-scale changes (e.g. alignment to a sharp geometry feature) to propagate when needed. The result of the coarse-to-fine pass is the updated field and a set of points whose field values have changed significantly. We use this information to extract a part of the mesh from the changed region and merge it with the final mesh (Section 4.4). The pipeline concludes the mesh update by calculating detail maps for colors and displacement for the new region (Section 4.5). Overall, we designed all update steps so that they produce results that are similar to those obtained by discarding the previous reconstruction and re-calculating everything from scratch (Figure 4).

4.1 Approximate k-nn

Efficient computation of the neighborhood of a point is essential for optimization of both fields, so we require a data structure that supports efficient neighbor queries, as well as frequent updates. Most data structures that store point data either do not support these queries (e.g. point lists), are hard to update (e.g. balanced k-D trees), or exhibit bad data locality (e.g. hash grids), making them unsuitable for our purposes. In particular, we experimented with spatial hashing techniques, which allow for constant-time updates, but found that lack of data locality, exacerbated by the

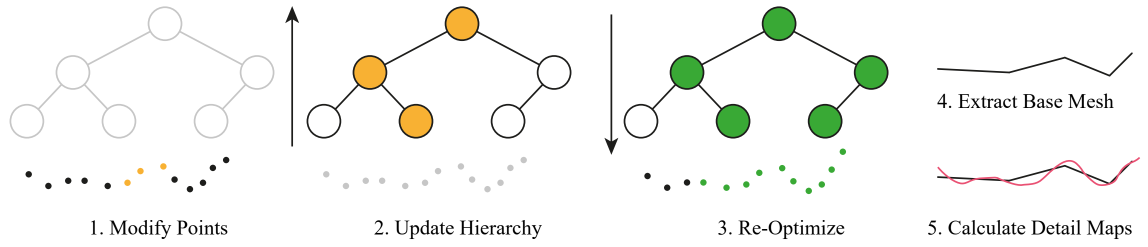


Fig. 3. Pipeline Overview

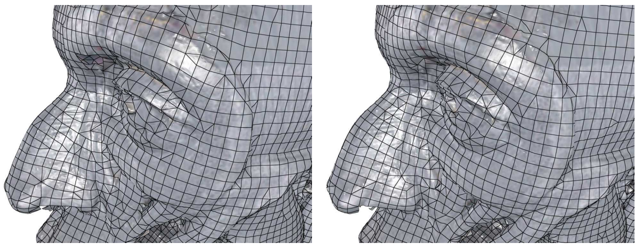


Fig. 4. Comparison of the final results obtained from successive addition of scans and a complete optimization of the entire point cloud. Although the base mesh's face layout exhibits minor differences, the geometries are indistinguishable.

need to perform multiple separate hash accesses for each query, downgraded performance and ultimately hindered the interactivity required for online reconstruction.

A key observation of our work is that the smoothing step in field optimization does not require the exact k-nn, allowing us to gain efficiency by using an approximate nearest neighbor representation.

We use a structure based on a combination of multiple Morton codes [Morton 1966], similar to [Li et al. 2012]. The Morton code of an n-dimensional integer vector is an easily computable integer that represents the position of the vector on the n-D z-order curve. Since nearby points tend to have nearby Morton code indices, sorting a set of points by their indices allows us to define an approximate notion of adjacency in terms of proximity within the sorted list. This is not perfect, however, as some nearby points may be far apart in the list. Since adding an offset s to each point's coordinate, re-encoding, and re-sorting generally leads to a different set of neighbors, we combine the information of multiple shifted z-curves to define an improved neighborhood relation that is fast to compute and suitable for online updates.

Shifted Grids. Four shifted grids and their corresponding z-order curves – each translated by an offset $s \in \mathbb{Z}^3$ (an integer multiple of the cell size) compared to the previous one – form the basis of our neighborhood data structure. The grids are represented as lists of points that are sorted by the corresponding cells' Morton codes (we use a notation that supports negative coordinates by including the inverted sign bit in the code). This representation provides fast queries and efficient updates. We add every point to all four grids, where the location in the list is determined by the difference of the Morton code of the point's quantized position and the respective grid's offset. We calculate the grid size g used for quantization in a way such that the average number of points in a cell is close to a target cell cardinality θ_c (15 in our implementation) when the first point set V is added. We start with an initial guess from the heuristic

$$g = 3 \cdot \max\left(\operatorname{diag}(V)\right) \sqrt[3]{\frac{\theta_c}{N(V)}}, \tag{1}$$

where $\operatorname{diag}(V)$ is the diagonal of the axis-aligned bounding box of V, $\max(\cdot)$ represents the maximum coefficient of a vector, and $N(V)$ is the number of points in V. We then refine this value by calculating the actual average cell cardinality c for the grid size g and update the initial guess g^* according to:

$$g^* = g \cdot \sqrt[3]{\frac{\theta_c}{c}} \tag{2}$$

In our implementation, we use $s = (5, 5, 5)^T$ cells because we found that this produces higher k-nn accuracy among a variety of data sets. We experimented with more shifted grids but found diminishing returns beyond four.

Implementation Notes. We store the original point data only once to conserve memory. New points are appended to the back of this contiguous vector and deletions create gaps that can be filled by other points later on. Adjacency information is defined indirectly using sorted lists that refer to points via their indices. Specifically, for every shifted grid, we store the permutation of the base list that generates the grid, where every entry is represented by the Morton code in the grid and an index into the original list. The rule for deletions ensures that point indices are preserved, which is important to avoid costly updates that touch the entire dataset.

Insertion. Adding a new set of points into this data entails sorting the new points according to their Morton codes and merging the result with the existing sorted list of points, which can be done very efficiently; this process must be repeated four times for each of the shifted grids. To further improve cache coherency, we also sort the point data that is appended to the base list by their Morton codes.

Query. To approximate the k-nearest neighbors of a given point, we locate the point in all the shifted grids using binary search over the permutation vectors. Note that since the first few elements accessed by the binary searches for close-by points are usually the same, they will stay in cache, leading to very fast lookups. After the point is located on a shifted grid, the m points immediately

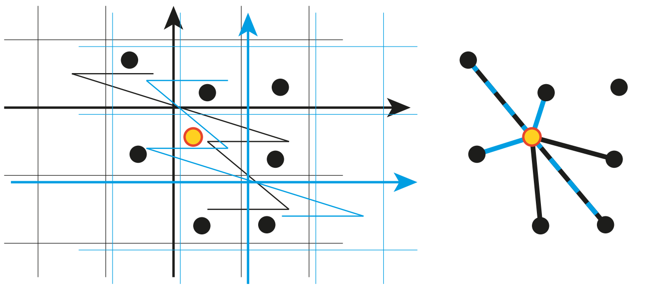
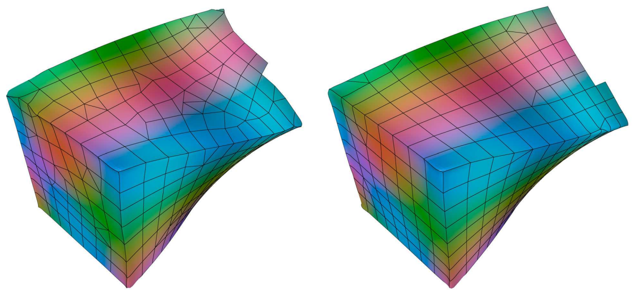


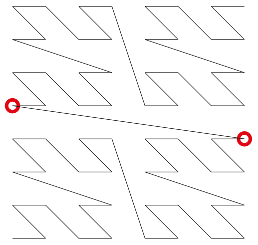
Fig. 5. A point cloud and a reference point (yellow) for which we seek neighbors. Two shifted grids (black and blue) with the relevant part of their z-order curve are superimposed. The two immediate neighbors on the curve are considered as adjacency candidates. The right figure shows the retrieved neighbor candidates (with color indicating the source z-curve).



(a) Optimization without hierarchy (b) Optimization using the hierarchy

Fig. 6. Comparison of the extraction results for fields that are optimized without a hierarchy (left) and with the hierarchy (right). Both optimizations use six iterations per level. Colors have been generated procedurally on the input point cloud.

preceding and following the point in the grid are considered as neighbor candidates if they do not exceed an upper distance limit. We set this upper limit, which is needed to filter out outliers, to the target edge length of the reconstruction. From the resulting set of candidates, we calculate the k nearest neighbors to find the final neighbor set (Figure 5). For our implementation, we used $k = 8$ and $m = 2$, resulting in a maximum candidate set size of $2 \cdot 2 \cdot 4 = 16$.



4.2 Multi-Resolution Hierarchy

The hierarchy's main purpose is to provide an initial guess for the finer levels during field optimization, where each level is optimized successively. Using a hierarchy allows us to limit the number of field optimization iterations per level to a small value (we use six). Figure 6 shows the importance of the hierarchy for a procedural data set. Hierarchy-less optimization is not able to produce a regular mesh even for these perfect data. While the meshing results near the features are adequate, the small number of iterations are not sufficient to propagate the information from the guiding edges to the face centers. Using the hierarchy avoids this problem because these guides can be efficiently propagated over long distances at coarser levels. Every node in the hierarchy stores geometry information (position and normal) and field values (directional and positional). Furthermore, nodes on the finest level also store color information.

To support neighbor queries for field optimization, every level of the hierarchy contains an instance of our neighbor data structure with original points and shifted grids. Since we are using Morton codes for the neighbor data structures on each level, an octree is a natural choice for the hierarchy because the coordinates of a cell's parent can be calculated efficiently with bitshift operations on the cell's Morton code. This allows us to keep the hierarchical relationships entirely implicit, which saves memory and improves memory coherency because access patterns are less random than with nodes linked by pointers. Except for the finest level, every cell at coarser levels stores the average of the data of its children. In total, our hierarchy requires about $130\frac{\text{bytes}}{\text{point}}$, whereas the original IM hierarchy (which stores neither colors nor references to mesh vertices) uses about $425\frac{\text{bytes}}{\text{point}}$.

All the operations for maintaining and updating the hierarchy explained above, including updates of the neighbor data structure, are very efficient, and we experimentally observed that the time spent on these operations is usually only about 1% of the total time of our pipeline (for details, refer to Figure 13).

4.3 Point Data Update and Field Re-Optimization

As points are added and removed from the input, we modify the representation at the finest level of the hierarchy and adjust the coarser resolutions by updating the average positions of points in the parent nodes. Storing the average at node n as the sum of positions and the sample count, (Σ_n, κ_n), the average of a parent node can be efficiently updated by adding the positions (and counts) of points inserted into the child and subtracting those removed. Specifically, if we denote by $p(n)$ the parent of node n, we get:

$$\left(\Sigma_{p(n)}, \kappa_{p(n)}\right) \leftarrow \left(\Sigma_{p(n)} + \Sigma_n^+ - \Sigma_n^-, \kappa_{p(n)} + \kappa_n^+ - \kappa_n^-\right) \tag{3}$$

$$\left(\Sigma_{p(n)}^+, \kappa_{p(n)}^+\right) \leftarrow \left(\Sigma_{p(n)}^+ + \Sigma_n^+, \kappa_{p(n)}^+ + \kappa_n^+\right) \tag{4}$$

$$\left(\Sigma_{p(n)}^-, \kappa_{p(n)}^-\right) \leftarrow \left(\Sigma_{p(n)}^- + \Sigma_n^-, \kappa_{p(n)}^- + \kappa_n^-\right) \tag{5}$$

where (Σ_n^+, κ_n^+) (resp. (Σ_n^-, κ_n^-)) is the sum and count of points added to (resp. removed from) node n.

As we update the averages in a fine-to-coarse manner, we add the visited nodes to a queue of nodes for which the directional and positional fields need to be updated. We then perform a subsequent coarse-to-fine pass to update the field values of queued nodes. Unfortunately, we cannot exclusively update the nodes visited when updating the averages as the prolongation of coarse solutions into finer children may require a modification of the fields in nearby regions as well. Failing to do so may result in suboptimal fields because the old field values pose a hard boundary constraint. We also cannot update *all* finer children, since this would require all nodes to be updated.

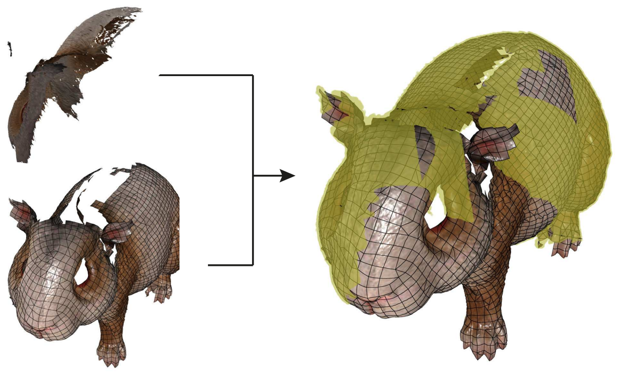


Fig. 7. Visualization of the re-optimized region (right part, shaded yellow) when adding a new scan (top left) to the current reconstruction (bottom left). The back of the guinea pig's left side is re-optimized to accommodate the new data while the left foreleg stays unaffected.

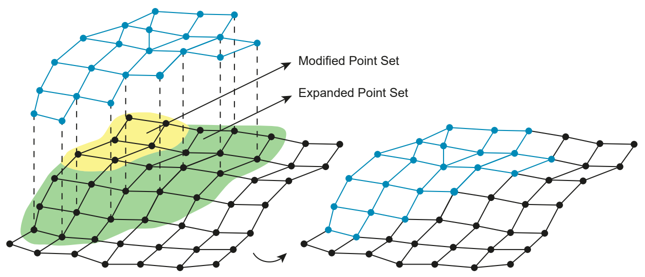


Fig. 8. Coarse mesh update. The set of modified points is grown to yield an expanded point set, from which a partial mesh is extracted. This partial overlay mesh is then integrated into the old mesh, replacing overlapping areas.

Instead, we track the extent to which the fields change as a result of the update step. More specifically, given the old and new field values o_{old}, o_{new}, p_{old}, and p_{new}, we calculate the changes as:

$$\begin{aligned} \Delta_o &= \|o_{old} - o_{new}\| \\ \Delta_p &= \frac{1}{\ell}\,\|p_{old} - p_{new}\|\,, \end{aligned} \qquad (6)$$

where ℓ is the target edge length. If one of these changes is above a prescribed threshold, we add all of the nodes' children to the queue for processing the next finer level. We achieve good results with the thresholds 0.1 for both fields on the finest level. The thresholds are doubled successively at each coarser level to account for the coarser resolution . In practice, this approach usually leads to re-optimization of only the area surrounding the modified points. However, in cases where there are no features in the input that guide the alignment field, new data may cause re-optimization of large parts of the model. Figure 7 shows the updated regions for an example data set.

4.4 Coarse Mesh Update

The locally-updated fields contain all the information required to extract the coarse mesh. As in previous steps, we want to process only the newly modified data to avoid costly passes over the entire point cloud. To extract the mesh locally, we propose a mesh merging procedure guided by the position field values — the key idea is to exploit the fact that in the untouched regions, the position field averages used by the IM extraction will lead to identical coordinates for the final mesh vertices. We thus identify the modified regions, grow them to capture a small strip of untouched elements, mesh the extended region, and zip the meshes together exploiting the vertex correspondences in the overlapping strip (Figure 8).

Active Region. Expansion is performed in two steps. First, we identify points on the boundary of the changed set. Then, we find the points that are outside of the changed set but within a fixed radius of this boundary. Using the nearest-neighbor graph (previously computed for updating the field values), we define the boundary of the changed set by identifying those points that are not in the changed set but which are neighbors of points that are.

In expanding the boundary, our goal is to find all points outside of the changed set whose associated mesh vertices can share a face with the mesh vertices associated to points inside the change set. We do this using the target edge length of IM as a heuristic. Specifically, assuming that all edges have a length equal to the target edge length, the distance between two points that define vertices sharing a quad is at most twice the target diagonal length (one diagonal inside the quad and another diagonal to account for the correspondences as the distance between a point and its positional field value is by definition at most half the target diagonal length). Since the optimization and clustering can cause slight deviations from this rigid scheme, we add a small margin and use the final heuristic $r = 3 \cdot \ell$, where ℓ is the target edge length. A similar argument shows that $r = 3 \cdot \ell$ suffices for triangle faces as well. Thus, by growing the modified set by a radius of $3 \cdot \ell$, we identify all points whose associated mesh vertices are within a one-ring neighborhood of the mesh vertices associated with boundary points (and possibly more). We refer to the set of added points as the *expanded points*.

Extraction. An *overlay mesh* is then extracted using the IM graph-based collapsing approach from the active region. We construct a local graph covering the active region, where every point generates a vertex at the location of its positional field value and edges are added according to the original point cloud's adjacency graph — since this graph will in general not be symmetric, we symmetrize it by adding the missing edges that are needed by the mesh extraction of IM. By construction, this graph forms clusters, which are then collapsed into a single node using length-based edge collapse. For every node in the collapsed graph, a mesh vertex is generated and faces are determined using a greedy search over the adjacency graph. Our pipeline supports triangles and quadrilaterals. We tessellate faces of higher degree as well as non-planar quads.

Stitching. The first stitching step deletes all vertices of the old mesh that have been previously created by points in the changed set. This generates a *punctured mesh* with a hole in the area of the changed set. In our representation, every point stores a reference to its associated mesh vertex. Thus, deleting mesh geometry during

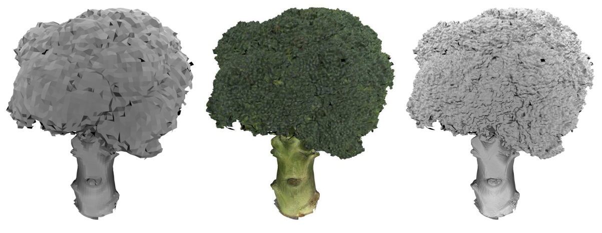


Fig. 9. High-frequency details are added to the coarse base mesh by the color and displacement maps.

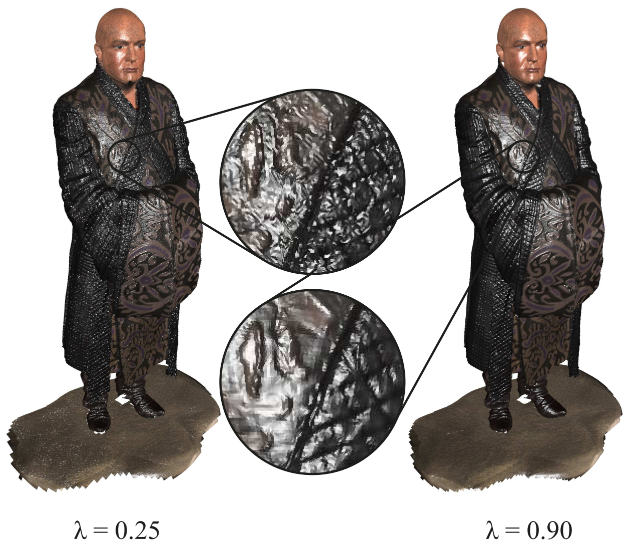


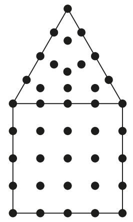
Fig. 10. Influence of the smoothness parameter λ on the extracted surface

the update is a matter of iterating over points in the changed set and removing the associated mesh vertices (and incident edges and faces). Since, by construction, vertices generated solely from the expanded points are duplicated in the punctured mesh, it is possible to glue the overlay mesh to it by merging the duplicate vertices.

Special care must be taken of vertices that are formed by points close to the boundary of the active region. The original clusters for those vertices may include points that are outside of the active region and averaging the positional field values of a partial cluster results in a different vertex position in general, preventing duplicate detection. To avoid this, we identify vertices formed from partial clusters with a breadth-first search on the overlay mesh starting from the vertices generated by the (unexpanded) changed set. The BFS proceeds until it reaches a duplicate vertex with respect to the punctured mesh and excludes every vertex beyond this point. The resulting front of duplicate vertices is sufficient to stitch the punctured mesh with the overlay mesh and we delete every vertex that has not been reached by the BFS as these vertices may be formed by partial clusters.

4.5 Detail Map Calculation

The final step of our reconstruction pipeline adds details in the form of a color and displacement map to the coarse mesh (Figure 9). The displacement map is a scalar height field over the mesh that displaces the surface in the direction of the interpolated vertex normals. We use linear and bilinear interpolation for triangles and quads, respectively. The vertex normals are calculated as the area-weighted average of incident face normals on the coarse mesh. As in previous steps, detail map calculation is only performed in regions of the mesh that have changed, i.e. on new faces that are generated during the creation of the coarse mesh.



Local Parametrization. Since a global UV parametrization is not necessary for our purposes, we opt for a local, per-face parametrization that allows us to store these data together with the geometry, similar to Mesh Colors [Yuksel et al. 2010]. We use the texel layout that is used by the GPU tessellation unit (see inset figure) to allow efficient rendering of the high-resolution mesh, similar to [Schäfer et al. 2013]. Using this representation, every face, edge, and vertex defines a set of texels storing the color and offset information, where texels of edges and vertices are shared by multiple faces.

Filtering. The texel values are defined so that the resulting tessellated mesh is at once faithful to the input points and robust to scanner noise. We achieve this by solving a linear least-squares system combining interpolation and regularization terms:

$$\arg\min_{\vec{t}} \left((1-\lambda) \cdot I(\vec{t}) \; + \; \lambda \cdot R(\vec{t}) \right) \tag{7}$$

Here, $\vec{t}$ is the vector containing texel data, $I(\vec{t})$ is the data fidelity term, and $R(\vec{t})$ measures the smoothness. The weight parameter $\lambda \in [0,1]$ is used to balance between the two terms. Figure 10 compares the reconstructed surface for two choices of λ. While small values reconstruct the input points more closely, larger values can reduce noise significantly. We perform all color calculations in *CIE La*b** space in order to measure smoothness and color similarity in a way that aligns with human perception.

Regularization Term. We use the bi-Laplacian to define the regularizer, setting:

$$R(\vec{t}) := \left\| L\vec{t} \right\|^2 \tag{8}$$

where L is the uniform Laplacian matrix for the tessellated mesh where every texel corresponds to one vertex. Note that as we are only regularizing for the smoothness of the normal offset, *not* the 3D displacement: this formulation will preserve creases in the input data as soon as the field optimization step aligns the edges of the coarse mesh to them (Figure 6). This formulation can result in a visible grid pattern on the reconstructed surface that reveals the underlying coarse mesh, especially for large smoothness values λ. To remedy this, we replace rows of the linear system for the displacement map that correspond to texels on non-crease edges (which we determine by a user-specified threshold on the dihedral angle) with a geometric

Laplacian, i.e. for the current texel i and its neighbors $N(i)$:

$$(p_i + n_i \cdot t_i) - \frac{1}{|N(i)|} \sum_{j \in N(i)} (p_j + n_j \cdot t_j) = 0, \tag{9}$$

where p_i, n_i, and t_i are the interpolated position, normal and the displacement of texel i, respectively.

Projection. To measure data fidelity, we project every point in the changed set onto the coarse mesh in the direction of the interpolated vertex normal, as proposed in [Kobbelt et al. 1999]. We do this efficiently, by only projecting a point onto the faces of the coarse mesh that are incident to the associated vertex.

We then find the projection of a point p by solving for the face f_p and bilinear coordinates (resp. barycentric coordinates for triangles), α_p, such that the vertex projects on the face at the interpolated point using the interpolated normal:

$$\big(p - \pi_f(\alpha)\big) \times n_f(\alpha) = 0, \tag{10}$$

where $\pi_f(\alpha)$ is the linearly (resp. bilinearly) interpolated position within the triangle (resp. quad) f at coordinates α and $n_f(\alpha)$ is the interpolated normal. For each face, we solve the non-linear equation using Newton iterations, discarding modified points from the interpolation constraints when they do not project inside any incident face. If there are multiple projections, we use the one that results in the smallest offset, measured as

$$\frac{\big\langle p - \pi_f(\alpha), n_f(\alpha)\big\rangle}{\|n_f\|^2}. \tag{11}$$

Data Term. Using the projection, we define the interpolation penalty in terms of the deviation of the point's attribute (color and offset) from the attribute obtained by sampling the texture map at the projected position:

$$I(\vec{t}) = \sum_{p \in P} \big(\vec{t}(f_p, \alpha_p) - a_p\big)^2, \tag{12}$$

where P is the set of modified points, $\vec{t}(f, \alpha)$ is the evaluation of the texture map at face f and coordinates α, and a_p is the attribute associated to point p.

We solve the sparse linear system in Equation (7) using Conjugate Gradients, locking the texel values of unmodified faces to define Dirichlet boundary constraints, and using as initial guess the point's attribute that is closest to a texel. In our experiments, the solver converges quickly, usually in less than 30 iterations. The final texel values are then uploaded to the GPU together with the coarse mesh and are rendered using the GPU's tessellation unit.

5 RESULTS

We use an *HP 3D Structured Light Scanner Pro S3* and an automatic turntable to acquire range scans as input for our pipeline. To provide a coarse registration without further user interaction, we attached an *HTC Vive* controller to the scanner rig whose orientation and position in a reference coordinate system can be tracked accurately (Figure 11). We use a workstation with a 6-core i7 processor clocked at 3.5 GHz to run all our experiments.

We use a semi-automatic calibration process to determine all relevant parameters of this system, which allows us to place acquired 3D scans in the Vive's reference coordinate system. The details of

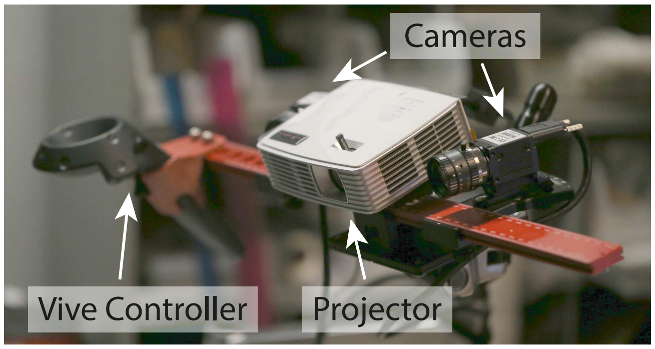


Fig. 11. Hardware setup

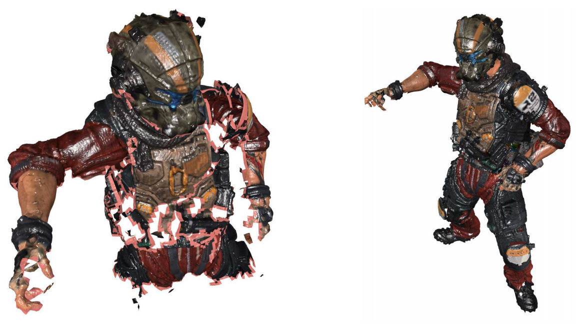
	Points	Coarse Faces	Fine Vertices
Eagle	796,825	21,296	1,955,789
King	1,876,034	17,440	1,549,559
Guinea Pig	2,532,694	5,019	472,602
Head (quads)	3,165,119	16,058	1,540,950
Head (tris)	3,165,119	33,539	2,526,406
Broccoli	3,433,542	9,406	832,670
Monk	5,661,497	9,441	907,846
Soldier	6,690,187	8,887	793,119

Table 1. Reconstruction input and output statistics

this calibration can be found in the supplementary material. Since small errors in the tracked controller orientation can lead to relatively large offsets in the scanned area (especially for small models), we provide an optional two-click coarse registration tool that lets the user specify one correspondence from which a correcting translation is calculated. The coarse registration obtained in this way (with or without user corrections) provides a good initial guess for a subsequent fine registration, for which we use Sparse ICP [Bouaziz et al. 2013] with the point-to-plane formulation. Table 1 shows statistics for all presented data sets, including the numbers of input points, extracted base mesh faces, and vertices in the tessellated mesh. (Our implementation and selected data sets are available at https://github.com/NSchertler/OnlineSurfaceReconstruction.)

Visual Aids. Our system highlights the boundaries of the reconstructed model after each scan is integrated (Figure 12, left), suggesting where data is required and helping to plan the next scan. While more advanced next-best-view optimizations could be integrated [Fan et al. 2016; Wu et al. 2014], we found the guidance provided by our direct visual feedback to be sufficient for all our experiments. We efficiently scanned models with complex shapes, combining an initial set of automatic scans (taken using a rotational stage), with a few manual scans of the occluded regions (Figure 12, right).

Normal Filtering. The point normals that are used for field optimization play an important role in the final result. Directly using the normals of the point clouds leads to noisy directional and positional fields in regions with high-frequency details, which breaks the extraction process in some places (Figure 14a.) We address this problem applying a smoothing filter with Gaussian weights to the



(a) Partial result with highlighted boundary (b) Final result

Fig. 12. Boundaries of the extraction result are highlighted with a flashing border during the scan session in order to help locate the next scan.

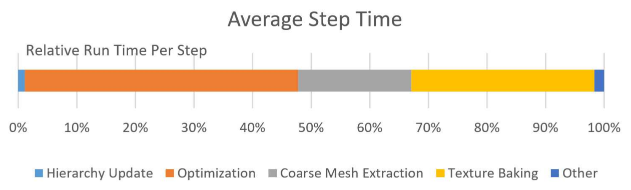
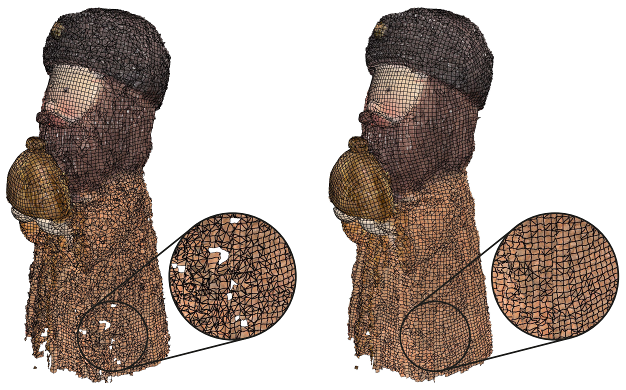


Fig. 13. Average relative time for each step of our pipeline

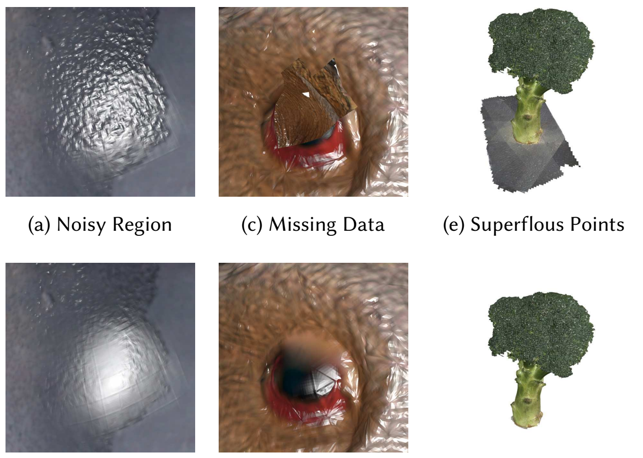


(a) Unmodified Normals (b) Cleaned with a Gaussian filter

Fig. 14. Comparison of the extraction result using normals calculated as the average of incident faces in the range scan (left) and after a cleaning pass using Gaussian filtering (right).

normals after integrating a scan into the hierarchy, using our approximate neighbor definition. We couple the variance of the Gaussian to the target edge length ℓ, setting $\sigma = 0.1\ell$. This smoothing produces much cleaner fields which results in a more regular mesh (cf. Figure 14b).

Direct Point Cloud Editing. The ability to update the reconstructed surface at interactive rates enables a user to correct problems in the scans (e.g. outliers or holes), with immediate visual feedback of the final reconstruction.



(a) Noisy Region (c) Missing Data (e) Superflous Points

(b) Smoothing Brush (d) Fill-Holes Brush (f) Delete-Points Brush

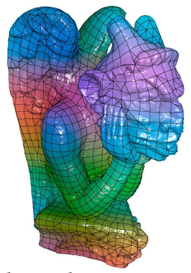
Fig. 15. Usage of interactive point cloud editing tools to resolve several problems in the scans

We implemented three different field-aligned brushes to showcase this feature (Figure 15).

The *smoothing brush* takes advantage of the hierarchy's efficient neighbor queries and applies a Gaussian filter to the point positions (Figure 15, b). Optionally, anisotropic smoothing can be applied, picking one of the directions of the directional field and specifying the filter strength in each of the three local dimensions, i.e. chosen direction, perpendicular tangential direction, and normal (see supplementary material for more details). For example, the region in Figure 15 was smoothed only along the normal direction.

The *fill-holes brush* reconstructs missing data in the scans. For example, we used it in our experiments to complete the eye of the guinea pig model, where the scanner failed to capture samples due to high specularity (Figure 15, d). The brush works in four stages: (1) the user marks a support region in the point cloud, where the reconstructed surface serves as a proxy for the 3D selection tool; (2) we fit a plane to this region using PCA and project the points of the support region onto this plane; (3) this produces a sampling of a height field $h : \mathbb{R}^2 \to \mathbb{R}$ over the plane, which we reconstruct using a thin plate spline; (4) the user can interactively sample points from this height field to fill the hole, until satisfied with the result.

Finally, the *remove-points brush* deletes undesired points in a spherical region around the 3D cursor (Figure 15, f).



Data Source. Our pipeline is not specific to our hardware setup, and can be used to process any range scan or point cloud. The inset shows a dataset acquired with a laser scanner and registered externally. The colors are procedural and depend on the vertex coordinates.

Comparison with IM. The original IM pipeline does not support updates of its underlying data structures. Therefore, if this pipeline were to be used, every new scan would initiate an entire rebuild of the hierarchy and extracted mesh. Our

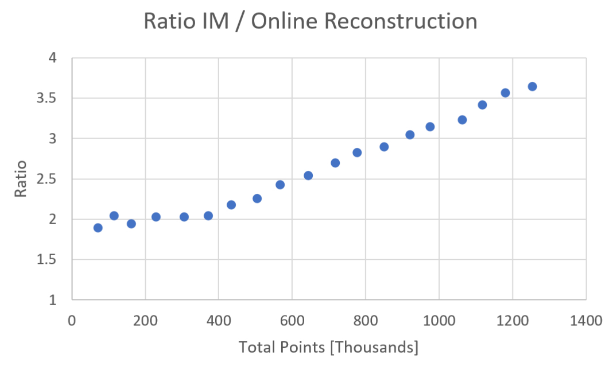


Fig. 16. Ratio of the total running times of IM and our pipeline for successive addition of new scans (Gargoyle dataset).

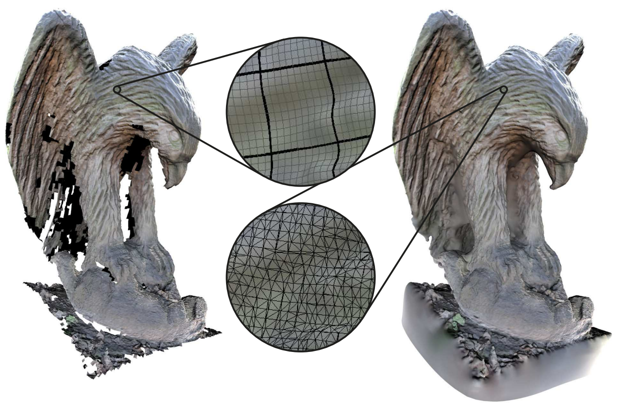


Fig. 17. Comparison of the extraction results of our method (left) and Screened Poisson Reconstruction (right) [data courtesy of LGG, EPFL, http://lgg.epfl.ch/statues.php].

pipeline updates only as much data as required. As a result, as more scans are added, the total runtime of the IM pipeline increases approximately quadratically. Figure 16 visualizes the ratio of runtimes of the IM pipeline and our solution. The increasing trend clearly shows that the IM pipeline becomes less efficient as more scans are added.

Comparison with Screened Poisson Reconstruction. In Figure 17, we compare our method with Screened Poisson Reconstruction (SPR) on the dataset used for Figure 8 of [Kazhdan and Hoppe 2013]. We used the reference implementation provided by the authors, and manually adjusted the resolution and smoothing parameter of both methods to produce a mesh with similar density and surface details. SPR took 39.5 seconds and required 910.4 MB to reconstruct the 800k points data set at the given resolution on a six-core Intel Core i7 machine, using all cores. Our pipeline outperforms SPR both in terms of running time, requiring 12.4 seconds, and peak memory usage (639.8 MB). Visually, the results are very similar with the main difference being that SPR fills holes where no data is present, while our method does not, introducing boundaries in the reconstructed mesh. The meshing pattern of our result is highly isotropic and semi-regular, and it is not plagued by the irregularity and sliver elements of the Marching Cubes step used by SPR.

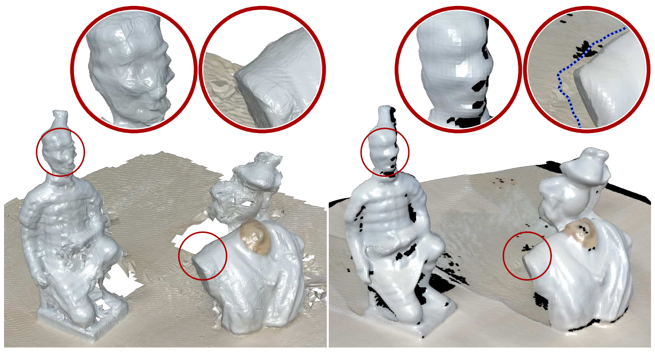


Fig. 19. Comparison of the extraction results of our method (left) and Kinect Fusion (right), computed from noisy range scanned data captured from a Kinect device. Results of Kinect Fusion are smoothed out, resulting in loss of detail (first close-up) and shrinkage (second close-up, where the silhouette of our result is superimposed as a dotted blue line, to highlight differences). Black regions in the Kinect Fusions results correspond to areas which were not visible to the scanner and for which color data is not available.

Figure 18 provides a more quantitative comparison with SPR using the benchmark of Berger *et al.* [2013]. The results were obtained by reconstructing surfaces from 240 virtual scans of 5 models (Anchor, Dancing Children, Daratech, Gargoyle, and Lord Quasimodo). The figure shows the ratio of running times (left), positional accuracy (center), and normal accuracy (right) of our method relative to SPR, and confirms that our method produces reconstructions more efficiently without sacrificing geometric quality. (We ran SPR at depth 9 to produce surfaces with resolution comparable to ours. We measured positional and normal accuracy using reconstruction-to-ground-truth errors in order to avoid bias due to reconstruction holes in regions that were not visible to the virtual scanners.)

Comparison with Kinect Fusion. In order to compare our pipeline with the online reconstruction of [Newcombe et al. 2011], we performed a scanning session with the Kinect Fusion implementation provided in the Microsoft Kinect SDK, capturing 800 frames at 30 fps, and reconstructed the triangle mesh from the acquired volumetric representation. We used every tenth captured depth map, which the Kinect Fusion system already smoothed with a bilateral filter, as input for our pipeline. We ran fine registration and extraction with a target edge length that approximately matches those in the Fusion reconstruction and a smoothness of $\lambda = 0.98$. Both results are shown in Figure 19. Kinect Fusion results are smoothed out, resulting in loss of detail and shrinkage. This aggressive smoothing is required to avoid artefacts caused by noisy input data and registration errors from the utilized SLAM (e.g. table surface). In addition, during the scanning section, Kinect Fusion results are only available as ray-traced distance fields, whereas our system offers a "final" on-the-fly feedback consisting of a quad-dominant, semiregular, field-aligned, displacement-mapped, explicit mesh representation.

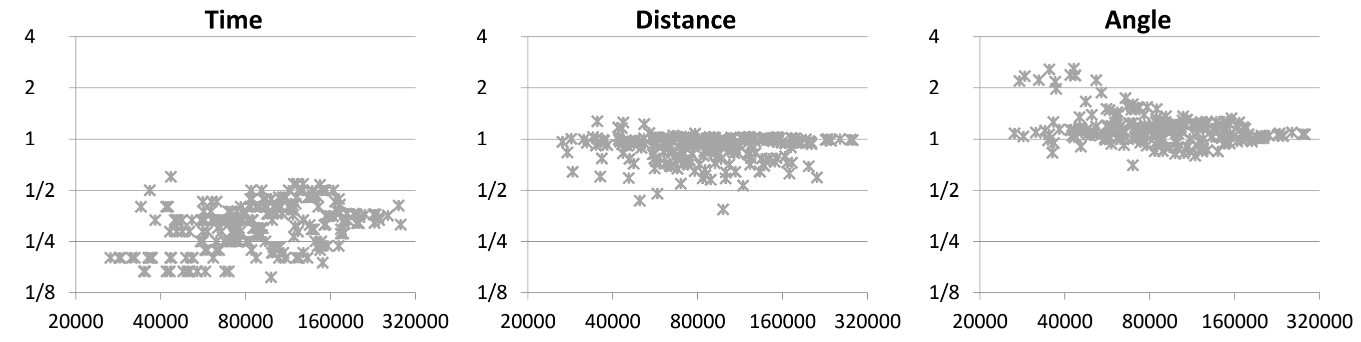


Fig. 18. Ratios of the running time (left), positional accuracy (center), and normal accuracy (right) of our method relative to SPR, obtained across 240 virtual scans, given as a function of the number of points in the scan.

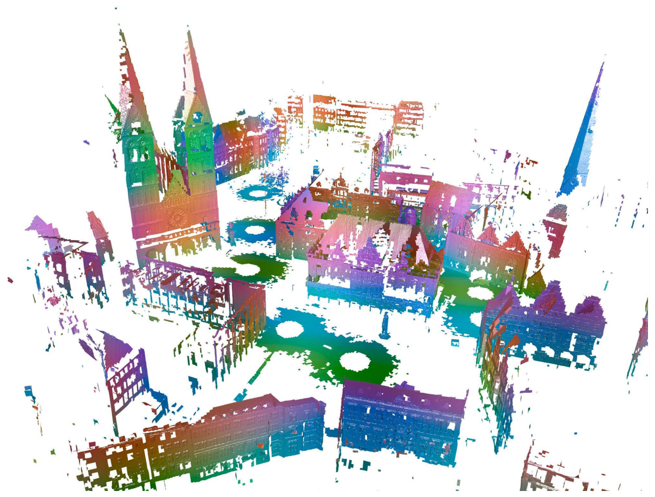


Fig. 20. Reconstruction of the large Bremen data set.

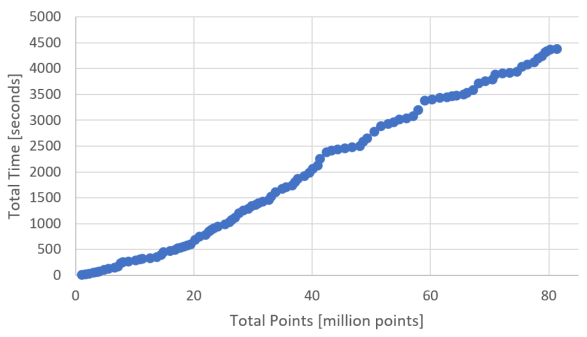


Fig. 21. Performance recording for integrating each of the 99 scans of the Bremen data set on an Intel Core i7 machine.

Large Dataset. The approximately linear cost of our reconstruction pipeline makes it ideal for large datasets. We show the reconstruction result for the *Bremen* data set [Borrmann and Nüchter 2016] in Figure 20. The dataset is comprised of 99 registered scans acquired by a laser scanner, for a total of about 80 million points. The approximate linear cost is clearly visible in Figure 21, where we plot the cumulative time as more and more range scans are integrated in the reconstruction.

6 LIMITATIONS AND CONCLUDING REMARKS

We presented the first online algorithm to convert range scans and point clouds to semi-regular, coarse, feature-aligned meshes. Our results are equipped with a local parametrization, which is used for generating color and displacement maps.

Similar to the original IM algorithm, our method is not guaranteed to produce manifold output, (though we have found non-manifold output to be rare in our experiments). Heavy undersampling can produce undesired holes approximately of the size of the target edge length: while this can be addressed by taking another scan or using our hole-filling brush, it would be interesting to combine our approach with the indicator function of a local Poisson reconstruction to automatically insert additional points and solve these problems.

Moderate levels of zero-mean noise with a standard deviation that is smaller than the target edge length can be handled well through our detail map generation. More severe noise can cause the base mesh generation to break, but basic filtering techniques can help significantly to overcome this problem as we have shown in Figure 14 and Figure 19. Registration errors can lead to noisy surfaces since their errors have non-zero mean. Outliers usually do not pose a challenge for our pipeline because these are already filtered out during base mesh generation – a feature that we adopted from the underlying IM framework.

Our results are made possible by leveraging the recent advancements in field-aligned parametrization and extending them, for the first time, to an online setting. We believe that this algorithm is useful in contexts other than range scanning, such as reconstruction of time varying datasets or interactive modeling sessions, where the data is represented as an implicit surface or a CSG tree. We plan to explore this direction in future work.

REFERENCES

N. Amenta, S. Choi, and R. Kolluri. 2001. Power Crust. In *ACM Symposium on Solid Modeling and Applications*. 249–260.

C. Bajaj, F. Bernardini, and G. Xu. 1995. Automatic Reconstruction of Surfaces and Scalar Fields from 3D Scans. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1995)*. 109–18.

Matthew Berger, Joshua A. Levine, Luis Gustavo Nonato, Gabriel Taubin, and Claudio T. Silva. 2013. A Benchmark for Surface Reconstruction. *ACM Transactions on Graphics* 32, 2 (2013), 20:1–20:17.

Matthew Berger, Andrea Tagliasacchi, Lee M. Seversky, Pierre Alliez, Joshua A. Levine, Andrei Sharf, and Claudio T. Silva. 2014. State of the Art in Surface Reconstruction from Point Clouds. In *Eurographics 2014 - State of the Art Reports*, Sylvain Lefebvre and Michela Spagnuolo (Eds.). The Eurographics Association. DOI:https://doi.org/10.2312/egst.20141040

F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. 1999. The Ball-Pivoting Algorithm for Surface Reconstruction. *IEEE Transactions on Visualization and Computer Graphics* 5 (1999), 349–359.

J.D. Boissonnat and S. Oudot. 2005. Provably good sampling and meshing of surfaces. *Graphical Models* 67 (2005), 405–451.

D. Bommes, B. LÃľvy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, and D. Zorin. 2012. State of the Art in Quad Meshing. In *Eurographics STARS*.

Dorit Borrmann and Andreas Nüchter. 2016. Robotic 3D Scan Repository. http://kos.informatik.uni-osnabrueck.de/3Dscans. (2016). http://kos.informatik.uni-osnabrueck.de/3Dscans

Sofien Bouaziz, Andrea Tagliasacchi, and Mark Pauly. 2013. Sparse iterative closest point. In *Computer graphics forum*, Vol. 32. Wiley Online Library, 113–123.

F. Calakli and G. Taubin. 2011. SSD: Smooth Signed Distance Surface Reconstruction. *Computer Graphics Forum* 30 (2011), 1993–2002.

J. Carr, R. Beatson, H. Cherrie, T. Mitchell, W. Fright, B. McCallum, and T. Evans. 2001. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 2001)*. 67–76.

P Cignoni, G Ranzuglia, M Callieri, M Corsini, F Ganovelli, N Pietroni, and M Tarini. 2011. MeshLab. http://www.meshlab.org/. (2011).

B. Curless and M. Levoy. 1996. A Volumetric Method for Building Complex Models from Range Images. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1996)*. 303–312.

T. Dey and S. Goswami. 2003. Tight Cocone: A Water-tight Surface Reconstructor. In *Proceedings of the Symposium on Solid Modeling and Applications*. 127–134.

G. Dziuk. 1988. Finite elements for the Beltrami operator on arbitrary surfaces. In *Partial Differential Equations and Calculus of Variations, Lecture Notes in Mathematics*. Vol. 1357. 142–155.

H. Edelsbrunner and E. Mücke. 1994. Three-dimensional Alpha Shapes. *ACM Transactions on Graphics* 13 (1994), 43–72.

S. Fuhrmann and M. Goesele. 2014. Floating Scale Surface Reconstruction. *ACM Transactions on Graphics* 33 (2014), 46:1–46:11.

H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. 1992. Surface Reconstruction from unorganized points. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1992)*. 71–78.

A. Hornung and L. Kobbelt. 2006. Robust reconstruction of watertight 3D models from non-uniformly sampled point clouds without normal information. In *Symposium on Geometry Processing*. 41–50.

Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and others. 2011. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 559–568.

Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust Inside-Outside Segmentation using Generalized Winding Numbers. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)* 32, 4 (2013), 33:1–33:12.

Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Instant Field-Aligned Meshes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA)* 34, 6 (Nov. 2015). DOI:https://doi.org/10.1145/2816795.2818078

Michael Kazhdan. 2005. Reconstruction of Solid Models from Oriented Point Sets. In *Proc. of the 3rd Eurographics Symp. on Geometry Processing (SGP '05)*. Eurographics Association, Article 73. http://dl.acm.org/citation.cfm?id=1281920.1281931

Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson Surface Reconstruction. (2006), 61–70. http://dl.acm.org/citation.cfm?id=1281957.1281965

M. Kazhdan and H. Hoppe. 2013. Screened Poisson surface reconstruction. *ACM Transactions on Graphics* 32 (2013), 29:1–29:13.

M. Kazhdan, A. Klein, K. Dalal, and H. Hoppe. 2007. Unconstrained Isosurface Extraction on Arbitrary Octrees. In *Symposium on Geometry Processing*. 125–133.

Leif Kobbelt, Jens Vorsatz, and Hans-Peter Seidel. 1999. Multiresolution Hierarchies on Unstructured Triangle Meshes. *Comput. Geom. Theory Appl.* 14, 1-3 (Nov. 1999), 5–24. DOI:https://doi.org/10.1016/S0925-7721(99)00032-2

R. Kolluri, J. Shewchuk, and J. O'Brien. 2004. Spectral Surface Reconstruction From Noise Point Clouds. In *Symposium on Geometry Processing*. 11–21.

P. Labatut, J.-P. Pons, and R. Keriven. 2009. Robust and efficient surface reconstruction from range data. *Computer Graphics Forum* 28 (2009), 2275–2290.

Shengren Li, Lance Simons, Jagadeesh Bhaskar Pakaravoor, Fatemeh Abbasinejad, John D Owens, and Nina Amenta. 2012. kANN on the GPU with shifted sorting. In *Proc. of the 4th ACM SIGGRAPH/Eurographics conf. on High-Performance Graphics*. Eurographics Association, 39–47.

W. Lorensen and H. Cline. 1987. Marching Cubes: A High Resolution 3D Surface Reconstruction Algorithm. In *Computer Graphics (Proceedings of SIGGRAPH 87)*. 163–169.

J. Manson, G. Petrova, and S. Schaefer. 2008. Streaming surface reconstruction using wavelets. In *Symposium on Geometry Processing*. 1411–1420.

Microsoft. 2010. Kinect. https://developer.microsoft.com/en-us/windows/kinect. (2010).

Guy M Morton. 1966. *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company New York.

P. Mullen, F. De Goes, M. Desbrun, D. Cohen-Steiner, and P. Alliez. 2010. Signing the Unsigned: Robust Surface Reconstruction from Raw Pointsets. *Computer Graphics Forum* 29 (2010), 1733–1741.

Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. 2011. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *Proceedings of IEEE ISMAR - 10th International Symposium on Mixed and Augmented Reality*. IEEE, 127–136.

Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H. Seidel. 2003. Multi-level partition of unity implicits. *ACM Transactions on Graphics* 22 (2003), 463–470.

Nico Pietroni, Marco Tarini, Olga Sorkine, and Denis Zorin. 2011. Global parametrization of range image sets. In *ACM Transactions on Graphics (TOG)*, Vol. 30. ACM, 149.

U. Pinkall and K. Polthier. 1993. Computing Discrete Minimal Surfaces and Their Conjugates. *Experimental Mathematics* 2 (1993), 15–36.

J. Podolak and S. Rusinkiewicz. 2005. Atomic volumes for mesh completion. In *Symposium on Geometry Processing*. 33–41.

Szymon Rusinkiewicz, Olaf Hall-Holt, and Marc Levoy. 2002. Real-time 3D Model Acquisition. *ACM Trans. Graph.* 21 (2002), 438–446.

Szymon Rusinkiewicz and Marc Levoy. 2000. QSplat: A Multiresolution Point Rendering System for Large Meshes. In *Proceedings of ACM SIGGRAPH 2000*. 343–352.

Henry Schäfer, Magdalena Prus, Quirin Meyer, Jochen Süßmuth, and Marc Stamminger. 2013. Multiresolution Attributes for Hardware Tessellated Objects. *IEEE transactions on visualization and computer graphics* 19, 9 (2013), 1488–1498.

Gabriel Taubin. 1995. A Signal Processing Approach to Fair Surface Design. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*. ACM, New York, NY, USA, 351–358. DOI:https://doi.org/10.1145/218380.218473

Xinyi Fan, Linguang Zhang, Benedict Brown, and Szymon Rusinkiewicz. 2016. Automated View and Path Planning for Scalable Multi-Object 3D Scanning. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 35, 6 (nov 2016).

Jonathan Palacios and Eugene Zhang. 2007. Rotational Symmetry Field Design on Surfaces. *ACM Trans. Graph. (SIGGRAPH 2007)* 26, 3, Article 55 (jul 2007). DOI:https://doi.org/10.1145/1276377.1276446

Shihao Wu, Wei Sun, Pinxin Long, Hui Huang, Daniel Cohen-Or, Minglun Gong, Oliver Deussen, and Baoquan Chen. 2014. Quality-driven Poisson-guided Autoscanning. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 33, 6, Article 203 (nov 2014), 12 pages. DOI:https://doi.org/10.1145/2661229.2661242

Amir Vaxman, Marcel Campen, Olga Diamanti, Daniele Panozzo, David Bommes, Klaus Hildebrandt, and Mirela Ben-Chen. 2016. Directional Field Synthesis, Design, and Processing. *Computer Graphics Forum* (2016), 15. http://graphics.tudelft.nl/Publications-new/2016/VCDPBHB16

Cem Yuksel, John Keyser, and Donald H House. 2010. Mesh colors. *ACM Transactions on Graphics (TOG)* 29, 2 (2010), 15.