Engineering with Computers (2011) 27:67–80
DOI 10.1007/s00366-010-0181-y

ORIGINAL ARTICLE

Real-time triangulation of point streams

Klaus Denker · Burkhard Lehner · Georg Umlauf

Received: 15 October 2008 / Accepted: 24 March 2009 / Published online: 11 March 2010

Abstract Hand-held laser scanners are commonly used in industry for reverse engineering and quality measurements. In this process, it is difficult for the human operator to scan the target object completely and uniformly. Therefore, an interactive triangulation of the scanned points can assist the operator in this task. In this paper, we describe the technical and implementational details of our real-time triangulation approach for point streams, presented at the 17th International Meshing Roundtable. Our method computes a triangulation of the point stream generated by the laser scanner online, i.e., the data points are added to the triangulation as they are received from the scanner. Multiple scanned areas and areas with a higher point density result in a finer mesh and a higher accuracy. On the other hand, the vertex density adapts to the estimated surface curvature. To guide the operator, the resulting triangulation is rendered with a visualization of its uncertainty and the display of an optimal scanning direction.

Keywords Online triangulation · Point streams · 3d Laser scanner · Quality visualization · User assistance

K. Denker (✉) · G. Umlauf
Computer Graphics Lab, Department of Computer Science, University of Applied Sciences Konstanz, Brauneggerstr. 55, 78462 Konstanz, Germany
e-mail: kdenker@htwg-konstanz.de

G. Umlauf
e-mail: umlauf@htwg-konstanz.de

B. Lehner
Geometric Algorithms Group, Department of Computer Science, University of Kaiserslautern, P.O. Box 3049, 67653 Kaiserslautern, Germany
e-mail: lehner@cs.uni-kl.de

1 Introduction

In industry, scanning of 3d objects is used in measurement and analysis of manufactured objects and in reverse engineering. Most scanning devices use a laser to sample points on the surface. Some scanners move the object while others move the laser device. While some scanning devices measure the sample points in a regular pattern, hand-held laser scanners have a movable scanning device that is moved along the surface by a human operator. These scanning devices generate a vast amount of data in very short time with very high precision. To process and triangulate this data the used method must preserve the precision while reducing the data quantity to an adequate level (see Figs. 1 and 2). Thus, it is necessary to allow for heterogeneous triangulations and point densities, especially when areas are scanned multiple times. This is particularly important for hand-held devices, with which the operator will likely scan the object from different directions and with different speeds. This generates disconnected triangulation fragments with highly different point densities.

Since the local point density can become arbitrarily high by multiple scans, the feature size that can be reconstructed is theoretically arbitrary small. Of course, the measuring accuracy/error sets a lower bound for the feature size.

For these hand-held laser scanners, the operator has to cover the complete surface of the object. Because the scanning may take a long time, it is difficult for the operator to keep track of the already scanned area. Furthermore, it is desirable to tell the operator to re-scan a region to increase the point density to improve the quality of the reconstructed surface. Therefore, a full-automatic real-time triangulation and visualization of the scanned points are

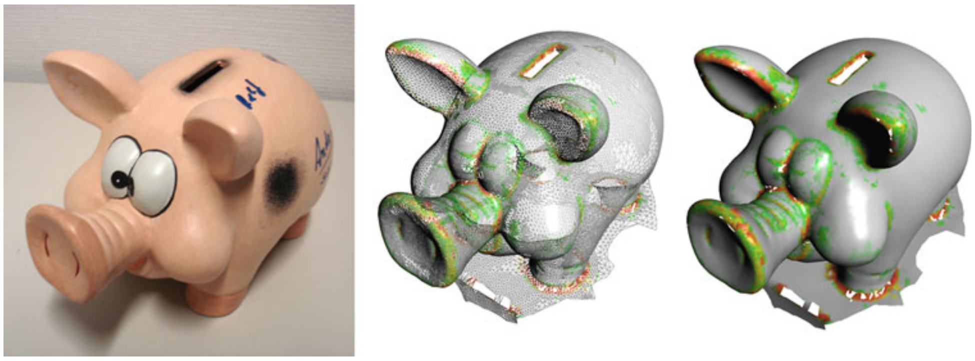
Fig. 1 A piggy bank. Original object (*left*), wire-frame (*center*), and smooth-shaded triangulation with uncertainty visualization (*right*)

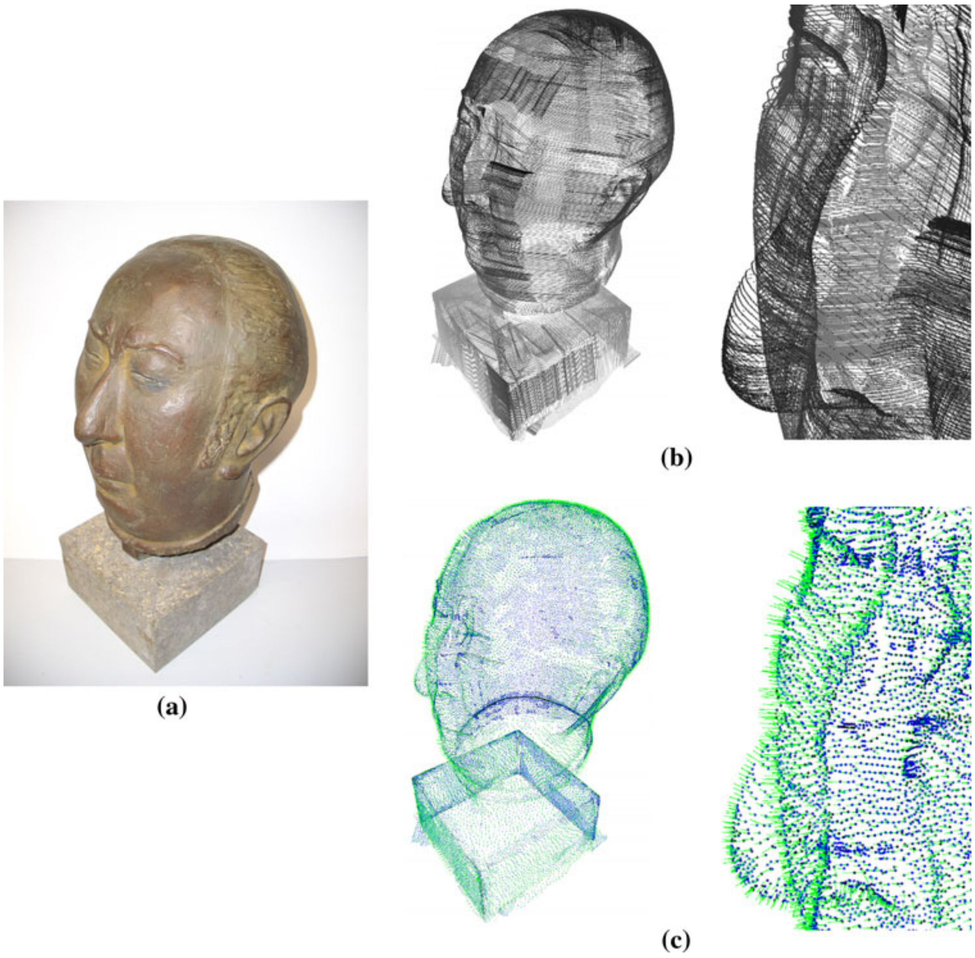

Fig. 2 Different levels of data reduction shown for the bronze bust "Bildnis Theodor Heuss" by Gerhard Marcks [19]. **a** Original object. **b** Raw points. Entire bust (*left*) and zoomed nose area (*right*). **c** Vertices and normals. Entire bust (*left*) and zoomed nose area (*right*)

crucial to assist the operator to improve the scans in less time. Thus, the crucial constraints for our triangulation task are:

A. Handling of large point sets.
B. Handling of heterogeneous point densities of incoherently scanned regions.
C. Handling of high precision point data with predefined measurement errors.
D. Handling of point streams of arbitrary order, i.e., online triangulation.
E. Fully automatic triangulation.
F. Triangulating in real-time.
G. Guiding the human operator during the scanning process.

In the rest of the paper we first discuss related work in Sect. 2 and describe the outline of our method in Sect. 3.

Then, we discuss various aspects of our method in detail in Sects. 4, 5, 6, 7, 8, 9, 10 with a particular focus on acceleration and implementation issues. In Sect. 9, we present two ways to assist the human operator during the scanning process. Finally, we show results of our method in Sect. 11 and give an outlook to future research plans in Sect. 12.

2 Related work

To contrast our approach to other methods for surface reconstruction from unorganized point clouds, we briefly describe alternative methods and discuss their pros and cons with respect to the constraints A.–G. given above.

One of the first methods in this field was proposed in [15]. Here, for every point a surface normal is estimated from its k nearest neighbors and its orientation is propagated from the orientation of one particular normal to all other normals using a global minimal spanning tree of the points. This allows to estimate tangent planes defining an estimated signed distance function to the surface. Its zero-set is used to compute a triangulation of the surface using marching cubes [18]. This method can deal with surfaces with boundaries and holes, and no additional information (such as surface normals) is necessary. On the other hand, it is not capable of dealing with incremental insertion of data points, since the orientation propagation is global, and the density of the points on the surface is pre-defined, otherwise spurious holes are introduced. The minimum feature size that can be reconstructed is fixed a priori by the edge length of the marching cubes algorithm and increasing the density of points does not reveal more details. Therefore, at least constraints B. and D. are not satisfied.

Alpha shapes were defined in [11, 12]. For a given real number $\alpha > 0$, the alpha shape S_α of a point set P is the set of all k-simplices $T \subset P$ $(k < d)$ with vertices lying on a sphere with radius α that does not contain any other point of P. The alpha shape can efficiently be determined from the Delaunay triangulation where α controls how many “details” of the point cloud are “cut” out of the convex hull of P. If α is too large, details remain hidden under larger faces, if it is too small, the object may be cut into disconnected pieces. Therefore, the choice of α is crucial for the optimal reconstruction of a surface from a point cloud. If the variation of the point density is too high, there may not even exist a suitable α value, violating constraint B.

For the so-called *weighted alpha shapes* of [1] every point of the set P gets an associated weight. This permits using different values of α for different regions of P. The weights have to be tuned to the point density very accurately to achieve a good reconstruction of the underlying surface, making it difficult to achieve B. Another extension was proposed in [23]. The sphere with radius α is deformed anisotropically into an ellipsoid, achieving a more accurate separation of surfaces close to each other. But their approach relies on user input, violating constraint E.

All methods based on alpha shapes can be used to reconstruct surfaces with borders and holes, and no additional information per vertex other than its position is necessary. But, the correct choice of α or the point weights, respectively, is crucial for the quality of the final triangulation, which is in contrast to constraint C. Furthermore, the computed triangulation is not guaranteed to be a two-manifold with border. It can contain edges with more than two adjacent faces, or isolated edges and vertices. Therefore, a post-processing clean-up step is necessary, violating also constraint F.

The so-called *power crust* of [4] is based on an approximation of the medial axis transform of the point set. It is computed from the Voronoi diagram and the poles of the input points. From this, it calculates in an inverse transformation of the original surface using the power diagram of the poles and taking the simplices dividing the interior and exterior cells of the power diagram from each other as triangulation for the surface. The power crust approach produces connected surfaces possibly with intentional holes. So, it is not suitable for online triangulations where the triangulation may consist of disconnected fragments (constraint B.).

The *eigencrust* method proposed in [17] is specialized to produce high quality surface reconstructions on noisy point clouds. It labels all tetrahedra in the 3d Delaunay triangulation of the sample points as either being inside or outside the surface based on a global optimization. The triangulation of the reconstructed surface is the set of faces that are adjacent to one inside and one outside tetrahedron. Because of the global optimization step, the results are of high quality even with the presence of noise and outliers. The final surface is always a two-manifold without border. Therefore, the eigencrust method cannot be used with constraint D.

The geometric convection approach for surface reconstruction described in [7] starts with a Delaunay triangulation of the point set, and shrinks the boundary surface by removing tetrahedra containing a boundary triangle that does not fulfill the *oriented Gabriel property*, i.e., the half-sphere centered at the triangle’s circumcenter and oriented to the inside contains point of the point set. This procedure is repeated until all boundary triangles fulfill the oriented Gabriel property. In some cases, cavities are not opened by this algorithm, so another property has to be defined to remove the involved tetrahedra. The approach requires the Delaunay triangulation of the complete point set, therefore it contradicts constraint D.

An extension of [7] for streams of point sets is proposed in [3]. The point set is divided into slices, and only a limited number of slices is kept in memory. A slice that cannot have impact on the current slice can be removed from memory, storing the triangles found for that slice. For the division into slices, all points have to be known in advance, because they have to be ordered according to one of the spatial coordinates. So, this method is not suitable for constraint D.

A method suitable not only for surface reconstruction but also for re-meshing of an existing mesh is presented in [21]. Using an advancing front approach, triangles are constructed that fulfill two user-defined constraints: the maximum edge length with respect to the curvature, and ratio bounds of adjacent edges. For surface reconstruction, a projection operator and a guidance field have to be defined, contradicting constraints B., D., and F.

A common problem of these methods [1, 3, 4, 7, 12, 17, 23] is their computational complexity, which is too high for real-time applications (constraint F.). Another disadvantage of these methods is the fact that the sample points or the same number of points are used to create the surface mesh. Thus, the complexity of the meshes increases rapidly while scanning, and the measurement errors are not corrected. Furthermore, if a region is scanned multiple times, the additional vertices decrease the area of the mesh faces, but the noise remains constant, leading to a bumpier surface after every scan pass. This is in contrast to constraints A. and C.

In [6] an interactive online triangulation method is proposed. The sampled points are processed in a pipeline. In the first stage the number of points is reduced by dropping every point that is closer than a specified radius from an already existing point. In the second stage, the normal at the point is estimated using the points in a local neighborhood. After another reduction stage with a larger radius, the points with a stable normal are inserted into the surface mesh which is re-triangulated locally with a shortest edge criterion to decide which edges to keep. This approach works effectively and efficiently, but has some major drawbacks:

- A large fraction of the input is ignored and not used to reduce the noise of the input data, violating constraint C.
- The size of the smallest features that can be modeled is bounded, violating constraint C.
- The size of the mesh triangles is not adapted to the density of sample points or the curvature of the surface, violating constraints B. and E.
- There is no immediate visual feedback for the human operator, violating constraint G.

The method described here and in [9] is based on [6], but satisfies all constraints A.–G. simultaneously.

3 Online triangulation

Our method is developed for a laser scanner like the Faro Laser ScanArm [11] described in Sect. 11. Scanners of this type generate a stream $D = (d_1, d_2, \ldots)$ of *data points* d_i. Each data point is a pair $d_i = (p_i, h_i) \in \mathbb{R}^3 \times \mathbb{R}^3$ of a *raw point* p_i, that is measured by the scanner on the scanned object, and the *scan position* h_i of the laser scanner at the moment of scanning p_i.

A laser scanner of this kind scans an object line by line measuring a certain number of data points per scan line. The scanner we used scans up to 30 lines per second measuring up to 640 data points per scan line (see Fig. 2b). These scan lines are arranged in scan passes that are triggered by the human operator by pressing a button on the scanner. The pauses between two scan passes are usually used by the operator to reposition the scanner for a different scan direction.

In order to triangulate this data stream online, the data points are reduced. They are classified by their distance and clustered into so-called *neighborhood balls* b_j that represent subsets of data points within a certain radius and with similar scan positions. The radius depends on the local point density and curvature estimate. Subsequently only the averages of data points of the neighborhood balls are used as vertex positions in the triangulation T approximating $\{p_1, p_2, \ldots\}$. The overall process is described schematically as follows

```
ONLINE-TRIANGULATION$(d_1, d_2, \ldots)$
Input: Data point stream $D = (d_1, d_2, \ldots)$;
Output: Triangulation $T$ approximating $\{p_1, p_2, \ldots\}$.
1: while ($D$ not terminated) do {
2:     ADD-TO-NEIGHBORHOOD-BALLS$(d_i)$;                    \\ see Sec. 4
3:     Update normals of affected neighborhood balls;     \\ see Sec. 6
4:     Update local approximation;                        \\ see Sec. 8
5:     Triangulate area of affected neighborhood balls;   \\ see Sec. 7
6:     Render triangulation for user guidance;            \\ see Sec. 9
7: }
```

4 Neighborhood balls

A bounding cube of edge length R enclosing the maximal scanning range is

$$O = [x_{\min}, x_{\min} + R] \times [y_{\min}, y_{\min} + R] \times [z_{\min}, z_{\min} + R],$$

i.e., $p_i \in O$ for all i. Furthermore, a *ball* with center $c \in \mathbb{R}^3$ and radius $r \in \mathbb{R}, r \geq 0$, is defined as the set

$$\beta(c, r) = \{x \in \mathbb{R}^3 : \|x - c\| \leq r\}$$

where $\|\cdot\|$ denotes the Euclidean norm.

As in [6] we use *neighborhood balls* $b_j = (c_j, r_j, D_j)$ to collect a set of n_j data points $D_j = \{d_{j,1}, \ldots, d_{j,n_j}\} \subset D$ contained in the ball $\beta_j = \beta(c_j, r_j)$. Every neighborhood ball corresponds to a local estimate N_j for the oriented surface normal, which might be undefined, and a vertex v_j of the triangulation T. Thus, neighborhood balls can overlap and serve three purposes:

- Collecting n_j data points to reduce the number of visualized data points.
- Estimating a local oriented surface normal.
- Averaging its data points gives the position of a vertex of the triangulation.

The minimal ball radius $R_{\min} = 0.75$ mm prevents ball sizes below scanner accuracy. A neighborhood ball b_j may contain up to $n_{\text{split}} = 40$ data points.

The set of all neighborhood balls b_j is denoted by B. It is initialized with the first data point $B = \{(p_1, R, \{d_1\})\}$. Then, new neighborhood balls are generated by adding one data point after the other with Add-To-Neighborhood-Balls(d_i), using the following steps:

1. To add data point $d_i = (p_i, h_i)$, first all k neighborhood balls $b_j = (c_j, r_j, D_j)$ are determined that contain $p_i \in \beta_j$ and correspond to normals N_j aligned to the scanning direction. This means, if N_j is defined,

   $$N_j^T(h_i - p_i) \geq 0. \quad (1)$$

   (a) If $k = 1$, d_i is added to D_j.
   (b) If $k > 1$, d_i is added to D_j of the neighborhood ball b_j with largest radius and smallest distance $|c_j - p_i|$.
   (c) If $k = 0$, a new neighborhood ball $b = (p_i, r, \{d_i\})$ is generated, with radius $r = R/2^{\mu}$ where μ is the smallest integer such that $\beta(p_i, r)$ does not contain the center of any other neighborhood ball b_j. The new ball b is added to B.

2. If in cases (a) and (b) n_j equals n_{split} after d_i is added and $r > R_{\min}$, the neighborhood ball b_j is removed from B and all data points in D_j are added using Step 1. If in this process a data point $d_l \in D_j$ is not contained in any other neighborhood ball of $B \setminus \{b_j\}$, a new ball $b = (p_l, r_j/2, \{d_l\})$ is generated and added to B.

To accelerate this process the recursive descent that may become necessary in step 2. is linearized. Instead of inserting the data points recursively they are added to a fifo-queue. This queue holds the data points and a maximal ball radius for each point. Before any new data point is added this queue is processed sequentially.

Remark 1 This definition of neighborhood balls has two advantages over the method of [6]: first, all data points are collected in neighborhood balls and not only the first one to support constraint C. Second, the radii of the balls can be adapted to the density of the data points and the estimated curvature of the surface to support B. and E.

5 The octree

To support the geometric neighborhood searches efficiently we use an octree data structure to manage the neighborhood balls. The root node of the octree represents the cube O. Every node in the tree represents a sub-cube

$$o = [x, x + \delta] \times [y, y + \delta] \times [z, z + \delta]$$

of O with side length δ. A node has either zero or eight child nodes holding the eight sub-cubes $o_1, \ldots, o_8$ with side length $\delta/2$ (see Fig. 3). To accelerate searches in the local neighborhood, every node stores additional links to the 26 face-, edge- and corner-neighbor nodes $o_{n_1}, \ldots, o_{n_{26}}$ on the same level, as in [6].

Remark 2 The main advantage of using an octree instead of a grid as in [6] is the use of different levels of detail corresponding to the levels of the octree to create a finer triangulation in regions with higher point density.

Every neighborhood ball b_i belongs to one cube in the octree which contains its center point c_i. The radius r_i equals the edge length of the cube. Every cube in the octree has a list of the neighborhood balls it contains.

Every data point $d_j = (p_j, h_j)$ is inserted into the octree by searching for the neighborhood ball b_i containing the

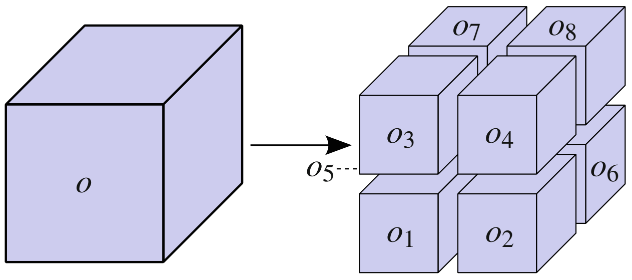

Fig. 3 The cube o of an octree node and its child-node's sub-cubes $o_1, \ldots, o_8$

raw point $p_j \in \beta_i$ and, if b_i is found, inserting it to that ball as in Sect. 4. To search for b_i the octree is descended from the root node O traversing on any level of the octree the cube o containing p_j. For the traversal all 26 neighbor cubes are tested to find the ball with c_i closest to p_j. If no b_i is found, the search descends one level in the tree and repeats the neighbor traversal. This is repeated until a ball is found or the leaf nodes are searched unsuccessfully. In the latter case a new neighborhood ball b containing d_j is created in the leaf node cube containing p_j.

To make b as large as possible the highest level in the octree with sufficient space is determined, such that the ball β_i does not contain the center of any other ball of B. These centers can only be in the siblings of the 26 indirect neighbor cubes, which share at least one corner with the actual cube o. The set of these cubes is denoted by $S(o)$. In $S(o)$ the center c_k that is closest to p_j is determined with $\Delta := |c_k - p_j|$.

1. If Δ is smaller than the radius of b, i.e., the edge length ℓ of o, the cube o is split into sub-cubes until the radius of b is small than Δ. Thus, b is added to a sub-cube o' that is $\lceil \log_2(\ell/\Delta) \rceil$ levels below the node of o in the octree.
2. If Δ is larger than the radius of b, it can be added to the cube o or one of its ancestors. Thus, the ancestors o' of o are tested if their siblings of $S(o')$ contain a center too close to p_j. Finally, b is added to the highest ancestor above o for which this test is negative.

5.1 Octree shortcuts

As data points that are scanned shortly after each other will probably be added to the same neighborhood ball, a global link to the last ball b_{last} a data point was added to is stored. For subsequent data points we first test if b_{last} also contains the new data point. In this case no search operations in the octree are necessary and the data point is added directly to the neighborhood ball b_{last}.

Similarly a global link o_{last} to the parent of the cube containing b_{last} is stored. Thus, if a search in the octree becomes necessary and o_{last} contains the new data point p_i, the search starts at o_{last} instead of the root node O. The link o_{last} is set to the parent cube, because neighborhood balls with large radius should be preferred. Although this strategy contradicts step 1.(b) of Add-To-Neighborhood-Balls, it accelerates the overall process.

6 Normal estimation

For every neighborhood ball b_j an estimated surface normal N_j is calculated. First, all n_l data points d_l contained in $\beta(\overline{b}_j, 2r_j) \ni p_l$ with

$$\overline{b}_j = \frac{1}{n_j} \sum_{p_k \in \beta_j} p_k$$

are collected in a set D_j^N. This provides a more stable normal estimation than using only the data points in D_j. The data points in D_j^N are used for a principal component analysis as in [6, 16]. Computing the eigenvalues $0 \le e_1 \le e_2 \le e_3$ of the 3×3 covariance matrix

$$C = \sum_{d_l \in D_j^N} (p_l - \overline{b}_j)(p_l - \overline{b}_j)^{\mathrm{T}}$$

using [22], yields the eigenvector v_{e_1} of C corresponding to the smallest eigenvalue e_1. The direction of this eigenvector is used to estimate the normal N_j

$$N_j = \frac{v_{e_1}}{\|v_{e_1}\|}.$$

To get a stable normal estimate, we require $e_2 \ge 2\ e_1$. Otherwise, b_j does not have a normal estimate and all subsequent computations requiring a N_j are rejected. Thus, highly curved regions with low point density are either not triangulated or marked as regions that need a further scan pass (see Sect. 9.1). This ensures a locally planar point distribution.

To get the orientation of N_j, we use the average scan direction of the data points $d_l \ \in D^N{}_j$

$$\overline{s}_j = \frac{1}{n_l} \sum_{d_l \in D_j^N} (h_l - p_l).$$

The normal orientation is correct if $N_j^T \cdot \overline{s}_j \ge 0$. Otherwise the orientation is inverted. Finally, all data points of D_j that do not satisfy (1) are removed from D_j and re-inserted later with Add-To-Neighborhood-Balls. Figure 2c displays the estimated normals of the neighborhood balls.

6.1 Update strategy

For efficiency the normal estimation is not recomputed every time a data point d_i is added to the neighborhood ball b_j. Instead, the update is only triggered for certain point counts n_j of b_j. We get good results doing an update every time the point count n_j exceeds $\sqrt{2}$ times the point count of the last normal estimation. Furthermore, we observed that it is not necessary to update very small neighborhood balls often, because a ball with less than eight points usually does not have a stable normal estimate. Based on these observations we trigger an update of the normal for b_j only if n_j is a multiple of eight and exceeds $\sqrt{2}$ times the point count of the last normal estimation. In our experiments we found this value to be a good compromise between performance of the process and actuality of the data structures.

6.2 Caching

Some of the data computed in this section will be reused in later stages of our triangulation process. These data are cached to avoid unnecessary re-computations.

- The sets D_j^N are cached for later use in Sect. 8 where we need these data points for smoothing.
- For the re-triangulation and the local least square fits in Sects. 7 and 8 the raw points p_l for all $d_l \in D_j^N$ are projected to the estimated tangent plane. This is computed by representing all p_l in a local tangential coordinate system spanned by N_j and discarding the coordinate in normal direction. For this change of the coordinate system the corresponding rotation matrix is cached every time the normal changes.

7 Local triangulation

Because every neighborhood ball corresponds to one vertex in the triangulation, the latter is updated in five steps if a neighborhood ball b_j is added or removed from B or if a normal N_j changes:

1. Collect all potential neighbor vertices of v_j of neighborhood ball b_j in a set V_j (see Sect. 7.1)
2. Project V_j onto estimated tangent plane (see Sect. 7.2)
3. Adjust the border polygon of V_j to T (see Sect. 7.3)
4. Determine the triangulation T_j of V_j (see Sect. 7.4)
5. Insert T_j into the triangulation T (see Sect. 7.5)

In order to optimize the efficiency of this triangulation approach, the triangulation is updated not immediately after an update becomes necessary. Instead, the update of the triangulation follows the schedule described in Sect. 7.6

7.1 Collecting potential neighbor vertices

Every neighborhood ball b_j corresponds to a vertex v_j in T with position $\overline{b}_j$. Thus, if b_j is added or removed from B, the corresponding vertex v_j is added or removed from T. In both cases the local neighborhood of v_j needs to be re-triangulated. To determine the geometric neighbors of v_j, we define a ball

$$\eta_j(r) = \{x \in \mathbb{R}^3 \\ : \|(x - \overline{b}_j) + ((x - \overline{b}_j)^{\mathrm{T}} N_j)(f_\eta - 1)N_j\| \le r\}.$$

of radius r around $\overline{b}_j$ flattened along the normal N_j by f_η to provide a better separation of close parallel surfaces sheets. Then, the geometric neighbors are all b_l with $\overline{b}_l \in \eta_j(5r_j)$ for $f_\eta = 3$ and $N_l^T \cdot N_j \ge 0.5$. This yields a set

$$V_j = \{v_{j_1}, \ldots, v_{j_m}\} \subset V \cup \{v_j\}$$

of vertices that will be re-meshed.

We found the radius $5r_j$ and the flattening factor f_η by experiments. The radius $5r_j$ depends on the mesh structure while the flattening factor f_η depends on how close parallel surface sheets can be distinguished. The latter should be re-calibrated for every laser scanner and application.

7.2 Projection onto the estimated tangent plane

Because the triangulation of the area around b_j is computed in the plane perpendicular to N_j, all vertices $v_{j_i} \in V_j$ are projected along N_j onto this plane, i.e., $\overline{b}_{j_i}$ is projected to t_{j_i}. Then, because of the one-to-one correspondence of v_{j_i} to t_{j_i}, triangulating the t_{j_i} is equivalent to triangulating the v_{j_i}. Therefore, we will speak of a triangulation of V_j although the triangulation is computed in the estimated tangent plane.

This projection to the estimated tangent plane is computed by a rotation of the z-axis of the global coordinate system to the estimated normal. Then, for the projection only the third coordinate needs to be discarded.

7.3 Adjusting the border of the local triangulation

Triangulating V_j yields a triangulation T_j of the local neighborhood of v_j. Because most vertices in T_j are also in T, the edges in T_j should match edges in T. Thus, the *border polygon* ∂T_j of T_j has to match edges in T. The border $\partial T_j = (v_{x_1}, \ldots, v_{x_{n_x}})$ is represented as a counter-clockwise oriented, ordered sequence of n_x *border vertices* $v_{x_l} \in V_j, l = 1, \ldots, n_x - 1$ with $v_1 = v_{x_{n_x}}$, i.e., the index of border vertices is cyclic above n_x. Every pair $(v_{x_l}, v_{x_{l+1}})$ is a so-called *border edge* and the border ∂T_j is initialized as the convex hull of V_j using “Jarvis’ March” [5]. Subsequently V_j and ∂T_j are modified until the border edges match edges in T as good as possible.

If a border edge $e_b = (v_{x_k}, v_{x_{k+1}})$ does not match any edge in T, determine the edge $e_s = (v_{x_k}, v_s)$ or $e_{\overline{s}} = (v_s, v_{x_{k+1}}) \in V_j \times V_j$ in T inside ∂T_j with smallest angle φ to e_b. Then v_s is inserted to ∂T_j between v_{x_k} and $v_{x_{k+1}}$, if e_s, respectively, $e_{\overline{s}}$ is either an inner edge or $\varphi < 20°$ and if this does not cause proper intersections of the interior of two edges of ∂T_j or any loops containing more than two border edges (see Fig. 4). The condition $\varphi < 20°$ is necessary to avoid sharp triangles near the border of the surface mesh.

This approach can lead to a border of the form $(\ldots, v_{x_l}, v_{x_{l+1}}, v_{x_{l+2}}, \ldots)$ with $v_{x_l} = v_{x_{l+2}}$, see Figure 4. This configuration is handled by removing v_{x_l} and $v_{x_{l+1}}$ from ∂T_j and $v_{x_{l+1}}$ from V_j.

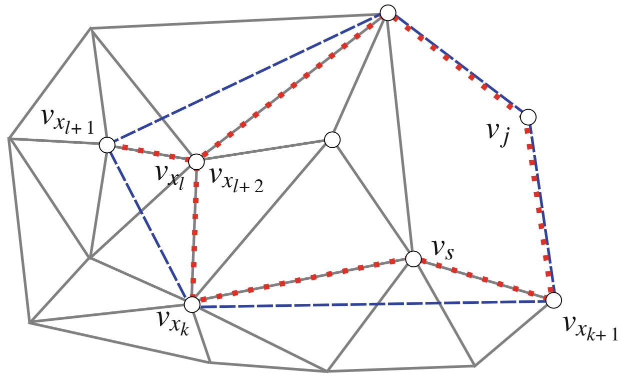

Fig. 4 The border ∂T_j before (*blue*) and after (*red*) the modification

Repeating these operations until there are no more edges that can be removed results in a border that fits the existing triangulation T better. This process terminates because the border shrinks monotonically in each step.

7.4 Triangulation of the border

To triangulate V_j first ∂T_j is split into monotone sub-polygons which are triangulated individually (see e.g., [5]). This determines a triangulation T_j' of ∂T_j. Second, all vertices of $v_l \in V_j \backslash \partial T_j$ are added to T_j' successively by splitting the triangle of T_j' that contains t_l at t_l into three new triangles. This is repeated for every vertex of $V_j \backslash \partial T_j$ yielding a triangulation T_j''. Finally, the Delaunay criterion [14] is applied repeatedly constrained by ∂T_j to improve the triangle quality generating a triangulation T_j of V_j.

7.5 Insertion of the local triangulation

Before T_j can be inserted into T, the triangles of T in conflict with T_j must be removed.

First, we remove all triangles of T incident to a vertex of $V_j \backslash \partial T_j$. Note that this also removes triangles from vertices in $V_j \backslash \partial T_j$ to vertices in $V \backslash V_j$ outside of $\eta_j(5r_j)$. Thus, the global topology of the surface is corrected due to the increased local point density.

At this stage T contains no triangles connected to vertices of $V_j \backslash \partial T_j$. All triangles remaining in T conflicting with T_j involve only vertices on ∂T_j. For such a vertex $v_l \in \partial T_j$ with border edges $e_1 = (v_{l\text{-}1}, v_l)$ and $e_2 = (v_l, v_{l+1})$, all faces potentially pointing to the inside of ∂T_j are deleted. So, there are three cases that are solved topologically:

1. If both edges e_1 and e_2 belong to T, all faces of the one-ring of v_l are removed if they are left of e_1 or e_2 or if they are not connected to the right faces of e_1 or e_2. This removes areas of T that are also covered by T_j and reduces the complexity of the one-ring (see Fig. 5, left).
2. If only e_1 belongs to T, the face of the one-ring of v_l left of e_1 and all faces not connected to the right face of e_1 are deleted (see Fig. 5, middle).
3. If both e_1 and e_2 do not belong to T and v_l has a closed one-ring, the face pointing the most inside the triangulated area is removed, which is, e.g., the triangle of the one-ring of v_l intersected by the bisector of e_1 and e_2 in the local estimated tangent plane (see Fig. 5, right).

Finally, if there are two vertices v_{l_1} and v_{l_2} from ∂T_j that are connected by an edge e that does not belong to T_j and lies inside of the polygon spanned by ∂T_j, the corresponding triangles are removed. To test if such an edge is inside the polygon spanned by ∂T_j the inner angle β of the polygon at v_{l_1} must be larger than the angle α between the incoming border edge at v_{l_1} and e, see Figure 6. Note that this also changes the global topology of the surface.

7.6 Triangulation update schedule

There are several events that require a subsequent update of the local triangulation. Since the local triangulation depends on the local normal estimate some of these events are triggered by normal changes:

- The first time a stable normal is estimated for a neighborhood ball, its corresponding vertex is added to the triangulation.
- The normal estimate for a neighborhood ball is updated when new data points are inserted (see Sect. 6.1). In this case the triangulation will be updated if the difference of the new normal N_i to the old normal N_{old_i}, that was used for the last triangulation update, is larger than $0.25 < \|N_i - N_{\text{old}_i}\|$. This constant is a compromise between performance and actuality. It directly affects the normal and mesh quality.
- When a neighborhood ball is removed, the corresponding vertex is removed from the triangulation, i.e., it is updated to close the hole.

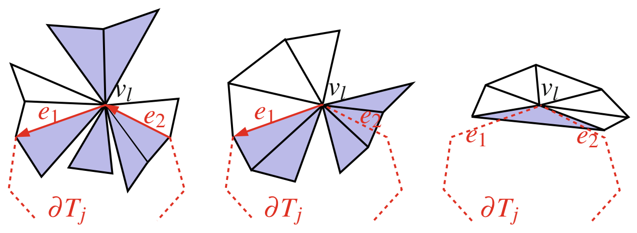

Fig. 5 Topologically preparing the border ∂T_j (*red*) for the new triangulation T_j by deleting the blue faces. All *triangles* are triangles of T, triangles of T_j are not shown

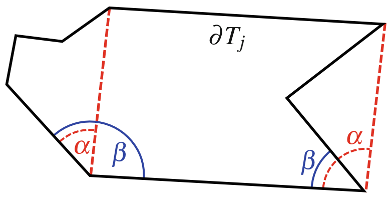

Fig. 6 Border ∂T_j (*black*) with edges (*red*) not connected by faces to the border

Because data points that are inserted to the data structures shortly after each other will most likely affect the same region of the triangulation, these update operations are not executed immediately. Instead these updates are delayed until the newly inserted data points will affect a different region of the triangulation. This strategy reduces the number of unnecessary updates of the triangulation.

To schedule the updates, the affected neighborhood balls are buffered in a so-called *unique queue*. Such a queue holds every of its elements only once. Multiple update operations for the same vertex within a short time interval are avoided this way. An individual thread is then executing the triangulation and dequeuing the neighborhood balls (see Sect. 10). To defer the updates during the scanning process, the queue holds at least 100 neighborhood balls.

8 Improving the approximation

For every neighborhood ball b_i a least square fit f_i to its raw points is computed to smooth the triangulation and to reduce the noise in the raw points. The fit f_i is a cubic approximation of the raw points p_l of all $d_l \in D_i^N$ parameterized over the estimated tangent plane of b_i as in [2] computed by a singular value decomposition. It serves three purposes:

- correction of the vertex positions,
- adjustment of the ball sizes depending on local curvature estimates, and
- denoising of the raw points.

8.1 Correction of vertex positions

The position of a vertex v corresponding to a neighborhood ball b_i is approximated by the arithmetic mean of all its raw points $\overline{b}_i$. On curved surfaces this results in a displaced position. In local coordinates of the estimated tangent plane, $\overline{b}_i$ has coordinates $(0, 0, 0)^{\mathrm{T}}$. Thus, the point $\widetilde{b}_i$ with local coordinates $(0, 0, f_i(0, 0))^{\mathrm{T}}$ is a better approximation of the raw points. In order to guarantee that the vertices of T are within scanner precision, the raw point p_j closest to $\widetilde{b}_i$ is determined. If ε is the scanner precision and $\widetilde{b}_i$ is not contained in $\beta(p_j, \varepsilon)$, $\widetilde{b}_i$ is projected onto $\beta(p_j, \varepsilon)$ in direction $p_j - \widetilde{b}_i$. This new point is the position of the vertex v_i corresponding to the ball b_i.

Remark 3 Because the laser scanner has different precisions in different directions, i.e., along the scan line, between scan lines, and in laser beam direction, $\widetilde{b}_i$ is projected onto an ellipsoid around p_j.

8.2 Curvature-dependent ball size

The fits f_i are also used to estimate the curvature of T in the vertex v_i. Then the size of the neighborhood balls is controlled by the curvature measure $C_i := (|\kappa_1| + |\kappa_2|)/2$ at v_i, where κ_1 and κ_2 are the principle curvatures of f_i at $\widetilde{b}_i$ before the projection onto $\beta(p_j, \varepsilon)$.

A small value of C_i indicates that the region is rather flat. So, for each neighborhood ball with a valid normal, the curvature C_i is computed and the neighborhood ball is split if

$$\arctan(4 r_j C_j) 2 n_j / \pi \geq n_{\text{split}}.$$

This results in larger triangles in flat regions.

8.3 Smoothing the raw points

The scanned data points d_i contain some inaccuracy and noise. Especially multiple scan passes of the same surface area can generate visible artifacts due to repeat accuracy. This noise can be reduced by projecting all raw points p_i onto a corresponding local least squares fit.

This projection is done independently for every neighborhood ball b_j. The fits f_j are only used for the projection of the raw points $p_k \in \beta(\overline{b}_j, r_{\text{smooth}})$. To achieve a good quality and performance we choose $r_{\text{smooth}} = r_j/4$. All the other raw points of b_j are projected by successively picking one of them p_l. Then, for p_l a new normal N_l and a new least square fit f_l are calculated. These calculations are based on all raw points within the ball $\beta(p_l, 2r_j)$. The chosen ball radius $2r_j$ affects the smoothness of the surface. All unprojected raw points within $\beta(p_l, r_{\text{smooth}})$ are projected to f_l. These steps are repeated until all raw points in b_j are smoothed.

The distance of the projection $\widetilde{p}_i$ of any raw point to its original position p_i must not exceed the vertical accuracy ε_v of the laser scanner. If $\|p_i - \widetilde{p}_i\| > \epsilon_v, \widetilde{p}_i$ is corrected to

$$p_i + \epsilon_v \frac{p_i - \widetilde{p}_i}{\|p_i - \widetilde{p}_i\|}.$$

Figure 7 shows the result of the data point smoothing.

This smoothing is an optional post-processing step if better raw points are required for subsequent applications. Calculating it online slows the scanning process down.

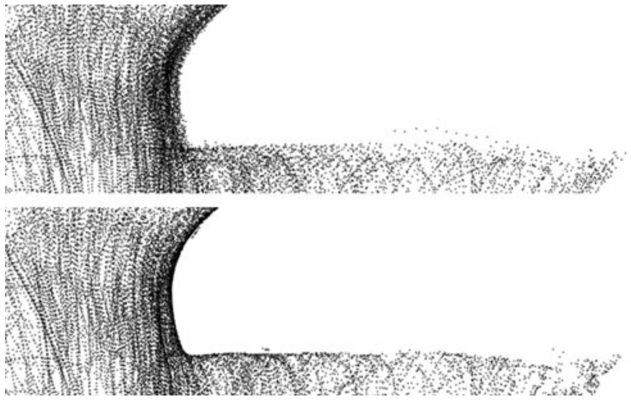

Fig. 7 Comparison between raw data points (*top*) and smoothed data points (*bottom*)

8.4 Improving the parameters

Usually all projections to the least square fit f_i are computed evaluating f_i at the parameters given by the point coordinates in the estimated tangent planes. However, ideally a point would be projected to the nearest point on the surface f_i. The method described in [20, Chapter 4.8] projects the raw point onto a local tangent and uses the resulting coordinates as new parameters. Iterating this three times gives a good approximation to the ideal solution. This technique is used to improve all projections to f_i.

8.5 Update strategy

Like the estimated normals N_i the least square fits f_i need to be updated during the scanning process. For this we use the same strategy as for the normal update. Every time a normal is re-estimated the associated least square fit is updated, too. Otherwise the fit would become invalid because it uses the estimated tangent for the local parameterization. Further updates are not required.

9 Operator guidance

9.1 Uncertainty visualization

Regions with a high uncertainty in the triangulation should be highlighted to enable the operator of the laser scanner to increase the point density by multiple scan passes. To measure the uncertainty, the stability of the normal estimation is used. It is estimated for each neighborhood ball b_i by the two smallest eigenvalues e_1 and e_2 of the principal component analysis of Sect. 6. The uncertainty u_i is defined as

$$u_i = \arctan((e_2/e_1 - 2)/20)2/\pi.$$

It is restricted to [0, 1] and visualized by coloring the vertices using a transition from red ($u_i = 0$), yellow ($u_i = 0.25$), green ($u_i = 0.5$) to white ($u_i \geq 0.75$).

9.2 Optimal scanning direction

Experiments show that the orientation of the scanning device during the scan has an impact on the quality of the input data. Scanning the same region with different orientations improves the stability of the computation significantly. In addition to the uncertainty visualization, the optimal orientation for the next scan pass of an uncertain region is visualized to assist the operator to achieve better results.

Each scan line defines a direction s_i between its start- and end-point. For each neighborhood ball b_j, an average direction $\bar{s}_j$ of all crossing scan lines is computed as their normalized sum. To stabilize the resulting direction during the scanning process, the directions of the last 20 scan lines are ignored in the summation.

If the neighborhood ball b_j has a normal N_j, the optimal scanning direction $Q_j = (\bar{s}_j \times N_j) \times N_j$ lies in the tangent plane. Without a normal, the direction of the eigenvector $Q_j = v_{e_3}/|v_{e_3}|$ of the largest eigenvalue e_3 of the principal component analysis is used. In uncertain regions the normalized optimal direction Q_j is drawn directly onto the surface (see Fig. 8).

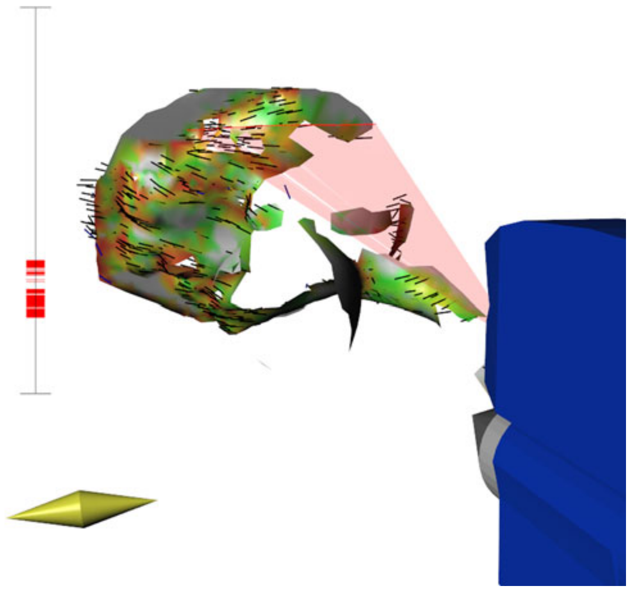

Fig. 8 Proposed scanning directions visualized on the surface (*small black lines*) and as a compass needle (*bottom left*). The *range bar* at the *left* shows the distance of the laser scanner to the current scan line on the surface

Additionally, we use a more intuitive way to display the optimal scanning direction to assist the operator to orient the scanner during a scan pass. Here, we use the optimal direction Q_j of the neighborhood ball b_j that contains the midpoint of the actual scan line. Then Q_j is rotated to the local coordinate system of the laser scanner and displayed as a compass needle at the bottom left of the screen (see Fig. 8).

For a faster computation of the optimal scanning direction, links to all scan lines crossing a neighborhood ball are cached. Every neighborhood ball stores $\overline{s}_j$ which is only recalculated when a new scan line is added.

10 Multi-threading

Today's computers allow the parallel execution of multiple threads. To use this advantage, our program is split into several parts running relatively independent.

- The first thread takes the data from the laser scanner and stores it in a simple list of all scan lines. This thread has high priority to assure that no data from the scanner is lost, because of performance problems in later stages of the scanning pipeline.
- A second thread reads these scan lines and inserts their data points to the neighborhood balls organized in an octree. It also adds the neighborhood balls to the unique queue for the local re-triangulation.
- In the third thread all triangulation operations are performed. This thread takes the neighborhood balls from the re-triangulation queue and re-triangulates the mesh around the corresponding vertices. It lags behind the second thread.
- The fourth thread renders the triangulation and computes the visualizations to assist the operator.

To make our implementation *thread-safe*, the third and the fourth thread are locking a common mutex protecting the triangulation data structure. By controlling how long one of the threads holds this lock and by threat-dependent priorities the computation time between the two threads is balanced.

11 Results

We used a hand-held laser scanner "Laser ScanArm" from Faro [13] (Fig. 9). That is a measurement arm with seven joints and an assembled laser scanner "Laser Line Probe". The scanner driver provides 3d point data relative to the foot of the measurement arm. Lines of up to 640 points can be scanned up to 30 times per second. For each scan line the position and the viewing direction of the laser scanner are tracked.

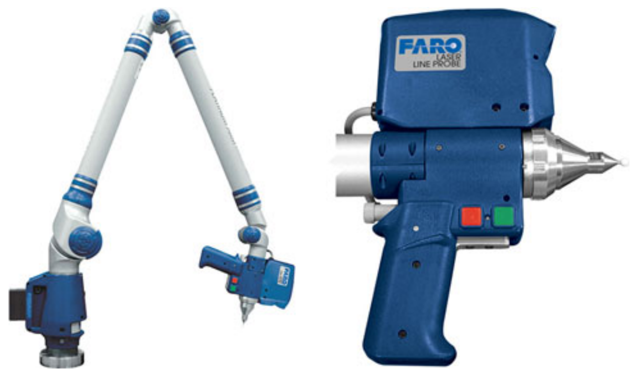

Fig. 9 The measuring arm "FaroArm" with laser scanner "Laser Line Probe" [13]

Our implementation uses the Qt framework, OpenGL for rendering, OpenMesh as mesh data structure, and the GNU Scientific Library for eigenvalue and SVD calculations. All other data structures and algorithms were implemented by ourselves. All examples are computed on an Intel Core 2 Quad Q6600, 2.4 GHz computer with 4 GB RAM.

In Fig. 1, a scanned piggy bank is shown, while Fig. 12 shows a simple mug. Figure 13 displays the result of scanning the bronze bust "Bildnis Theodor Heuss" by Gerhard Marcks. The uncertainty visualization in the wire-frame and smooth-shaded representations reveal the regions that can be improved by additional scan passes.

The reduction of the input points is demonstrated in Fig. 2, where Fig. 2b shows the data generated by the scanner, and Fig. 2c shows the averages $\overline{b}_j$ of the neighborhood balls and the estimated normals for every ball.

Figure 10 shows how the triangulation is adaptively refined for several scan passes of the same region. The edge of the protruded digit becomes more precise after each scan pass. Furthermore, the wire-frame representation (bottom row) shows that the triangles near the protruded edge are smaller, because the neighborhood balls are dissolved earlier in this region of higher curvature.

Another example of increased accuracy for multiple scan passes is shown in Fig. 11. The reconstructed surface of a piece of paper with holes of different size is shown after the first, the second and the third scan pass. While after the first scan pass two small holes are still closed by the triangulation procedure, after the third scan pass all holes are correctly detected.

To demonstrate the efficiency of the proposed method, we list in Table 1 the sizes of the models on the finest level in terms of data points d_i and vertices v_i and the times spend for the different computations: number of data points processed per second, number of vertices processed per second, and the overall times for the computation of the triangulation and the scanning process. The times for the

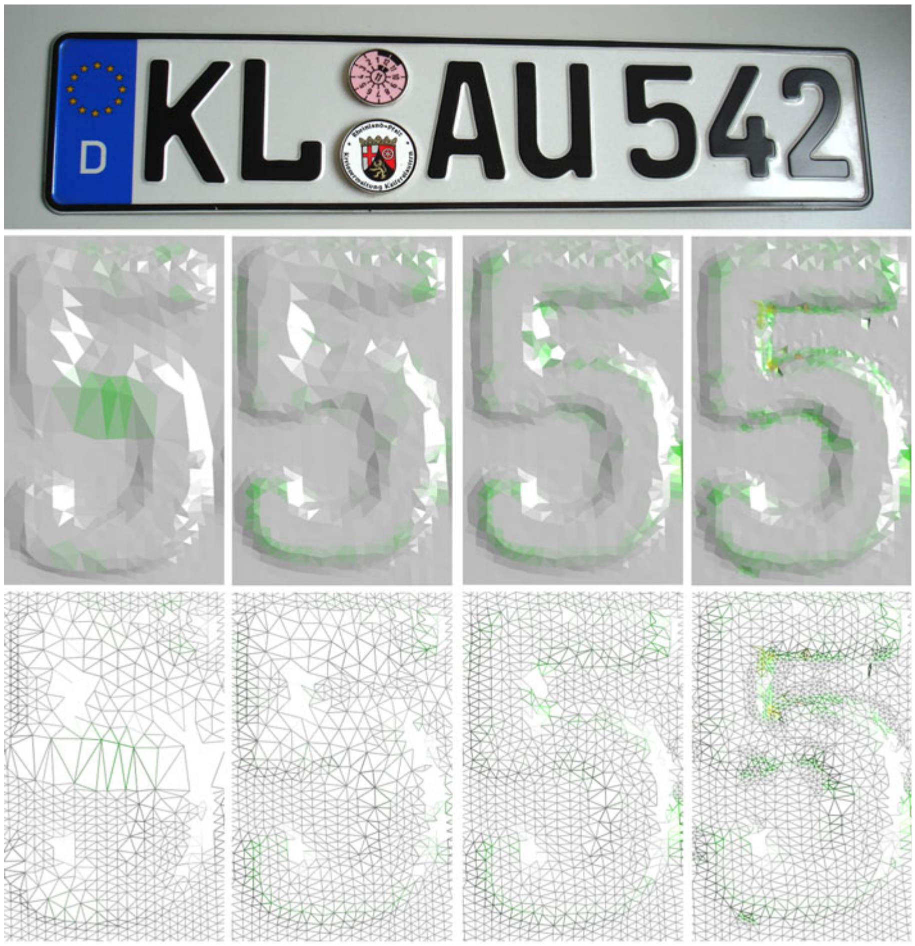

Fig. 10 Flat shading (*top*) and wire-frame (*bottom*) of triangulation of a license plate after 250, 2,000, 5,000, 9,000 scan lines with uncertainty visualization

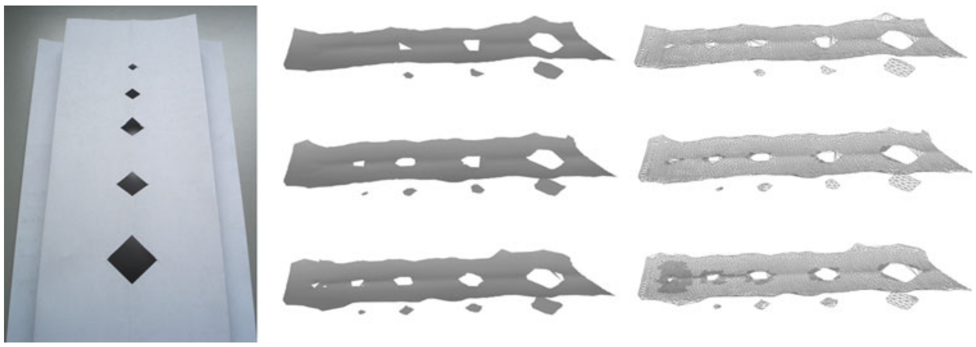

Fig. 11 Smooth shading and wire-frame representation of triangulation of a sheet of paper with holes after 585, 1,895, and 3,149 scan lines

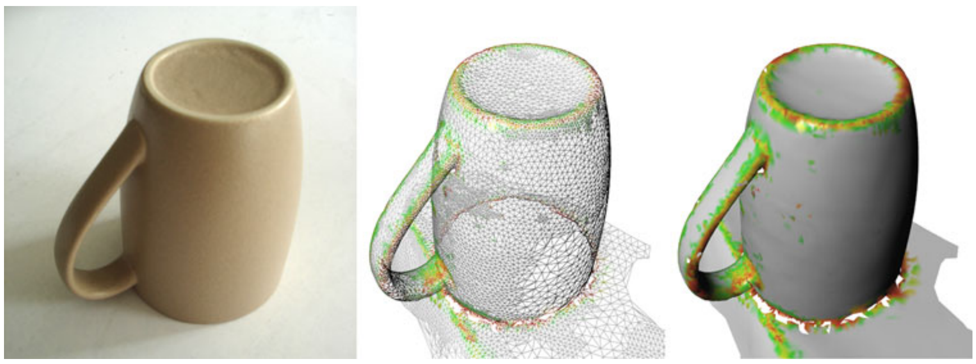

Fig. 12 A simple mug. Original object (*left*), wire-frame (*center*), smooth-shaded triangulation with uncertainty visualization (*right*)

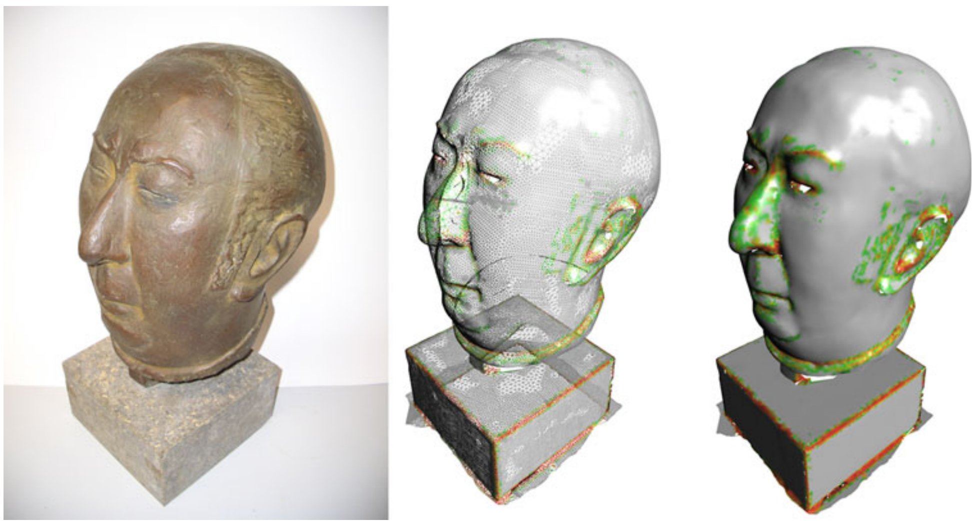
Fig. 13 Gerhard Marcks-bronze bust “Bildnis Theodor Heuss” [19]. Original object (*left*), wire-frame (*center*), smooth-shaded triangulation with uncertainty visualization (*right*)

Table 1 Table of number of data points d_i and vertices v_i in the final triangulation, times for processing data points and vertices, overall time for the computation of the triangulation and the scanning process for all models presented in this paper

	Data points	Vertices	Data points per second	Vertices per second	CPU time (s)	Scan time (s)
Piggy bank (Fig. 1)	1,280,387	30,130	4,198	98.9	304.6	318
License plate (Fig. 10)	1,866,413	29,703	4,392	69.9	424.8	504
Sheet of paper (Fig. 11)	478,900	2,538	7,483	39.9	63.6	105
Mug (Fig. 12)	358,232	8,866	5,778	143.7	61.7	105
Theodor Heuss (Fig. 13)	2,451,014	35,412	4,300	62.1	570.0	623
Bronze bird [9]	181,731	6,247	4,432	152.0	41.1	143

scanning process do not include the pauses between scan passes. It is apparent from Table 1 that our method works in real time even for complex objects. A video of a live scanning process is available at [8].

12 Conclusion and outlook

Our experiments show that the proposed method is suitable to assist the operator of a hand-held laser scanner to produce fast and complete high-quality triangulations satisfying all constraints A.–G. of Sect. 1. The online visualization helps to reduce the time for the manual scanning process significantly. The final surface mesh is a correct triangulation that can be used without further post-processing for measurement, surface analysis or reverse engineering.

The robustness of our method is derived from the fact that the human operator can increase the point density by additional scan passes in regions that are not yet reconstructed correctly or where important features are missing.

Nevertheless, there are some requirements for the proposed method to work satisfactorily. Only those parts of the object that are covered by scan lines can be reconstructed. Because of the interactive rendering the human operator can easily detect uncovered regions, fill the remaining holes and scan a topologically correct reconstruction of the target object. Regions of the object that cannot be scanned, because of occlusions or limitations of the scan arm, cannot be reconstructed and remain as holes in the triangulation.

For the future we plan to cover the following aspects.

- A different method for meshing and re-meshing similar to [10] could avoid some unnecessary mesh modifications. This would further improve the performance of our method.
- The collection of vertices in Sect. 7.1 searches on a much larger scale than the used octree (see Sect. 5) is intended for. Using a second octree with larger cubes containing the same neighborhood balls could improve the performance of the search in this step. We will experiment if this gain justifies the effort of maintaining two octrees simultaneously.
- A thread-safe mesh-implementation could allow the rendering thread to do dirty reads. This would reduce the blocking with the meshing thread completely. Furthermore, extensive parallelization of octree and mesh operations could improve the overall performance very much on modern computer hardware.

Acknowledgments This work was supported by DFG IRTG 1131 "Visualization of large and unstructured data sets." We also thank Faro Europe for lending out their "Laser ScanArm" and "Pfalzgalerie Kaiserslautern" and "Vereinigung Pfälzer Kunstfreunde (VKP)" for their generous support with the sculpture [19]. In particular, we thank Peter Salz for his help on the implementation of the visualizations of the optimal scanning direction.

References

1. Akkiraju N, Edelsbrunner H, Facello M, Fu P, Mücke E, Varela C (1995) Alpha shapes: definition and software. In: 1st International computational geometry software workshop, pp 63–66
2. Alexa M, Behr J, Cohen-Or D, Fleishman S, Levin D, Silva C (2003) Computing and rendering point set surfaces. IEEE Trans Vis Comput Graphics 9(1):3–15
3. Allègre R, Chaine R, Akkouche S (2007) A streaming algorithm for surface reconstruction. In: Symposium on Geometry Processing, pp 79–88
4. Amenta N, Choi S, Kolluri R (2001) The power crust. In: 6th ACM symposium on solid modeling and applications, pp 249–266
5. de Berg M, van Kreveld M, Overmars M, Schwarzkopf O (2000) Computational geometry. Springer, Berlin
6. Bodenmüller T, Hirzinger G (2004) Online surface reconstruction from unorganized 3d-points for the DLR hand-guided scanner system. In: 2nd Symposium on 3D data processing, visualization and transmission, pp 285–292
7. Chaine R (2003) A geometric convection approach of 3-d reconstruction. In: Sympoisum on geometry processing, pp 218–229
8. Denker K, Lehner B, Umlauf G (2008) Live scanning video. http://cg.cs.uni-kl.de/denker/scanning.avi
9. Denker K, Lehner B, Umlauf G (2008) Online triangulation of laser-scan data. In: Garimella R (ed) 17th International Meshing Roundtable, pp 415–432
10. Dyer R, Zhang H, Möller T (2007) Delaunay mesh construction. In: Symposium on geometry processing, pp 273–282
11. Edelsbrunner H, Kirkpatrick D, Seidel R (1983) On the shape of a set of points in the plane. IEEE Trans Inf Theory 29(4):551–559
12. Edelsbrunner H, Mücke E (1994) Three-dimensional alpha shapes. ACM Trans Graph 13(1):43–72
13. Faro Europe GmbH & Co. KG (2008) http://www.faro.com
14. Hjelle Ø, Dæhlen M (2006) Triangulations and applications. Springer, Berlin
15. Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W (1992) Surface reconstruction from unorganized points. Comput Graph 26(2):71–78
16. Jolliffe I (2002) Principal component analysis. Springer, Berlin
17. Kolluri R, Shewchuk JR, O'Brien J (2004) Spectral surface reconstruction from noisy point clouds. In: Symposium on geometry processing, pp 11–21
18. Lorensen WE, Cline HE (1987) Marching cubes: a high resolution 3d surface construction algorithm. SIGGRAPH Comput Graph 21(4):163–169
19. Marcks G (1952) Bildnis Theodor Heuss. In: "Pfalzgalerie Kaiserslautern", permanent loan from "Vereinigung Pfälzer Kunstfreunde" (VPK). Bronze sculpture, 31.5 × 21 × 23.5 cm
20. Prautzsch H, Boehm W, Paluszny M (2002) Bezier and B-spline techniques. Springer, Berlin
21. Schneider J, Scheidegger CE, Fleishman S, Silva CT (2006) Direct (re)meshing for efficient surface processing. Comp Graph Forum 25(3):527–536
22. Smith O (1961) Eigenvalues of a symmetric 3 × 3 matrix. Comm ACM 4:168
23. Teichmann M, Capps M (1998) Surface reconstruction with anisotropic density-scaled alpha shapes. In: IEEE conference on visualization, pp 67–72