Coarse-to-fine approximation of range images with bounded error adaptive triangular meshes

Angel D. Sappa
Edifici O Campus UAB
Computer Vision Center
08193 Bellaterra, Barcelona, Spain
E-mail: angel.sappa@cvc.uab.es

Miguel A. Garcia
Autonomous University of Madrid
Department of Informatics Engineering
Cra. Colmenar Viejo, Km. 15
28049 Madrid, Spain

Abstract. *A new technique for approximating range images with adaptive triangular meshes ensuring a user-defined approximation error is presented. This technique is based on an efficient coarse-to-fine refinement algorithm that avoids iterative optimization stages. The algorithm first maps the pixels of the given range image to 3D points defined in a curvature space. Those points are then tetrahedralized with a 3D Delaunay algorithm. Finally, an iterative process starts digging up the convex hull of the obtained tetrahedralization, progressively removing the triangles that do not fulfill the specified approximation error. This error is assessed in the original 3D space. The introduction of the aforementioned curvature space makes it possible for both convex and nonconvex object surfaces to be approximated with adaptive triangular meshes, improving thus the behavior of previous coarse-to-fine sculpturing techniques. The proposed technique is evaluated on real range images and compared to two simplification techniques that also ensure a user-defined approximation error: a fine-to-coarse approximation algorithm based on iterative optimization (Jade) and an optimization-free, fine-to-coarse algorithm (Simplification Envelopes).* © *2007 SPIE and IS&T.* [DOI: 10.1117/1.2731824]

1 Introduction

A range image is a digital image (2D array) in which each pixel keeps a measure related to the distance (range) from a range sensor to a point on a surface that is being observed by the sensor. Range images are rapidly gaining popularity as an efficient source of 3D information for a wide variety of computer vision applications due to the increasing availability of commercial range sensors that are fast, accurate, and affordable. However, dense range images are highly redundant representations in which, for instance, thousands of pixels can be utilized to represent large planar areas. The burden of processing all those pixels can be significantly lessened if more compact representations, such as adaptive triangular meshes,[1] are utilized to approximate the surfaces of the objects present in the original range images.

Adaptive triangular meshes allow us to model the surfaces represented in a range image with triangles of variable size. The size of the triangles is related to the local curvature of the surfaces to be modeled. Thus, small triangles are utilized to represent areas of high curvature, while smooth regions are approximated by rather large triangles. Subsequent processing can then be accelerated by carrying it out at a higher abstraction level, by dealing with triangles instead of with individual pixels (e.g., Refs. 2–6).

The problem of approximating a range image with an adaptive triangular mesh can be cast as the more general problem of approximating a surface modeled by a dense triangular mesh, which has been extensively studied in computer graphics and image processing with the purpose of approximating closed surfaces corresponding to individual objects (e.g., Refs. 7 and 8) as well as open surfaces corresponding to height fields (e.g., terrain[9]). A dense triangular mesh can be trivially obtained from a range image by linking neighboring points along rows, columns, and diagonals. Only one of the two diagonals (left or right) is chosen. Alternatively, a dense triangular mesh can be computed from a set of unorganized points.[10] In this case, no additional information is required other than the space coordinates of the points. In both cases, these dense meshes can then be simplified by applying any of the techniques developed for both closed surfaces and height fields.

Two alternative goals can be pursued when adaptive triangular meshes are generated. The first goal (*bounded size approximation*) consists of obtaining meshes with a predefined number of points. The second goal (*bounded error approximation*) consists of generating meshes whose approximation error with respect to the original mesh is bounded by a given value known as *tolerance*. In this case, the number of points contained in the generated mesh is unknown in advance, although the aim is to satisfy the error constraint with a minimum or, at least, a reduced number of points. A few of the many surface approximation algorithms that have been proposed in the literature can guarantee that the adaptive triangular meshes they generate

Paper 06043RR received Mar. 16, 2006; revised manuscript received Nov. 14, 2006; accepted for publication Dec. 5, 2006; published online Apr. 24, 2007.
1017-9909/2007/16(2)/023010/11/$25.00 © 2007 SPIE and IS&T.

have a real approximation error with respect to the original model below a given bound.

The fastest surface approximation algorithms, such as *Qslim*[11] and *Out-of-Core*,[12] ensure a given number of vertices (bounded size) but not an approximation error. For instance, Ref. 11 characterizes the geometric error between the decimated mesh and the original one through a heuristic value based on a quadric error metric. On the other hand, Ref. 12 applies a vertex clustering and repositioning technique that does not consider the error with which the original surface is approximated. These techniques are especially designed for real-time visualization of very large meshes. Recently, Ref. 13 has proposed another real-time visualization technique for large models. However, similarly to Refs. 11 and 12 the final triangular mesh does not ensure a user-defined approximation error, since it is based on a quadric error metric.

Independently of the final goal, approximation algorithms can be classified into three broad categories depending on the way they proceed: *fine-to-coarse, coarse-to-fine*, and *subsampling* techniques. Fine-to-coarse (also known as *decimation*) techniques start with a dense triangular mesh and, at every iteration, remove a point or subset of points that are chosen based on a certain approximation error or energy criterion (e.g., the point whose removal leads to a minimum error increase is deleted[14,15]). Alternatively, coarse-to-fine (also known as *refinement*) techniques start with a few triangles and, at every iteration, they add a point or subset of points that are chosen according to some approximation error or energy criterion (e.g., the point whose inclusion leads to the largest error reduction is added[16,17]). Finally, subsampling techniques directly start with a triangular mesh with the desired number of points (e.g., Refs. 12 and 18). Those points are conveniently placed in order to adapt to the curvatures associated with the original surfaces being approximated.

The majority of fine-to-coarse, coarse-to-fine, and subsampling techniques apply some type of iterative optimization procedure. Most of the proposals are based on *discrete optimization*, in the sense that they choose for inclusion or removal the point or points that best satisfy a given optimality criterion at every iteration. Additionally, some proposals apply *continuous optimization*, by adjusting the positions of the chosen points with the aim of improving the quality of the final approximation. A few optimization-free techniques have also been proposed. By avoiding optimization, the approximation process can become more efficient. Actually, only two of the reviewed techniques are free of optimization stages: a fine-to-coarse technique (*Simplification Envelopes*[19]) and a coarse-to-fine one.[20] The other algorithms apply some kind of optimization, whether they are fine-to-coarse (e.g., *Jade*[21]) or coarse-to-fine (e.g., Ref. 22) techniques.

This paper presents a new coarse-to-fine technique for approximating a given range image with an adaptive triangular mesh of bounded approximation error without applying iterative optimization stages. It is important to emphasize that this algorithm is applicable only to height fields, not to closed surfaces, and it works independently of the technology used during the scanning stage, provided the whole surface has been uniformly scanned with a static sensor. The proposed algorithm first maps the pixels of the given range image to 3D points defined in a "curvature space" (the curvature space can be thought of as a curvature image in which the range originally associated with each pixel is substituted for a measure of its local curvature). The points defined in the curvature space are then tetrahedralized with a 3D Delaunay algorithm.[23] Finally, an iterative process starts digging up (sculpturing) the convex hull of the obtained tetrahedralization by progressively removing triangles that do not fulfill the specified user-defined approximation error with respect to the original 3D triangular mesh. Throughout that iterative process, the approximation error is *validated* (determined) in the original 3D space. The generated meshes can be progressively refined whenever necessary by iterating from the previously obtained result. Thus, any intermediate solution between the coarsest mesh and the finest one can be generated.

The utilization of the 3D Delaunay triangulation as the basis for coarse-to-fine mesh approximation techniques is not new. It was originally introduced by Boissonnat[24] in order to determine the surface from which a dense cloud of 3D points were sampled. The basic idea of his algorithm consists of tetrahedralizing the input points by applying the 3D Delaunay algorithm. The resulting convex hull is then successively "sculptured" by removing external tetrahedra from it until a triangular surface that contains all the original 3D points is reached. No approximation error is considered, since the goal is to obtain the highest-resolution triangular mesh that contains all the input points. Afterwards, the same algorithm was applied to the reconstruction of polyhedral surfaces from 3D data obtained by stereo vision.[25] Again, no approximation error was considered since the highest-resolution mesh was sought.

The organization of this paper is as follows. Section 2 introduces conventions and definitions utilized throughout this work. The proposed optimization-free, course-to-fine technique is described in Sec. 3. Section 4 shows experimental results with real range images and the comparison with public implementations of two techniques that also guarantee a given tolerance: a fine-to-coarse technique based on discrete optimization (*Jade*[21]) and an optimization-free, fine-to-coarse technique (*Simplification Envelopes*[19]). Conclusions and further improvements are presented in Sec. 5.

2 Conventions and Definitions

A range image is generally a rectangular sampling of the surfaces present in an observed scene. Its usual representation is a two-dimensional array $\mathbf{R}$, where each array element $\mathbf{R}(r,c)$, $r \in [0,R)$, and $c \in [0,C)$, is a scalar that represents a surface point $\mathbf{P}_{rc}$ of coordinates: $\mathbf{P}_{rc} = (f_x(r), f_y(c), f_z(\mathbf{R}(r,c)))$ is referred to a local coordinate system associated with the range sensor. The definition of the scaling functions f_x, f_y, and f_z depends on the properties of the actual range sensor being utilized—they could be trivially defined as the identity function: $\mathbf{P}_{rc} = (r,c,\mathbf{R}(r,c))$. The set of $\mathbf{P}_{rc}$ points constitutes a *3D range image*.

In general, $\mathbf{R}(r,c)$ can be considered to be the distance between a surface point and a *reference plane* that is orthogonal to the axis of the range sensor and placed opposite it at a specified distance. Invalid points in the image will be considered to have the background value β. The *viewing*

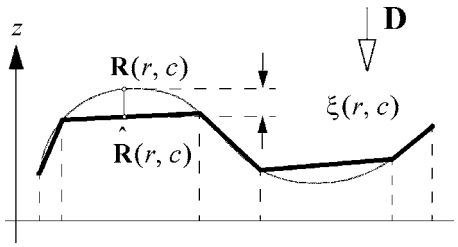

Fig. 1 Computation of the approximation error $\xi(R,C)$ corresponding to a range image point $\mathbf{R}(r,c)$. The thick polygonal line represents a 2D section of the 3D triangular mesh M that approximates the original range image $\mathbf{R}$. Vector $\mathbf{D}$ represents the viewing direction of the range sensor.

direction $\mathbf{D}$ is aligned with the z-axis of the local coordinate frame and points toward the scene. Thus, the vector $\mathbf{D}$ can be defined as $\mathbf{D}=(0,0,-1)$.

A *3D triangular mesh* is a piecewise linear surface consisting of triangular faces connected along their edges. Formally, a 3D triangular mesh M is a set $\{V,T\}$, where $V=\{v_1,\ldots,v_m\}$; $v_i \in R^3$ is a set of vertex positions that define the shape of the mesh in R^3; and T is a description of the mesh topology. Each vertex is defined by three coordinates: (x, y, and z).

The approximation error $\xi(r,c)$ associated with each range image point $\mathbf{R}(r,c)$ is defined as the distance between that point and a point $\hat{\mathbf{R}}(r,c)$ contained in the adaptive triangular mesh: $\xi(r,c)=|\mathbf{R}(r,c)-\hat{\mathbf{R}}(r,c)|$. $\hat{\mathbf{R}}(r,c)$ is obtained as the intersection between the 3D triangular mesh M and a straight line orthogonal to the reference plane of $\mathbf{R}$ and passing through point $\mathbf{R}(r,c)$ (Fig. 1).

3 Generation of Bounded Error Triangular Meshes from Range Images

This section presents a coarse-to-fine algorithm to compute a triangular mesh M that approximates a given range image $\mathbf{R}$, such that every point of M is within a maximum distance τ (tolerance) of some point of $\mathbf{R}$. The flowchart in Fig. 2 illustrates the algorithm's stages.

Broadly speaking, the algorithm consists of three main stages. In the first stage, the original range image is mapped to a 3D image space whose reference frame is attached to the range sensor. Thus, each pixel $\mathbf{R}(r,c)$ of the range image is converted to a 3D point $\mathbf{P}_{rc}$. The set of those 3D points constitutes a 3D range image. Afterwards, each $\mathbf{P}_{rc}$ is associated with a curvature value obtained by estimating the surface that would pass through that point in the 3D range image. Taking these curvatures into account, all the points $\mathbf{P}_{rc}$ of the 3D image space are mapped to a 3D curvature space.

Fig. 3 Refinement effect of sculpturing: each exterior triangle (i.e., tetrahedron) removed from a tetrahedralization leads to the addition of three new triangles and a new vertex to the external surface.

The second stage of the algorithm tetrahedralizes the points defined in the curvature space through a 3D Delaunay triangulation algorithm. The effect of applying the tetrahedralization in the curvature space instead of in the 3D image space is that the coarsest (most external) triangles in the tetrahedralization tend to join distinctive points (points of high curvature), which are the ones that usually define the shape of the objects being approximated. Hence, working in the curvature space allows us to obtain a coarse representation in the 3D image space in a fast and straightforward way.

Finally, in the third stage, an iterative process starts digging up (sculpturing) the triangular mesh that constitutes the external surface of the previous tetrahedralization (its convex hull), removing those exterior triangles that do not fulfill a given user-defined approximation error. Each removed triangle is substituted for the three triangles that belong to its corresponding tetrahedron (see Fig. 3).

While the digging process is carried out in the tetrahedralization generated in the curvature space, the process that determines whether or not the external triangles fulfill the desired tolerance (*validation process*) is performed in the 3D image space. Therefore, the external mesh of the tetrahedralization is successively refined until all its triangles satisfy the approximation error when they are considered in the 3D image space. These three stages are further described below.

3.1 *3D Curvature Space Mapping*

As mentioned, all points $\mathbf{P}_{rc}$ originally defined in the 3D image space are mapped to a 3D curvature space that is then tetrahedralized. The benefits of introducing this curvature space are twofold: on the one hand, that space naturally highlights the points that convey the largest shape information (the ones that belong to areas of high curvature) and those that can be made redundant as they belong to planar regions. On the other hand, the proposed curvature space allows us to deal with both convex and nonconvex surfaces in a similar way. Otherwise, if the subsequent tetrahedralization were performed in the original 3D space, while nonconvex surfaces were correctly triangulated at

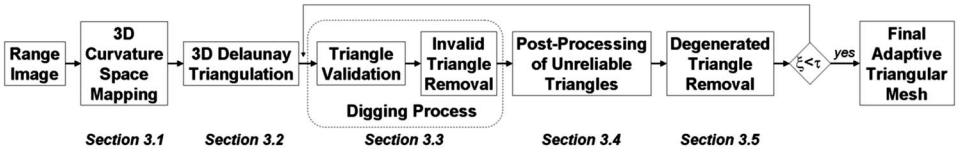

Fig. 2 Illustration of the algorithm's stages.

different resolution levels, convex surfaces would always be triangulated at the finest resolution, rendering thus the subsequent carving process useless.

Initially, the surfaces of the objects present in the given range image are approximated through a trivial triangulation of the 3D range image points, by joining them along the rows, columns, and diagonals associated with their corresponding pixels in the range image. By convention, one of the two diagonals (e.g., the right diagonal) is always chosen. This triangular mesh will be referred to as the original triangular mesh.

An estimation of the curvature of every point $\mathbf{P}_{rc}=(x,y,z)$ of the 3D range image is then computed as $K_{rc}=|8z-\Sigma_{\forall i}z_i|$, where z_i denotes the z-coordinate of each of the eight neighbors of $\mathbf{P}_{rc}$. A more accurate curvature estimation, such as the one proposed in Ref. 3 could be alternatively used. However, experiments have shown that the slight improvement in accuracy obtained with the latter does not justify the significant increase in CPU time that is attained.

Once the curvature K_{rc} associated with every point $\mathbf{P}_{rc}$ has been estimated, each $\mathbf{P}_{rc}$ is mapped to a point $\hat{\mathbf{P}}_{rc}=(\hat{x},\hat{y},\hat{z})$ defined in the following 3D curvature space:

$$\hat{x}=\alpha r,$$

$$\hat{y}=\alpha c,$$

$$\hat{z}=\log(1+\Re+K_{rc}^2), \tag{1}$$

with $\Re$ being a random real number ranging between 0 and 1, and being a constant necessary to avoid degenerated tetrahedra due to precision errors (good tetrahedralizations were obtained with $\alpha>20$). The random component is necessary to avoid degenerated tetrahedra in areas of constant curvature, in which all points lie on the same plane when they are mapped to the 3D curvature space.

The new points $\hat{\mathbf{P}}_{rc}$ obtained above are tetrahedralized by applying a 3D Delaunay triangulation algorithm.[23] The digging process applied to the convex hull obtained as a result of the Delaunay algorithm should eventually proceed until reaching the original triangular mesh, since the latter represents the lowest possible approximation error (tolerance $\tau=0$). In order to guarantee that the tetrahedralization generated after applying Delaunay contains this original triangular mesh, which is a triangulation that joins adjacent points in the 3D range image, each $\hat{\mathbf{P}}_{rc}$ must be scaled along its $\hat{z}$-direction by a constant factor γ:

$$\gamma=\frac{\alpha\sqrt{7}}{\Delta\hat{z}_{\max}}, \tag{2}$$

with $\Delta\hat{z}_{\max}$ being the maximum vertical distance along the $\hat{z}$-direction between every $\hat{\mathbf{P}}_{rc}$ and its eight neighbors (see Fig. 4), by considering all the points $\hat{\mathbf{P}}_{rc}$ contained in the 3D curvature space, and α being the constant utilized in the definition of $\hat{\mathbf{P}}_{rc}$, which represents the distance between two consecutive points along both the $\hat{x}$- and $\hat{y}$-directions. Thus, $\hat{\mathbf{P}}_{rc}$ is finally defined as $\hat{\mathbf{P}}_{rc}=(\alpha r,\alpha c,\gamma\log(1+\Re+K_{rc}^2))$.

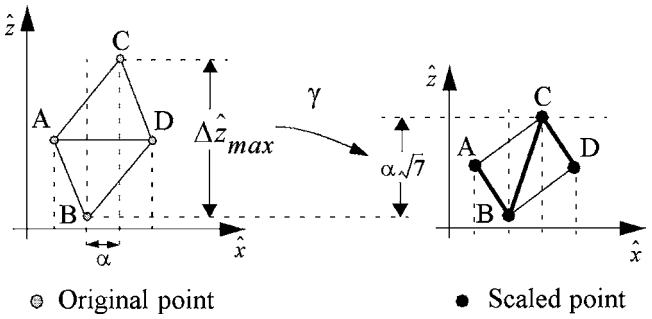

Fig. 4 (left) 2D representation of four consecutive points (along the $\hat{x}$-direction) in the curvature space and their Delaunay triangulation. (right) The previous points scaled down in $\hat{z}$ by a factor γ and their corresponding triangulation. The thickened line represents segments that join adjacent points.

In order to understand the definition of γ, let us suppose that four neighboring points (A, B, C, D) are defined in a 2D space, as shown in Fig. 4 (left). Since the Delaunay triangulation algorithm maximizes the minimum angle of the triangles it generates and since α is a constant, the vertical distance $\Delta\hat{z}$ between two consecutive points must be modified in order to guarantee that the Delaunay algorithm generates a triangulation that links every pair of consecutive points, as shown in Fig. 4 (right).

The worst case occurs at the limit position in which the Delaunay algorithm produces a swap between segments AD and BC. According to the *Delaunay criterion*, this happens when both segments have the same length (see Fig. 5): $\overline{\mathrm{AD}}=\overline{\mathrm{BC}}$. By definition (Fig. 4), $\overline{\mathrm{BC}}^2=\alpha^2+\Delta z_{\max}^2$ and $\overline{\mathrm{AD}}=3\alpha$. At the worst case, since $\overline{\mathrm{BC}}=\overline{\mathrm{AD}}$, the previous expression can be rewritten as $9\alpha^2=\alpha^2+\Delta z_{\max}^2$. Hence, the worst case occurs when $\Delta\hat{z}_{\max}=\alpha\sqrt{8}$. Therefore, if $\Delta\hat{z}_{\max}=\alpha\sqrt{7}$, it is guaranteed that $\overline{\mathrm{AD}}>\overline{\mathrm{BC}}$ and, hence, the Delaunay algorithm will always join adjacent points B and C instead of the nonadjacent ones A and D [Fig. 4 (right) and Fig. 5 (right)]. The definition of the scaling factor γ as the fraction introduced above ensures that the maximum $\Delta\hat{z}$ ($\Delta\hat{z}_{\max}$) is mapped to the upper bound $\alpha\sqrt{7}$ when considering all the points $\hat{\mathbf{P}}_{rc}$, while the other $\Delta\hat{z}$ are mapped to new values below that bound.

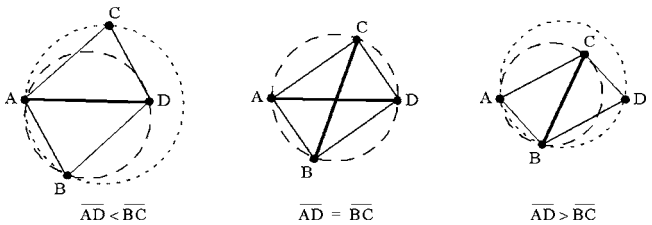

Fig. 5 2D Delaunay triangulation of four points (the distance between A and D is the same in the three examples). According to the Delaunay criterion, the circumcircle of a triangle can contain only the three vertices of that triangle. The example in the middle corresponds to an ambiguous triangulation in which both diagonals ($\overline{\mathrm{AD}}$ and $\overline{\mathrm{BC}}$) are possible.

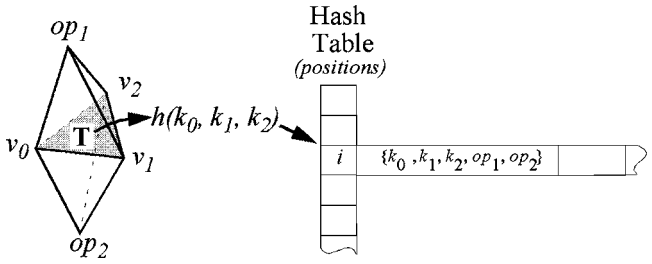

Fig. 6 Information related to triangle **T**, which is stored in position *i* of the hash table.

3.2 *3D Delaunay Triangulation of Points in the Curvature Space*

At this stage, the points $\hat{\mathbf{P}}_{rc}$ defined in the 3D curvature space are tetrahedralized by using a 3D Delaunay algorithm[23] that produces a list of tetrahedra. Each of the four triangles that constitute every tetrahedron is defined by three identifiers, (v_0,v_1,v_2), which denote three different points. Each point has two alternative representations: one in the image space, $\mathbf{P}_{rc}$, and another in the curvature space, $\hat{\mathbf{P}}_{rc}$.

In order to speed up further operations, all triangles obtained above are inserted into a *hash table*. Hash tables are data structures that allow the storage of large amounts of records identifiable by a certain numeric or alphanumeric key, guaranteeing that those records can be retrieved in constant asymptotic time $O(1)$. In our case, the hash table is defined by a vector with B entries, where B is the prime number closest to the total number of triangles obtained by the 3D Delaunay algorithm. A triangle $\mathbf{T}=(v_0,v_1,v_2)$ is stored at the entry determined by the following hash function:

$$h(k_0,k_1,k_2)=\left(k_0+k_1\frac{B}{3}+k_2\frac{B^2}{4}\right)\bmod B, \tag{3}$$

with (k_0,k_1,k_2) being the permutation of (v_0,v_1,v_2) that satisfies $k_0<k_1<k_2$. This ordering is chosen to guarantee that the same triangle will be stored at the same entry of the hash table, regardless of the order of its vertices. The application of the hash function may produce collisions (different triangles being assigned to the same entry). Therefore, a list of triangles is kept at each entry of the hash vector. Fig. 6 illustrates a hash table.

A triangle **T** belongs to two tetrahedra at most. Thus, it is associated with two *opposite points*, (op_1,op_2), which constitute the apices of those tetrahedra. As a particular case, triangles that belong to the external surface of the tetrahedralization are only contained in a single tetrahedron. Those triangles are referred to as *exterior triangles*. After applying the 3D Delaunay algorithm, the set of exterior triangles constitutes the convex hull of the tetrahedralization.

Every exterior triangle $\mathbf{T}=(v_0,v_1,v_2)$ is associated with a unitary normal vector **N** orthogonal to the plane defined by the three points $\mathbf{P}_{rc}$ that constitute **T**. The vector **N** is oriented in such a way that it points toward the half-space that does not contain the vertex opposite **T**. This vertex is the apex of the tetrahedron whose base is **T**.

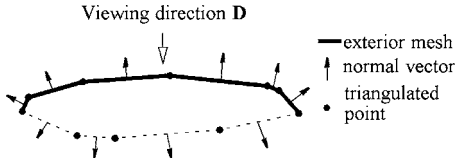

Fig. 7 2D section of the convex hull and the exterior mesh associated with it for a set of points represented in the 3D image space.

In order to reduce computations, all triangles stored in each entry of the hash table keep: (1) the list of their vertex identifiers in ascending order (k_0,k_1,k_2); (2) their opposite point identifiers (op_1,op_2), and (3) the normal **N** in case of triangles that are exterior. Notice that since exterior triangles belong to only one tetrahedron, they have only a valid opposite point identifier. In this situation, one of the two identifiers will be a negative integer. Hence, it is very efficient to determine whether or not a certain triangle is exterior or given the indices to its vertices.

As we are dealing with range images obtained from a certain viewing direction **D**, only those exterior triangles whose normals have an angle with respect to **D** greater than 90 deg belong to the surfaces of the objects present in the image. The exterior triangles that fulfill the previous condition form the *exterior mesh* (Fig. 7).

The algorithm proceeds by digging the original tetrahedralization in the curvature space, starting with the triangles contained in the first exterior mesh (the convex hull) until all the triangles contained in the current exterior mesh fulfill the specified user-defined approximation error when they are considered in the 3D image space. This digging process is described next.

3.3 *Digging Process*

Let M be the exterior mesh obtained as the union of exterior triangles determined as described in Sec. 3.2. We will consider this mesh to be *validated* and, therefore, that a valid solution to the approximation problem has been found when all its triangles have an approximation error less than or equal to the specified tolerance τ. That approximation error is measured in the 3D image space.

The iterative digging process goes over all triangles of the exterior mesh M. If a triangle has an approximation error above τ, this triangle is marked for removal. After all exterior triangles have been considered, each triangle marked for removal is substituted for the other three triangles that belong to its tetrahedron. Thus, a new exterior mesh M is obtained and the process is iterated. These two steps are detailed next.

3.3.1 *Triangle validation*

The validation process is carried out in the 3D image space and consists of measuring the approximation error associated with each triangle of the current exterior mesh. The approximation error is the distance between a triangle and its farthest *control point* measured along the viewing direction **D**. The control points utilized to validate a triangle **T** are the points $\mathbf{P}_{rc}$ of the 3D range image whose projection along **D** onto the range image reference plane is contained in the projection of **T** onto the same plane (Fig. 8). Those control points can be quickly determined since they are a

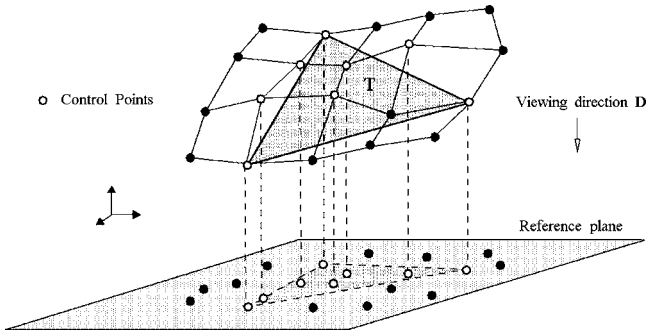

Fig. 8 Control points corresponding to a triangle **T** in the 3D image space.

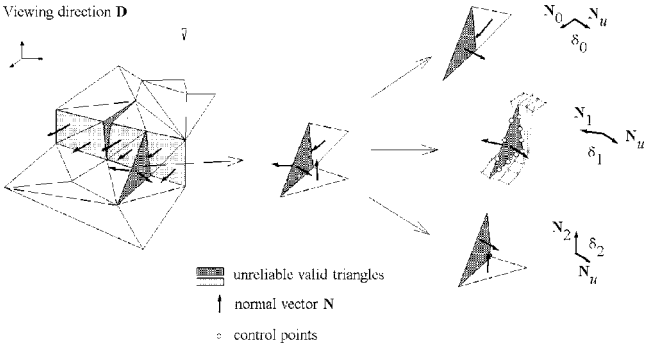

Fig. 10 Postprocessing of unreliable triangles. Unreliable triangles that should be removed are represented in dark gray, while triangles that will be marked as reliable are represented in light gray.

subset of the points contained in the bounding box of the projection of **T** onto the reference plane.

The validation of an exterior triangle **T** consists of verifying that the control points $\mathbf{P}_{rc}$ associated with **T** are located at a distance from **T** along the viewing direction below or equal to the tolerance τ. If any of these control points is farther away from **T** than τ, triangle **T** is considered to be invalid. As described in the next section, that triangle will be removed after all its current exterior triangles are validated. On the other hand, if all of **T**'s control points are located at a distance less than or equal to τ, **T** is considered to be valid (Fig. 9).

After an exterior triangle has been considered to be valid according to the previous procedure, it must be further classified as either *reliable* or *unreliable*. Recall that the triangular mesh has been generated in the curvature space, so it could happen that when a triangle is represented in the 3D image space it has only three control points or looks like a sliver that does not correspond to the original surface. Therefore, a valid triangle **T** is unreliable when it only has three control points, which correspond to the vertices of the triangle itself, or, alternatively, when the normal vector **N** associated with **T** is almost orthogonal to the viewing direction **D** (triangles with an angle between **N** and **D** less than 93 deg have been classified as unreliable in this work; this value is directly related to the sensor resolution and has been experimentally obtained). Unreliable triangles are not considered during the following iterations of the digging process. They have to undergo a postprocessing stage, described in Sec. 3.4, in order to determine whether or not they belong to the final solution. On the other hand, if the number of control points is greater than 3 and the angle between **N** and **D** is greater than or equal to 93 deg, **T** is marked as reliable. This implies that **T** already belongs to the final solution and will not be considered during the following digging iterations or the postprocessing stage.

The triangle validation stage is applied to all the triangles that belong to the current exterior mesh M. After it, triangles labeled as invalid if any are removed, as described next.

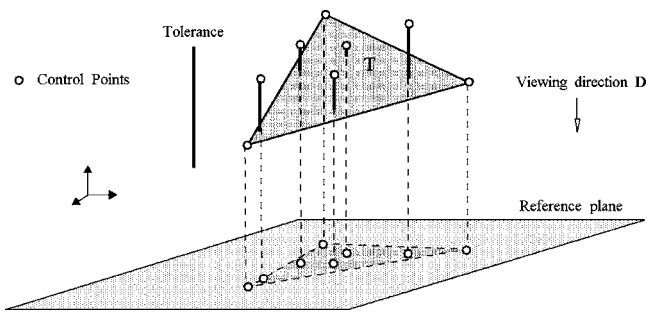

Fig. 9 Triangle **T** is *valid*: the vertical distances from **T** to its control points in the 3D image space are less than the specified tolerance.

3.3.2 *Invalid triangle removal*

The aim of this stage is to remove all the triangles of the current exterior mesh M that have been classified as invalid by the previous stage. Those are the triangles whose distance with respect to the 3D range image is above the specified tolerance.

Given an invalid exterior triangle **T** with indices (v_0, v_1, v_2), the identifier of its opposite point op_i is extracted from the hash table. That identifier corresponds to the apex of the tetrahedron that contains **T**. Triangle **T** is then substituted in the current exterior mesh M for the other three triangles that form its tetrahedron $\{(v_0, v_1, op_i), (v_1, v_2, op_i), (v_2, v_0, op_i)\}$. If any of those new triangles is already contained in the exterior mesh, the new triangle is not inserted and the existing one is removed from the mesh. The normal vector associated with each new exterior triangle is computed, and its corresponding entry in the hash table is updated.

If, after having removed all invalid triangles from the exterior mesh M, no new triangles have been included, the algorithm proceeds by checking all valid triangles that have been classified as unreliable, if any. Otherwise, if new exterior triangles have been included, the digging process starts over from the triangle validation stage (Sec. 3.3.1).

3.4 *Postprocessing of Unreliable Triangles*

This stage is responsible for determining whether the unreliable triangles contained in the current exterior mesh M must be included in the final solution or discarded. A triangle is unreliable either when it has three control points (its vertices) or the angle of its normal **N** with respect to the viewing direction **D** is close to 90 deg. Figure 10 illustrates two different types of unreliable triangles (convex and nonconvex silvers).

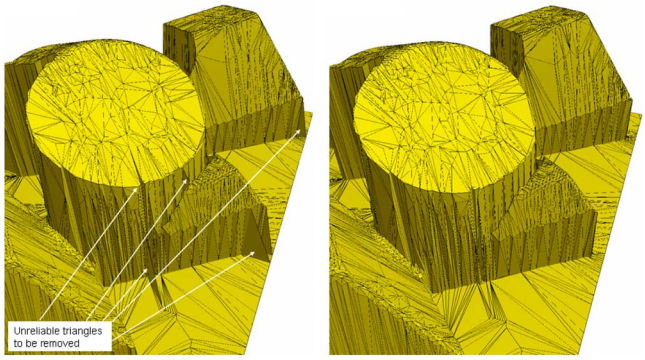

Fig. 11 (left) Final triangular mesh containing valid triangles (both reliable and unreliable). (right) The previous triangular mesh after the postprocessing stage (all its triangles are reliable).

Given an unreliable triangle $\mathbf{T}_u=(v_0,v_1,v_2)$, the algorithm proceeds by selecting the three triangles, $\{\mathbf{T}_0,\mathbf{T}_1,\mathbf{T}_2\}$, that meet $\mathbf{T}_u$ along one of its edges (alternatively, two triangles in case $\mathbf{T}_u$ has two exterior vertices or a single triangle when the three vertices of $\mathbf{T}_u$ are exterior). Next, the angles between the normal vector $\mathbf{N}_u$ associated with $\mathbf{T}_u$ and each of the normal vectors $\mathbf{N}_i$ associated with its neighboring triangles are computed: $\delta_i=angle(\mathbf{N}_u,\mathbf{N}_i)$, $i=\{0,1,2\}$.

Notice that the aim of this stage is to find triangle configurations that look like either convex or nonconvex slivers, which do not correspond to the scanned surface. These configurations are easily detected and removed by checking the angles δ_i. If all those δ_i are below a certain threshold (it has been experimentally set to 160 deg in the current version), triangle $\mathbf{T}_u$ is classified as reliable, since it is assumed to belong to the surface. Otherwise, if any of those angles is above the given threshold, a fine validation process is applied to $\mathbf{T}_u$.

The fine validation process consists of decomposing the unreliable triangle to be checked, $\mathbf{T}_u$, into a list of nine control points distributed along its edges (see Fig. 10). The distance from those points to the original triangular mesh along the viewing direction is computed. The maximum distance computed in that way is considered to be the approximation error associated with $\mathbf{T}_u$. If that error is less than or equal to the given tolerance τ, $\mathbf{T}_u$ is marked as a reliable triangle, and, hence, it will belong to the final solution. Otherwise, this triangle is classified as invalid, and the digging process starts over from the invalid triangle removal stage (Sec. 3.3.2).

This postprocessing stage is applied to all the valid, unreliable triangles contained in the current exterior mesh M. Figure 11 (*left*) shows a zoom into a final triangular mesh containing some unreliable triangles. Figure 11 (*right*) shows the same area after the stage in the left part of Fig. 11 has been applied.

3.5 *Removal of Degenerated Triangles*

The last stage of the proposed algorithm applies a verification process to all the triangles that form the exterior triangular mesh. At this point, these triangles are both valid and reliable. This process is necessary since some of those triangles can be *degenerated* in the 3D image space. This means that their area will be null due to the alignment of their three vertices. The latter may occur due to the fact that the Delaunay triangulation algorithm has been carried out in the 3D curvature space; hence, some triangles obtained in that space could be degenerated when they are represented in the 3D image space.

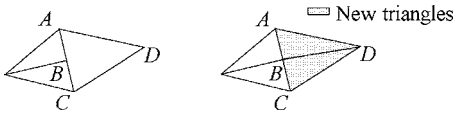

Fig. 12 Removal of degenerated triangle *ABC* and retriangulation of *ACD*.

In order to handle such degenerated triangles throughout the various stages of the digging process described in previous sections, their normal vectors **N** will be considered to be null. Moreover, all tests in which a normal vector intervenes will fail when the latter is null. In this way, the triangle validation stage (Sec. 3.3.1) will classify degenerated triangles as valid and unreliable, since they only have three control points that correspond to their vertices. Afterwards, the postprocessing stage (Sec. 3.4) will apply the fine validation process, which will eventually assess whether or not the distance of the degenerated triangles to the original triangular mesh is below the tolerance and, thus, whether those degenerated triangles are valid or invalid.

The final verification process removes every valid, degenerated triangle and retriangulates the triangle adjacent to it along its longest edge. This retriangulation process generates two new triangles for every retriangulated triangle, both of which lie on the same plane as the latter. For that reason, the approximation error is kept unchanged. Figure 12 illustrates the process of removing null-area triangles. Triangle *ABC* is removed, and two new triangles are generated; *ABD* and *BCD*. These triangles occupy the same area as the original triangle, *ACD*.

4 Experimental Results

The proposed technique has been tested with real range images from the OSU (MSU/WSU) database.[26] The figures included in this section show the original (dense) triangular meshes corresponding to the given range images as well as the adaptive triangular meshes computed from them. In all those meshes, triangles that correspond to regions of the range image that contain invalid pixels (shadows, sensor errors, etc.) have not been displayed. In particular, if the centroid of a triangle projects onto an invalid pixel of the given range image, that triangle has not been drawn.

3D Delaunay triangulations have been performed with *Qhull*, a public implementation of the *Quickhull* algorithm,[23] which is publicly available from the Geometry Center at the University of Minnesota (http://www.qhull.org). All CPU times have been measured on a SGI Indigo II with a 175-MHz R10000 processor.

The computational complexity of *Quickhull* is $O(n \log v)$ when applied in both 2D and 3D, with n the number of input points and v the number of vertices in the convex hull. Computing the 3D Delaunay triangulation of the input points is the hardest stage of the proposed algorithm. The remaining stages are much faster. Thus, the original points are mapped to the curvature space in $O(n)$

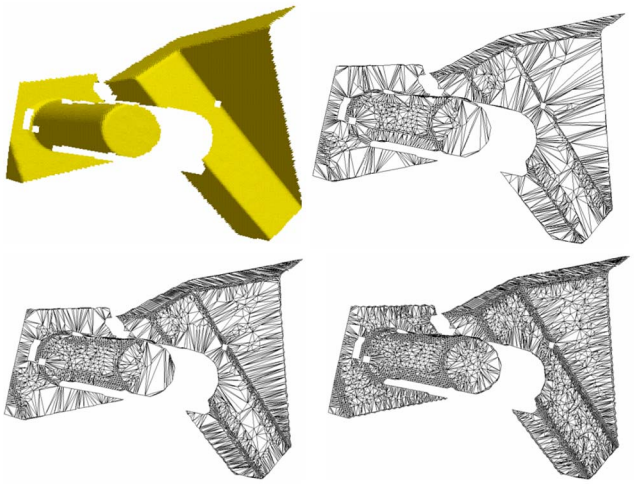

Fig. 13 (top left) Original (rendered) range image whose associated dense triangular mesh contains 21,452 triangles. (top right) Final exterior mesh with 2,759 triangles (τ=0.11%). (bottom left) Final exterior mesh with 4,959 triangles (τ=0.035%). (bottom right) Final exterior mesh with 8,268 triangles (τ=0.011%).

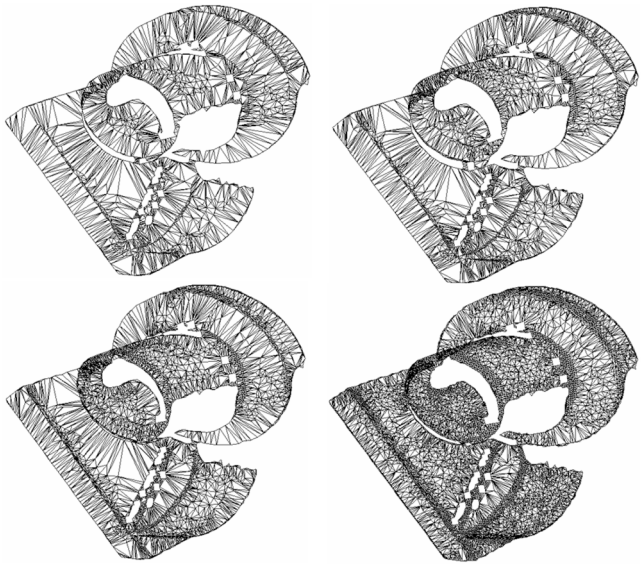

Fig. 14 Several adaptive triangular meshes that approximate a range image whose dense triangular mesh contains 26,028 triangles. (top left) Final exterior mesh with 2,697 triangles (τ =0.28%). (top right) Final exterior mesh with 4,069 triangles (τ =0.14%). (bottom left) Final exterior mesh with 5,590 triangles (τ =0.069%). (bottom right) Final exterior mesh with 11,158 triangles (τ=0.013%).

time. After the 3D triangulation is applied, the digging (sculpturing) stage removes subsequent triangles from the external surface of the tetrahedralization. For every triangle that is removed, three new triangles are added in constant time. The use of a BSP tree allows the determination of the approximation error of any triangle in constant time. Hence, since the number of triangles in the external surface is $O(n)$, the whole digging process runs in $O(n \log n)$ asymptotic time. In practice, if a coarse resolution mesh is sought, the number of iterations of the digging process will be relatively low. Therefore, the actual time corresponding to the digging process is significantly smaller than the 3D triangulation time in practice. Finally, the postprocessing stage runs in $O(n)$ time. This means that the computational cost of the proposed technique is $O(n \log n)$, with the 3D triangulation stage being the one that, in practice, consumes the largest percentage of the processing time.

Figure 13 (*top left*) shows a range image whose original triangular mesh contains 21,452 triangles. The mapping to the curvature space was done in 1.33 s. The 3D Delaunay triangulation took 47.69 s. The hash table was loaded in 4.75 s. Figure 13 (*top right*) shows the final triangular mesh that approximates the original range image with a user-defined approximation error τ of 10 units, which corresponds to 0.11% of the maximum z-value of the given range image. This triangular mesh contains 2,759 triangles, which were obtained after 44 digging iterations. The CPU time of the digging process was 6.67 s. The postprocessing stage took 0.72 s, and 65 triangles with null areas were finally removed in 0.1 s.

Figure 13 (*bottom left*) shows another triangular approximation of the same original range image with 4,959 triangles. Its approximation error τ was set to 3 units, which corresponds to 0.035% of the maximum z-value. The CPU time for performing the 45 digging iterations was 11.12 s. Unreliable triangles were checked in 0.89 s. The 134 degenerated triangles that were detected were removed in 0.19 s. Figure 13 (*bottom right*) shows a final exterior mesh containing 8,268 triangles. It was obtained after 56 iterations in 20.72 s. Its approximation error τ was set to 1 unit, which corresponds to 0.011% of the maximum z-value. The postprocessing stage took 1.05 s, and 208 triangles with null areas were removed in 0.57 s.

Figure 14 shows four adaptive triangular meshes that approximate the same range image at different resolutions. The dense triangular mesh corresponding to that range image contains 26,028 triangles. The CPU time for mapping the points of that mesh to the 3D curvature space was 0.93 s. The 3D Delaunay triangulation of those points in the curvature space took 31.56 s and the hash table was loaded in 3.28 s.

Figure 14 (*top left*) shows the final triangular mesh obtained after 41 digging iterations. It approximates the original range image with an error of 0.28% with respect to the maximum z-value. The digging process took 5.58 s, and the postprocessing stage ran in 1.66 s. Finally, 58 degenerated triangles were removed in 0.06 s. Figure 14 (*top right*) shows another approximation with 4,069 triangles and a user-defined approximation error of 0.14%. This triangular mesh was obtained after 34 digging iterations in 7.05 s. Unreliable triangles were checked in 1.33 s. Sixty-three triangles with null areas were removed in 0.07 s.

Figure 14 (*bottom left*) shows an adaptive triangular mesh that approximates the original range image with a user-defined approximation error of 0.069% of the maximum z-value. This mesh was obtained after 39 digging iterations. The CPU time for computing these iterations was 9.51 s. Unreliable triangles were checked in 0.96 s. Eighty-six degenerated triangles were removed in 0.15 s. Finally, Fig. 14 (*bottom right*) shows the triangular mesh obtained after 37 iterations in 23.43 s. It has a user-defined approximation error of 0.013%. Unreliable triangles were checked

Fig. 15 (top left) Original (rendered) range image whose associated dense triangular mesh contains 27,798 triangles. (top right) Final exterior mesh with 2,628 triangles (τ=0.13%). (bottom left) Final exterior mesh with 5,630 triangles (τ=0.039%). (bottom right) Final exterior mesh with 9,545 triangles (τ=0.013%).

in 1.31 s, and 166 degenerated triangles were removed in 0.55 s.

Figure 15 shows three triangular approximations corresponding to the range image shown in Fig. 15 (*top left*). The original triangular mesh contains 27,798 triangles. The CPU time to perform the curvature space mapping was 1.27 s. The 3D Delaunay triangulation was computed in 43.21 s, and the hash table was loaded in 4.51 s. Figure 15 (*top right*) shows the final triangular mesh obtained after 39 digging iterations in 5.90 s. It contains 2,628 triangles. Its approximation error was set to 10 units, which corresponds to 0.13% of the maximum z-value of the given 3D range image. The postprocessing stage took 0.95 s. Finally, 42 degenerated triangles were removed in 0.05 s. Figure 15 (*bottom left*) shows another approximation of the same range image. It has a user-defined approximation error of 0.039% and contains 5,630 triangles. This final representation was obtained after 52 digging iterations in 12.74 s. Unreliable triangles were checked in 1.27 s, and 136 triangles with null areas were removed in 0.31 s. Figure 15 (*bottom right*) shows an approximation with a user-defined error of 0.013%, obtained after 58 digging iterations. The final triangular mesh contains 9,545 triangles, which were obtained in 24.01 s. The postprocessing stage took 0.96 s, and 200 degenerated triangles were finally removed in 0.78 s.

All the aforementioned experimental results are summarized in Table 1. For each of the adaptive triangular meshes shown above, the following data are displayed: (1) number of triangles in the original dense mesh; (2) 3D triangulation time (including mapping to curvature space, 3D Delaunay triangulation, and loading of hash table); (3) specified tolerance (user-defined approximation error); (4) number of triangles in the final adaptive mesh; (5) sculpturing time (including digging process and removal of unreliable and degenerated triangles); (6) total CPU time (3D triangulation+sculpturing). The tolerance is given as a percentage of the vertical distance between the points that in the original 3D range image are farthest away along the viewing direction.

The proposed technique has been compared with two public iterative decimation algorithms that also guarantee a user-defined approximation error between the final mesh and the original model: *Jade*,[21] which is an optimization-based technique, and *Simplification Envelopes*,[19] which is

Table 1 Summary of experimental results for each of the adaptive triangular meshes generated with the proposed technique.

Final Adaptive Mesh	Original # Triangles	3D Triang. Time (s)	Tolerance	Final # Triangles	Sculpt. Time (s)	Total Time (s)
Fig. 13 (*top right*)	21,452	53.77	0.11%	2,759	7.49	61.26
Fig. 13 (*bottom left*)	21,452	53.77	0.035%	4,959	12.20	65.97
Fig. 13 (*bottom right*)	21,452	53.77	0.011%	8,268	22.34	76.11
Fig. 14 (*top left*)	26,028	35.77	0.28%	2,697	7.30	43.07
Fig. 14 (*top right*)	26,028	35.77	0.14%	4,069	8.45	44.22
Fig. 14 (*bottom left*)	26,028	35.77	0.069%	5,590	10.62	46.39
Fig. 16 (*left*)	26,028	35.77	0.041%	6,787	13.07	48.84
Fig. 14 (*bottom right*)	26,028	35.77	0.013%	11,158	25.29	61.06
Fig. 15 (*top right*)	27,798	49.00	0.13%	2,628	6.90	55.90
Fig. 15 (*bottom left*)	27,798	49.00	0.039%	5,630	14.32	63.32
Fig. 15 (*bottom right*)	27,798	49.00	0.013%	9,545	25.75	74.75

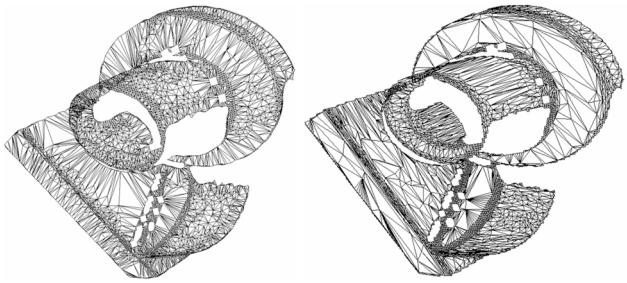

Fig. 16 (left) Final exterior mesh computed with the proposed technique. In total, 6,787 triangles were obtained after 31 digging iterations in 48 s. (right) Adaptive triangular mesh obtained with *Jade*. Here 5,205 triangles were generated in 142 s.

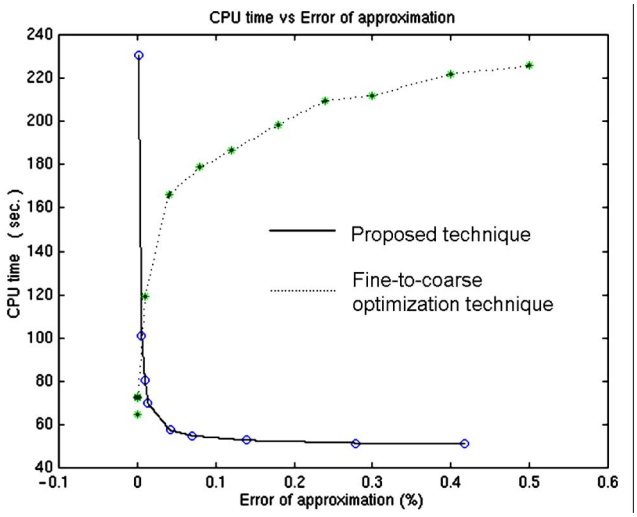

Fig. 17 Approximation error versus CPU time for the range image used in Fig. 14, by considering the proposed coarse-to-fine technique and a fine-to-coarse optimization technique (*Jade*).

optimization-free. For example, the 3D range image corresponding to the examples shown in Fig. 14 was approximated with the three methods using the same user-defined approximation error, $\tau=0.041\%$. *Jade* generates a triangular mesh that approximates the 3D range image with 5,205 triangles. The result, in Fig. 16 (*right*), is slightly better than the one obtained with the proposed technique, in Fig. 16 (*left*), but *Jade* takes 142.7 s, while the proposed technique takes 48.84 s instead. The final exterior mesh obtained with the proposed technique contains 6,787 triangles. *Simplification Envelopes* is much more inefficient than *Jade*, since it generates a similar approximation with 5,495 triangles in 1,315 s. Note that although there is a small difference in the number of triangles contained in the final meshes, the most significant difference lies in the required CPU time. Therefore, the efficiency of the techniques in order to compute the required representations is used as the comparison criterion.

In order to determine when a coarse-to-fine technique, such as the one proposed in this paper, is more advantageous than a fine-to-coarse approximation technique, such as *Jade*, the original 3D range image utilized in Fig. 14 was approximated with different adaptive triangular meshes corresponding to various tolerances, by applying both the proposed technique and *Jade*. It is assumed that both techniques generate similar results; hence, comparisons are based on CPU time. The geometry and number of triangles of the final meshes are similar in general, since, in all cases, they have to fit the original cloud of points according to the same user-defined approximation error. Figure 17 shows CPU times versus approximation errors for both algorithms.

As expected, when an adaptive triangular mesh with a very low approximation error is required, the fine-to-coarse technique is much faster than the proposed technique. The reason is that the final exterior mesh is closer to the original dense mesh, which is the starting point of the fine-to-coarse algorithm. On the other hand, when a rather coarse adaptive triangular mesh is required, the proposed technique is more efficient than the fine-to-coarse approach; in this case, the solution is closer to the coarse mesh (the convex hull) utilized as the starting point of the iterative digging process applied by the proposed technique. Hence, fewer iterations are necessary in order to reach the desired solution.

5 Conclusions and Further Improvements

A new coarse-to-fine algorithm for generating bounded-error triangular meshes from range images without optimization has been presented. This technique maps the pixels of a range image into a 3D curvature space. After tetrahedralizing those points with a 3D Delaunay algorithm, a digging process starts eroding the external surface of the resulting tetrahedralization until a mesh that fulfills the desired tolerance is obtained. The approximation error of that external surface is verified in the 3D image space.

By avoiding iterative optimization stages, the proposed technique is advantageous with respect to previous coarse-to-fine techniques based on iterative optimization, such as Refs. 17 and 22. Those techniques refine an initial triangular mesh of low resolution by adding one vertex at a time. At each iteration, the vertex that produces the highest reduction of approximation error must be found and inserted into the mesh, making them computationally intensive. Moreover, by working in a 3D curvature space, the proposed technique allows the adaptive approximation of both convex and nonconvex surfaces. This improves the behavior of previous coarse-to-fine sculpturing methods based on the 3D Delaunay triangulation, such as Refs. 20 and 24. By tetrahedralizing in the original 3D space instead of in the proposed curvature space, the latter methods approximate convex surfaces with the highest resolution, disregarding their curvature.

Experimental results have shown that the proposed technique is more efficient than two well-known bounded-error simplification algorithms whenever a rather coarse resolution mesh is sought: a fine-to-coarse approach based on iterative optimization (*Jade*[21]) and an optimization-free technique (*Simplification Envelopes*[19]). Obviously, since the proposed iterative digging process starts refining the coarsest-resolution triangular mesh (the convex hull), the proposed technique will require fewer iterations to reach a relatively coarse resolution mesh than iterative techniques that start with the highest-density triangular mesh and proceed by removing one point at a time.

In general, coarse-to-fine approaches are more appropriate than fine-to-coarse ones for applications that require the generation of low-resolution representations that are subsequently refined only when necessary. This is a common procedure in order to accelerate a wide variety of algorithms in different fields, such as collision detection and path planning in robotics, scene visualization and ray tracing in computer graphics, or model-based image recognition in computer vision.

As experimental results show, planar surfaces tend to be oversampled due to the presence of noise in the input range images, especially when low tolerances are specified. This undesired effect is likely to be reduced by applying discontinuity-preserving filtering techniques, such as in Ref. 27. Further work is necessary to evaluate the impact of these techniques on the whole approximation process.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Education and Science under projects TRA2004-06702/AUT and DPI2004-07993-C03-03. The first author was supported by The Ramón y Cajal Program.

References

1. P. Heckbert and M. Garland, "Survey of polygonal surface simplication algorithms," in *Multiresolution Surface Modeling Course, SIGGRAPH'97*, Available online at ftp://ftp.cs.cmu.edu/afs/cs/project/anim/ph/paper/multi97/release/heckbert/simp.pdf (1997).
2. I. Cheng and P. Boulanger, "Feature extraction on 3D texmesh using scale-space analysis and perceptual evaluation," *IEEE Trans. Circuits Syst. Video Technol.* **15**(10), 1234–1244 (2005).
3. P. Csákány and A. Wallace, "Representation and classification of 3D objects," *IEEE Trans. Syst. Man Cybern.* **33**(4), 638–647 (2003).
4. K. Park, K. Lee, and U. Lee, "Models and algorithms for efficient multiresolution topology estimation of measured 3D range data," *IEEE Trans. Syst. Man Cybern.* **33**(4), 706–711 (2003).
5. Y. Sun, J. Paik, A. Koschan, D. Page, and M. Abidi, "Point fingerprint: A new 3D object representation scheme," *IEEE Trans. Syst. Man Cybern.* **33**(4), 712–717 (2003).
6. Z. Yan, S. Kumar, and J. Kuo, "Mesh segmentation schemes for error resilient coding of 3D graphic models," *IEEE Trans. Circuits Syst. Video Technol.* **15**(1), 138–144 (2005).
7. M. Andreetto, N. Brusco, and G. Cortelazzo, "Automatic 3D modeling of textured cultural heritage objects," *IEEE Trans. Image Process.* **13**(3), 354–369 (2004).
8. G. Guidi, J. Beralding, and C. Atzeni, "High-accuracy 3D modeling of cultural heritage: The digitizing of Donatello's 'Maddalena'," *IEEE Trans. Image Process.* **13**(3), 370–380 (2004).
9. R. Whitaker and E. Juarez-Valeds, "On the reconstruction of height functions and terrain maps from dense range data," *IEEE Trans. Image Process.* **11**(7), 704–716 (2002).
10. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *Proc. SIGGRAPH'92*, pp. 71–78 (1992).
11. M. Garland and P. Heckbert, "Surface simplification using quadric error metrics," *Proc. SIGGRAPH'97*, pp. 209–216 (1997).
12. P. Lindstrom, "Out-of-core simplification of large polygonal models," in *Proc. SIGGRAPH'00*, pp. 259–262 (2000).
13. Y. Zhu, "Uniform remeshing with an adaptive domain: A new scheme for view-dependent level-of-detail rendering of meshes," *IEEE Trans. Vis. Comput. Graph.* **11**(3), 306–316 (2005).
14. H. Hoppe, "Progressive meshes," *Proc. SIGGRAPH'96*, 99–108 (1996).
15. P. Lindstrom and G. Turk, "Evaluation of memoryless simplification," *IEEE Trans. Vis. Comput. Graph.* **5**(2), 98–115 (1999).
16. D. Brodsky and B. Watson, "Model simplification through refinement," *Proc. Graphics Interface*, pp. 221–228 (2000).
17. L. De Floriani and E. Puppo, "Hierarchical triangulation for multiresolution surface description," *ACM Trans. Graphics* **14**(4), 363–411 (1995).
18. M. Garcia and A. Sappa, "Efficient generation of discontinuity-preserving adaptive triangulations from range images," *IEEE Trans. Syst. Man Cybern.* **34**(5), 2003–2014 (2004).
19. J. Cohen *et al.*, "Simplification envelopes," *Proc. SIGGRAPH'96*, pp. 119–128 (1996).
20. L. De Floriani, P. Magillo, and E. Puppo, "On-line space sculpturing for 3d shape manipulation," *Proc. 15th IAPR Intl. Conf. Patt. Recog.*, pp 105–108 (2000).
21. A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno, "Multiresolution decimation based on global error," *The Visual Computer* **13**(5), 228–246 (1997).
22. L. De Floriani, "A pyramidal data structure for triangle-based surface description," *IEEE Comput. Graphics Appl.* **9**(2), 67–78 (1989).
23. C. Barber, D. Dobkin, and H. Huhdanpaa, "The Quickhull algorithm for convex hulls," *ACM Trans. Math. Softw.* **22**(4), 469–483 (1996).
24. J. Boissonnat, "Geometric structures for three-dimensional shape representation," *ACM Trans. Graphics* **3**(4), 266–286 (1984).
25. O. Faugeras, E. LeBras-Mehlman, and J. Boissonnat, "Representing stereo data with the Delaunay triangulation," *Artif. Intell.* **44**(2), 41–87 (1990).
26. OSU (MSU/WSU), "Range image database," available online at http://sampl.eng.ohiostate.edu/sampl/data/3DDB/RID/inddex.htm.
27. S. Li, "Highclose-form solution and parameter selection for convex minimization-based edge-preserving smoothing," *IEEE Trans. Pattern Anal. Mach. Intell.* **20**(9), 916–932 (1998).

Angel D. Sappa received his electromechanical engineering degree in 1995 from National University of La Pampa, General Pico-La Pampa, Argentina, and his PhD degree in industrial engineering in 1999 from Polytechnic University of Catalonia, Barcelona, Spain. From 1999 to 2002 he undertook postdoctorate research at LAAS-CNRS, Toulouse, France, and at Z+F UK Ltd., Manchester, UK. From September 2002 to August 2003, he was with the Informatics and Telematics Institute, Thessaloniki, Greece, as a Marie Curie Research Fellow. Since September 2003 he has been with the Computer Vision Center, Barcelona, Spain, as a Ramón y Cajal Research Fellow. His research interests are focused on range image analysis, 3D modeling, and model-based segmentation.

Miguel A. Garcia received his BS, MS, and PhD degrees in computer science from Polytechnic University of Catalonia, Barcelona, Spain, in 1989, 1991, and 1996, respectively. He joined the Department of Software at the Polytechnic University of Catalonia in 1996 as an assistant professor. In 1997 he joined the Department of Computer Science and Mathematics at, Rovira i Virgili University, Tarragona, Spain, as an associate professor, as head of Intelligent Robotics and Computer Vision Group. In 2006 he joined the Department of Informatics Engineering at Autonomous University of Madrid, where he is currently associate professor. His research interests include image processing, 3D modeling, and mobile robotics.