Efficient Generation of Discontinuity-Preserving Adaptive Triangulations from Range Images

Miguel Angel Garcia, *Member, IEEE* and Angel Domingo Sappa, *Member, IEEE*

***Abstract*—This paper presents an efficient technique for generating adaptive triangular meshes from range images. The algorithm consists of two stages. First, a user-defined number of points is adaptively sampled from the given range image. Those points are chosen by taking into account the surface shapes represented in the range image in such a way that points tend to group in areas of high curvature and to disperse in low-variation regions. This selection process is done through a noniterative, inherently parallel algorithm in order to gain efficiency. Once the image has been subsampled, the second stage applies a two and one half-dimensional Delaunay triangulation to obtain an initial triangular mesh. To favor the preservation of surface and orientation discontinuities (jump and crease edges) present in the original range image, the aforementioned triangular mesh is iteratively modified by applying an efficient edge flipping technique. Results with real range images show accurate triangular approximations of the given range images with low processing times.**

***Index Terms*—Adaptive triangular meshes, discontinuity-preserving triangulation, range images, three-dimensional (3-D) shape representation and recovery.**

I. Introduction

Computer vision has traditionally relied on intensity image processing for obtaining three-dimensional (3-D) shapes. Different techniques have been proposed for that purpose, such as shape-from-focus, shape-from-stereo, or shape-from-shadows, to mention a few. However, this process is conceptually involved since, in the end, 3-D information must be inferred from two-dimensional (2-D) projections.

A simpler and more direct approach for obtaining shape information of 3-D objects consists of utilizing *range images*. A range image is a 2-D array of pixels. Each pixel does not represent a light intensity level but the distance from a point on the surface of a 3-D object to a virtual plane referred to the range sensor utilized to acquire the image. The ability to directly obtain 3-D shapes and the availability of increasingly inexpensive and fast range sensors explain the popularity that range images are gaining in the robotics and computer vision communities.

Manuscript received June 19, 2003; revised January 19, 2003. This work was supported in part by the Government of Spain under the CICYT project DPI2001-2094-C03-02. The second author was supported by The Ramón y Cajal Program. An implementation in C of this algorithm is available by contacting the authors. This paper was recommended by Associate Editor D. Goldgof.

M. A. Garcia is with the Intelligent Robotics and Computer Vision Group, Department of Computer Science and Mathematics, Rovira i Virgili University, 43007 Tarragona, Spain (e-mail: magarcia@etse.urv.es).

A. D. Sappa is with the Computer Vision Center, Edifici O Campus UAB, 08193 Bellaterra, Barcelona, Spain (e-mail: angel.sappa@cvc.uab.es).

Digital Object Identifier 10.1109/TSMCB.2004.832156

Range images are dense representations containing tens of thousands of data points. Processing all those points is costly and that may hinder the utilization of range images in fields with real-time constraints, such as robotics, computer vision, or augmented reality. However, the processing associated with range images can be significantly reduced by converting the original dense images into efficient data representations able to keep the same surface shapes with fewer data, and then performing the actual processing upon those simplified representations. This strategy is advantageous not only in terms of processing speedup, but also in terms of storage efficiency, since those reduced representations will also be appropriate for keeping and updating world models of large and complex workspaces in an efficient way.

Adaptive triangular meshes are a popular data representation. They are advantageous since they can adapt to intricate shapes efficiently. Thus, large planar areas, which in a range image may occupy tens or hundreds of pixels, can be represented by a few triangles. In fact, a triangular mesh approximating a range image can be interpreted as a first level of abstraction of that image, since planar areas are identified and interrelated. Then, further processing algorithms (e.g., segmentation, integration, recognition) can take advantage of such preprocessing to perform more efficiently. For instance, [1] presents a fast technique for segmenting range images approximated by triangular meshes. Besides the speedup of further processing algorithms, triangular meshes are a convenient representation in order to integrate the surfaces described by the range image in a world model or to include them in computer-aided design (CAD) packages.

The common way to generate an adaptive triangular mesh from a dense range image is through a dense-to-coarse algorithm that consists of two stages: First, a dense (nonadaptive) triangular mesh is generated by connecting all the pixels of the range image along rows, columns and diagonals; second, the dense mesh is iteratively decimated (coarsened) based on a measure of its approximation error with respect to the original image (e.g., [2]–[6]) or some appearance metric (e.g., [7]–[9]). Recent developments (e.g., [6], [7], [10], and [11]) allow the simplification of very large triangular meshes at very high speeds. However, these techniques do not guarantee that surface and orientation discontinuities are preserved when the final meshes are relatively coarse.

Instead of starting with a dense triangular mesh that is then decimated. DeFloriani proposed a coarse-to-fine technique that starts with a coarse triangular mesh that is successively refined by iteratively adding new points [12]. This technique also suffers from efficiency problems due to the application of iterative optimization (at every iteration, the algorithm must select the triangle whose subdivision produces the largest reduction of approximation error with respect to the original image). Another coarse-to-fine approach was proposed in [13]. This method starts with a coarse mesh approximating the surface through triangular elements covering the boundary of the domain, then iteratively adds new points from the data set until a specified error tolerance is achieved.

A different approach was proposed by Terzopoulos and Vasilescu in [14]. Instead of starting with either a dense or coarse mesh and iteratively modifying it, they start with a uniformly distributed triangular mesh with the desired number of vertices. The positions of the vertices of this mesh are then iteratively displaced toward surface and orientation discontinuities detected from the image through a gradient-based operator. This technique is very efficient if the original mesh is small, but the relaxation process may be time consuming and expensive in case of larger meshes. That technique was extended in [15] in order to enhance the approximation of discontinuities. Besides moving positions of the mesh vertices, triangles can also be either subdivided or merged, adding thus further processing cost.

Adaptive triangular meshes that approximate data points of closed surfaces were proposed in [16]–[18]. The so-called Marching Cubes algorithm, introduced in [16], creates constant density triangular meshes from 3-D data points. Extensions to this algorithm were presented in [19] and [20]. Alternatively, balloon models were proposed in [17] and [18]. Both methods start with an initial small triangulated balloon placed inside the closed surface. Then, the balloon is inflated until it reaches the data points, approximating thus the shape of the 3-D model.

The concept of frequency was used in [21] to generate a triangular mesh. Thus, nodes are automatically placed in regions of the image containing high-frequency features, while coarse triangles are generated in smooth regions. A similar approach was proposed in [22] with the goal of obtaining a quadrilateral mesh from a given range image without applying any optimization stages (without decimations, coarsenings or point movements). According to this approach, the number of selected points and thus, the mesh size can be specified *a priori*. Besides, since the final mesh can be thought of (although it is not) the deformation of a uniform mesh, the shape of the obtained quadrilateral cells varies gently. Later on, an extension of that technique to obtain triangular meshes was proposed in [23]. The idea was to split each quadrilateral cell obtained above into two triangles by choosing the diagonal that mostly agrees with the discontinuities present in the range image.

However, the latter technique presents two basic problems: 1) the adaptation to discontinuities only considers the flip of the two diagonals inside each of the previously obtained quadrilateral cells. This means that the original edges of the quadrilateral mesh are not modified. Hence, those edges can cross surface and orientation discontinuities; 2) the sampling technique guarantees that all cells are quadrilateral, but does not prevent the appearance of degenerated or even twisted quadrilateral cells. Hence, degenerated and twisted triangles may also appear in the final mesh.

This paper presents a new approach for obtaining adaptive triangular meshes from range images without applying costly decimation, refinement, or point movement operations. The aim is the generation of a final mesh with the same number of points as if uniform sampling was applied, but obtaining a more accurate model of the surfaces present in the range image by concentrating more points in those areas that contain more surface details.

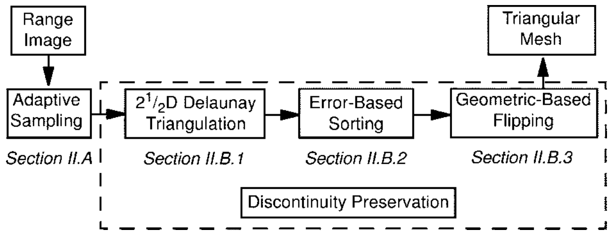

Fig. 1. Illustration of the algorithm's stages.

The algorithm consists of two stages. The first stage is a direct application of the deterministic sampling process introduced in [22] and [23], which adaptively selects a user-defined number of points by taking into account an estimation of the curvature associated with each pixel of the given range image. In the current work however, pixel curvatures are calculated through an extension of the technique utilized in [22], [23] by applying multiresolution. In that way, crease edges are better highlighted. The second stage meshes the previously sampled points to produce the final result. First, a two and one half-dimensional (2.5-D) Delaunay triangulation is applied to the set of selected points. Then an efficient iterative process flips triangle edges in order to preserve the discontinuities present in the original range image.

This paper is organized as follows. The proposed technique is described in Section II. Section III presents experimental results with real range images. Finally, conclusions and further improvements are given in Section IV.

II. Discontinuity-Preserving Adaptive Triangulation of Range Images

A range image is usually a rectangular sampling of a scene surface. Its typical representation is a 2-D array $\mathbf{R}$, where each element $\mathbf{R}(r,c)$ is a scalar that represents a surface point of coordinates: $(x,y,z) = (f_x(r), f_y(c), f_z(\mathbf{R}(r,c)))$ referred to a local coordinate system. The definition of f_x, f_y and f_z depends on the properties of the actual range sensor being utilized. In general, $\mathbf{R}(r,c)$ can be considered to be the distance between a surface point and a given *reference plane* which is orthogonal to the axis of the sensor and placed opposite it at a specified distance.

The proposed algorithm generates discontinuity-preserving adaptive triangular meshes from range images in two stages. In a first stage, the range image is adaptively sampled and a user-defined number of points is obtained. The second stage triangulates the previously sampled points through a conventional 2.5-D Delaunay algorithm and then applies an iterative process that flips triangle edges in order to preserve underlying jump and crease edges. Fig. 1 presents a chart flow illustrating the algorithm's stages, which are described below.

A. Adaptive Pixel Sampling

Let $\mathbf{R}(r,c)$, $r \in [0,R)$, and $c \in [0,C)$ be a range image with R rows and C columns in which each valid pair (r,c) denotes a pixel located at row r and column c. Pixels with no available depth information receive a constant value β that corresponds to the background.

The generation of a set of points whose density agrees with the features present in the image consists of three steps. First, an estimation of curvature is computed for every pixel of the given range image. Second, the range image is tessellated into a user-defined number of tiles that overlap along one line of pixels. Finally, an adaptive sampling process is separately run for each tile based on the range image and its curvature estimation. These steps are described next.

1) Gap Filling and Curvature Estimation: Given a range image $\mathbf{R}(r,c)$, a first stage removes single pixel gaps, usually due to sensor errors and inaccuracies. This is done by substituting each background pixel that is surrounded by nonbackground pixels for the average of those neighbors. Gaps larger than one pixel are not removed in considering that they are real gaps in the sensed 3-D surface. An optional filtering of the whole range image, by applying the same procedure to nonbackground pixels, has been discarded since it did not lead to significant improvements of the final mesh and also due to its tendency to remove details, such as sharp edges.

From the filtered range image $\mathbf{R}_f(r,c)$, a curvature image $\mathbf{K}(r,c)$ is computed. Each pixel in $\mathbf{K}(r,c)$ is a function of the curvature associated with its corresponding pixel $\mathbf{R}_f(r,c)$. The curvature image is obtained through a multiresolution process in order to highlight details, such as crease edges, sometimes difficult to perceive at full resolution but that can become apparent at lower resolutions. This process is described next.

Let $\mathbf{R}_f^1(r^1,c^1) = \mathbf{R}_f(r,c)$, $r^1 = r \in [0,R)$ and $c^1 = c \in [0,C)$, be an $R \times C$ full resolution range image and $\mathbf{R}_f^2(r^2,c^2)$, $r^2 \in [0,R/2)$ and $c^2 \in [0,C/2)$, an $R/2 \times C/2$ half resolution one obtained by sampling one out of two pixels from $\mathbf{R}_f^1(r^1,c^1)$. In general, a range image at a certain resolution $\mathbf{R}_f^i(r^i,c^i)$ with $i = 2^k$, $k > 0$, $r^i \in [0,R/i)$ and $c^i \in [0,C/i)$ is obtained from its corresponding double resolution image as $\mathbf{R}_f^i(r^i,c^i) = \mathbf{R}_f^{i/2}(2r^i,2c^i)$.

Let $\mathbf{K}^i(r^i,c^i)$ be the curvature estimation image computed from $\mathbf{R}_f^i(r^i,c^i)$ as indicated below. For instance, $\mathbf{K}^1(r^1,c^1) = \mathbf{K}(r,c)$ represents the full resolution curvature image, while $\mathbf{K}^2(r^2,c^2)$ is the half resolution one. For the sake of simplicity, we will describe the computation of $\mathbf{K}(r,c)$ from $\mathbf{R}_f(r,c)$. The same procedure is applied to images of any resolution.

First, horizontal $\mathbf{K}_{rc}^R$ and vertical $\mathbf{K}_{rc}^C$ curvature estimations are obtained as

$$\begin{aligned}\mathbf{K}_{rc}^R &= |\mathbf{R}_f(r-1,c) - 2\mathbf{R}_f(r,c) + \mathbf{R}_f(r+1,c)| \\ \mathbf{K}_{rc}^C &= |\mathbf{R}_f(r,c-1) - 2\mathbf{R}_f(r,c) + \mathbf{R}_f(r,c+1)|.\end{aligned}$$

The sought curvature is finally obtained as the addition of both estimations: $\mathbf{K}(r,c) = \mathbf{K}_{rc}^R + \mathbf{K}_{rc}^C$.

Pixels with an estimated curvature below a small threshold (e.g., 20) are set to that threshold, ensuring that these low curvature regions are also sampled with a minimum amount of points. By doing this, unnecessary concentrations of points along the boundary between valid regions and the background or between low and high variation regions are prevented from happening.

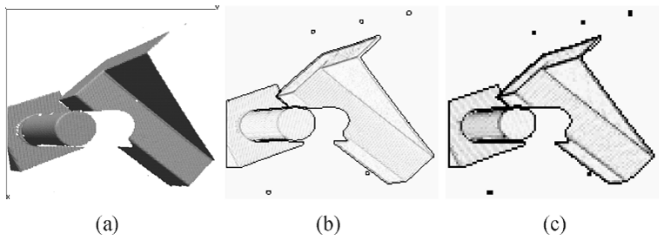
Fig. 2. (a) Original range image (rendered with perspective). (b) Curvature image from the filtered range image. (c) Curvature image after merging estimations at both full and half resolution.

Given the original (filtered) range image sampled at various resolutions, a curvature estimation image is computed for each resolution according to the aforementioned algorithm. The final objective is to obtain a curvature estimation at full resolution that merges the estimations obtained at the various levels of detail. After trying various schemes, we have opted for adding the different estimations without any specific weights that distinguish among the various resolutions. Given the size of range images dealt with (under 200 by 200 pixels), the curvature estimations at less than half the original resolution are too coarse and do not help improve the final result. Thus, the curvature estimation is finally obtained by adding the curvature estimations at the original and half resolutions: $\mathbf{K}(r,c) = \mathbf{K}^1(r,c) + \mathbf{K}^2(r/2,c/2)$.

Fig. 2 shows an example of the application of this technique. The left image is the rendered original range image. The middle image shows the curvature image at full resolution. Dark regions represent areas of high curvature. The right image shows the final result when the full and half resolutions are combined according to the proposed scheme. Notice how crease edges are better highlighted now. Since the remaining process maps curvatures into point densities, more points will be chosen along those edges, helping increase the accuracy with which those areas are modeled.

2) Range Image Tessellation: In order to obtain a final triangular mesh adapted to the different regions of the range image according to their particular characteristics, and also in order to favor parallelism, both the given range image and its associated curvature image are partitioned into a user-defined number of rectangular *tiles*. Each tile is considered to be a small range image upon which further processing is independently applied. In particular, each range image is divided into H horizontal and V vertical stripes, giving rise to $H \times V$ different tiles. The partition in tiles is not disjoint. A tile shares one row or column of pixels with each of its adjacent tiles. If the number of tiles is excessively large, the adaptive sampling degenerates into uniform sampling. Instead, if the number of tiles is very low, high curvature areas may produce inappropriate undersampling of distant regions. An intermediate number of tiles (e.g., 4 by 4) has shown to produce good results without compromising processing speed.

3) Curvature-Based Pixel Sampling: Given a range image tile $\mathbf{R}_{vh}(r,c)$, $v \in [0,V)$, $h \in [0,H)$, and its corresponding curvature image $\mathbf{K}_{vh}(r,c)$, the objective of this step is to sample

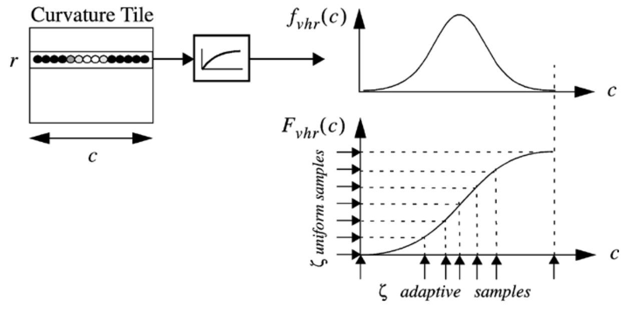

Fig. 3. (*Top*) Unnormalized probability density function $f_{vhr}(c)$ that represents curvature associated with every pixel c from a certain row r. (*Bottom-right*) Uniform sampling of the image space of the corresponding unnormalized probability distribution function $F_{vhr}(c)$ produces a set of points whose density varies according to $f_{vhr}(c)$.

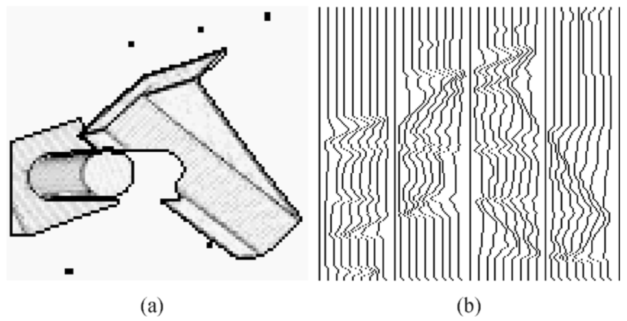

Fig. 4. (a) Curvature image. (b) Vertical curves after adaptive horizontal sampling, with $H = V = 4$ and $\zeta = 10$.

the given tile at $R \times \zeta$ positions (defined by the user) that cover the whole tile and adapt to the shape of the surfaces comprised in it.

In order to generate this array, each row of the given tile is adaptively sampled at ζ different positions so that points tend to concentrate in high-curvature areas and to disperse in low-variation ones. This is done as follows.

Let $\mathbf{R}_{vhr}(c)$ be a row of pixels and $\mathbf{K}_{vhr}(c)$ its corresponding *curvature profile*. This profile is converted into an *unnormalized probability density function* $f_{vhr}(c)$ that denotes the probability of selecting a given pixel based on its curvature. Logically, points with high curvature will be more likely to be selected than low curvature ones. However, if this mapping is linear, areas of high curvature would be oversampled while planar regions would be undersampled. Therefore, sudden changes of point density would occur, giving rise to inadequate distributions of points that would ultimately lead to the proliferation of degenerated triangles in the final mesh. In order to avoid that problem, the curvature profile is smoothed by applying a logarithmic transformation function that attenuates high curvatures while keeping low ones.

$$f_{vhr}(c) = \Gamma \log(\mathbf{K}_{vhr}(c)) \tag{1}$$

with Γ being a constant weighting factor (e.g., $\Gamma = 500$). Other underlinear transformation functions can also be applied, being also possible the definition of customized behaviors by utilizing, for instance, B-splines.

Once the previous density function has been defined, an unnormalized probability distribution function $F_{vhr}(c)$ is obtained by "integrating" $f_{vhr}(c)$

$$F_{vhr}(c) = \sum_{i=0}^{c} f_{vhr}(i) - f_{vhr}(0). \tag{2}$$

$F_{vhr}(c)$ is unnormalized since it does not range between zero and one but between zero and a maximum value M.

If the image space of $F_{vhr}(c)$ is sampled at ζ uniformly distributed points, the application of the inverse distribution function $F_{vhr}^{-1}(y)$ to those points leads to a set of ζ points adaptively distributed according to $f_{vhr}(c)$. This principle is illustrated in Fig. 3. In our case, as the probability distribution function is related to the curvature estimation from the range image, the density of the obtained points will depend on that curvature and, hence, on shape variations.

This process is repeated for every row r of the range image tile $\mathbf{R}_{vh}(r, c)$, leading to an $(R/V+1) \times \zeta$ array: $HS_{vh}(r, j) = HS_{vhr}(j)$, $r \in [0, R/V]$, $j \in [0, \zeta)$.

For each value j, a collection of points $(r, HS_{vh}(r, j))$ that determine a "vertical" curve in the range image is obtained by iterating over r. Going over the different values of j, we obtain a collection of j vertical curves that tend to adapt to the shape of the underlying objects contained in the range image. For example, Fig. 4 shows the set of vertical curves obtained by applying this procedure to all the tiles of the range image shown in Fig. 2. We are considering a tessellation into 4 by 4 tiles with ten sampled columns per tile ($\zeta = 10$).

Owing to the overlap of one pixel between adjacent tiles, the same distribution of selected points is obtained at the common boundary. Hence, curves belonging to adjacent tiles join smoothly. The final result is a complete range image sampled in vertical curves that adapt to the shape of the individual tiles that make up the whole range image.

The next step consists of adaptively sampling each of the vertical curves at R positions (R is the input parameter that indicates the number of sampled rows per tile). The process is similar to the previous one. The difference now is that a curvature profile is obtained from the positions of the points that belong to one of the vertical curves instead of from the positions corresponding to a row of pixels. Again, each tile is processed separately.

Let $VC_{vhj}(r) = HS_{vh}(r, j)$, $r \in [0, R/V]$, represent the vertical curve corresponding to a certain column j, $j \in [0, \zeta)$. The 2-D positions of the points that belong to that curve are $(r, VC_{vhj}(r))$.

The unnormalized probability density function corresponding to the curvature profile associated with each curve is an adaptation of (1)

$$f_{vhj}(r) = \Gamma \log(\mathbf{K}_{vh}(r, VC_{vhj}(r))). \tag{3}$$

Similarly, to (2), an unnormalized probability distribution function $F_{vhj}(r)$ is computed. Then, the image space of this distribution function is uniformly sampled at R positions y and the inverse distribution function $F_{vhj}^{-1}(y)$ is applied to them to obtain a set of R points r such that $F_{vhj}(r) = y$. A vector of vertical sampled points $VS_{vhj}(i)$, $i \in [0, R)$, keeping the different r' s is computed in this way. In the end, an $R \times \zeta$ array of horizontal and vertical sampled points is obtained

$$HVS_{vh}(i, j) = (VS_{vhj}(i), HS_{vh}(VS_{vhj}(i), j)).$$

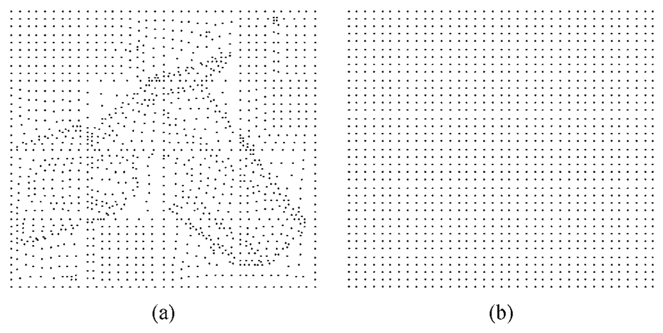
Fig. 5. (a) Horizontal and vertical sampled points with $H = V = 4$ and $R = \zeta = 10$. (b) Same number of points after uniform sampling.

In summary, given a range image tile $\mathbf{R}_{vh}(r,c)$ and its associated curvature image $\mathbf{K}_{vh}(r,c)$, the array $HVS_{vh}(i,j)$ contains the 2-D coordinates (r,c) of $R \times \zeta$ points selected from the given tile. Fig. 5(a) shows the set of vertical and horizontal sampled points obtained for the 4 by 4 tiles in which the original range image was tessellated. Fig. 5(b) shows the same number of sampled points but considering uniform sampling instead. In the adaptive distribution, points tend to concentrate in areas of high curvature, highlighting the shape of the objects contained in the image. Owing to the overlap between adjacent tiles, the same distribution of selected points is obtained at the common boundary.

B. Discontinuity-Preserving Triangulation of Sampled Points

The outcome of the previous stage is an array of $(R-1)V+1$ rows and $(\zeta-1)H+1$ columns, with H and V being the number of horizontal and vertical partitions of the range image into tiles and R and ζ the number of rows and columns in which every tile is sampled. Each position of this array contains the row and column of a range image pixel and its corresponding value (depth measure). By construction, the final number of selected points is predefined by the user. This allows to obtain a final triangular mesh with a bounded number of points and thus, with a foreseeable maximum size.

From the set of points obtained above, only those points that do not belong to the background of the range image will be considered for the final triangulation.

Once a set of points has been chosen, the objective now is to triangulate them in such a way that triangle edges agree with the orientation of the discontinuities contained in the range image (jump and crease edges). In this line, several *data-dependent triangulations* have been proposed in the literature (e.g., [24], [25]). However, as discussed in [23], discontinuities are not necessarily preserved if the triangulation process is only based on minimizing the approximation error.

The algorithm proposed in this paper triangulates the set of points obtained after the adaptive sampling stage as follows. First a 2.5-D Delaunay triangulation is applied to all the non-background points. Then, all triangle edges are sorted out according to their approximation error with respect to the original range image. Finally, all edges are considered in turn according to the previous ordering and a set of geometric tests are applied to them to decide whether they are flipped or not in order to keep underlying discontinuities. These stages are described next.

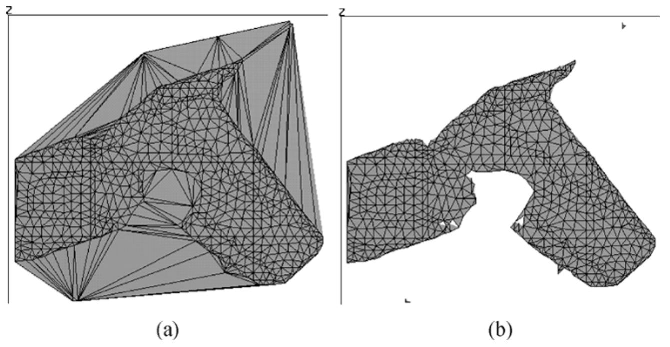
Fig. 6. (a) Two-dimensional Delaunay triangulation of a collection of adaptively sampled points (range image pixels). (b) Same triangulation but removing triangles that belong to the background.

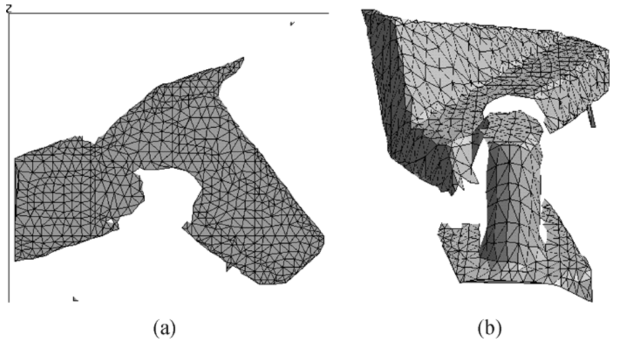
Fig. 7. (a) Two-dimensional Delaunay triangulation after removing background and dangling triangles. (b) Two and one half-dimensional triangulation obtained from the previous result.

1) 2.5-D Delaunay Triangulation: Given a set of 3-D points consisting of two image coordinates (row and column number) and a depth value, a 2-D Delaunay triangulation [27] is applied to the set of image coordinates. Fig. 6(a) shows the 2-D Delaunay triangulation of the set of adaptively sampled points corresponding to the example utilized so far. The triangulation covers the convex hull of the given set of points.

From this triangulation, triangles that belong to the background are removed. In particular, the image coordinates (row and column) of the barycenter of each triangle are computed by averaging the image coordinates of its three vertices. If the resulting coordinates correspond to a pixel labeled as background in the input range image, the triangle is removed. The result is shown in Fig. 6(b).

This procedure can generate *dangling triangles*, or triangles that join together by just a single shared vertex. Two groups of such triangles appear in the previous example. Those triangles are removed in order to produce a *valid triangular mesh* in which all triangles join together along a shared edge. Interior holes may also appear—the previous example has one—but they do not pose any problems regarding the correctness of the mesh topology. Fig. 7(a) shows the mesh of Fig. 6(b) after removing dangling triangles present in it.

When the previous triangulation (without dangling triangles) is applied to the 3-D points obtained by considering not only the 2-D image coordinates (row, column) but also the associated depth measure, and these three coordinates are conveniently scaled and interpreted as X, Y and Z coordinates, the result is the so called 2.5-D Delaunay triangulation. In order to do that, each component (row, column, depth) must be converted to 3-D Euclidean space by multiplying it by an appropriate scale factor that depends on the calibration of the range sensor being utilized

$$\mathrm{X} = K_x r \quad \mathrm{Y} = K_y c \quad \mathrm{Z} = K_z \mathbf{R}_f(r,c). \tag{4}$$

At this point, we can appreciate that the obtained triangulation does not preserve the underlying discontinuities—basically crease edges—present in the image, since many triangle edges cross such discontinuities producing noticeable artifacts on the resulting surface. The obvious reason is that the obtained triangulation takes into account a criterion that only considers the size of the triangles when they are projected onto the XY plane, disregarding the shapes of the underlying surfaces. Therefore, a further stage must flip some of those edges to better align them along the existing discontinuities. This is done in two steps. First, all triangle edges are sorted out according to a measure of the error between them and the original range image. Then, each edge is considered in turn, deciding whether it is flipped or not based on the shape of the result. These steps are described next.

2) Error-Based Sorting of Triangle Edges: All edges of the previously obtained triangulation are possible candidates to be flipped. However, the order in which those flips are performed is important. For instance, consider the example shown in Fig. 8. Given the configuration depicted on the left, edge *AD* can be swapped (flipped) for edge *EB*. The latter might improve the approximation of a certain discontinuity present in the range image. However, if *AB* was initially swapped for *CD*, *EB* would not be able to be swapped any more, even though *EB* was a better solution.

Clearly, this is a problem of priorities that will appear in any strategy. A possible solution would consist of applying global nonlinear optimization techniques, such as simulated annealing. However, besides an important downfall in performance, if the variable being optimized was the approximation error, the results would not guarantee that discontinuities are preserved [23].

In order to tackle this problem efficiently, we have opted for an eager strategy that sorts out all triangle edges according to the expected approximation gain that they would attain in case they were flipped. This gain is defined in terms of the variation of approximation error between the range image and each triangle edge as it will be explained below. It is important to note that this gain does not determine whether edges are flipped or not, but only the order in which they will be subsequently considered for being flipped. A top-down approach is followed, such that edges with highest gains are considered first.

Approximation Gain of an Edge: Let us consider that the given range image is mapped into 3-D space by applying the linear mapping defined in (4) to all its pixels. This means that X, Y, and Z coordinates are computed for every pixel from both its row and column coordinates and its associated depth. We will refer to this mapping as the *3-D range image*. Once the range image and its triangular approximation are in the same space, we can define the *approximation error* of a triangle edge as the cumulative 3-D distance between that edge and the 3-D range image points closest to it. The lower the approximation error of an edge is the better that edge will model the underlying surfaces present in the range image.

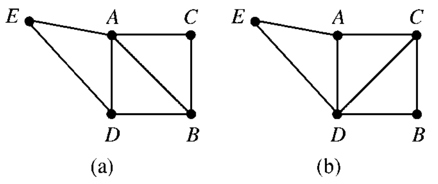

Fig. 8. (a) Configuration in which edge *AD* can be flipped. (b) Configuration in which edge *AD* is no longer a candidate to be flipped.

In order to gain efficiency, the approximation error of an edge *AB* is estimated as follows. Its two extremes, *A* and *B*, are 3-D points whose X and Y coordinates are proportional (4) to the row and column coordinates of two range image pixels: (r_A, c_A) and (r_B, c_B), respectively. Instead of computing which points of the 3-D range image are closest to the given edge, we determine which pixels from the range image are traversed by edge *AB* when it is projected onto the image plane. This is done by applying Bresenham's algorithm [28] to "draw" a straight (rasterized) line between (r_A, c_A) and (r_B, c_B). All pixels traversed by this line are the ones considered to be closest to *AB*. Then, those pixels are mapped to 3-D points by applying (4). Finally the Euclidean distances between those 3-D points and the 3-D edge *AB* are computed and added together, obtaining thus an estimation of the sought approximation error.

The *approximation gain* of an edge *AB* is computed as follows. Let us consider the triangular mesh shown in Fig. 8(a). Let E_{AB} be the approximation error of *AB* computed as indicated above. Edge *AB* can be swapped for edge *CD*. Let E_{CD} be the approximation error of the latter. The approximation gain of *AB* is defined as: $G_{AB} = E_{AB} - E_{CD}$. According to this formulation, when the approximation error of the alternative edge *CD* is lower than the approximation error of *AB*, the approximation gain of *AB* will be positive, implying that a swap of *AB* for *CD* is advantageous in terms of approximation accuracy. The higher that difference, the higher the gain. Thus, this gain can be utilized to rank all edges of the triangular mesh in order to determine the order in which they will be considered for flipping: edges with high approximation gain will be considered first.

However, there are two cases in which an edge does not have an alternative edge for which it can be swapped. The first case affects *exterior edges*, or edges that belong to the boundary of an open triangular mesh. For example, this is the case of edge *DE* in Fig. 8(a). The second case is when the alternative edge does not intersect the original edge. This happens when the two triangles adjacent to the given edge do not form a convex quadrilateral. For instance, this situation arises in the case of *AD* in Fig. 8(b). If *AD* is swapped for *CE*, the resulting triangulation would not be topologically correct. This second situation is easily detected by checking if the two edges (segments) intersect when their end-points are projected onto the image plane according to the inverse of (4).

In both cases, when no alternative edge is found, the approximation error of the alternative edge is considered to be infinity, leading to a minus infinity approximation gain that will prevent the given edge from being flipped.

Optimal Ranking of Triangle Edges Based on Approximation Gain: At this point, all edges of the given triangulation are considered in turn and sorted out in decreasing order according to their approximation gain. If two edges have exactly the same approximation gain, one of them is randomly chosen to be processed first. The final objective is to iterate over all those edges in the given order until there are no more edges whose gain is above a certain threshold. The nature of this last iterative stage determines the process utilized to perform the ranking, which is described below.

If all edges depicted in Fig. 8(a) were sorted out according to their respective gains and *AB* was the edge with the highest gain, the iterative algorithm mentioned above would try to swap *AB* for *CD*. If this flip was feasible, we would get the triangular mesh shown in Fig. 8(b). Obviously, the old edge *AB* is removed from the sorted list of edges and a new edge *CD* is introduced with its corresponding approximation gain. However, besides these two operations, the four edges that form cell *ADBC* may suffer from a variation of their respective approximation gains, since the mesh topology will have changed. In this example, only the gain of *AD* would be modified from a certain gain to minus infinity, as *EC* is not feasible. The other three edges are exterior and their gains would remain at minus infinity. In general, however, all those edges could be interior and would potentially have to be recomputed, leading to possible relocations inside the sorted list of edges.

This means that when an edge is flipped, besides removing that edge from the sorted list and including the new edge in it, four edges must be found inside the list and, if necessary, relocated according to their newly computed approximation gains. Obviously, this process must be done as efficiently as possible. In order to sort out all edges efficiently and to allow for fast relocations, insertions and deletions of edges, we have opted for an optimal sorting strategy based on the use of a *priority queue (max-heap)* [29]. The advantage of priority queues is that insertion, deletion and relocation operations are performed in logarithmic time, $\Theta(\log N)$ with N being the number of elements being processed.

In order to rank the edges of the triangular mesh according to their approximation gains, all edges are sequentially inserted into the queue in optimal asymptotic time $\Theta(N \log N)$. The ordering key is the edge's approximation gain computed as indicated above. To get the sequence of ordered edges, each edge must be extracted in turn leading to a total optimal time cost $\Theta(N \log N)$. The objective is to extract edge after edge from the head of the queue until the approximation gain is below a certain threshold. This process is described next.

3) Discontinuity Preserving Iterative Flip of Triangle Edges: As a result of the previous process, all edges of the given triangular mesh are inserted in a priority queue. The edge at the head of the queue is the one with the largest approximation gain, meaning that a swap for its alternative edge leads to the maximum drop of approximation error between the triangular mesh and the given range image. Edges will be considered for flipping according to this heuristic, which corresponds to an eager policy as discussed earlier in Section II.B2.

Since the approximation gain is based on an error measure that considers the distance between the triangular mesh and the 3-D range image, as discussed in [23], a flipping strategy based on this gain would have the same drawbacks than other methods based on similar principles, not guaranteeing the preservation of discontinuities, especially of crease edges. Instead, once an edge is extracted from the queue, a set of heuristic tests are carried out to determine whether that edge must be flipped or not. If all tests are passed, the edge is swapped for its alternative one and the queue is internally reorganized in order to reflect the changes. In particular, the old edge is removed, the new edge is inserted according to its approximation gain, and the gains of the four edges that form the quadrilateral cell containing the old edge are recomputed and their corresponding positions in the queue eventually recalculated. If the extracted edge is not swapped, it is inserted back into the queue with a minus infinity gain. The algorithm is very efficient since the total asymptotic cost of these operations is $\Theta(\log N)$, with N being the number of stored edges. For example, for 2,000 edges, each operation would require a maximum of 11 internal displacements in the queue. Only four more displacements could be required if the number of edges was ten times higher.

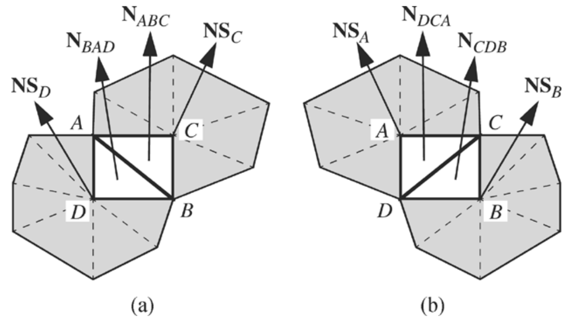

Fig. 9. Spans, normals of spans and normals of triangles for the two possible configurations of a convex quadrilateral cell.

Given a certain edge *AB*, a set of local geometric tests aimed at detecting the presence of discontinuities are sequentially applied in order to decide whether *AB* is swapped for its alternative edge *CD*. Both edges are contained in a convex quadrilateral cell *ADBC*, Fig. 9(a). Those tests are based on the local geometry around the cell rather than on the approximation error, which, as mentioned above, is not a reliable indicator of discontinuities. The tests make use of some magnitudes that are introduced next.

Let Fig. 9 represent the two possible triangulations of a convex quadrilateral cell. Let us suppose the original triangulation is done through diagonal *AB*. The subsequent tests are carried out in order to assess if diagonal *CD* is more advantageous than diagonal *AB* in the sense that *CD* agrees with the orientation of an underlying discontinuity more than *AB*. For each vertex opposite the cell's diagonal, we define its *span* as the set of triangles that contain that vertex, excluding the cell's triangle. For example, the span of vertex *C* in Fig. 9 contains all the triangles around *C* excluding triangle *ABC*. Spans are shown as dark polygons in Fig. 9. For each span, we define the *normal of span* as the sum of all the unitary normal vectors corresponding to the triangles that belong to that span. For example, $\mathbf{NS}_C$ is the normal of the span of vertex *C*. The normal of span gives an estimate of the predominant orientation of the surface that surrounds a certain vertex. When the span of a vertex does not contain any triangles, the corresponding normal of span is the null vector. This situation arises when the three vertices of a triangle are exterior, for instance, for vertex *E* in the triangular meshes shown in Fig. 8.

Given both the normal of span of a certain vertex *C*, $\mathbf{NS}_C$, and the normal of a triangle *ABC* containing that vertex, $\mathbf{N}_{ABC}$, we define the *disparity of the triangle* as: shown in (5) at the bottom of the page, where Δ is an *upper disparity threshold* (e.g., $\Delta = 20°$). The disparity of a triangle indicates how much the orientation of that triangle deviates from the overall orientation of the surface around it. Low disparities are an indicator of the absence of a discontinuity. When the angle between the normal of span and the triangle normal is above the upper disparity threshold, the disparity is set to its maximum value in considering that a possible discontinuity is likely to exist. The upper disparity threshold is statistically computed from the mean and standard deviation ($\Delta = \mu + 2\sigma$) of the disparities corresponding to triangles that are known to belong to a single surface in a manually segmented triangular mesh (ground-truth segmentation) that approximates a given range image utilized as a training dataset.

Given two triangles *ABC* and *BAD* sharing a diagonal *AB*, two disparities, $disp_{ABC}$ and $disp_{BAD}$, are computed as indicated above. Taking those separate disparities into account, we define the *disparity of the diagonal* as the maximum of the disparities of its associated triangles: $disp_{AB} = \max(disp_{ABC}, disp_{BAD})$. In general, a diagonal between two triangles will be likely to belong to a discontinuity if its disparity is low while the angle between the triangles is high. Similarly, given the two alternate diagonals inside a quadrilateral cell, the diagonal with the lowest disparity will be likely to be the one that best preserves the discontinuities in that region. Actually, this criterion is the final heuristic test we use to decide whether a flip is performed or not. However, prior to it, several geometric tests are sequentially carried out to deal with several particular situations.

Test 1: Deadlock Avoidance: The objective is to avoid algorithm deadlocks by preventing old edges from being subsequently chosen. In the previous example, if the alternative edge *CD* had already been considered, it would have to be discarded in future iterations. In order to do that, each vertex of the triangular mesh keeps a list of identifiers of its former adjacent vertices. An edge between two former neighbors is rejected.

Test 2: Significant Disparity Variation: The objective is to reject a swap of the current diagonal *AB* for its alternative diagonal *CD* if the difference between their corresponding disparities is below a *lower disparity threshold,* $|disp_{AB} - disp_{CD}| < \varepsilon$. This situation occurs when the improvement obtained after the flip is negligible (e.g., $\varepsilon = 0.3°$). In that case, performing the flip is not worthwhile and usually leads to triangles with worse shapes than the original ones, which were the result of a Delaunay triangulation. If there is a significant disparity variation, the third test is applied.

Test 3: Triangle Degeneration: At this point, a significant disparity variation exists between the two diagonals. The objective now is to reject a swap of the current diagonal *AB* for its alternative diagonal *CD* if any of the two new triangles, *DCA* or *CDB*, are degenerated and both lie at a smooth region. A triangle is degenerated if one of its internal angles is above a maximum angle (e.g., $160°$). Two triangles potentially lie at a smooth region if the angle between their respective normals is below a certain threshold. This threshold has been chosen to coincide with the upper disparity threshold Δ utilized above (e.g., $\Delta = 20°$). If no degeneration is found or it is found but accepted because the triangles do not presumably lie at a smooth region, the fourth test is applied.

Test 4: Noticeable Shape Variation: At this point, a significant disparity variation exists and the new triangles are not degenerated (or if they are, they do not presumably belong to a smooth region). The objective now is to determine if the flip produces a noticeable variation of shape. Specifically, the normals of the new triangles, $\mathbf{N}_{DCA}$ and $\mathbf{N}_{CDB}$, must have a difference of angles above a certain threshold (e.g., 5°). If this does not occur, the flip is worthless since it will not contribute to highlighting any discontinuities (the new triangles after the flip would almost be planar). If the previous condition is not satisfied, the flip is rejected. If no rejection is decided, the fifth test is applied.

Test 5: Noticeable Discontinuity Existence: At this point, a noticeable disparity variation exists, the new triangles are not degenerated and the flip produces a noticeable change of shape. The objective now is to decide whether the flip will likely lead to the probable loss or highlight of an orientation discontinuity (crease edge).

The presence of a discontinuity can be hypothesized by measuring the angle between the normals of corresponding spans. Particularly, if the angle between $\mathbf{NS}_C$ and $\mathbf{NS}_D$ (Fig. 9) is high, diagonal *AB* will likely be part of a crease edge separating two surfaces with different orientations. Thus, if the angle between those two normals is significantly larger than the angle between the normals of span of the alternative triangles obtained after the flip, there is a high probability that a discontinuity may be lost as a result of that flip. In practice, if the angle between $\mathbf{NS}_C$ and $\mathbf{NS}_D$ is larger than twice the angle between $\mathbf{NS}_A$ and $\mathbf{NS}_B$, the flip is rejected. This test is especially suitable for detecting triangle edges (diagonals) that cross crease edges joining planar regions.

If this test is applied and no decision is made about the convenience of the flip, the last test is responsible for reaching the final decision.

Final Test: Difference Between Disparities: At this point, all previous tests have been unable to determine whether the swap of edge *AB* for *CD* is advantageous or not in order to highlight surface discontinuities. The final test consists of comparing the disparities between the two alternative diagonals. If the disparity of the given diagonal *AB*, $disp_{AB}$, is larger than the disparity of *CD*, $disp_{CD}$, the flip is performed. On the contrary, the flip is rejected. In other words, the configuration in which

$$\mathrm{disp}_{\mathrm{ABC}} = \begin{cases} \mathrm{Angle}(\mathbf{NS}_C, \mathbf{N}_{ABC}) & \|\mathbf{NS}_C\| > 0 \wedge \mathrm{Angle}(\mathbf{NS}_C, \mathbf{N}_{ABC}) < \Delta \\ \pi & \text{otherwise} \end{cases} \tag{5}$$

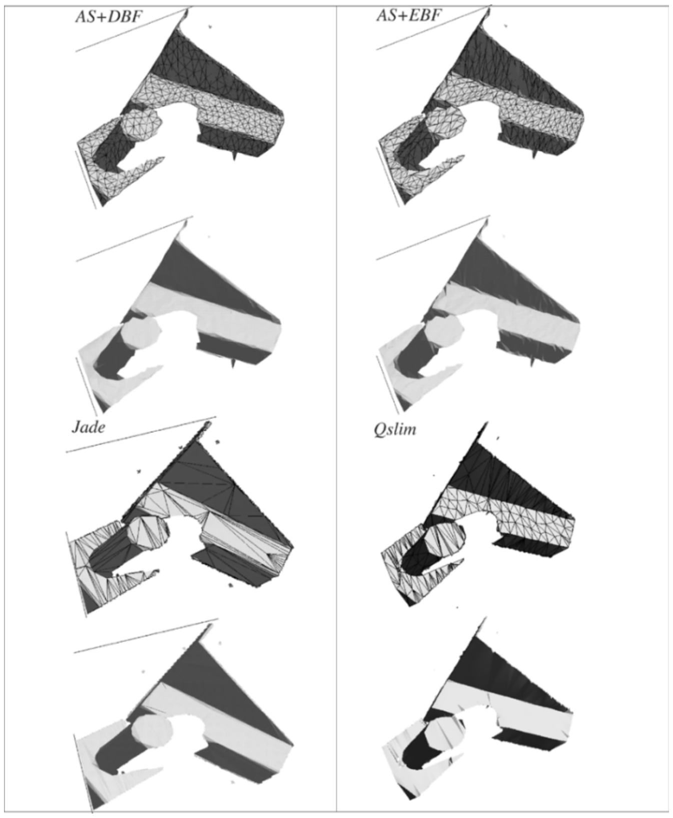

Fig. 10. Comparison of the proposed flipping technique. Triangles are shown both in solid and wireframe at the top and only in solid at the bottom. (*Top-left*) Adaptive mesh through the proposed adaptive sampling plus the discontinuity-based flipping technique—*AS+DBF*. (*Top-right*) Adaptive mesh through adaptive sampling plus an error-based diagonal selection algorithm—*AS+EBF*. (*Bottom-left*) Adaptive mesh computed with *Jade*. (*Bottom-right*) Adaptive mesh computed with *Qslim*.

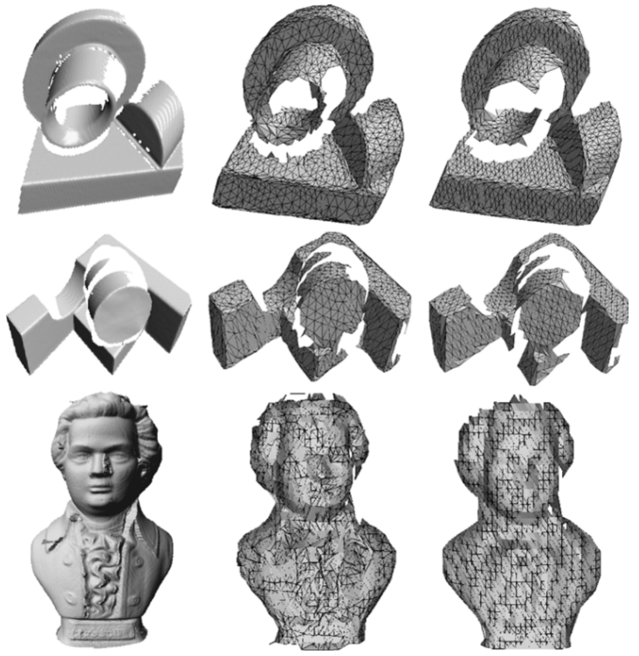

Fig. 11. Examples of the application of the proposed flipping technique. (*Left*) Original range image (rendered with perspective). (*Middle*) Triangulation after adaptive sampling plus discontinuity-preserving diagonal flipping. (*Right*) Triangulation after regular sampling plus error-based diagonal selection.

the triangles have an orientation more similar to the orientation of their respective neighborhoods (spans) produces the lowest

TABLE I
EXPERIMENTAL RESULTS FOR THE TRIANGULAR MESHES PRESENTED IN FIG. 11

Original range image	Sampled points	Triangular Mesh (*without background*) Vertices	Edges	Triangles	Flipped Edges	Total CPU time (sec.)	Relative error (%)
Fig. 11 (*top*) *154x183* (*28,182* pixels)	1,369 adaptively	894	2,248	4,696	366	0.44	0.18
	1,369 uniformly	761	1,669	4,076	——	0.4	0.23
Fig. 11 (*middle*) *197x187* (*36,839* pixels)	1,369 adaptively	725	1,696	3,613	251	0.37	0.22
	1,369 uniformly	548	1,293	2,745	——	0.28	0.40
Fig. 11 (*bottom*) *399x263* (*104,937* pixels)	1,369 adaptively	853	2,269	4,671	488	0.43	0.33
	1.369 uniformly	773	2,100	4,296	——	0.04	0.34

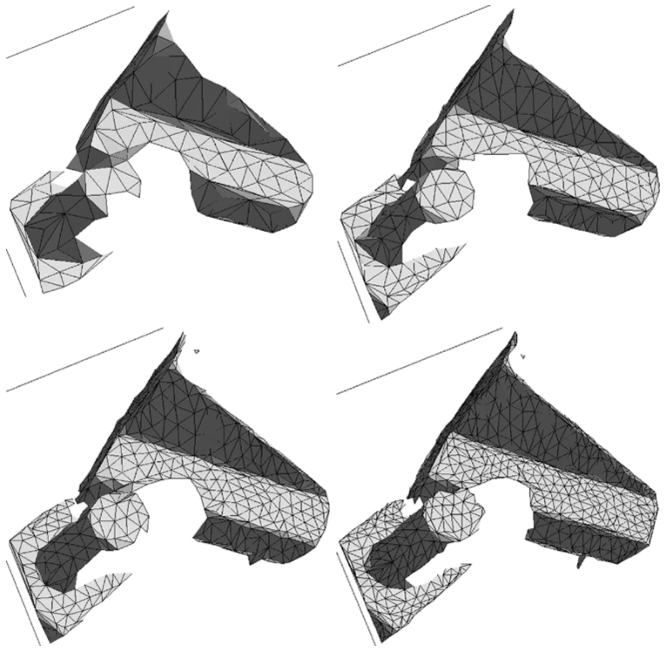

Fig. 12. Approximation of a range image at four different resolutions using the proposed technique. (*Top-left*) 125 vertices (5×5 samples per tile), 0.13 s. (*Top-right*) 276 vertices (7×7 samples per tile), 0.20 s. (*Bottom-left*) 385 vertices (8×8 samples per tile), 0.25 s. (*bottom-right*) 639 vertices (10×10 samples per tile), 0.34 s.

disparity and is likely to be the one that most highlights the underlying discontinuity. For example, if edge *CD* in Fig. 9(a) is part of a crease edge that separates two planes, the disparity of edge *CD* will be zero, since triangle *DCA* and the span of vertex *A* will lie on the first plane, and triangle *DCB* and the span of *B* will lie on the second plane. However, the initial configuration, in which diagonal *AB* will cross the discontinuity, will tend to have a disparity above zero, since the triangles and their spans will cover an area across the discontinuity and will tend to have quite disparate normals.

The previous tests are applied in sequence to all the edges extracted from the priority queue. As mentioned above, each flip decision involves the extraction of the old edge from the queue, the inclusion of the new edge and the recomputation of the approximation gain (and possible internal relocation in the queue) of the edges that delimit the involved quadrilateral cell.

TABLE II
SUMMARY OF EXPERIMENTAL RESULTS FOR EACH OF THE ADAPTIVE TRIANGULAR MESHES PRESENTED IN FIG. 12 (4 BY 4 TILES IN ALL EXAMPLES)

Fig. 12	Sampled pixels per tiles	Vertices	Edges	Triangles	Flipped edges	Total CPU time (sec.)	Relative error (%)
(*top-left*)	5 x 5	125	255	561	55	0.13	0.64
(*top-right*)	7 x 7	276	625	1,344	109	0.20	0.34
(*top-right*)	8 x 8	385	897	1,911	160	0.25	0.26
(*bottom-left*)	10 x 10	639	1,563	3,286	275	0.34	0.18

This iterative process stops when the edge at the head of the queue, which is the edge with the largest approximation gain, is below a certain threshold. Interestingly enough, the best results are not obtained when edges with positive approximation gains are only considered, but when edges with negative gains are also accepted. The reason is related to the fact that the approximation error is not a reliable indicator of discontinuities [23]. Therefore, edges that exhibit a large approximation error may be necessary in order to highlight a discontinuity. In practice, all edges are considered except for those with a minus infinity approximation gain, which correspond to diagonals of nonconvex quadrilateral cells and edges already flipped—the latter edges are inserted in the priority queue since, eventually, they can be recomputed as a result of a new flip. Notwithstanding, edges with very negative approximation gains are usually associated with edges that correctly model discontinuities and, if they were flipped, they would cross those discontinuities leading to larger approximation errors. This means that edges with very low (negative) approximation gains will not tend to produce flips and will only be relocated in the queue with a minus infinity gain without producing any considerable overload.

III. Experimental Results

The proposed technique has been applied to real range images obtained with the MSU Prip Lab's Technical Arts 100 × scanner and the OSU's Minolta 700 range scanner. These images contain planar and curved surfaces with surface and orientation discontinuities (jump and crease edges). Since the proposed technique is mainly devised to improve the quality of approximation of crease and jump edges and they usually cover a small percentage of the overall range image, improvements are qualitative rather than quantitative and, hence, they are better appreciated through images of the final result. However, quantitative results also show an improvement as it will be described below. CPU times have been measured on a 650 MHz Pentium III. This is the typical processing power of the computers currently embedded in commercial mobile robots.

The example used so far corresponds to a 188 row by 208 column range image (39,104 pixels). This image was split up into 4 by 4 tiles ($H = V = 4$) and 10 by 10 points were adaptively sampled per tile. After applying the sampling technique described in Section II.A, an adaptive quadrilateral mesh with 1,369 points is obtained in 0.07 s. Then this mesh is triangulated in 0.04 s., producing 3,286 valid triangles, 1,563 edges,

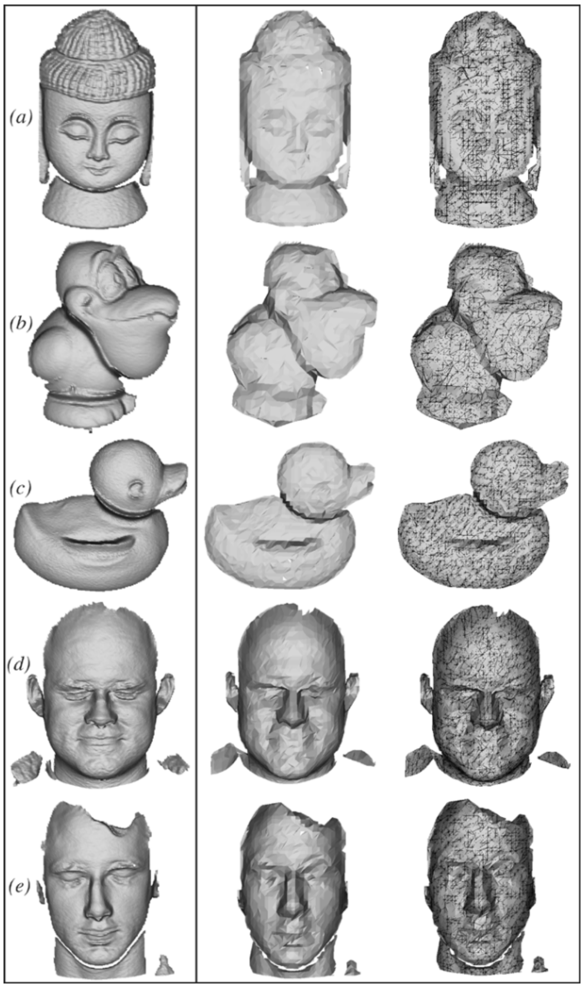

Fig. 13. (*Left*) Original range images (rendered with perspective) obtained with the OSU's Minolta 700 range scanner. Each range image is defined by 40 000 points. (*Right*) Final result (in solid and wireframe) computed with the proposed technique (adaptive sampling plus discontinuity-based flipping).

and 639 3-D vertices. Considering that each pixel of the original range image represents a 3-D point, a point reduction of 98% is obtained. This reduction must not be interpreted as a data compression ratio though, since the storage space to represent a dense range image and a triangular mesh are not comparable. However, this figure is significant in terms of geometrical reduction and post-processing speedups, since many range image processing algorithms deal with all pixels of the range image, frequently converting it to a dense triangular mesh. In that case, the point reduction ratio would be directly reflected into an equivalent processing time reduction. In case the range

TABLE III
EXPERIMENTAL RESULTS FOR THE TRIANGULAR MESHES PRESENTED IN FIG. 13

Fig. 13	Vertices	Edges	Triangles	Flipped edges	Total CPU time (sec.)	Relative error (%)
(*a*)	1,249	3,450	7,046	721	0.36	0.06
(*b*)	798	2,146	4,396	439	0.23	0.04
(*c*)	1,359	3,848	7,809	698	0.4	0.04
(*d*)	1,683	4,635	9,449	866	0.48	0.05
(*e*)	1,308	3,482	7,147	688	0.36	0.05

image is intended to build a world model, it will usually be converted into a triangular or quadrilateral mesh. In that case, the storage reduction is certainly significant.

Once the triangular mesh was obtained, 275 edges were flipped according to the technique described in Sections II.B2 and II.B3 in 0.23 s. Thus, the total CPU time to generate the final triangular mesh from the given range image was 0.34 s.

Fig. 10 compares the proposed edge-flipping technique (*top-left*) with an error-based technique (*top-right*) and two iterative (optimization-based) decimation techniques (*bottom-left*) and (*bottom-right*). The error-based technique determines whether an edge is flipped or not based on the approximation error between the mesh triangles and the original range image. Specifically, edges with high approximation gain are flipped as they are extracted from the priority queue without applying any of the aforementioned geometric tests. In both cases, the original triangular mesh was obtained through the adaptive sampling technique presented in Section II.A. Notice that the majority of artifacts that appear along crease edges when the error-based technique is applied are removed by the proposed edge-flipping technique.

The iterative decimation techniques are based on different optimization algorithms that fulfill a user defined point reduction while minimizing an approximation measure—the approximation error with the original data in the first case and a quadratic error metric in the second one. These algorithms are described in [2] and [8], and they are available as public software packages (*Jade* and *Qslim*, respectively). In both algorithms, until the number of points of the approximating mesh equals the amount of points specified by the user, the algorithms sequentially identify the points whose removal produces the lowest approximation measure and extract them from the mesh, retriangulating their corresponding neighbors. In the example shown in Fig. 10(*bottom-left*), *Jade* obtains 638 points in 138 s. (the same amount of points is obtained in 0.34 s with the proposed adaptive technique). Fig. 10(*bottom-right*) has been generated with *Qslim*. It contains 639 points and was computed in 1.18 s. *Qslim* is faster than *Jade* but it produces more artifacts in the final representation (see solid model).

As mentioned above, quantitative results also show that the proposed technique improves the quality of approximation of the original range image. Thus the *relative approximation error* between each triangular mesh and the original range image has been computed as follows. For every pixel of the 3-D range image, the vertical distance between its 3-D position and the first triangle intersected by a vertical line passing through that pixel is computed. This absolute distance is divided by the maximum height of the objects contained in the range image. In this way, the final relative approximation error is computed by averaging the relative errors of all nonbackground pixels of the range image. The relative errors for the examples shown before are: 0.24% (adaptive sampling + error-based flipping, Fig. 10(*top-right*) and 0.18% [adaptive sampling + proposed flipping, Fig. 10(*top-left*)].

Some more examples are shown in Fig. 11. The middle column shows the triangular meshes that result from applying the proposed technique (adaptive sampling plus discontinuity-preserving diagonal selection). The right column shows the triangular meshes that result from applying a regular sampling of the range image and then a triangulation with an error-based diagonal selection. The corresponding original range images are presented in the left column. Experimental results are summarized in Table I.

Fig. 12 shows the same example utilized throughout this paper but with a different number of adaptively sampled points. All examples were run with 4 by 4 tiles per range image. Table II summarizes the experimental results corresponding to this example.

Finally, Fig. 13 shows five different range images (*left*) obtained with the OSU's Minolta 700 range scanner. Each range image consists of 200 by 200 points. The corresponding adaptive triangular meshes computed with the proposed technique are presented both in solid and wireframe in the middle and right columns. In addition, information, such as CPU time, relative error and triangular mesh definition, is presented in Table III.

IV. CONCLUSIONS AND FURTHER IMPROVEMENTS

This paper presents an efficient technique aimed at the generation of adaptive triangular meshes from range images preserving surface and orientation discontinuities (crease and jump edges). Initially, the algorithm generates an adaptive quadrilateral mesh from the range image. This mesh is more accurate than a uniformly sampled one since it distributes the same number of points by considering the curvature of the surfaces contained in the range image. Then, all points of this mesh are triangulated through a 2.5-D Delaunay algorithm. Finally, an iterative process goes over all edges of that triangulation determining whether they must be flipped or not according to a set of geometric tests that highlight underlying discontinuities present in the original range image. The algorithm is very efficient and appropriate for fields with tight timing constraints such as robotics. The obtained triangular meshes can be integrated in world-models of complex workspaces or to serve as the input data for further segmentation, integration or recognition algorithms that will perform more efficiently than if they directly worked on the original dense range images.

References

[1] M. A. García and L. Basañez, "Fast extraction of surface primitives from range images," in *Proc. 13th IAPR Int. Conf. Pattern Recognition, Vol. III: Applications Robotic Systems*, Vienna, Austria, August 1996, pp. 568–572.

[2] A. Ciampalini *et al.*, "Multiresolution decimation based on global error," in *The Visual Computer*. New York: Springer-Verlag, 1997, vol. 13, pp. 228–246.

[3] M. Soucy, A. Croteau, and D. Laurendeau, "A multi-resolution surface model for compact representation of range images," in *Proc. IEEE Int. Conf. Robotics Automation*, May 1992, pp. 1701–1706.

[4] H. Hoppe *et al.*, "Mesh optimization," in *Proc. SIGGRAPH*, 1993, pp. 19–26.

[5] H. Hoppe, "Progressive meshes," in *Proc. SIGGRAPH*, 1996, pp. 99–108.

[6] E. Shaffer and M. Garland, "Efficient adaptive simplification of massive meshes," *IEEE Visual.*, pp. 127–134, Jan. 2001.

[7] P. Lindstrom, "Out-of-core simplification of large polygonal models," in *Proc. SIGGRAPH*, 2000, pp. 259–262.

[8] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proc. SIGGRAPH*, 1997, pp. 209–216.

[9] P. Lindstrom *et al.*, "Real-time continuous level of detail rendering of height fields," in *Proc. SIGGRAPH*, 1996, pp. 109–118.

[10] V. Volkov and L. Li, "Real-time refinement and simplification of adaptive triangular meshes," in *Proc. IEEE Visualization*, Oct. 2003, pp. 155–162.

[11] M. Hussain, Y. Okada, and K. Niijima, "A fast and memory-efficient method for LOD modeling of polygonal models," in *Proc. IEEE Int. Conf. Geometric Modeling Graphics*, July 2003, pp. 137–142.

[12] L. DeFloriani, "A pyramidal data structure for triangle-based surface description," *IEEE Comput. Graphics Applicat.*, vol. 9, pp. 67–78, Mar. 1989.

[13] O. Van Kaick, M. Da Silva, W. Schwartz, and H. Pedrini, "Fitting smooth surfaces to scattered 3-D data using piecewise quadratic approximation," in *Proc. IEEE Int. Conf. Image Processing*, vol. 1, Sept. 2002, pp. 493–496.

[14] D. Terzopoulos and M. Vasilescu, "Sampling and reconstruction with adaptive meshes," in *Proc. IEEE Int. Conf. Computer Vision Pattern Recognition*, June 1991, pp. 70–75.

[15] M. Vasilescu and D. Terzopoulos, "Adaptive meshes and shells: Irregular triangulation, discontinuities and hierarchical subdivision," in *Proc. IEEE Int. Conf. Computer Vision Pattern Recognition*, June 1992, pp. 829–832.

[16] W. Lorensen and H. Cline, "Marching cubes: A high resolution 3-D surface construction algorithm," *Comput. Graph.*, vol. 21, no. 4, pp. 163–169, July 1987.

[17] H. Zha, S. Tahira, and T. Hasegawa, "Multi-resolution surface description of 3-D objects by shape-adaptive triangular meshes," in *Proc. Int. Conf. Pattern Recognition*, vol. 2, Aug. 1998, pp. 954–958.

[18] C. Yang and G. Medioni, "Surface description of complex objects from multiple range images," in *Proc. IEEE Int. Conf. Computer Vision Pattern Recognition*, June 1994, pp. 153–158.

[19] G. Nielson, A. Huang, and S. Sylvester, "Approximating normals for marching cubes applied to locally supported isosurfaces," in *Proc. IEEE Visualization*, Oct.–Nov. 2002, pp. 459–466.

[20] G. Nielson, "On marching cubes," *IEEE Trans. Visual. Comput. Graph.*, vol. 9, pp. 283–297, July–Sept. 2003.

[21] J. Lee, Y. Yang, and M. N. Wernick, "A new approach for image-content adaptive mesh generation," in *Proc. IEEE Int. Conf. Image Processing*, vol. 1, Sept. 2000, pp. 256–259.

[22] M. A. García, A. D. Sappa, and L. Basañez, "Fast generation of adaptive quadrilateral meshes from range images," in *Proc. IEEE Int. Conf. Robotics Automation*, Apr. 1997, pp. 2813–2818.

[23] ——, "Efficient approximation of range images through data-dependent adaptive triangulations," in *Proc. IEEE Int. Conf. Computer Vision Pattern Recognition*, June 1997, pp. 628–633.

[24] S. Kumar, "Surface Triangulation: A Survey. Tech. Rep.," Dept. Comput. Sci., Univ. North Carolina, Chapel Hill, NC, 1996.

[25] L. L. Schumaker, "Triangulations in CAGD," *IEEE Comput. Graphics Applicat.*, vol. 13, pp. 47–52, Jan. 1993.

[26] M. Rutishauser, M. Stricker, and M. Trobina, "Merging range images of arbitrarily shaped objects," in *Proc. IEEE Int. Conf. Computer Vision Pattern Recognition*, June 1994, pp. 573–580.

[27] S. Fortune, "A sweepline algorithm for Voronoi diagrams," *Algorithmica*, no. 2, pp. 153–174, 1987.

[28] T. Pavlidis, *Algorithms for Graphics and Image Processing*. San Francisco, CA: MD Computer Scientific Press, 1982.

[29] A. V. Aho, J. E. Hopcroft, and J. D. Ullmann, *Data Structures and Algorithms*. Reading, MA: Addison-Wesley, 1983.

Miguel Angel Garcia (M'94) received the B.S., M.S., and Ph.D. degrees in computer science from Polytechnic University of Catalonia, Barcelona, Spain, in 1989, 1991, and 1996 respectively.

He joined the Department of Software at Polytechnic University of Catalonia in 1996 as an Assistant Professor. In 1997, he joined the Department of Computer Science and Mathematics, Rovira i Virgili University, Tarragona, Spain, where he is currently Associate Professor and Head of Intelligent Robotics and Computer Vision Group. His research interests include image processing, 3-D modeling, and mobile robotics.

Angel Domingo Sappa (S'94–M'00) received the electro-mechanical engineering degree in 1995 from National University of La Pampa, General Pico-La Pampa, Argentina, and the Ph.D. degree in industrial engineering in 1999, from the Polytechnic University of Catalonia, Barcelona, Spain.

From 1999 to 2002, he did his post-doctorate research at the LAAS-CNRS, Toulouse, France and at Z+F UK Ltd., Manchester, U.K. 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 Ramon y Cajal Research Fellow. His research interests are focused on range image analysis, 3-D modeling, and model-based segmentation.