Proceedings of the 1997 IEEE
International Conference on Robotics and Automation
Albuquerque, New Mexico - April 1997

Fast Generation of Adaptive Quadrilateral Meshes from Range Images

Miguel Angel García‡ Angel Domingo Sappa† Luis Basañez†

‡ Department of Software (Computer Graphics Section)
† Institute of Cybernetics
Polytechnic University of Catalonia
Diagonal 647, planta 8. 08028 Barcelona, SPAIN
fax: +34 3 401 60 50, E-mail: garcia@turing.upc.es

Abstract

This paper proposes a fast technique for generating adaptive quadrilateral meshes from range images with no optimization. The obtained meshes adapt to the features of the input images by concentrating points in areas of high curvature and by dispersing them in low variation regions. This leads to more accurate approximations of the given range images than when uniform sampling with the same number of points is applied. Experimental results with real range images representing both free-form and polyhedral and cylindrical objects are presented.

1 Introduction

Range sensors are becoming a popular way of obtaining 3D information in robotics owing to the development of low-cost devices able to provide dense images at high speeds [9]. However, the processing of dense range images containing hundreds of thousands of pixels is still costly and difficult to apply to real-time applications.

One way of speeding up the processing of range images consists of reducing the amount of data contained in the images while keeping enough details that allow the application of further algorithms, such as segmentation or shape recognition. This can be done by removing points in areas of low variation and keeping them in areas of high curvature. In general, the objective becomes the generation of a mesh that approximates the given range image with fewer data points. Further processing algorithms can then be applied to that mesh instead of to the dense range image directly [8].

Meshes generated from range images can be either regular (quadrilateral) or irregular. Irregular meshes do not impose any conditions upon the distribution of the data points they hold. Therefore, they allow the representation of both scattered and regularly distributed points in space. Among them, irregular triangular meshes are a widely-used representation owing to the availability of efficient triangulation algorithms. Several iterative optimization techniques have been proposed for approximating range images by triangular meshes (e.g., [2][10]). A faster approach which avoids such costly optimizations was proposed by García [6].

Quadrilateral meshes are a less flexible data representation compared to irregular meshes, since they require that the data points are distributed in rows and columns. However, quadrilateral meshes are convenient and sometimes necessary for different applications, for example, for estimating the surfaces of the original objects contained in the image by applying tensor-product B-splines or NURBS, which are the "de facto" standard for free-form surface modeling in CAD/CAM. Although many techniques have been devised for reconstructing surfaces from irregular meshes (e.g., [7]), they are unable to perform as good as tensor-product methods. Besides, it is always possible to obtain a triangular mesh from a quadrilateral one by splitting each rectangular cell into two triangles. This argument and the existence of algorithms that work with quadrilateral meshes (e.g., [4][5]) justify the utility of generating quadrilateral meshes from range images.

The simplest way of generating a quadrilateral mesh from a range image is by sampling that image at specific intervals both horizontally and vertically. However, this process produces aliasing effects, since details in the image between two sampled points will be ignored. Thus, the objective is the generation of non-uniform quadrilateral grids, that vary their point density depending on the amount of information (details) present in the image. The underlying goal is to be able to capture more information with the same amount of data points.

Similarly to the irregular case, previous methods for generating adaptive quadrilateral meshes from range images are based on iterative optimization techniques. Variational formulations and elastic models are some examples (e.g., [1][11]). A recent algorithm segments the given image into patches and fits quadrilateral meshes to each patch [3]. However, those previous techniques are

This work has been partially supported by the Government of Spain under the CICYT project TAP96-0868. The second author has been supported by the Spanish Agency for International Cooperation and the National University of La Pampa (Argentina).

computationally costly. Therefore, their application to fields with real-time requirements, such as robotics, may be more complicated.

This paper presents a fast technique for generating adaptive quadrilateral meshes from range images with no optimization. The aim is the generation of a quadrilateral mesh with a user-specified number of rows and columns so that the distribution of the points is better than a uniform distribution, in the sense that, with the same number of points in the mesh, the range image can be approximated keeping more details.

The proposed method divides the original range image into a set of rectangular regions or tiles. For each tile, an adaptive quadrilateral mesh is generated considering the curvature associated with the tile's image, such that points tend to concentrate in high-curvature areas and to disperse in low variation regions. Since each tile can be processed independently, the algorithm can benefit from the application of parallel architectures. Contrary to the randomized method proposed in [6], the new algorithm is fully deterministic given the number of rows and columns desired for each tile. The algorithm is described in section 2. Section 3 gives experimental results with images containing both free-form and planar and cylindrical objects. Finally, conclusions are given in section 4.

2 Adaptive Quadrilateral Mesh Generation

A range image is a sampling, usually rectangular, of a scene surface. Its usual representation is a two dimensional array **R**, where each array 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 to it at a specified distance. Invalid points in the image will be considered to have the background value β.

This section presents an algorithm for generating a quadrilateral mesh that approximates a given range image adaptively. The algorithm consists of two main stages. The first stage filters small gaps and computes a curvature estimation of the whole range image. Then, the second stage divides the range image into a user-specified number of equal-sized rectangular regions or tiles. Taking into account the previous curvature estimation, an adaptive quadrilateral mesh, with a user-specified number of rows and columns, is independently generated for every tile. Finally, the meshes generated for every tile are recombined into a final mesh which approximates the given range image. These stages are described below.

2.1 Gap Filling and Curvature Estimation

Given a range image **R** with R rows and C columns, a first stage of the algorithm removes single pixel gaps. This is done by substituting every background pixel surrounded by non-background pixels for the average of those neighbors. Let $S(r, c)$ be the set of non-background pixels that surround a given pixel $\mathbf{R}(r, c)$,

$$S(r,c) = \{\mathbf{R}(i,j) \neq \beta \mid i \in (0,R), j \in (0,C) \\ \mid i-r \mid = 1, \mid j-c \mid = 1 \}$$

Then, a filtered range image $\mathbf{R}_f(r, c)$ is computed as the average of all the pixels belonging to $S(r, c)$.

This produces a filtered range image which compensates for sensing errors leading to undesired single-pixel gaps. Gaps larger than one pixel have not been removed in considering that they are real gaps in the sensed 3D surface. Notwithstanding, the previous filtering can be successively applied to remove larger gaps if necessary. An optional filtering of the whole range image, by applying the same procedure to non-background pixels, can also be utilized in case of noisy range images. However, it has been discarded in our current implementation since it did not lead to significative improvements of the final mesh for the range images we have tested and also due to its tendency to remove details, such as sharp edges. This aspect is negative since the rest of the algorithm relies on the existence of distinctive features that determine the density of points of the final mesh. The problem can be overcome by applying multiresolution analysis (wavelets) instead of by simply averaging neighboring points.

From the filtered range image $\mathbf{R}_f(r, c)$, a curvature image $\mathbf{K}(r, c)$ is computed. $\mathbf{K}(r, c)$ is a function of the curvature associated with the pixel corresponding to $\mathbf{R}_f(r, c)$. $\mathbf{K}(r, c)$ is generated by merging the estimations of the curvature components along the horizontal $\mathbf{K}_{rc}^{R}$ and vertical $\mathbf{K}_{rc}^{C}$ directions of the image. First, two initial estimations are calculated,

$$\mathbf{K}_{rc}^{C'} = |\mathbf{R}_f(r, c-1) - 2\mathbf{R}_f(r, c) + \mathbf{R}_f(r, c+1)|$$
$$\mathbf{K}_{rc}^{R'} = |\mathbf{R}_f(r-1, c) - 2\mathbf{R}_f(r, c) + \mathbf{R}_f(r, c+1)|$$

Next, a threshold operator $(\alpha > 0)$ is applied,

$$\mathbf{K}_{rc}^{C} = \begin{cases} \mathbf{K}_{rc}^{C'} & \mathbf{K}_{rc}^{C'} \leq \alpha \\ \alpha & \text{otherwise} \end{cases}$$

$$\mathbf{K}_{rc}^{R} = \begin{cases} \mathbf{K}_{rc}^{R'} & \mathbf{K}_{rc}^{R'} \leq \alpha \\ \alpha & \text{otherwise} \end{cases}$$

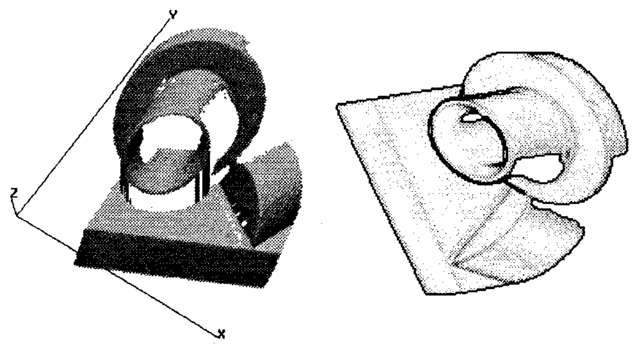

Figure 1: (*left*) Original range image (rendered). (*right*) Associated curvature image.

The curvature estimation is finally calculated as the logical addition of the binary representation of the previous terms: $\mathbf{K}(r, c) = \mathbf{K}^R_{rc} \vee \mathbf{K}^C_{rc}$.

According to this formulation, $\mathbf{K}(r, c)$ is a scalar varying between zero and α. The larger that value is, the larger the curvature at $\mathbf{R}_f(r, c)$ is. Parameter α has been set to 255 in order to store the curvature estimation as an 8-bit image while keeping enough resolution. Background pixels have been given a constant curvature value to guarantee that background regions are sampled and to prevent large concentrations of points at jump edges—as a quadrilateral mesh is sought, the background must also be covered. Actual background curvature settings are described in section 3.

With this approach, pixels with high values of $\mathbf{K}(r, c)$ correspond to areas with high surface variation. On the contrary, pixels with $\mathbf{K}(r, c)$ equal to zero correspond to planar regions. As in [6], the aim is to make the point density of the final mesh in a certain region of the range image be proportional to the curvature associated with the pixels of that region. Thus, points will tend to concentrate in high-variation areas, ensuring that details present in those areas are accounted for.

Fig. 1 shows a range image and its associated curvature image according to this technique. Black areas represent high curvature and white areas planar regions.

2.2 Range Image Tessellation

Both the filtered range image $\mathbf{R}_f(r, c)$ and its curvature estimation $\mathbf{K}(r, c)$ are partitioned into equal-sized rectangular tiles. Later on, an adaptive quadrilateral mesh will be independently defined for every tile. In this way, the density of points is locally adjusted to the shape of different regions of the image. That process can be run in parallel, taking advantage of high-performance architectures.

The number H of horizontal and V of vertical partitions of the range image (and its associated curvature image) is an input parameter of the algorithm.

A *window* W_{vh}, $v \in [0, V-1], h \in [0, H-1]$, is defined by two 2D points: an upper-left corner (ulr_{vh}, ulc_{vh}) and a bottom-right corner (brr_{vh}, brc_{vh}). Their coordinates are computed as:

$$ulr_{vh} = \frac{R}{V} v \qquad ulc_{vh} = \frac{C}{H} h$$

$$brr_{vh} = \begin{cases} \frac{R}{V}(v+1) & v < V-1 \\ R-1 & v = V-1 \end{cases}$$

$$brc_{vh} = \begin{cases} \frac{C}{H}(h+1) & h < H-1 \\ C-1 & h = H-1 \end{cases}$$

A window W_{vh} determines a *tile* (rectangular region) in the range image. The next stage of the algorithm computes an adaptive quadrilateral mesh for every tile independently. Adjacent windows (and therefore tiles) have an overlap of one line of pixels. Hence, the boundaries of the quadrilateral meshes generated for adjacent tiles will coincide.

2.3 Adaptive Mesh Generation

Let $\mathbf{R}_{vh}(r', c')$ be the range image tile defined by a window W_{vh} when the latter is applied to the filtered range image $\mathbf{R}_f(r, c)$:

$$\mathbf{R}_{vh}(r', c') = \mathbf{R}_f(r' + ulr_{vh}, c' + ulc_{vh})$$

The two parameters (r', c') are local to the tile: $r' \in [0, R/V]$, $c' \in [0, C/H]$. Let also $\mathbf{K}_{vh}(r', c')$ be the curvature image tile defined by the same window on the curvature image $\mathbf{K}(r, c)$. The objective now is the definition of a quadrilateral mesh which adapts to the surface contained in $\mathbf{R}_{vh}(r', c')$ based on its corresponding curvature estimation $\mathbf{K}_{vh}(r', c')$. Notice that each tile can be considered to be a small range image with its corresponding curvature estimation.

For every image tile $\mathbf{R}_{vh}(r', c')$, an adaptive quadrilateral mesh $HVS_{vh}(i, j)$ with $\mathcal{R}$ rows and ζ columns is generated. The number of rows and columns of that mesh is an input parameter of the algorithm and determines the number of rows and columns of the mesh corresponding to the whole range image.

Given the image tile $\mathbf{R}_{vh}(r', c')$ and its associated curvature estimation $\mathbf{K}_{vh}(r', c')$, two steps are necessary to generate a quadrilateral mesh. The first stage generates ζ vertical curves that adapt to the shape of the image tile. These curves tend to approach in areas of high curvature and to disperse in low-variation regions. The second stage samples each of these curves at $\mathcal{R}$ different points, so that the points also tend to concentrate in high-curvature areas and to disperse in low-variation ones. The outcome of this stage is an $\mathcal{R}$ x ζ array of points for every tile. Similar results are obtained by generating horizontal curves in the first stage.

In order to define a collection of vertical curves that adapt to the shape of the image tile $\mathbf{R}_{vh}(r', c')$, a set of ζ pixels is selected from each row of pixels of the tile based on its curvature estimation. This is done as follows.

Let $\mathbf{R}_{vh}(r', c')$, $c' \in [0, C/H]$ and r' fixed, be a row of pixels and $\mathbf{K}_{vh}(r', c')$ their corresponding curvatures. Following the principles utilized in [6], this curvature profile is mapped to an unnormalized probability density function that expresses the probability of selecting each pixel of the range image tile, so that pixels with high curvature will have a higher chance of being selected for the mesh. The discrete, unnormalized probability density function $f_{vhr'}(c')$ is defined by applying a transformation function to the curvature profile:

$$f_{vhr'}(c') = \mathcal{T}(\mathbf{K}_{vh}(r', c')) \qquad (1)$$

The transformation function $\mathcal{T}$ determines the variation of density of points with respect to a variation of curvature. A linear variation implies that for high curvature areas, the density of points is very high. This linear behavior produced final meshes with excessive concentration of points. However, if a logarithmic function is used, the density of points varies more gently. Thus, the transformation function that has been finally chosen is

$$\mathcal{T}(x) = K_{\mathcal{T}} \log x \qquad (2)$$

where $K_{\mathcal{T}}$ is a proportionality constant which determines the maximum value of the density function. It has been experimentally set to 500. Alternative sublinear functions can also be utilized, resulting in different point density variations. Even a customized transfer function can be devised by utilizing B-splines.

Next, a discrete, unnormalized probability distribution function $F_{vhr'}(c')$ is obtained as:

$$F_{vhr'}(c') = \sum_{i=0}^{c'} f_{vhr'}(i) - f_{vhr'}(0) \qquad (3)$$

If the value range of $F_{vhr'}(c')$ is sampled at ζ points uniformly distributed, the application of the inverse distribution function $F^{-1}{}_{vhr'}(y)$ to those points leads to a set of ζ points that are adaptively distributed according to $f_{vhr'}(c')$. This principle is illustrated in Fig. 2. In our case, since the probability distribution corresponds to the curvature estimation, the density of points will be correlated to the curvature and, hence, to shape variations.

In order to obtain $F^{-1}_{vhr'}(y)$ given the set of ζ points y uniformly distributed between 0 and the maximum $F_{vhr'}(c')$, a table keeping the values $F_{vhr'}(c')$ for all the $c' \in [0, C/H]$ is computed. Then, a single iteration

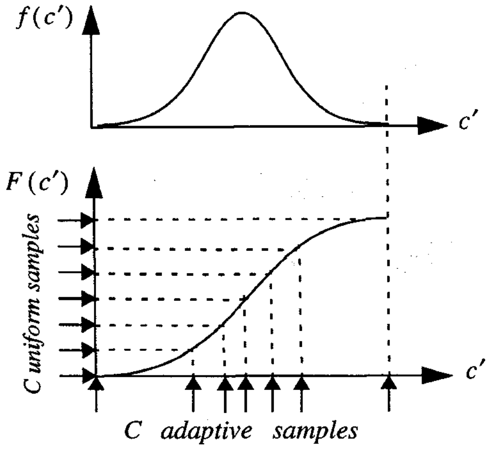

Figure 2: *(top)* An unnormalized probability density function $f(c')$ that represents curvature associated with every pixel c'. *(bottom)* Uniform sampling of the range of the corresponding unnormalized probability distribution function $F(c')$ gives a set of points whose density varies according to $f(c')$.

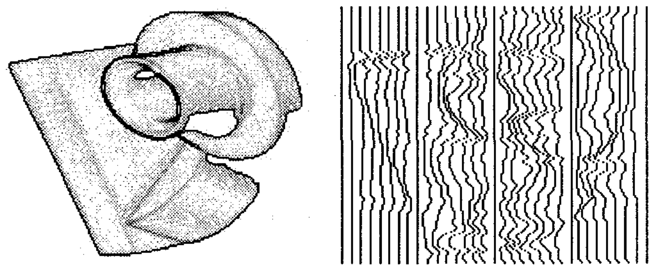

Figure 3: *(left)* Curvature image *(right)* Vertical curves computed after adaptive horizontal sampling, with $H = V = 4$ and $\zeta = 9$.

traverses this table, extracting those positions c' such that $F_{vhr'}(c') = y$. A vector of horizontal sampled points $HS_{vhr'}(j)$, $j \in [0, \zeta)$, keeping the different c's is obtained in this way.

This process is repeated for every row r' of the range image tile $\mathbf{R}_{vh}(r', c')$, producing an $(R/V+1)$ x ζ array:

$$HS_{vh}(r', j) = HS_{vhr'}(j), \quad r' \in [0, R/V], \; j \in [0, \zeta)$$

For each value j, if we iterate over r', we obtain a collection of points $(r', HS_{vh}(r', j))$ that determine a "vertical" curve in the range image. Going over all the different j values, we obtain a collection of vertical curves that tend to adapt to the shape of the underlying objects contained in the range image, moving together in areas of fast shape variation. Fig. 3 shows the set of vertical curves obtained by applying this procedure to all the tiles of the range image shown in Fig. 1, considering a tessellation in 4 by 4 tiles with 9 columns per tile ($\zeta = 9$).

Each vertical curve obtained above corresponds to one of the columns of the final quadrilateral mesh associated with the tile being processed. In order to obtain the rows of the quadrilateral mesh, each of these curves is adap-

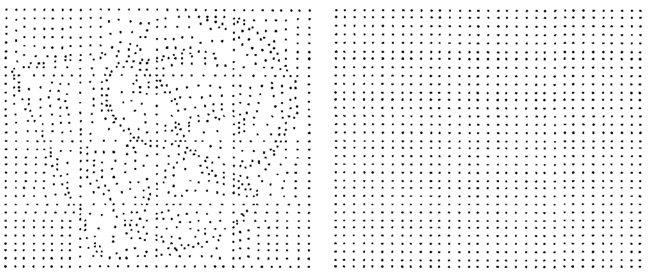

Figure 4: (*left*) Horizontal and vertical sampled points with $H = V = 4$ and $\zeta = 9$. (*right*) Same number of points with uniform sampling.

tively sampled at $\mathcal{R}$ positions ($\mathcal{R}$ is the input parameter that indicates the number of rows per tile). The process is similar to the previous one, which leads to horizontal samples from rows of pixels extracted from the tile. The difference now is that a curvature profile is obtained from the positions of the points that belong to one of the curves instead that from the positions corresponding to a horizontal 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 2D positions of the points belonging 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): $f_{vhj}(r') = \mathcal{T}(\mathbf{K}_{vh}(r', VC_{vhj}(r')))$. Similarly to (3), an unnormalized probability distribution function $F_{vhj}(r')$ is computed. Then the image space of this distribution function is uniformly sampled with $\mathcal{R}$ points y and the inverse distribution function $F_{vhj}^{-1}(y)$ applied to them to obtain a set of $\mathcal{R}$ points r' such that $F_{vhj}(r') = y$. A vector of vertical sampled points $VS_{vhj}(i)$, $i \in [0, \mathcal{R})$, keeping the different r's is obtained in this way. In the end, we obtain an $\mathcal{R} \times \zeta$ array of horizontal and vertical sampled points:

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

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 2D coordinates (r', c') of the point selected for each row i and column j of the adaptive quadrilateral mesh corresponding to the given tile. Fig. 4(*left*) shows the set of vertical and horizontal sampled points obtained for all the tiles in which the original range image has been tessellated (4 by 4). Fig. 4(*right*) shows the same number of sampled points considering uniform sampling. In the adaptive distribution, points tend to concentrate in areas of high curvature, highlighting the shape of the objects contained in the image. Notice that since the sought mesh is rectangular, background areas are also sampled.

In order to generate the final mesh corresponding to the range image as a whole, the quadrilateral meshes associated with each tile are merged. By construction, the boundaries of adjacent tiles overlap. Therefore, the final mesh will contain a total of $(\mathcal{R} - 1)V + 1$ rows and $(\zeta - 1)H + 1$ columns, with $(\mathcal{R}, \zeta)$ being the number of rows and columns per tile, and (V, H) the number of vertical and horizontal partitions of the original range image into tiles.

This mesh is stored in a 2D array HVS which keeps, for every position, the (r, c) coordinates of the sampled points:

$$HVS(v(\mathcal{R} - 1) + i, h(\zeta - 1) + j) = HVS_{vh}(i,j) + (ulr_{vh}, ulc_{vh})$$

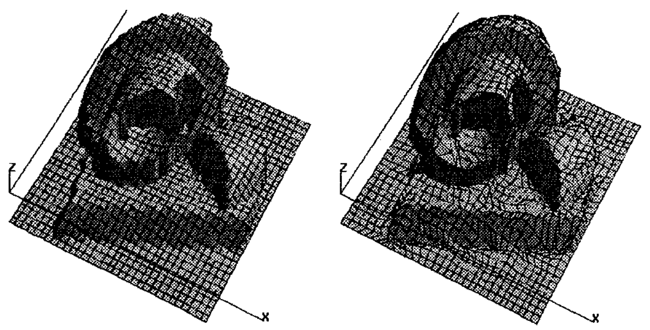

Figure 5: (*left*) Uniform quadrilateral mesh with 31 x 31 points. (*right*) Adaptive quadrilateral mesh with 31 x 31 points.

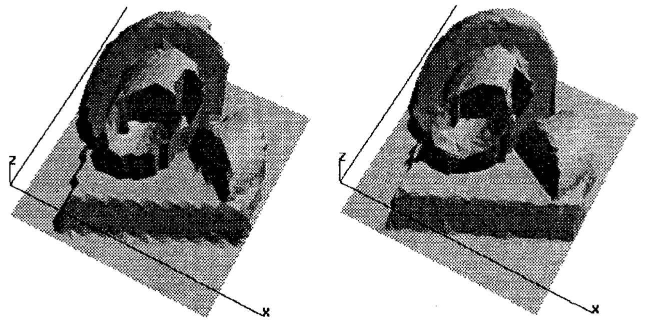

Figure 6: (*left*) Rendering of the uniform quadrilateral mesh with 31 x 31 points. (*right*) Rendering of the adaptive quadrilateral mesh with 31 x 31 points. CPU generation time = 0.21 sec.

3 Experimental results

Fig. 5 shows a 3D view of both the uniform (*left*) and the adaptive (*right*) quadrilateral meshes corresponding to the range image shown in the previous figures. The proportionality constant of the transformation function in (2) was experimentally set to 500.

The curvature associated with the background pixels was set to 5. This constant determines the density of points sampled over background regions when these regions join with non-background regions in a same tile. So far, this value is heuristically set. An automatic procedure is being studied that adjusts this constant independently for every tile as a function of the overall curvature of the non-background regions of the tile. The aim is to sample the background uniformly and, at the same time, not to concentrate many points in it so that non-background areas may result undersampled.

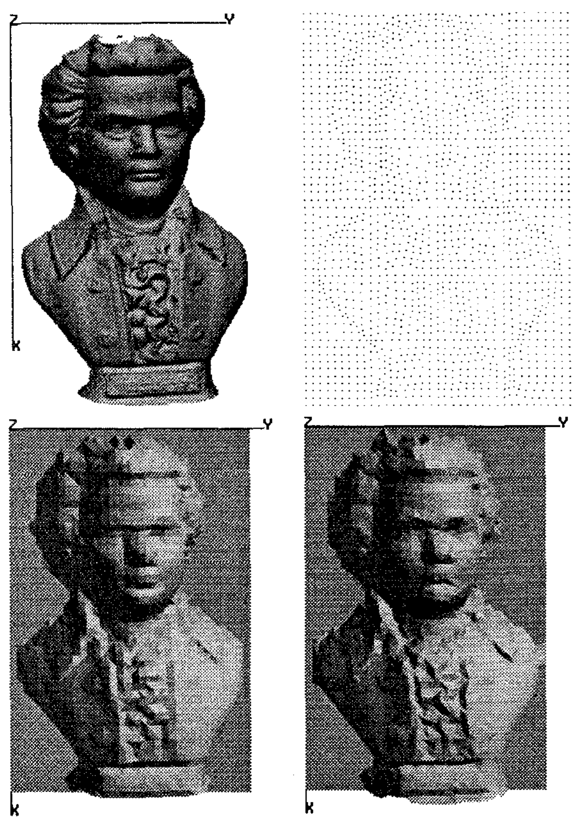

Figure 7: (top-*left*) Range image (399x263). (top-*right*) Adaptive sampling with 55x37 points. (bottom-*left*) Uniform mesh. (bottom-*right*) Adaptive quadrilateral mesh. CPU time = 0.82 sec.

Fig. 6 shows a rendered version of both the uniform (*left*) and adaptive (*right*) meshes. To display a 3D solid, quadrangular cells have been split into two triangles by cutting each cell along the diagonal that minimizes the error with the original range image. Notice that the uniform mesh exhibits artifacts along crease and jump edges. These artifacts are reduced in the adaptive sampling.

The range image of the previous example contains 154 rows and 183 columns while the final quadrilateral meshes (uniform and adaptive) contain 31 rows and columns. The CPU time to compute the adaptive quadrilateral mesh is 0.21 sec. using an SGI Crimson workstation with a 100MHz R4000 processor. A CPU time of 0.1 sec. was measured with an SGI Indigo II with a 200MHz R4400 processor.

Since the proposed technique tends to improve the approximation of the details of the range image (jump and crease edges, high-curvature shapes) by reducing the aliasing effect of uniform sampling, and these features usually represent small details in the overall range image, the *global relative error of approximation* [6] does not show big differences between the uniform and adaptive meshes. Differences are qualitative rather than quantitative. Even though, the improvement is also noticeable, with a relative error of 0.33% for the adaptive mesh versus a 0.45% for the uniform one.

Fig. 7 shows an example of the application of this technique to a range image containing a free-form surface. The original image has 399 rows and 263 columns. This image is tessellated into 4x6 tiles with 10 rows and columns per tile. The background curvature was set to 20. The final mesh of 55 rows and 37 columns was generated in 0.82 sec. (0.41 sec. with the 200 MHz SGI Indigo II). The relative error of the adaptive mesh is 0.26% versus 0.32% of the uniform mesh.

4 Conclusion

A fast technique for obtaining adaptive quadrilateral meshes from range images has been presented. The generated meshes improve the quality of approximation of quadrilateral meshes obtained through uniform sampling, by concentrating points in areas of high curvature while keeping the regularity constraint. The proposed technique is fully deterministic and avoids costly iterative optimization algorithms, making it suitable for real-time applications. Besides, it is inherently parallel. Hence, it can benefit from the application of parallel architectures.

An implementation of the proposed technique in C language is available by contacting the authors.

5 References

[1] R.M. Bolle and B.C. Vemuri, On three-dimensional surface reconstruction methods, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 13, no. 1, pp. 1-13, Jan. 1991.

[2] L. DeFloriani, A pyramidal data structure for triangle-based surface description. *IEEE Computer Graphics and Applications*, pp. 67-78, March 1989.

[3] M. Eck and H. Hoppe, Automatic reconstruction of B-spline surfaces of arbitrary topological type, *SIGGRAPH '96*, 325-334.

[4] P. Fillatreau, M. Devy, and R. Prajoux, Modelling of unstructured terrain and feature extraction using B-spline surfaces, *Int. Conf. on Advanced Robotics*, 1993, 279-284.

[5] H. Gagnon, M. Soucy, R. Bergevin and D. Laurendeau, Registration of multiple views for automatic 3-D model building, *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 1994, 581-586.

[6] M. A. García, Fast approximation of range images by triangular meshes generated through adaptive randomized sampling. *IEEE Int. Conf. on Robotics and Automation*, Nagoya, Japan, May 1995, 2043-2048.

[7] M. A. García and L. Basañez, Efficient free-form surface modeling with uncertainty. *IEEE Int. Conf. on Robotics and Automation*, Minneapolis, USA, April 1996, 1825-1830.

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

[9] K. Hattori and Y. Sato, Handy rangefinder for active robot vision, *IEEE Int. Conf. on Robotics and Automation*, 1995, 1423-1428.

[10] M. Soucy, A. Croteau and D. Laurendeau, A multi-resolution surface model for compact representation of range images, *IEEE Int. Conf. on Robotics and Automation*, 1992, 1701-1706.

[11] D. Terzopoulos and M. Vasilescu, Adaptive surface reconstruction, *SPIE Vol. 1383 Sensor Fusion III: 3-D Perception and Recognition*, 1990, 257-264.