Approximation and Processing of Intensity Images with Discontinuity-Preserving Adaptive Triangular Meshes

Miguel Angel García[1], Boris Xavier Vintimilla[2] and Angel Domingo Sappa[3]

[1] Department of Computer Science and Mathematics
Rovira i Virgili University
Ctra. Salou s/n. 43006 Tarragona, Spain
magarcia@etse.urv.es
[2] Institute of Organization and Control of Industrial Systems
Polytechnic University of Catalonia
Diagonal 647, planta 11. 08028 Barcelona, Spain
vintimi@ioc.upc.es
[3]LAAS - CNRS, Office B157
7, Avenue du Colonel Roche
31077 Toulouse, Cedex 4, France
asappa@laas.fr

Abstract. A new algorithm for approximating intensity images with adaptive triangular meshes keeping image discontinuities and avoiding optimization is presented. The algorithm consists of two main stages. In the first stage, the original image is adaptively sampled at a set of points, taking into account both image discontinuities and curvatures. In the second stage, the sampled points are triangulated by applying a constrained 2D Delaunay algorithm. The obtained triangular meshes are compact representations that model the regions and discontinuities present in the original image with many fewer points. Thus, image processing operations applied upon those meshes can perform faster than upon the original images. As an example, four simple operations (translation, rotation, scaling and deformation) have been implemented in the 3D geometric domain and compared to their image domain counterparts.[1]

1 Introduction

The standard formats commonly used for image compression, such as GIF and JPEG, were not originally devised for applying further processing. Thus, images codified in those formats must be uncompressed prior to being able to apply image processing operations upon them, no matter how big and redundant the images are. Nonetheless, some researchers have managed to apply various basic operations upon compressed representations. For example, [1] presents a technique for applying arithmetic operations directly to JPEG images. Several techniques for image manipulation and feature extraction in the DCT domain are also presented in [2].

An alternative to the problem of compactly representing images consists of the

[1] 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 ESPOL University (Ecuador).

utilization of geometric representations, such as triangular meshes. Those meshes allow the modeling of large areas of pixels with basic geometric primitives. For example, a large white region can be represented by a few triangles instead of by hundreds of pixels. Geometric representations are applicable since the pixels of an image can be considered to be 3D points in a space in which coordinates x and y are functions of the rows and columns of the image, and coordinate z is a function of the gray level. An additional advantage of using geometric representations is that they allow the application of techniques that have been successfully utilized in other fields, such as computer graphics or computer vision (e.g., [3][6][7][15]).

Some algorithms have been proposed for generating geometric approximations from images (e.g., [8][9]). These algorithms generate an initial high-resolution mesh by mapping each pixel of the original image to a point in a 3D space and by linking those points according to some topological criteria. Then, an iterative, optimization-based algorithm decimates the obtained mesh until either a certain maximum error between the current mesh and the original image is reached or a certain number of points is attained. A recent algorithm [5] decimates an initial mesh based on a non-optimization-based iterative algorithm that ensures a maximum approximation error. The inconvenience of that method is that, since image discontinuities are not explicitly modeled, an oversampling of the given images is generated.

Besides the use of geometric representations as tools for image modeling, little research has been done regarding their utilization for simplifying and accelerating general image processing operations. For instance, [4] presents a technique for segmenting range images approximated with adaptive triangular meshes. Furthermore, [10] and [16] present an algorithm for segmenting intensity images. This algorithm is based on an initial triangulation of corners detected in the image, followed by an iterative, optimization-based split-and-merge technique applied to the triangles of the mesh.

This paper presents an efficient algorithm for approximating intensity images with adaptive triangular meshes without applying iterative optimization. The obtained meshes preserve the shades and discontinuities present in the original image. Furthermore, an efficient technique for converting triangular meshes to intensity images is also described. Finally, it is shown how the previously obtained triangular meshes can be utilized to accelerate image processing operations through four basic translation, rotation, scaling and deformation operations.

This paper is organized as follows. The proposed approximation algorithm is described in section 2. Section 3 presents a technique for generating intensity images from triangular meshes. Section 4 describes the implementation of four simple image processing operations applied upon the previously obtained triangular meshes. Experimental results are shown in section 5. Conclusions and further improvements are presented in section 6.

2 Approximation of Intensity Images with Adaptive Triangular Meshes

This section presents a technique for approximating intensity images with discontinuity-preserving adaptive triangular meshes. The triangular meshes are composed of points sampled over the original image. Each point corresponds to a certain pixel and is defined by three coordinates: row, column and gray level. The proposed technique consists of two stages. The first stage adaptively samples the given image at a set of pixels. First, the image edges are detected, adaptively sampled and approximated by polylines. Then, the internal regions comprised between image edges are adaptively sampled. The second stage triangulates the points sampled over internal regions, using the previously obtained polylines as constraints for the triangulation. Both stages are described below.

2.1 Image Adaptive Sampling

The aim of this stage is to sample the given image, obtaining a set of points that are distributed according to the shades present in the image. The triangulation of those points will be used as a higher abstraction-level representation of the original image. Edges (contours) and internal regions in the image are approximated separately.

2.1.1 Edge Adaptive Sampling

Intensity images usually contain sudden changes in gray level due to region boundaries. The edge sampling stage approximates those boundaries with adaptive polylines. First, the edges present in the original image are found by applying Canny's edge detector [11] and then by thresholding the result so that all pixels with a value above zero are set to gray level 0 (black) while the other pixels are set to gray level 255 (white). Thus, an *edge image* is generated, such as it is shown in Fig. 1 (*right*).

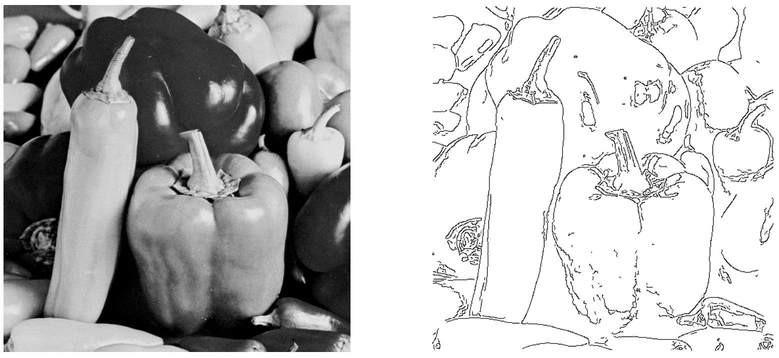

Fig. 1. (*left*) Original image of 512x512 pixels. (*right*) Edge image generated from the previous image**.**

Then, each edge in the edge image is adaptively approximated by a collection of segments that constitute a *polyline*. The points that define the segments of a polyline are obtained through the following iterative procedure.

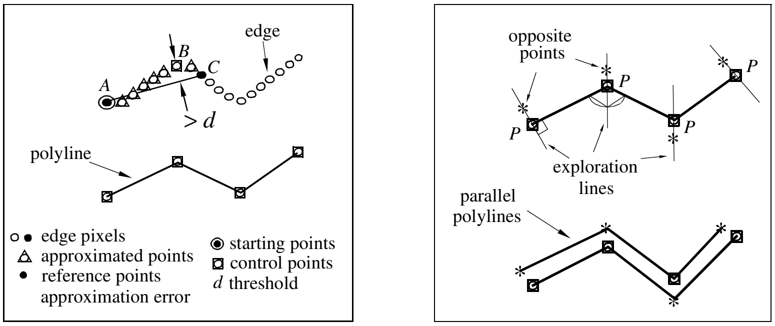

Fig. 2. Edge sampling process.(*left*) Edge extraction. (*right*) Edge unfolding.

First, the edge image is scanned from left to right and top to bottom until a pixel contained in an edge is found. This pixel is chosen as the *starting point, A*. The chosen edge is traversed from the starting point, and a second pixel contained in the same edge and placed at a user defined number of pixels away from the starting point is selected. This second pixel is the *reference point, C*. Both, the starting point and the reference point generate an *approximating segment*, $\overline{AC}$. The pixels that belong to the chosen edge comprised between the starting point and the reference point constitute the *approximated points*. The distance in image coordinates between each approximated point and the approximating segment is the *approximation error*, . If all the current approximation errors are below a given threshold *d*, a new reference point is selected by advancing the previous reference point *C* a fixed number of pixels along the chosen edge. Then, a new segment, joining the starting point and this new reference point is defined and the previous procedure is iterated.

When an approximated point is found to have an error above *d*, that point is chosen as the new starting point, *B*. The edge is traversed in this way until either one of its extremes is reached or a bifurcation is found. The polyline that approximates the previous edge with an error bounded by *d* is the set of segments that join all the starting points found during the exploration of the edge, plus the final point in the edge (extreme or bifurcation). The points that define the polyline constitute the *control points*. Fig. 2 (*left*) shows an example of the previous procedure.

When an edge has been successfully approximated by a polyline, all the points traversed during the process are removed from the edge image so that they do not intervene in further edge approximations. This polyline extraction procedure is applied until all edges have been approximated and, therefore, the edge image is white. Since the starting points of the polylines are found by applying the aforementioned scan-line algorithm, different executions of the edge sampling process upon a same image will produce the same polylines.

Each polyline obtained above delimits a boundary between two neighboring regions, indicating a discontinuity in the gray level values. The points that form the polyline (control points) correspond to pixels that can be located at both sides of the discontinuity. However, the final 3D triangular mesh requires that these discontinuities be modeled as vertical "walls". These walls can only be produced by unfolding each obtained polyline into two parallel polylines, each at a different gray-level (height). This process is done by computing an *opposite point* for each polyline's control point. Each opposite point will be located at the other side of the discontinuity in which its corresponding control point lies.

Given a control point *P*, its corresponding opposite point is obtained as follows. First, an *exploration line* that bisects the polyline's segments that meet at *P* and that passes through *P* is computed, Fig. 2 (*right*). The pixels traversed by this line are explored in both directions starting with *P*. The first pixel along that line where a significative change of gray level occurs is chosen as *P*'s opposite point. The opposite points corresponding to the extremes of the polyline are determined by considering as exploration lines the lines perpendicular to the segments that abut at those extremes, and then by applying the previous criterion. Fig. 2 (*right*) shows an example of this procedure.

A new polyline is obtained for each original polyline by linking its corresponding opposite points. Since the control points that define the original polyline may not be located at the same side of the discontinuity, the new polyline and the original one may not be parallel and, therefore, may have some segments that self-intersect. To avoid this problem, the two polylines are traversed exchanging corresponding pairs of control and opposite points, such that each polyline only contains the points that are located at the same side of the discontinuity (all the points of a polyline must have a similar gray level). Thus, two parallel polylines are finally generated from each original polyline, one completely lying on one side of the discontinuity (at the region with the highest gray level) and the other completely lying on the other side (at the region with the lowest gray level), Fig. 2 (*right*). Fig. 3 shows the set of control points and opposite points corresponding to the example utilized so far.

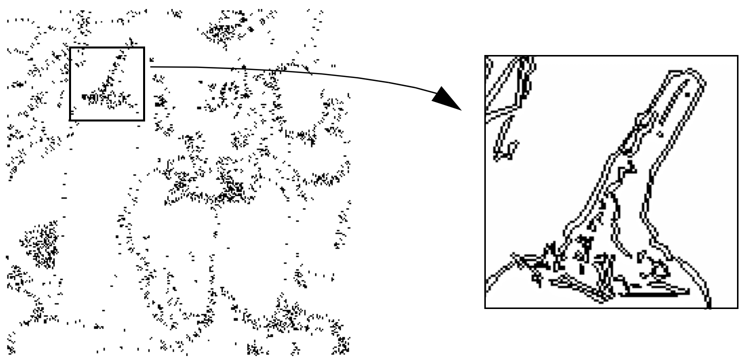

Fig. 3. (*left*) Set of both control and opposite points obtained by the edge adaptive sampling process (4,900 points). (*right*) Parallel polylines obtained from the previous points.

2.1.2 Region Adaptive Sampling

This stage aims at obtaining a set of points adaptively distributed over the image,

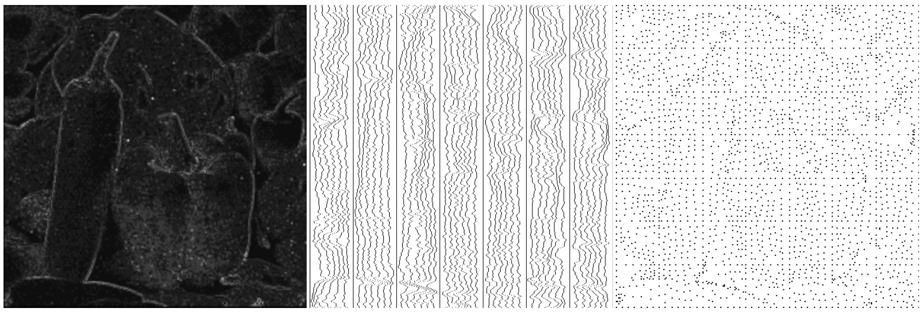

Fig. 4. (*left*) Curvature image. (*center*) Adaptively sampled vertical curves (7x7 tiles). (*right*) Set of points obtained by region adaptive sampling (2,324 points).

such that they concentrate in high-curvature regions and scatter over low variation regions. The sampling process must be applied by taking into account that all the edges in the image have already been considered by the previous step and do not have to be resampled.

This process is done by applying a non-optimization, adaptive sampling technique previously developed [3]. That technique, which was originally proposed for range images, efficiently samples a predefined number of pixels over the given image adapting to the curvatures present in it. In order to apply that sampling technique to intensity images, it suffices to consider each pixel as a 3D point with three coordinates, which correspond to the pixel's row, column and gray level. The adaptive sampling process is summarized below. A complete description can be found in [3].

First, a *curvature image* [3] is estimated from the original image, Fig. 4 (*left*). Since the image contours have already been approximated with parallel polylines (section 2.1.1), the curvatures of the pixels that belong to the contours of the original image, detected through Canny's edge detector, and their adjacent neighbors are reset so that they do not cause further resampling.

Both, the original and curvature images are then divided into a predefined number of rectangular *tiles*. The following steps are independently applied to each tile. First, a predefined number of points is chosen for each row of every tile, in such a way that the point density is proportional to the curvatures previously computed for the pixels of that row. After that *horizontal sampling*, a set of *vertical curves* is obtained, Fig. 4 (*center*).

Then, each vertical curve is adaptively sampled at a predefined number of points whose density is again proportional to the curvature estimated for the pixels contained in the curve. In the end, a predefined number of adaptively sampled points is obtained for every tile, Fig. 4(*right*). The number of both tiles and points per tile is defined by the user. Many tiles lead to uniformly-sampled meshes, while a few tiles lead to degenerated meshes. An intermediate value must be experimentally set.

The merging of the two previous sets of points, obtained after both edge sampling and region sampling, produces the final result of the adaptive sampling stage.

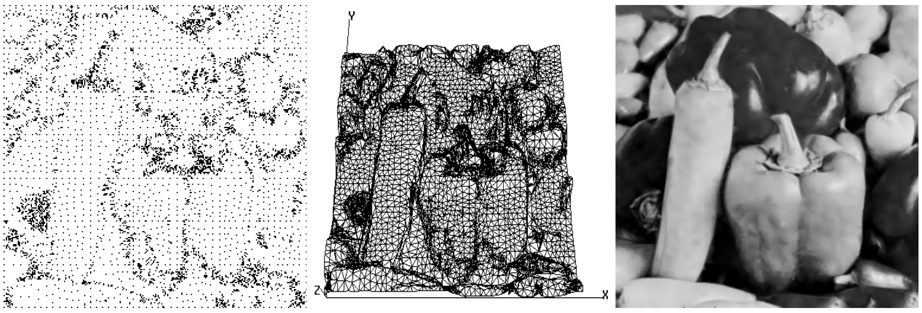

Fig. 5. (*left*) Final set of points obtained after the adaptive sampling process (7,224 points). (*center*) Adaptive triangular mesh generated from the previous points. (*right*) Approximating image obtained from the previous triangular mesh through z-buffering in 0.31 sec.

Fig. 5 (*left*) shows the set of sampled points that approximate the given original image.

2.2 Triangular Mesh Generation

The final aim of this stage is the generation of an adaptive triangular mesh from the set of points obtained after the adaptive sampling stage. This triangular mesh is an approximation of the given intensity image obtained by means of a constrained 2D Delaunay triangulation algorithm [12]. The triangulation is applied to the x and y coordinates of the sampled points. The constraints of the triangulation are the parallel polylines that approximate the detected contours of the image. These polylines can be either open or closed. In this way, it is guaranteed that the discontinuities of the gray level values are preserved as edges in the generated triangular mesh. Fig. 5 (*center*) shows the final adaptive triangular mesh obtained for the current example. The z coordinates of the vertices of that mesh correspond to the gray levels associated with them in the original image.

3 Generation of Intensity Images from Triangular Meshes

Any compression or coding algorithm requires a corresponding decompression or decoding counterpart that allows the recovery of data in the original format. Likewise, a tool for generating intensity images from adaptive triangular meshes is necessary.

Triangular meshes are utilized to represent intensity images by assuming that the first two dimensions of the points in the mesh correspond to row and column image coordinates, and the third dimension to a gray level. An intensity image can be generated from a triangular mesh by considering that each triangle of the mesh represents a plane that contains a set of points (pixels). The z coordinates of the points inside that plane represent gray level values in the resultant image. Hence, the image generation process can be based on computing the bounding box of every triangle *T* of the mesh, then finding the pixels of this box which are contained within *T*, and finally obtaining the *z* coordinates of these points by using the plane equation corresponding to *T*.

However, by taking advantage of the 3D nature of these triangular meshes, the computational cost of the previous algorithm can be reduced almost by half by applying a well-known algorithm, *z-buffering*, which is extensively utilized in computer graphics. Thus, the image generation stage has been finally implemented as follows. First, the triangular mesh is visualized in a window with the same size as the desired image through functions of the standard 3D OpenGL library. Next, the z-buffer obtained after this visualization is read with another OpenGL function (*glReadPixels*). Finally, the intensity image is obtained by linearly mapping the values of the z-buffer to gray levels in the range [0-255].

Since the implementations of the OpenGL library take advantage of hardware acceleration in most current computers (including PCs), the whole process turns out to be very fast. For example, the approximating image (with 262,144 pixels) corresponding to the example used so far was obtained in 0.31 seconds on a SGI Indigo II, Fig. 5 (*right*). Other examples are shown in Fig. 6 and Fig. 7.

The approximation error (RMS) of the image shown in Fig. 5 (*right*) is 12.1. This RMS error can be bounded to a desired tolerance by applying a previously developed algorithm [5], which approximates intensity images through bounded error triangular meshes.

4 Geometric Processing of Triangular Meshes

The triangular meshes obtained above are representations of intensity images at a higher level of abstraction. This allows the application of many image processing operations more efficiently than if they were applied upon the individual pixels of the original images. For example, translation, scaling, rotation and deformation operations are trivially implemented by applying affine transformations to the 3D coordinates of the points that constitute the meshes (see Fig. 6). Since those adaptive meshes contain a fraction of the original amount of pixels, these operations perform faster than their pixel-to-pixel counterparts. The actual results corresponding to the examples shown in Fig. 6 are presented in the next section.

Another operation that can benefit from the previously obtained triangular meshes is image segmentation. Image segmentation algorithms based on split-and-merge techniques must iteratively process all the pixels of the input images, grouping and ungrouping them according to some uniformity criteria. Alternatively, the triangles of an adaptive triangular mesh already capture some of those criteria. Therefore, triangles can be merged instead of pixels, leading to an overall speed-up. In order to implement such an image segmentation algorithm in the geometric domain, it is possible to apply a fast technique previously developed for segmenting range images [4]. This approach is more efficient than the one presented in [10][16], since the latter generates an adaptive triangular mesh by applying an optimization-based split-and-merge algorithm that considers the pixels contained in every triangle. Conversely, the proposed technique does not require any costly optimization

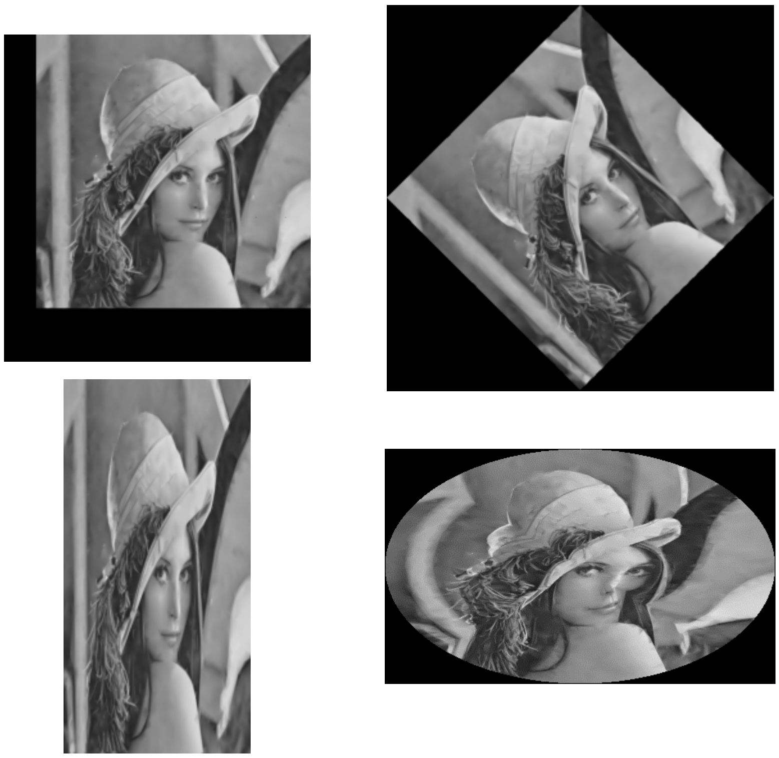

Fig. 6. Approximating images obtained by applying simple geometric operations. (*top-left*) Translation. (*top-right*) Rotation. (*bottom-left*) Scaling. (*bottom-right*) Ellyptical deformation.

stages.

These basic operations are just a few examples of how triangular meshes can help speed up conventional image processing. Many other computer vision and image processing tasks are also susceptible to be accelerated with geometric processing of adaptive triangular meshes. This will be the subject of further research.

5 Experimental Results

The proposed approximation algorithm has been tested with intensity images of different size and also compared to both a uniform (non-adaptive) sampling technique and a mesh decimation technique based on iterative optimization [13]. A public implementation of the latter technique (*Jade*) has been utilized.

The uniform sampling technique consists of choosing one pixel out of a predefined number of pixels along the rows and columns of the image. On the other hand, the optimization-based technique (Jade) starts with a high resolution triangular mesh containing all the pixels from the image, and decimates it until either a certain number of points is obtained or the approximating error is above a threshold. In

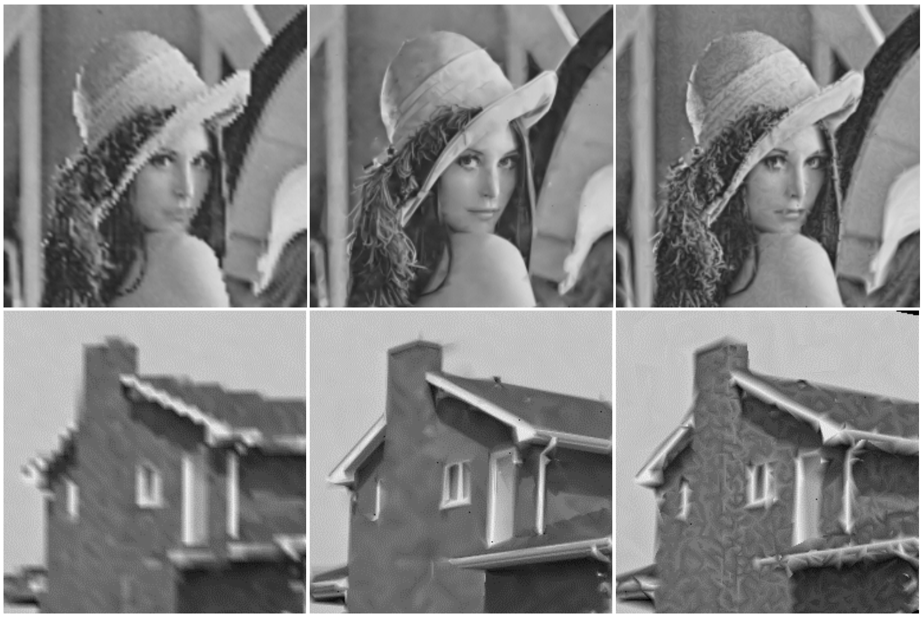

Fig. 7. Approximating images. (*left column*) Uniform (non-adaptive) sampling. RMS errors: 14.3 (*top*) and 18.3 (*bottom*). (*center column*) Proposed technique. RMS errors: 9.8 (*top*) and 10.6 (*bottom*). (*right column*) Optimization-based technique (Jade). RMS errors: 11.7 (*top*) and 15.3 (*bottom*).

order to be able to compare these techniques with the proposed one, Jade and the uniform sampling process were run to produce triangular meshes with the same number of points as the ones obtained with the proposed technique.

Fig. 7 shows two of the test images. The triangular meshes generated from these images with the proposed technique were obtained in 7.64 (Lenna) and 1.26 (House) seconds. All CPU times were measured on a SGI Indigo II with a 175MHz R10000 processor. These meshes contain 7,492 and 1,649 points respectively. The RMS errors of the approximating images are 9.8, Fig. 7 (*top-center*), and 10.6, Fig. 7 (*bottom-center*). If subsequent image processing operations were applied upon these triangular meshes, those points would be the only ones to be processed. The same operations applied upon the original images would require the processing of 262,144 and 65,536 pixels respectively, which is between one and two orders of magnitude larger.

The proposed technique produced better image approximations than both the uniform sampling technique and the optimization based technique (Jade). For example, given the same number of points, the proposed technique always produced lower RMS errors (e.g., 9.8 and 10.6 in Fig. 7) than the uniform sampling technique (e.g., 14.3 and 18.4 in Fig. 7) and than the optimization based technique (e.g., 11.7 and 15.3 in Fig. 7). The reason is that the proposed technique explicitly models the discontinuities in the image, while optimization-based techniques, such as Jade, are only able to keep those discontinuities by concentrating large numbers of points. Moreover, the CPU times necessary to generate the adaptive triangular meshes corresponding to the two previous examples were two orders of magnitude faster with the proposed technique (e.g., 7.64 and 1.26 sec) than with Jade (e.g., 1,580 and 675 sec).

Finally, the CPU times to perform the basic operations shown in Fig. 6 were measured and compared with the times to perform similar operations with a conventional image processing software (CVIPtools) publicly available [14]. The images of Lenna shown in Fig. 6 are approximating images obtained from an adaptive triangular mesh of 7,492 points computed with the proposed technique. This corresponds to an RMS error of 9.84 with respect to the original image, which contains 512x512 (262.144) pixels. The CPU times to perform the translation with CVIPtools was 0.32 sec., while the same operation in the geometric domain took 0.00087 sec. The rotation operation took 2.23 sec. with CVIPtools and 0.02 sec. with the proposed technique. Finally, the scaling operation took 0.33 sec. with CVIPtools and 0.02 sec. with the proposed technique in the 3D geometric domain. CVIPtools does not include any routines for producing deformations such as the elliptical one shown in Fig. 6. Therefore, it should be implemented with a user program that would access the given image, pixel after pixel, with the subsequent time penalty. Similarly, all the image deformations typically found in Adobe's Photoshop-like image processing packages are easily implementable in the 3D geometric domain by trivial mesh deformations, requiring a fraction of the time utilized in the image domain.

In all the examples considered in this section, the given times do not include the mesh generation and image reconstruction stages. The reason is that these stages must only be applied once: to map the original image to the geometric domain and to map the resulting mesh back to image space. If many operations are performed (chained) in the geometric domain, the overhead of those two stages will become negligible.

6 Conclusions and Future Lines

This paper presents a technique for approximating intensity images with discontinuity-preserving adaptive triangular meshes without optimization, and explores the use of those meshes to accelerate conventional image processing operations. The adaptive meshes generated with this algorithm are obtained faster than with optimization-based algorithms and, since image discontinuities are explicitly handled, the results are better than the ones obtained through both uniform (non-adaptive) sampling and optimization-based algorithms. The paper also explores the utilization of adaptive triangular meshes for accelerating image processing operations. Basic translation, rotation, scaling and deformation operations have been developed in the geometric domain and compared to a conventional image processing software, showing a faster performance.

We are currently developing new algorithms for implementing conventional processing operations (e.g., feature extraction, image enhancement, pattern recognition) directly in the geometric domain. The application of this technique to color images will also be studied.

We thank Jonathan R. Shewchuk for providing us with the 2D Delaunay triangulation algorithm (*Triangle*) [12].

7 References

[1] B. Smith and L. Rowe. "Algorithms for Manipulating Compressed Images". *IEEE Computer Graphics and Applications*, vol. 13, no. 5, September 1993, 34-42.

[2] Shih-Fu Chang. "Compressed-domain techniques for image/video indexing and manipulation". *IEEE Int. Conf. on Image Processing*, Washington DC, October 1995.

[3] M. A. García, A. D. Sappa and L. Basañez. "Efficient approximation of range images through data dependent adaptive triangulations". *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, Puerto Rico, June 1997, 628-633.

[4] M. A. García and L. Basañez. "Fast extraction of surface primitives from range images". *IEEE Int. Conf. on Pattern Recognition*, Vienna, 1996, 568-572.

[5] M. A. García, Boris X. Vintimilla and A. D. Sappa. "Efficient approximation of gray-scale images through bounded error triangular meshes". *IEEE Int. Conf. on Image Processing*, Kobe, Japan, October 1999.

[6] Leila De Floriani. "A pyramidal data structure for triangle-based surface description". *IEEE Computer Graphics & Applications*, 9 (2), March 1989, 67-78

[7] R. C. Wilson and E. R. Hancock. "A minimum-variance adaptive surface mesh". *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, Puerto Rico, June 1997, 634-639.

[8] D. Terzopoulos and M. Vasilescu. "Sampling and reconstruction with adaptive meshes". *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, Hawaii, USA, June 1991, 70-75.

[9] M. Vasilescu and D. Terzopoulos. "Adaptive meshes and shells: irregular triangulation, discontinuities and hierarchical subdivision". *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, Champaign, USA, June 1992, 829-832.

[10] T. Gevers and V. K. Kajcovski. "Image segmentation by directed region subdivision". *IEEE Int. Conf. on Pattern Recognition*, Jerusalen, Israel, Octuber 1994, 342-346.

[11] J. Canny. "A computational approach to edge detection". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, November 1986, 679-698.

[12] J. R. Shewchuk. "Triangle: engineering a 2D quality mesh generator and Delaunay triangulator". *First Workshop on Applied Computational Geometry*, Philadelphia, Pennsylvania, ACM, May 1996, 124-133.

[13] A. Ciampalini, P. Cignoni, C. Montani and R. Scopigno. "Multiresolution decimation based on global error". The Visual Computer, Springer-Verlag, 13(5), June 1997.

[14] S. E. Umbaugh. Computer Vision and Image Processing. Prentice-Hall International Editions, 1998.

[15] D. Molloy and P. F. Whelan. "Active-mesh self-initialisation". *Irish Machine Vision and Image Processing Conference*. P. Whelan (Ed.), ISBN 1 872 327 222, Dublin, Ireland, September 1999, 116-130.

[16] T. Gervers and A. W. Smeulders. "Combining region splitting and edge detection through guided Delaunay image subdivision". *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, Puerto Rico, June 1997, 1021-1026.