Imesh: An Image Based Quality Mesh Generation Technique

Alex J. Cuadros-Vargas, Luis Gustavo Nonato, Rosane Minghim and Tiago Etiene
Instituto de Ciências Matemáticas e de Computação
CP 668, São Carlos 13560-970, São Paulo, Brazil
{alexj, gnonato, rminghim}@icmc.usp.br, tiago@lcad.icmc.usp.br

Abstract

Generating triangular meshes from images is a task important to many applications. Usually, techniques that can do that either take as starting point a segmented image or generate a mesh without distinguishing different structures contained in the image. In both cases the results can be satisfactory for a number of applications, but the pre-segmentation and the absence of well defined structures imply in difficulties using the resulting mesh for simulations. Furthermore, guarantee of good quality meshes is also a common problem in previous results. In this work we present a new technique for mesh generation that handles these problems well. First, it eliminates the need for pre-processing by building the segmentation into the mesh generation process. Second, the mesh generation process takes into acount the quality of the mesh elements, producing as result meshes of better quality than previous techniques.

1 Introduction

Numerical simulation in domains defined from images has stimulated the technological development in many branches of science. Typical examples of this kind of application are blood flow simulation, elastic deformation of organic structures, and studies of structure cracks and their propagation. A main drawback in this context is to generate a mesh that fits the structures of interest contained in the images while being adequate for numerical simulation.

Opposite to the classical mesh generation problem [1], little has been achieved towards automating the process of generating a mesh directly from image data. Most algorithms described in the literature strongly rely on extensive pre-processing steps. A typical approach consists in segmenting regions of interest from the images, using the boundaries of such regions as input to a mesh generator [4]. Although largely employed, such an approach demands user intervention and specific segmentation softwares in order to obtain well defined models.

Recently, some techniques that act directly on images have been presented [10, 18, 19], but these techniques are neither able to distinguish different structures in the image, nor to ensure meshes of good quality. The identification of distinct structures in the image is essencial in applications such as multi-fluid flows [16] and fluid-structure interaction [3], since these applications take into account the interface among regions to define interfacial strengths. Additionally, good quality meshes are primordial for accurate and efficient numerical solutions of natural phenomena modeled by partial differential equations. Therefore, in order to be effective, any mesh generator dedicated to numerical applications in domains defined from images must be concerned with both aspects discussed here.

In this work we present a novel algorithm, named *Imesh*, which can automatically generate a mesh from a given image while identifying different regions and maintaining good quality of the triangular elements. The regions are identified by building a segmentation strategy into the mesh generation process. In fact, this segmentation first guides an initial refinement step that aims at tracking the boundaries of the regions contained in the image. Then, it groups such regions by employing a region growing strategy. The quality of the mesh is obtained by adapting Ruppert's algorithm [13] to work in the "segmented" mesh. Therefore, the proposed algorithm assembles the two aspects mentioned above, namely, the automatic generation of a mesh that distinguishes different structures in the image and the generation of good quality mesh elements.

Section 2 gives an overview of related work on image based mesh generation. Basic definitions necessary to a better understanding of *Imesh*are given in Section 3. *Imesh*is detailed in Section 4. Section 5 shows the results obtained with our approach as well as some numerical simulations carried out from the meshes produced by our method. Conclusions and discussion of future work are given in Section 6.

2 Related Work

The techniques devoted to generate meshes from images either aim at building meshes to represent images or are concerned with numerical simulation. In the following, we present a brief description of work related with both approaches.

The literature has presented a large set of techniques to generate meshes for image representation (such techniques are also called mesh modeling). In general, these techniques intend to build a mesh that minimizes an error measure between the original image and the approximate image generated by interpolation based on pixel values in the vertices of the mesh. Garcia et al. [7], for example, have presented an algorithm that controls the maximum root-mean-square error (RMS) by choosing the vertices of the mesh from a curvature image, that is, more vertices are placed in areas with high curvatures. The mesh model is built by generating the Delaunay triangulation [6] from the chosen vertices. Regions with high RMS error are resampled and the Delaunay triangulation updated. Garcia's method is a typical example of an adaptive approach, which is characterized by begining with an initial mesh that is iteratively refined in order to reduce the interpolation error. Many algorithms to represent images by meshes are based on adaptive approaches [11, 8, 9]. Alternatively, some techniques have adopted an opposite strategy, i.e., a fine mesh is successively coarsened until the approximation error reaches a tolerance [5]. Mixed approaches that combine refinement and coarsening [12] as well as optimization for re-positioning the vertices in minimum error places have also been developed [17]. Still envisioning mesh modeling, Yang et al. [18] proposed a one pass method that makes use of zero-crossing jointly with an error diffusion algorithm to choose a set of vertices from which the Delaunay triangulation is built. Besides reducing the approximation error, the authors argue that this strategy produces a better quality mesh.

Image based mesh generation for numerical simulation, in general, comprises two main steps, namely, preprocessing and mesh generation itself. The pre-processing step aims at filtering and segmenting the image in order to detect the regions of interest, which are "meshed" in the mesh generation step. Cebral and Lohner [4] for example, binarize the original image in order to extract well defined contours from which a mesh is generated. Other approaches that also employ a pre-processing strategy are those by by Zhang et al. [19] and Berti [2]. In both algorithms the mesh is built by defining, an implicit function that guides an space partitioning strategy based on quadtree. They also add a post-processing step to improve the quality of mesh elements. A different approach has been presented by Hale [10]. Hale's algorithm makes use of a preprocessing step to reduce noise and highlight sharp features. Following that, a potential energy function is employed to align a lattice of points with the image features. The mesh is finally generated by Delaunay triangulation from the lattice points. The goal of this strategy is that of representing images and thus distinct regions are not individually identified, imparing its use in simulation applications.

The method proposed in this work differs from the techniques described above in two main aspects: the segmentation strategy, commonly employed as a pre-processing step, is built in the mesh generation process and a theoretically guaranted mesh is produced. The fundamental difference between this approach and pre-processed segmentation is that the regions are defined based on geometric properties of the mesh (such as lenghts and areas) together with image attributes, providing additional information to support the decision process involved in segmentation. Furthermore, the algorithm can distinguish different structures contained in the image, producing an independent mesh for each detected structured.

3 Basic Concepts

In this section we present some basic definitions and terminology used in the remaining of the text.

Let S be a set of points in $\mathbb{R}^2$. A *triangulation (mesh)* of S is a two-dimensional simplicial complex T whose vertices are the points of S, and any k-simplex of T, $k = 0, 1$, is contained in at least a 2-simplex (triangle) of T. If the union of all simplices in T makes up the convex hull of S and the circumcircle of each triangle in T does not contain any point of S in its interior then T is called *Delaunay triangulation* [6].

A *good quality mesh* is a triangulation where all triangles satisfy a measure of quality, as for example *circumradius-to-shortest edge* ratio, i.e., the ratio between the radius of the circumcircle and the length of the shortest edge is limited by a constant in all triangles of the mesh.

A *planar graph* is a graph G with vertices in $\mathbb{R}^2$ where each edge is a straight-line segment with ends in G and if e_1 and e_2 are two edges of G, $e_1 \cap e_2$ is either empty or a vertex of G. Given a planar graph G, a *mesh constrained by* G is a triangulation M where each vertex of G is in M and if e is an edge in G then $|e| = |e_1 \cup e_2 \cup \cdots \cup e_k|$, where $e_i \in M$, $i = 1, \ldots, k$ are edges of M and $|\cdot|$ represents the underling space, i.e., each edge of G can appear subdivided as a set of edges in M.

Let S be a set of points and M a mesh (triangulation) of S, if $M = M_1 \cup M_2 \cup \cdots \cup M_k$, where each M_i is a triangulation and $M_i \cap M_j$, $i \neq j$ is either empty or a planar graph then $\{M_1, M_2, \cdots, M_k\}$ is said a *k-partitioning* of M in *submeshes* M_i, $i = 1, \ldots, k$.

An $m \times n$ *image* is a function $I : [0, \ldots, m] \times [0, \ldots, n] \rightarrow \mathbb{R}^+$ that assigns to each point $p \in [0, \ldots, m] \times [0, \ldots, n] \subset \mathbb{Z}^2$ a non-negative scalar $I(p)$. The pair $(p, I(p))$ is called pixel.

4 The *Imesh* Method

In this section, we present a description of the proposed algorithm *Imesh*, which consists in three main steps: initialization, partitioning, and mesh improvement. The first step aims at generating an initial mesh that fits image features. Partitioning is concerned with the segmentation. The final step refines the mesh in order to get a quality mesh. A detailed description of each step is presented in the following text. The color figures presented in the remaining of the paper can be also viewed at: http://www.lcad.icmc.usp.br/~alex/imesh/imesh-sibgrapi-05/

4.1 Initialization

The initialization step starts by creating a Delaunay triangulation M from points evenly distributed (based on distance between them) on the border of the input image. Over each triangle $t_i \in M$ the image I can be approximated by:

$$\hat{I}(p) = \sum_{k=1}^{3} I(p_k) b_{i,k}(p), \qquad \text{for every point } p \in |t_i| \quad (1)$$

where $b_{i,k}(p)$ is the interpolation basis function associated with the k^{th} vertex p_k, $k = 1,2,3$ of t_i.

If $\max\{|I(p) - \hat{I}(p)|\}$, for every point p in $|t_i|$, is higher than a user defined scalar denoted by *min_error* then the image I is said not well approximated in t_i. In order to improve the approximation, a new point p_n in $|t_i|$ is inserted in the Delaunay triangulation.

The strategy adopted to define the new point p_n is one of the inovative aspects of our approach. Different from other adaptive approaches that choose the new point by maximizing either $\{|I(p) - \hat{I}(p)|\}$ or an edge detection operator applied over all points in $|t_i|$, our scheme selects p_n by scanning only a few lines in the interior of the triangle, thus avoiding to traverse all pixels of $|t_i|$.

Let h_1, h_2, h_3, and c be the three altitudes and the circumcenter of t_i, respectively (see Figure 1). Consider the set of points

$$P_{t_i} = \{p \in h_j,\ j = 1,2,3 \mid \mathscr{E}(p) \geq min\_border\}$$

where $\mathscr{E}$ is an edge detection operator and *min_border* is an user defined scalar. Therefore, P_{t_i} is the set of pixels on the lines h_j where the operator $\mathscr{E}$ exceeds a thresholding. The new point p_n is choosen from the points in P_{t_i} so as to minimize $\|c - p\|$, $p \in P_{t_i}$, i.e., p_n is the point of P_{t_i} closer to the circumcenter of t_i. In Figure 1 the darker square illustrates the point p_n choosen by our scheme.

The adaptive scheme finishes when either the image is well approximated in all triangles of M or $\mathscr{E}(p)$ is lower than *min_border* in all triangles that must be refined. Both

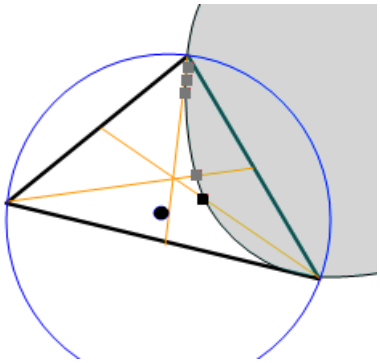


Figure 1. Choosing a new vertex to be inserted in the triangulation.

parameters *min_error* and *min_border* affect the triangle density. Figure 2 illustrates this behavior. In Figure 2a) and b) the parameter *min_error* is set with a small value and with a high value respectively and *min_border* is fixed in such a way that the refinement finishes due to *min_error*. Note that in Figure 2a), due to noise, the background is also refined. In figure 2c) and d) the parameter *min_error* is fixed close to zero and *min_border* receives small and high values respectively, so the refinement is stopped by *min_border*.

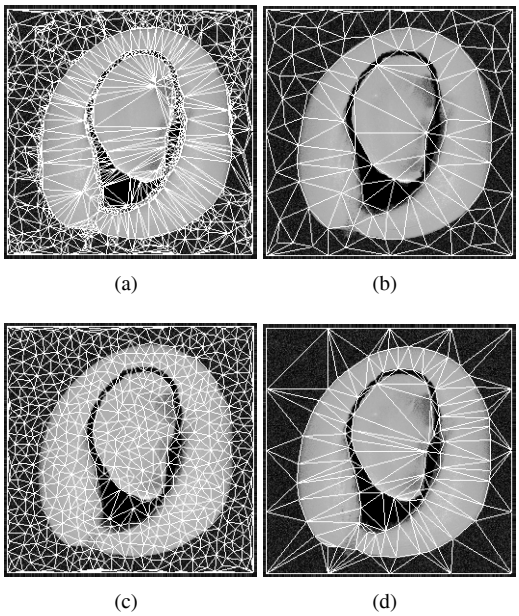


Figure 2. Handling triangle densities

From Figure 2 we can see that small values for *min_error* and *min_border* tend to generate denser meshes while courser meshes are produced with high values.

Our adaptive approach presents some desirable properties not found in other traditional schemes: by inserting new

points closer to circumcenters we avoid the acumulation of points around already exisisting vetices. Furthermore, as we will see later (subsection 4.3), such strategy tends to generate triangles with a better quality, thus improving the interpolated image $\hat{I}$ (see Yang [18] for details in how the quality of triangles affects the interpolated image).

4.2 Partitioning

The partitioning step aims at generating a k-partitioning of the mesh produced in the initialization step, where k is previously specified. Such partitioning is carried out by a region growing approach, which is accomplished as described as follows.

Let $\mathscr{H}(T)$ be a mapping that associates an array of characteristics to a triangulation. $\mathscr{H}(T)$ may be an array of texture features, histogram of the pixels in $|T|$, or just the average of I in $|T|$.

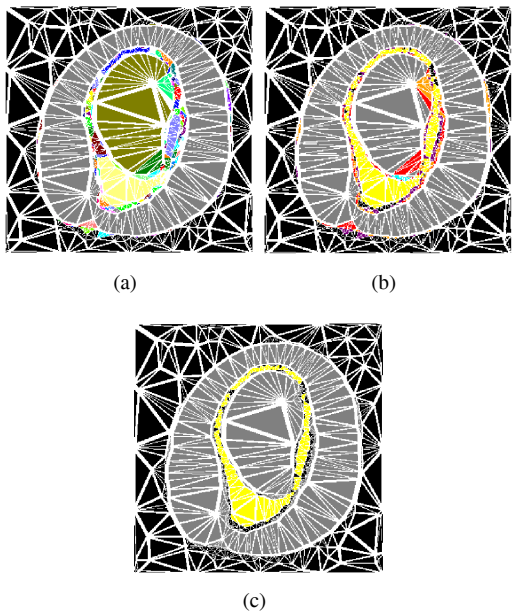
The partitioning is started by arbitrarily selecting a triangle $t_i \in M$, computing $\mathscr{H}(t_i)$ and initializing a first submesh M_1 with t_i. For each triangle t_j sharing an edge with M_1 we compute $|\mathscr{H}(M_1) - \mathscr{H}(t_j)|_{\mathscr{H}}$, where $|\cdot|_{\mathscr{H}}$ is a distance measure defined from $\mathscr{H}$. If $|\mathscr{H}(M_1) - \mathscr{H}(t_j)|_{\mathscr{H}}$ is smaller than a user defined parameter *min_diff* then t_j is added to M_1. Therefore, M_1 "grows" until there is no triangle to be added. A new submesh M_2 is then started with a triangle not in M_1 and the process is repeated. In the end, when all triangles in M are in some submesh M_i, we have a partitioning $M = M_1 \cup M_2 \cup \cdots \cup M_m$. In order to keep submeshes with the same characterist $\mathscr{H}$ grouped, we compute $|\mathscr{H}(M_i) - \mathscr{H}(M_j)|_{\mathscr{H}}, i, j = 1, \ldots, m; i \neq j$, merging the submeshes whose distances $|\cdot|_{\mathscr{H}}$ are smaller than *min_diff*, resulting in a partition $M = M_1 \cup M_2 \cup \cdots \cup M_s, s \leq m$.

If $s > k$ we must merge some submeshes in order to obtain a k-partitioning. This final merge is carried out taking into account the area of the submeshes and their number of triangles. Adding the submeshes to a priority queue sorted by those two criteria simultaneously, the smallest submesh is merged with the "closest" submesh regarding the distance $|\cdot|_{\mathscr{H}}$. The process carries on until the number of submeshes is k. Using the priority queue avoids sorting the submeshes every time a merge is executed. The algorithms admits other merging criteria but this has been the most effective in the tests performed.

Figure 3 illustrates the three stages of the partitioning step. In this example $\mathscr{H}$ is the average color of the submesh and $|\cdot|_{\mathscr{H}}$ is the euclidean distance.

4.3 Mesh Improvement

This final step of the algorithm aims at refining the submeshes in order to produce a triangulation whose triangles respect a minimum angle criterion. A variant of Ruppert's



(a) (b)

(c)

Figure 3. The three stages of the partitioning step: (a) building the initial submeshes (258 submeshes); (b) Merging similar submeshes (8 submeshes); (c) obtaining a k-partitioning (k=3).

algorithm [13] is employed to achieve a good quality triangulation.

Ruppert's algorithm refines a Delaunay mesh by inserting the circumcenters of "poor" quality triangles. The quality of the triangles is measured by the circumradius-to-shortest edge ratio, i.e., the radius of the circumcircle divided by the length of the shortest edge of the triangle. It can be proven that the circumradius-to-shortest edge ratio r/d of a triangle is related to its smallest angle α by $\sin\alpha = d/(2r)$. As the insertion of the circumcenters tends to generate triangles with smaller circumradius, the smallest angle of the new triangles tends to be bigger than the old ones, thus improving the quality of the triangulation. Ruppert's algorithm inserts circumcenters until all triangles satisfy a quality constraint, i.e., all triangles have the ratio r/d limited by a constant.

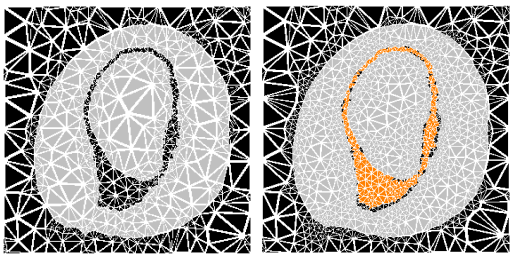
The strategy to insert new vertices in Ruppert's algorithm is governed by two main rules thus described: let G be a planar graph and M be a mesh whose vertices of G are vertices of M. The first rule of Ruppert's algorithm verifies, for each edge e in G, if a vertex of M lies strictly inside of the diametral circle (the smallest circle enclosing the edge) of e. In the affirmative case, the edge e is split in two segments by inserting a vertex at its midpoint. The process follows until the diametral circles of the edges (or subdivided segments)

in G are empty. The second rule aims at inserting a vertex at the circumcenter of each triangle whose circumradius-to-shortest edge ratio is greater than a bound B. However, if the new vertex lies inside of the diametral circle of some segment (or subsegment) of G, then such a vertex is not inserted and the segment is split as in the first rule.

It can be shown that if $B \geq \sqrt{2}$ then Ruppert's algorithm terminates; furthermore, if a circumcenter v is inserted as a new vertex then v lies inside the planar graph [14] bounding the mesh.

We generalize Ruppert's algorithm, originally designed for a single planar graph, to work on a k-partitioning. The main problem in such a generalization in the fact that new vertices inserted on the edges of the planar graphs bounding submesh may eliminate other subsegments, damaging the partitioning. We overcome this problem by "locking" the segments of the planar graphs. Although the triangulation may become non-Delaunay momentarily, it is not difficult to show that the Delaunay property will be recovered as locked segments have also diametral circles not empty and thus will be subdivided. As circumcenters are guaranteed to lie inside the planar graphs, their insertion do not affect the partitioning.

We can apply the mesh improvement strategy in all submeshes or in a subset of submeshes. Figure 4 shows an example of the refinement in a mesh with 2 and 3-partitioning, respectivelly. In this figure the mesh improvement has not been applied in the submesh that comprises the background of the image. The B parameter has been set to ensure a minimal angle equal to 22^o.



(a) (b)

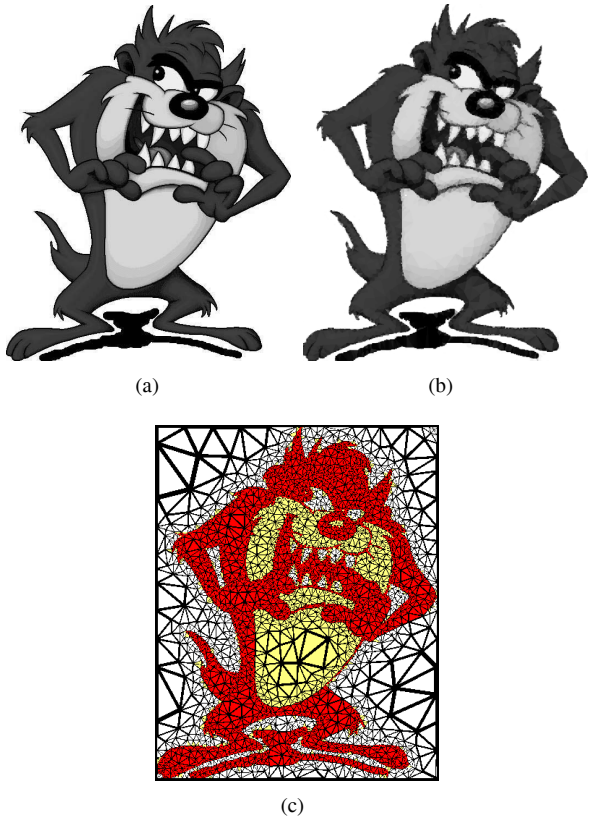
Figure 4. Sample refinements. (a) 2-partitioning. (b) 3-partitioning

Ruppert's algorithm presents problems when que planar graph bounding the initial mesh has small angles. Ruppert proposes a simple solution for the problem [13], which subdivides the edges comprising small angles taking into account the local feature size of the vertices. Such an strategy has been implemented in our code. Although simple, the strategy adopted by Ruppert to deal with small angles is not computationally efficient, increasing computational times.

5 Results

In this section we present examples of meshes produced by the proposed algorithm. It is worth mentioning that the images have not been pre-processed, i.e., they are input directly into the algorithm. Average color and euclidean distance have been used as measures of similatiry among submeshes.

The two first examples (Figures 5 and 6) show the results of applying our algorithm in synthetic images. Notice in Figure 5 that thin details, as the mustache and fur, have been captured by the algorithm. Figure 6a) presents an image with light effect from left to right, which, in general, hinders segmentation processes. As it can be seen in Figure 6b), the algorithm successfully detects the structures, partitioning the image in two submeshes.

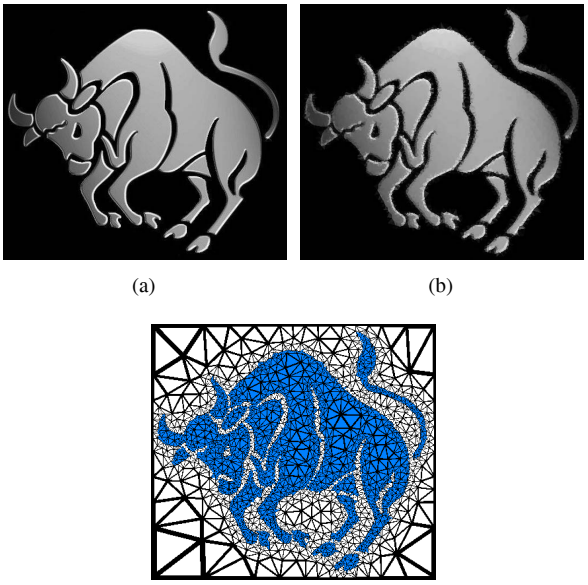


(a) (b)

(c)

Figure 5. Tasmanian devil (TAS): (a) Original image; (b) Interpolated image from the mesh; (c) mesh wiht 4044 vertices. Parameters: *min_border* = 150, *min_error* = 3, and $k = 3$.

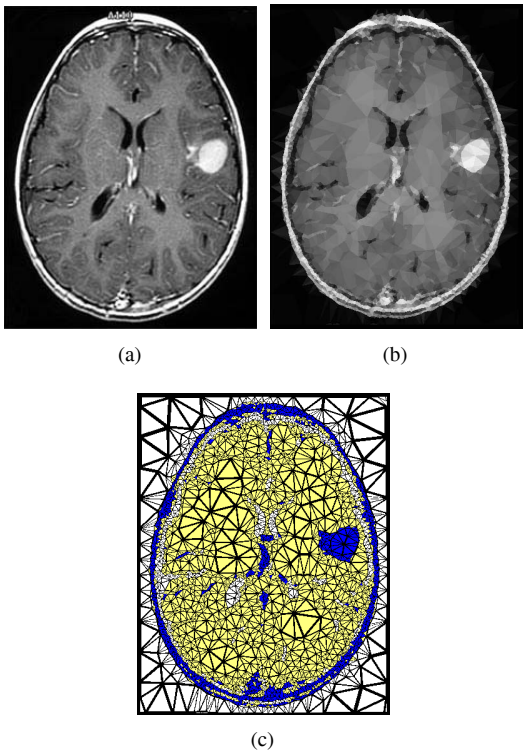
Figure 7 shows the behavior of the algorithm in a medical image. In this example we set the algorithm to generate a 3-partitioning, aiming at detecting the tumor on the right side of the brain (figure 7a)). Notice in Figure 7b) that, even



(a) (b)

(c)

Figure 6. Bull: (a) Original image; (b) Interpolated image from the mesh; (c) mesh with 4674 vertices. Parameters: *min_border* **= 50,** *min_error* **= 9, and** $k = 2$**.**



(a) (b)

(c)

Figure 7. Brain slice: (a) Original image; (b) Interpolated image from the mesh; (c) mesh wiht 2302 vertices. Parameters: *min_border* **= 140,** *min_error* **= 15, and** $k = 3$**.**

with a small number of vertices (2302), the algorithm has detected the tumor while interpolating the image properly.

The computational times regarding the meshes presented in figures 5,6, and 7 are shown in table 1. The code, implemented in C++ and OpenGL, has been executed in a Xeon 2.8Ghz processor with 1Gb RAM. First column shows the size of the original images. The other columns present times, in seconds, of each step of the algorithm. It is worth mentioning that we have not implemented any optimization strategy for the mesh improvement step, justifying the still high computational time of this step. As pointed out by Shewchuk [15], optimization strategies, which include a more efficient treatment for small angles, can improve considerably the performance of the algorithm.

Images/steps	Initialization	Partitioning	Mesh Improvement
TAS 523×655	8.047	0.56	129.90
Bull 528×475	5.765	0.29	60.75
Brain 235×300	5.5	0.34	18.89

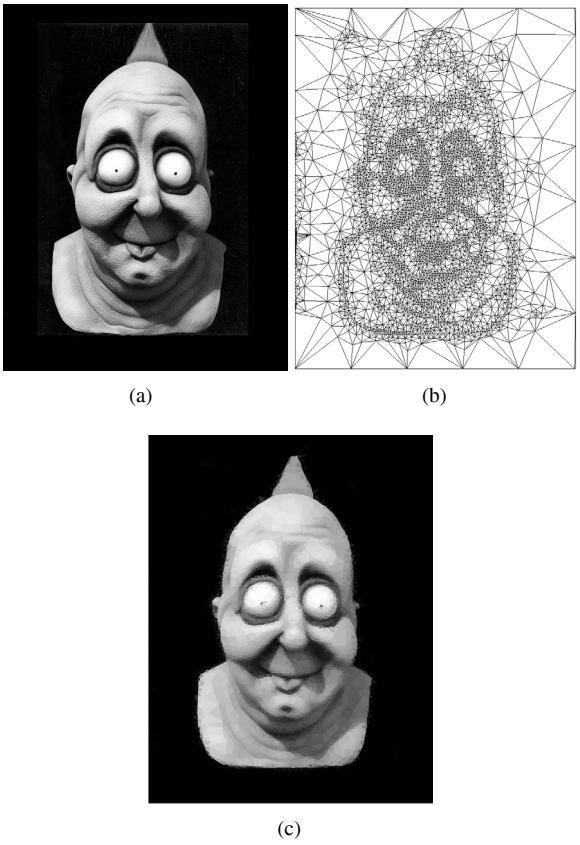
Table 1. Computational times (s)

We finish this section with an example of numerical simulation using a mesh generated by our algorithm. In fact, the example shows an elastic model acting on the mesh, producing thus a deformation on the image represented by the mesh.

Figure 8a) shows the original image (called Boingo), which we shall call Boingo. Figures 8b) and 8c) present the Boingo's interpolated image and mesh, respectively.

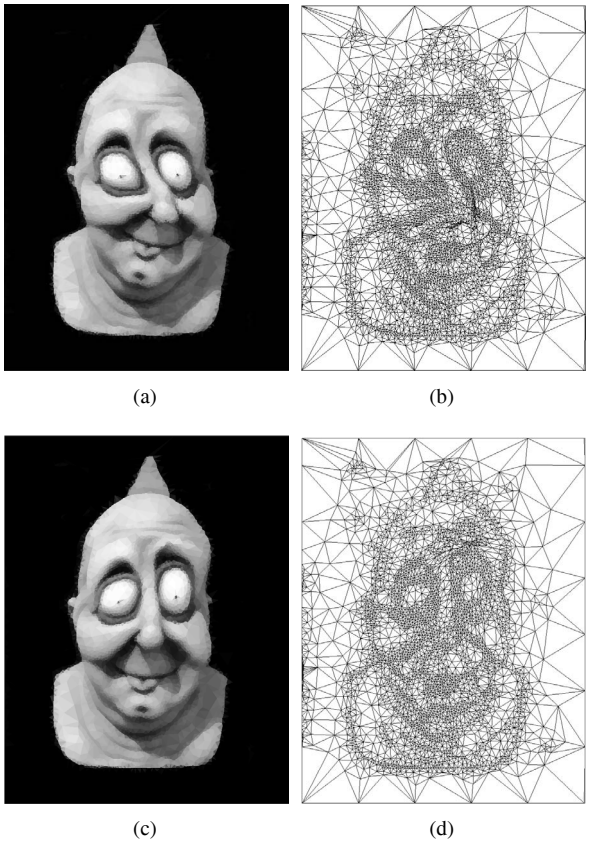
Figure 9 shows two elastic deformation of the mesh presented in figure 8b). Figures 9a) and 9c) are interpolated images (the colors stored in the vertices are used in the interpolation) and figures 9b) and 9d) are the deformed meshes.

The elastic model employed in figure 9 is based on a mass-spring sistem with constant mass in each vertex and stiffness (spring on the edges) proportional to $1/l$, where l is the length of the edge. This example illustrates the importance of a good quality meshe, once that large deformations, as the ones presented in figure 9, would certainly produce invalid elements in a mesh containing bad shaped triangles. The reason for invalid elements is that bad quality triangles have a vertex close to an opposite edge, thus, even small perturbations can make this vertex cross the edge, inverting the triangle orientation, damaging the quality of the interpolated image.



(a) (b) (c)

Figure 8. Boingo: (a) original image; (b) mesh; (c) interpolated image.



(a) (b) (c) (d)

Figure 9. Boingo's elastic deformation.

6 Conclusions and Future Work

In this work we have presented an image based mesh generation algorithm that automatically distinguishes different structures contained in an image while ensuring good quality of the resulting triangular elements. The approach does not require pre-segmentation of the image. Rather, the segmentation is built in the mesh generation step and is supported by geometric features along with image properties. The algorithm unifies concepts from mesh generation and image processing in an integrated framework for automatic meshing from images.

The strategy adopted in the first step of the algorithm, which inserts points close to circumcenters, turns out to be very efficient, as it avoids accumulating points around already existing vertices and tends to improve the quality of the triangles. The partitioning strategy, which generates submeshes based on similarity as well as in area, has also produced good results in the tests we have run. The adaptation of Ruppert's algorithm to work in a partitioned mesh has given rise to a refinement strategy that improves the quality of all submeshes simultaneously.

The mesh improvement step of the algorithm is currently undergoing optimization and should improve performance considerably.

We have been working in an strategy to estimate the parameters of the algorithm automatically, taking into account the characterists of the image. The optimization strategies proposed by Shewchuk [15] are also being incorporated in the mesh improvement step, including the treatment of small angles present in the planar graph. We are also extending the two-dimensional algorithm presented here to the three-dimensional case.

Acknowledgements

We acknowledge the financial support of FAPESP - the State of São Paulo Research Funding Agency (Grant# 03/02815-0 and #02/05243-4), and CNPq, the Brazilian National Research Council (Grants #521931/97-5 and # 300531/99-0). The brain image is courtesy of Saind Mary's Hospital - London. The Taz image was obtained in http://jupiter.ucsd.edu/ rstevens/villano/Taz.gif and the Taurus image in http://www.free-horoscope-

reading.com/taurus.jpg The Boingo mask was downloaded from http://www.halloween-mask.com/boingo_lubati.htm.

References

[1] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In H. F.K. and D. D.Z., editors, *Computing in Euclidean Geometry*, pages 23–90. World Scientific, 1992.

[2] G. Berti. Image-based unstructured 3d mesh generation for medical applications. In *ECCOMAS - European Congress On Computational Methods in Applied Sciences and Engeneering*, 2004.

[3] S. Canic, A. Mikelic, and J. Tambaca. A two-dimensional effective model describing fluid-structure interaction in blood flow: analys is, simulation and experimental validation. In *Special Issue of Comptes Rendus Mechanique Acad. Sci. Paris*. 2004.

[4] J. Cebral and R. Lohner. From medical images to cfd meshes. In *Proceedings of the 8th International Meshing Roundtable*, pages 321–331, 1999.

[5] A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno. Multiresolution decimation based on global error. *The Visual Computer*, 13(5):228–246, 1997.

[6] S. Fortune. Voronoi diagrams and delaunay triangulation. In H. F.K. and D. D.Z., editors, *Computing in Euclidean Geometry*, volume 1 of *Lecture Notes Series on Computing*, pages 193–233. World Scientific, Singapore, 1992.

[7] M. García, A. Sappa, and B. Vintimilla. Efficient approximation of gray-scale images through bounded error triangular meshes. In *IEEE Intern. Conf. on Image Processing*, pages 168–170, 1999.

[8] M. Garland and P. Heckbert. Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, Carnegie Mellon University, 1995.

[9] T. Gevers and A. Smeulders. Combining region splitting and edge detection through guided delaunay image subdivision. In *IEEE Proceedings of CVPR*, pages 1021–1026, 1997.

[10] D. Hale. Atomic images - a method for meshing digital images. In *10th International Meshing Roundtable*, pages 185–196, 2001.

[11] C. Huang and C. Hsu. A new motion compensation method for image sequence coding using hierarchical grid interpolation. *IEEE Trans. Circuits Syst. Video Technol.*, 4:44–51, 1994.

[12] H. Pedrini. An improved refinement and decimation method for adaptive terrain surface approximation. In *WSCG'2001*, pages 5–9, 2001.

[13] J. Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1995.

[14] J. Shewchuk. *Lecture Notes on Delaunay Mesh Generation*. Department of Electrical Engineering and Computer Science - Berkeley, CA 94720, 2000.

[15] J. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(2-3):21–74, 2002.

[16] F. Sousa, N. Mangiavacchi, L. Nonato, A. Castelo, M. Tome, V. Ferreira, J. Cuminato, and S. McKee. A front-tracking/front-capturing method for the simulation of 3d multi-fluid flow with free surface. *Journal of Computational Physics*, 198:468–499, 2004.

[17] D. Terzopoulos and M. Vasilescu. Sampling and reconstruction with adaptive meshes. In *IEEE Int. Conf. Comp. Vision, Pattern Recog.*, pages 829–831, 1992.

[18] Y. Yang, M. Wernick, and J. Brankov. A fast approach for accurate content-adaptive mesh generation. *IEEE Trans. on Image Processing*, 12(8):866–881, 2003.

[19] Y. Zhang, C. Bajaj, and B.-S. Sohn. Adaptive and quality 3d meshing from imaging data. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 286–291, 2003.