Template-based Remeshing for Image Decomposition

Mario A. S. Lizier
ICMC-USP
São Carlos, Brazil
lizier@gmail.com

Marcelo F. Siqueira
DIMAp-UFRN
Natal, Brazil
mfsiqueira@dimap.ufrn.br

Joel Daniels II
SCI-Utah
Salt Lake City, USA
jdaniels@cs.utah.edu

Claudio T. Silva
SCI-Utah
Salt Lake City, USA
csilva@sci.utah.edu

Luis G. Nonato
ICMC-USP
São Carlos, Brazil
gnonato@icmc.usp.br

***Abstract*—This paper presents a novel quad-based remeshing scheme, which can be regarded as a replacement for the triangle-based improvement step of an existing image-based mesh generation technique called *Imesh*. This remeshing scheme makes it possible for the algorithm to generate good quality *quadrilateral* meshes directly from imaging data. The extended algorithm combines two ingredients: (1) a template-based triangulation-to-quadrangulation conversion strategy and (2) an optimization-based smoothing procedure. Examples of meshes generated by the extended algorithm, and an evaluation of the quality of those meshes are presented as well.**

***Keywords*-Image-based mesh generation; template-based mesh; remeshing; mesh smoothing; mesh segmentation; Bézier patches;**

I. Introduction

Generating meshes from 2D digital images is an important problem, which has been investigated in several different contexts such as image representation and numerical simulation. In numerical simulation, most methods rely on a two-stage scheme: image segmentation and mesh generation itself. The image segmentation stage is responsible for partitioning the image in well defined regions, which are then "meshed" in the mesh generation stage.

An alternative approach was adopted by the *Imesh* algorithm [1], [2] (Figure 1). The *Imesh* approach combines image segmentation and mesh generation into a single processing stage, requiring only a couple of parameters to trigger the meshing process directly from the input image. Moreover, *Imesh* is able to segment the mesh in accordance with image features, making it possible to identify and build a correspondence between regions of the image and partitions of the mesh. The *Imesh* output is a provably good quality triangulation which contains smaller triangles along the boundary of image regions, and larger triangles in their interior.

Triangle meshes have been extensively investigated by the meshing community, and their theoretical properties are now well understood [3]. In addition, algorithms for generating good triangle meshes of polygonal and curved planar domains, such as the one used by *Imesh*, have been proposed and implemented [4], [5]. In contrast, the generation of good quality quadrilateral meshes is not so well understood [6]. Directly generating quadrilateral meshes from a description of the domain is intrinsically harder than generating triangle meshes. Yet, quadrilateral meshes are more appropriate than triangle meshes for some numerical methods and simulations [7].

Contributions. We describe an extension of *Imesh* which generates quadrilateral meshes directly from imaging data. Our extension combines two ingredients: a template-based triangulation-to-quadrangulation conversion strategy, and an optimization-based smoothing procedure. The former aims at generating quadrilateral meshes that respect the image object boundaries (as defined by the mesh partition step of *Imesh*). The latter improves the quadrilateral shape quality.

The use of a template-based meshing approach makes it possible for *Imesh* to generate a quadrilateral mesh *indirectly*, i.e., from a triangle mesh rather than directly from a description of a polygonal domain. This indirect strategy makes the quadrilateral mesh generation task easier. Templates also enabled us to devise a novel and simple smoothing procedure to locally improve the quality of the quadrilaterals, while preserving the previously defined image object boundaries. Our experimental results indicate that the combination of our template-based mesh generation approach with the new smoothing procedure is very effective, rendering *Imesh* one of the few techniques able to generate quadrilateral mesh *straightly* from images.

II. Related Work

In this section we summarize the main techniques devoted to generate meshes from images as well as triangle-to-quadrilateral meshes techniques in 2D. A comprehensive overview, mainly in the context of surfaces, is beyond the scope of this paper and can be found in [8].

Mesh generation from images. Techniques for generating meshes from digital images can be grouped into two main classes: mesh-based image representation and image modeling for simulation. *Mesh-based image representation techniques* build meshes that minimize the approximation error between the original image and the image represented by the mesh. In this class one finds adaptive methods, which iteratively refine the mesh until a lower bound error is reached [9]–[11], mixed methods [12], and error diffusion schemes [13]. A main drawback of most mesh-based image

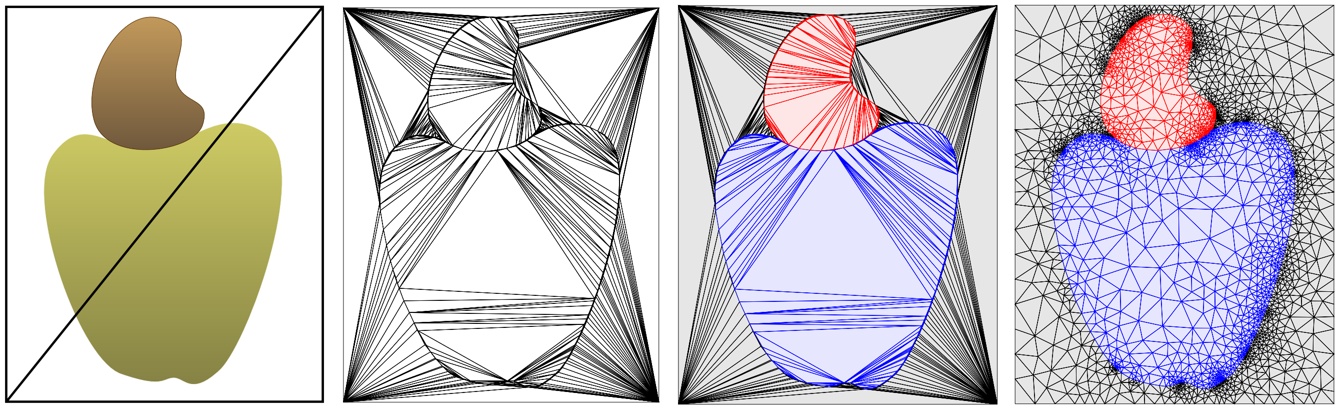

(a) Original Image (b) Mesh Generation (c) Mesh Partition (d) Mesh Improvement

Figure 1. The three main steps of *Imesh*.

representation methods is the use of interpolation error to guide the mesh generation process, which is not effective in textured and color images, impairing the use of such methods in a wide class of problems. *Image modeling for simulation techniques* divide the mesh generation process in two main steps: pre-processing and mesh generation. The pre-processing step aims at filtering and segmenting the image in order to detect regions of interest, which are meshed in the mesh generation step [14]. Binarization combined with implicit function reconstruction [15], [16], pre-segmentation with Delaunay meshing [17], and shape deformation [18] figure among the most popular approaches, all of them relying on a segmentation stage.

Quadrilateral mesh generation. Generating a quadrilateral mesh of a polygonal domain is intrinsically harder than producing a triangular mesh. Indeed, if we require the set of vertices of the mesh to be the set of vertices of the input polygon, then a triangular mesh can always be obtained. In contrast, additional vertices may be necessary in order to generate a quadrilateral mesh. In addition, the theoretical properties to generate good quality quadrilateral meshes are not as well understood as the ones for producing good quality triangular meshes [6]. So, several researchers adopted an *indirect approach* to produce quadrilateral meshes [19]–[21]: a triangle mesh of the domain is generated, and later converted into a quadrilateral mesh. This approach relies on the premise that a quadrilateral mesh can be more easily generated from an existing triangle mesh. Here, we adopt a two-stage indirect approach. First, we combine adjacent triangles to form quadrilaterals and produce a hybrid, triangle-quadrilateral mesh. Second, we convert the hybrid mesh into an all-quadrilateral mesh using *template subdivisions* of triangles and quadrilaterals. Templates subdivisions of mesh elements have been used before for refining quadrilateral meshes [22] and respecting domain boundaries [6]. We use template subdivisions for mesh conversion and optimization purposes. To our best knowledge, this is the first work that exploits the potential of template subdivisions for both purposes.

III. Preliminaries

This section introduces basic concepts from Computational Geometry and Digital Topology, which are used in the description of the *Imesh* algorithm in the following section. We refer the reader to [4], [21], [23] for a detailed discussion of those concepts.

Let S be a finite set of points in $\mathbb{R}^2$. A *triangulation*, $\mathcal{T}(S)$, of S is a set of triangles, along with their edges and vertices, such that (1) the set of vertices of $\mathcal{T}(S)$ is exactly S, and (2) the intersection of any two triangles, σ and τ, of $\mathcal{T}(S)$ is either empty or a common vertex or edge of σ and τ. The *underlying space*, $|\mathcal{T}(S)|$, of $\mathcal{T}(S)$ is the point set consisting of all points of $\mathbb{R}^2$ that belong to the triangles of $\mathcal{T}(S)$. Similarly, we define a *quadrangulation*, $\mathcal{Q}(S)$, of S as a set of quadrilaterals, along with their edges and vertices, such that (1) the set of vertices of $\mathcal{Q}(S)$ is exactly S, and (2) the intersection of any two quadrilaterals, μ and ν, of $\mathcal{Q}(S)$ is either empty or a common vertex or edge of μ and ν. The underlying space, $|\mathcal{Q}(S)|$, of $\mathcal{Q}(S)$ is the point set consisting of all points of $\mathbb{R}^2$ that belong to the quadrilaterals of $\mathcal{Q}(S)$. Hereafter, we will use *quad* as an abbreviation for quadrilateral.

A *Delaunay triangulation*, $\mathcal{DT}(S)$, of S is a triangulation of S such that (1) the underlying space, $|\mathcal{DT}(S)|$, of $\mathcal{DT}(S)$ is the convex hull of S (i.e., the smallest convex set that contains S), and (2) the interior of the circumcircle of every triangle of $\mathcal{DT}(S)$ does not contain any vertex of $\mathcal{DT}(S)$. Given a planar straight-line graph (PSLG), $G = (V, E)$, where V is a set of points of $\mathbb{R}^2$ and E is a set of line segments in $\mathbb{R}^2$ with endpoints in V, we define a *conforming Delaunay triangulation* of G as *any* Delaunay triangulation, $\mathcal{DT}(S)$, for some $S \subset \mathbb{R}^2$, such that (1) $V \subseteq S$ and (2) each edge e of E is an edge of $\mathcal{DT}(S)$ or a union of edges of $\mathcal{DT}(S)$. We say that $\mathcal{DT}(S)$ *conforms* to the vertices in V and edges in E (edges in E are called *constrained edges*).

As customary in Digital Topology, we call each $p \in \mathbb{Z}^2$ a *grid point*, and we regard p as the center of a grid square, denoted by $\square(p)$, with edges of unit length and oriented parallel to the Cartesian coordinate axes. We commonly refer to $\square(p)$ as a *pixel*. A *2D digital (multivalued) image* is a function $\mathcal{I} : \mathbb{G}_{n_1,n_2} \to C$ from a nonempty and finite subset of $\mathbb{Z}^2$, $\mathbb{G}_{n_1,n_2} = \{(g_1, g_2) \in \mathbb{Z}^2 \mid g_i \in [1, n_i], i = 1, 2\}$, to a nonempty and finite subset, C, of $\mathbb{R}$, where n_1 and n_2 are positive integers. The domain $\mathbb{G}_{n_1,n_2}$ of $\mathcal{I}$ is called a *2D grid* of size $n_1 \times n_2$. The elements of the co-domain C of $\mathcal{I}$ are called *colors*. So, the image $\mathcal{I}$ is a function that assigns a color $\mathcal{I}(p)$ from C to each $p \in \mathbb{G}_{n_1,n_2}$. The union set, $\bigcup_{p\in\mathbb{G}_{n_1,n_2}} \square(p)$, of all pixels whose centers are in $\mathbb{G}_{n_1,n_2}$ is the *continuous analog* of $\mathbb{G}_{n_1,n_2}$, which is denoted by $\square(\mathbb{G}_{n_1,n_2})$. By definition, we have that $\square(\mathbb{G}_{n_1,n_2})$ is a rectangle.

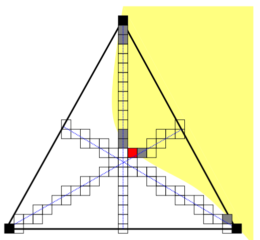
Given a 2D digital image, $\mathcal{I} : \mathbb{G}_{n_1,n_2} \to C$, we define a *triangle mesh* of $\mathcal{I}$ as *any* triangulation, $\mathcal{T}(S)$, for some finite subset S of $\mathbb{R}^2$, such that $|\mathcal{T}(S)| = \square(\mathbb{G}_{n_1,n_2})$. We can define a *quad mesh* of $\mathcal{I}$ in a similar way. In the following section we describe the *Imesh* algorithm for computing a triangulation $\mathcal{T}(S)$ from a given image.

IV. Imesh Overview

This section presents an overview of *Imesh*, an algorithm for directly generating triangle meshes from 2D digital images (see [2] and [1] for a more complete and detailed description).

Imesh is comprised of three main steps: mesh generation, mesh partition, and mesh improvement, as illustrated in Figure 1. It takes in a 2D digital image, $\mathcal{I} : \mathbb{G}_{n_1,n_2} \to C$, a *threshold* t_e, with $t_e \in [0, 1] \subset \mathbb{R}$, and a *classifier*, which is defined as a function, $g : \mathbb{G}_{n_1,n_2} \to L$, where L is a set of "labels". For any $p \in \mathbb{G}_{n_1,n_2}$, the value of g at p is in general determined by $\mathcal{I}(p)$ and the values of $\mathcal{I}$ in the neighborhood of p.

Mesh Generation: In the mesh generation step *Imesh* builds a Delaunay triangulation, $\mathcal{DT}(S)$, by successively adding vertices so as to match features of the input image $\mathcal{I}$. More precisely, suppose that a Delaunay triangulation $\mathcal{DT}(S_{i-1})$ has already been generated from a previously computed point set $S_{i-1} \subseteq \square(\mathbb{G}_{n_1,n_2})$ (S_0 contains the four corners of $\square(\mathbb{G}_{n_1,n_2})$, as shown in Figure 1(a)). For each triangle σ in $\mathcal{DT}(S_{i-1})$, let $e(\sigma)$ be an error measure defined in terms of the classifier g and all image pixels intersected by the three medians, m_1^σ, m_2^σ, and m_3^σ, of σ. In mathematical terms, the error $e(\sigma)$ is defined as follows: denote by $P_{m_j^\sigma}$ the set of pixels where the classifier g changes its value when one traverses m_j^σ (gray squares in the illustration on the right) and



let $\alpha_\sigma(p)$ be the smallest barycentric coordinate of point p with respect to σ. The error $e(\sigma)$ is given by:

$$e(\sigma) = \max\left\{ \alpha_\sigma(p) \in \mathbb{R}_+ \middle| \square(p) \in \bigcup_{j=1}^{3} P_{m_j^\sigma} \right\},$$

Note that the value of $e(\sigma)$ is related to how far the pixels in $P_{m_j^\sigma}$ are from the edges of σ, the farther they are the larger the value of $e(\sigma)$. So, a large value of $e(\sigma)$ means that σ is not well fitted in a single region of the image, and thus it should be subdivided. In fact, if $e(\sigma) > t_e$ then the point p such that $e(\sigma) = \alpha_\sigma(p)$ (red square in the illustration) is added to S_{i-1}, producing $S_i = S_{i-1} \cup \{p\}$, and the Delaunay triangulation $\mathcal{DT}(S_i)$ is generated. The above process is repeated until the error $e(\sigma) < t_e$, for all σ in $\mathcal{DT}(S_i)$. Since $\mathbb{G}_{n_1,n_2}$ is a finite set and the same point from $\mathbb{G}_{n_1,n_2}$ is never considered for insertion twice, the termination of the refinement stage is assured. Figure 1(b) shows a typical mesh generated by the mesh generation step of *Imesh*. From now on, we omit the set S, and denote $\mathcal{DT}(S)$ by simply $\mathcal{DT}$.

Mesh Partitioning The mesh partitioning step generates a partition, $\mathcal{P}$, of the set of triangles, $\mathcal{DT}_t$, of the Delaunay triangulation, $\mathcal{DT}$, produced by the mesh generation step. To build $\mathcal{P}$, the *Imesh* algorithm makes use of a function, $h : \mathcal{DT}_t \to L$, which assigns a label from L to each triangle $\sigma \in \mathcal{DT}_t$. Two triangles, σ and τ, of $\mathcal{DT}_t$ belong to the same element set of $\mathcal{P}$ iff $h(\sigma) = h(\tau)$. The label $h(\sigma)$ is defined as the most frequent one among the labels (given by the classifier g) of the pixels in the medians m_j^σ of σ. Figure 1(c) shows the partitioning obtained from the mesh in Figure 1(b).

Mesh Improvement The mesh generated by the first two steps of *Imesh* is in general a mesh with poor quality triangles, i.e., triangles with very small and/or very large angles. This kind of triangle is extremely unsuitable for many mesh-based applications [24]. To overcome the poor quality mesh problem, *Imesh* further refines the Delaunay triangulation, $\mathcal{DT}$, using an adaptation of Ruppert's algorithm [4]. In other words, the improvement algorithm inserts the circumcenter of every poor quality triangle (i.e., angle smaller than 26.4^o) into $\mathcal{DT}$. However, if the circumcenter of a poor quality triangle σ lies in a triangle τ such that $h(\sigma) \neq h(\tau)$ (or outside the image domain), then such a circumcenter is called *invalid* and is not inserted into $\mathcal{DT}$. In this case, the invalid circumcenter will be inside the diametral circle of one or more edges of $\mathcal{DT}$, which are in the boundary of two distinctly labeled partitions (or in the boundary of the image domain). The midpoint of such edges are computed and inserted in the Delaunay triangulation. Termination and quality guarantee can be ensured under certain conditions, which are discussed in [2]. Figure 1(d) shows the result of executing the mesh improvement step on the mesh in Figure 1(c).

V. Remeshing

We now describe the new remeshing mechanism for *Imesh*, which enables us to generate good quality quad meshes. This extension can be regarded as a replacement

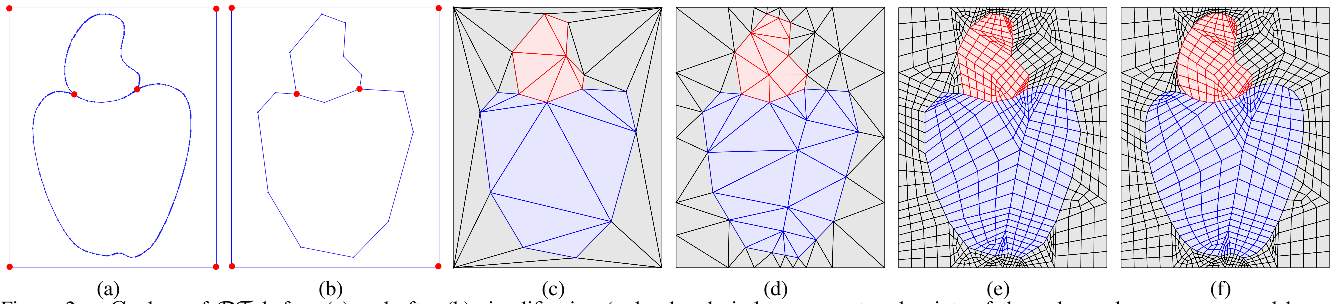


Figure 2. C-edges of $\mathcal{DT}$ before (a) and after (b) simplification (red colored circles represent end points of the polygonal curves generated by our preprocessing). (c) A Delaunay triangulation that conforms to the PSLG in (b). (d) A better quality Delaunay triangulation that also conforms to the PSLG in (b). (e) A template-based quad mesh that conforms to the PSLG in (b). (f) A template-based quad mesh that approximates the edges of the PSLG in (a).

for the mesh improvement step, and it is carried out in two stages, namely: template mapping and optimization. The template mapping stage converts the triangle mesh into a quad one and it consists of three main steps: boundary and mesh simplification, triangle pairing, and the template-based subdivision. The optimization stage aims at improving the quality of quad elements and it comprises two main steps: boundary adaptation and relaxation. In what follows we detail those steps.

A. *Boundary and Mesh Simplification*

Consider the PSLG $G = (V, E)$ given as input for the remeshing step, that is, edges and vertices in G are shared by triangles from different regions of the partitioned mesh (or they are incident to image boundary edges). Each edge in E is a constrained edge, which we call *c-edge*. The goal of the boundary simplification step is to simplify the polygonal curves defined by the set of all c-edges and their vertices. To do that we use a well-known line-simplification algorithm, which can only handle simple, open polygonal curves [25]. Unfortunately, the set of all c-edges defines polygonal "curves" that are not necessarily simple nor open (i.e., they are closed and may form T-junctions). So, we preprocess the set of all c-edges in order to define a set of maximally longer simple open polygonal curves (see Figure 2(a)), and then execute the aforementioned line-simplification algorithm on the resulting curves.

Formally, let $C_G = V \cup E$. We partition C_G into a set of simple curves by first cutting the graph at all vertices with valence greater than two and then cutting all remaining closed curves open. Next, the line simplification algorithm is executed on each open polygonal curve c, producing a simplified curve c' from c. Curve c' approximates c and its vertex set is a subset of the vertex set of c. It is important to point out that the simplification algorithm [25] provides error bounds that allow us to precisely drive the simplification. As a result we obtain a set of simplified c-edges, which defines a PSLG, $G' = (V', E')$, such that V' and E' are the vertex and edge sets of all simplified curves, respectively. The edges and vertices of G' also delimit the image objects, as shown in Figure 2(b). Finally, a conforming Delaunay triangulation, $\mathcal{DT}'$, is generated from G' (see Figure 2(c)).

B. *Triangle Pairing and Template-Based Subdivision*

The goal of the template-based subdivision stage is to generate a quad mesh from the previously computed conforming Delaunay triangulation, $\mathcal{DT}'$. To do that, we first pair up adjacent triangles of $\mathcal{DT}'$ using an adaptation of the *triquad* procedure in [26] to planar triangulations. The modified procedure maintains a max-heap H of ordered pairs, (e, k), where e is an *unconstrained* edge from $\mathcal{DT}'$ and k is the length of e. Initially, all edges in H are said to be unmarked. The procedure removes one pair (e, k) at a time from the top of the heap. If e is unmarked, then the two triangles of $\mathcal{DT}'$ sharing e are paired up to form a quad, and all unconstrained edges of this quad are marked. However, if the quad is not strictly convex, the pairing is discarded. The procedure ends when H is empty. It turns out that our procedure eventually leaves out several unpaired triangles, so that $\mathcal{DT}'$ may not be converted into an *all*-quad mesh.

It might be the case that $\mathcal{DT}'$ contains some poor quality triangles. If so, before executing the modified triquad procedure, we run Ruppert's algorithm on $\mathcal{DT}'$ in order to remove poor quality triangles (see Figures 2(c)-(d)). However, to avoid creating a dense triangulation (which would cause the resulting all-quad mesh to be overly dense), we relax the quality measure threshold of the algorithm and limit the number of point insertions. Unfortunately, we have currently no way of setting a value for this threshold that is suitable for every possible input mesh. So, in our experiments, we have manually tuned and selected the quality threshold for each example. It is worth noticing that the insertion of new vertices is not essential for our algorithm and it is only employed to improve the quality of the quad elements.

Let $\mathcal{M}$ be the collection of triangles and quads resulting from the pairing procedure. Regardless of whether $\mathcal{M}$ consists of quads only, each triangle or quad in $\mathcal{M}$ is subdivided into a small and fixed amount of quads to produce an all-quad mesh. The subdivision of a triangle (or a quad) into quads is based on the templates shown in Figures 3(a)-(b). More specifically, each template is a fixed subdivision of a canonical triangle or square, which is then mapped by an affine map or a bilinear map to triangles or quads in $\mathcal{M}$, respectively. After mapping the canonical templates to the triangles and quads in $\mathcal{M}$, we obtain a quadrangulation, $\mathcal{Q}$, that *conforms* to the edges of G' and respect the triangle mesh partition, i.e., every edge of G' is either an edge or a union of edges of $\mathcal{Q}$. Furthermore, we have that $|\mathcal{Q}| = |\mathcal{DT}'|$ and the partition is preserved. Figure 2(e) shows a template-based mesh.

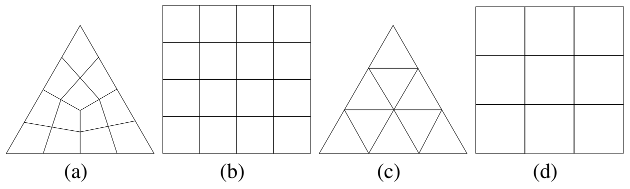


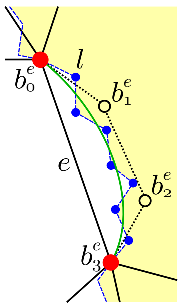
Figure 3. Templates for the canonical (a) triangle and (b) square subdivisions, and the uniform subdivisions of the canonical (c) triangle and (d) square for defining the control net of triangular and rectangular Bézier patches.

C. *Optimization-Based Smoothing*

The final stage of the remeshing step carries out two inter-related tasks. First, the quad mesh, $\mathcal{Q}$, resulting from the previous stage is adapted to the image object boundaries, i.e., to the original polygonal curves from C_G (see Section V-A). By adapting, we mean to move mesh vertices toward the original polygonal curves of C_G. As a result the quad mesh of each mesh partition element faithfully approximates its corresponding region defined by the PSLG G. Second, the quality of the adapted mesh quads is improved by a new optimization-based relaxation. Both tasks are related with each other by the fact that we adapt and optimize the mesh by moving mesh vertices with the guidance of Bézier surface patches.

Boundary Adaptation. Recall that every quad of $\mathcal{Q}$ belongs to either a triangle or a quad from the set $\mathcal{M}$ resulting from the triquad procedure (see Section V-B). To move the vertices of $\mathcal{Q}$, we assign a triangular (rectangular) Bézier patch b_σ (b_μ) of total degree 3 (bi-degree $(3,3)$) to each triangle σ (quad μ) from $\mathcal{M}$. The patch b_σ (b_μ) is responsible for guiding the movement of the vertices of the quads inside triangle σ (quad μ). Since $\mathcal{Q}$ is a planar mesh, we set the z coordinates of all control points of b_σ (b_μ) to 0. So, vertex movements are constrained to the plane xy.

To compute the control points of each b_σ (b_μ), we distinguish two cases. If σ (μ) does not contain any c-edge of G', then we uniformly subdivide the canonical triangle t (square q) associated with b_σ (b_μ), as shown in Figures 3(c)-(d), and let the control points of b_σ (b_μ) be the image of the subdivision vertices under the affine (bilinear) map that takes t (q) onto σ (μ). If σ (μ) contains a c-edge of G', then we first proceed as in the previous case, and then consider each c-edge e of σ (μ) at a time. From the simplification stage, c-edge e is an edge of a simplified polygonal curve from $C_{G'}$, which corresponds to a chain of one or more edges of a polygonal curve, say l, from C_G. So, we redefine the control points of b_σ (b_μ) in e in such a way that the boundary Bézier curve of b_σ (b_μ) closely approximates l as shown in illustration on the right. In what follows we use b_τ to denote both b_σ and b_μ.



To position the control points of the boundary Bézier curve of b_τ associated with c-edge e in $C_{G'}$, we solve a curve fitting problem using a least squares-based procedure. More specifically, let $b_\tau^e : [0,1] \to \mathbb{R}^2$ be the boundary cubic Bézier curve of b_τ associated with e. Then,

$$b_\tau^e(t) = \sum_{i=0}^{3} B_i^3(t) \cdot b_i^e \,,$$

for every $t \in [0,1]$, where $B_i^3(t)$, $i \in \{0,1,2,3\}$, is the i-th Bernstein polynomial of degree 3 and b_i^e are the control points on the boundary of the Bézier patch b_τ. Let $(t_i)_{i=0}^n$ be a list of $n+1$ *parameter* values in $[0,1]$, which correspond to the $n+1$ points $(p_i)_{i=0}^n$ defining l, where $n \geq 4$, $t_0 = 0$, $t_n = 1$, and $t_j < t_{j+1}$, for every $j \in \{0,\ldots,n-1\}$. By setting $b_\tau^e(t_0) = b_0^e = p_0$ and $b_\tau^e(t_n) = b_3^e = p_n$, our fitting problem boils down to find the control points $b_1^e = (b_{1,x}^e, b_{1,y}^e)$ and $b_2^e = (b_{2,x}^e, b_{2,y}^e)$ so that $b_\tau^e(t)$ is the "best" fitting (in the least squares sense) to l. To find b_1^e and b_2^e, we assemble two systems, $K\boldsymbol{x} = \boldsymbol{d}$ and $K\boldsymbol{y} = \boldsymbol{f}$, of $n+1$ linear equations each, where K is a $(n+1)\times 2$ matrix such that the i-th line of K consists of the elements $B_1^3(t_i)$ and $B_2^3(t_i)$, $\boldsymbol{x}$ is the x-coordinate vector $[b_{1,x}^e \; b_{2,x}^e]^{\mathrm{T}}$, $\boldsymbol{y}$ is the y-coordinate vector $[b_{1,y}^e \; b_{2,y}^e]^{\mathrm{T}}$, $\boldsymbol{d}$ is a vector with $(n+1)$ coordinates such that its i-th coordinate is $p_{i,x} - (B_0^3(t_i)\cdot b_{0,x}^e + B_3^3(t_i)\cdot b_{3,x}^e)$, and $\boldsymbol{f}$ is a vector with $(n+1)$ coordinates such that its i-th coordinate is $p_{i,y} - (B_0^3(t_i)\cdot b_{0,y}^e + B_3^3(t_i)\cdot b_{3,y}^e)$, with $p_i = (p_{i,x}, p_{i,y})$, $b_0^e = (b_{0,x}^e, b_{0,y}^e)$, and $b_3^e = (b_{3,x}^e, b_{3,y}^e)$. Since $n \geq 4$, both systems are overdetermined, and thus we seek the least squares solutions, $\boldsymbol{x}$ and $\boldsymbol{y}$, that minimize $\|K\boldsymbol{x} - \boldsymbol{d}\|_2$ and $\|K\boldsymbol{y} - \boldsymbol{f}\|_2$.

For our purposes, the set of parameter values, $(t_i)_{i=0}^n$, is obtained by a chord length parametrization of the vertices $(p_i)_{i=0}^n$ of l over e. Indeed, each polygonal curve c from C_G is the union of one or more polygonal chains, each of which corresponds to a c-edge e of G'. So, the entire curve c is approximated by a set of cubic Bézier curves. Figure 2(f) shows the result of applying the boundary fitting scheme to the mesh of Figure 2(e).

From the definition of the control points of the Bézier patch b_σ (b_μ) associated with each triangle σ (quad μ) of $\mathcal{M}$, the boundary Bézier curves of adjacent patches are exactly the same (i.e., they have the same control points). This is obviously true for the boundary curves associated with c-edges. For the curves associated with edges that are not constrained, our claim follows from the facts that (1) the curve control points are placed along the edges, and (2) they are images of canonical triangle (quad) subdivision vertices under an affine (bilinear) map, both of which preserve distance ratio along a line.

Optimization. The goal of the optimization task is to improve the *shape quality* of the quads of $\mathcal{Q}$. Recall that each quad of $\mathcal{Q}$ is the image of a quad defined in a canonical triangle or quad in $\mathcal{M}$. Triangular and rectangular Bézier patches are also defined on the canonical domain. So, each vertex of $\mathcal{Q}$ can be written in terms of the Bézier patches and by imposing that adjacent Bézier patches share the same cubic Bézier curve, one can modify the quad mesh $\mathcal{Q}$ by moving the control points of the patches.

To improve the shape quality of the quads of $\mathcal{Q}$, we judiciously move the control points of all Bézier patches to improve the shape quality of all quads with respect to the *shape quality measure*. In our case, the *Shape and Size* quadrilateral quality metric [27]. In general, we can view the shape measure as a function, $s : \mathcal{Q}_q \to [0,1]$, where $\mathcal{Q}_q$ is the set of quads of $\mathcal{Q}$. Function s is defined in such a way that for each quad $\nu \in \mathcal{Q}_q$, the larger the value of $s(\nu)$ the better the quality of ν. Therefore, the optimal positioning of the control points can be found by minimizing the following energy:

$$q_s = \sum_{\nu \in \mathcal{Q}'_q} (1 - s(\nu))^2 \,,$$

To this end, we used Powell's method [28] defined on the space of the coordinates of the control points. Because the total number of control points is smaller than the number of vertices in $\mathcal{Q}$, the proposed optimization mechanism turns out to be more effective than directly using the coordinates of the quad vertices. In order to avoid folding (inversion of a quad element), a control point movement must be *feasible*, i.e., every control point must be in the interior of the polygon defined by its adjacent control points in the Bézier control net. This constraint is imposed when applying Powell's method. Our procedure is iterative and runs until q_s is below a predefined threshold or the number of iterations exceeds another predefined threshold. Figure 4 shows the result of applying the optimization mechanism to the mesh of Figure 2(f).

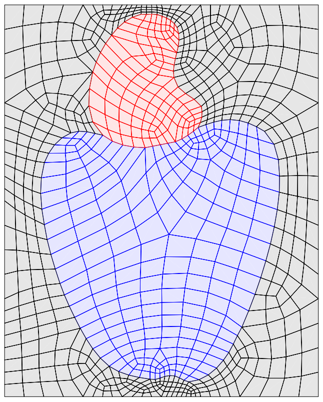


Figure 4. An optimized template-based quad mesh.

VI. Experimental Results

This section presents the results of some experiments we carried out with the proposed *Imesh* extension. To illustrate the effectiveness of our extension, we generated quad meshes from two images from the Berkeley Segmentation Dataset[1] (pyramid and lake) and a range image of a hub wheel from the Stuttgart Range Image Database[2]. These images are considered "real data". All experiments were conducted in an AMD Athlon X2 4450B 2.3GHz with 3GB RAM. We used an implementation of the *Shape and Size* quadrilateral quality metric available in the VERDICT library[3].

The three images used in our experiments are shown in Figure 5. Colored squares in Figures 5(a) and 5(b) correspond to samples used by the built-in texture classifier of *Imesh*. Each color represents a region of interest (label). Five and four distinct regions of interest are defined in Figures 5(a) and 5(b), respectively. Since the three regions in Figure 5(c) can easily be identified by thresholding, we needed not use the texture classifier for this particular example.

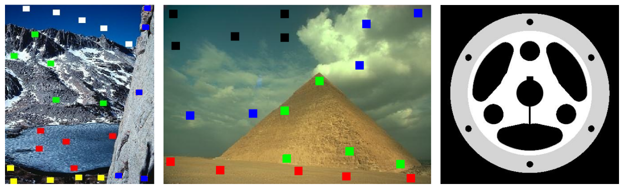

(a) Lake (b) Pyramid (c) Hub wheel

Figure 5. Images used in our experiments. The colored squares in (a) and (b) correspond to samples used by the texture classifier built in the *Imesh* (each color corresponds to a label).

Figure 6 presents the quad meshes resulting from applying our *Imesh* extension to the images in Figure 5. Notice that the segmentation provided by *Imesh* is naturally preserved by the proposed remeshing scheme. Moreover, as a consequence of the boundary adaptation mechanism, curves between distinctly labeled regions are precisely represented. Furthermore, the "jagged" effect, usually observed in triangle meshes generated by the original version of *Imesh*, is now avoided.

The quality histograms for the meshes in Figure 6 are shown in Figure 7. The vertical dashed line represents the lower bound (value equal to 0.2) from which a quad element is considered of bad quality in accordance with the Shape and Size measure [27]. The "redish" and blue histograms correspond to mesh quality before and after optimization, respectively. Notice that the proposed control-point-based optimization mechanism was able to improve mesh quality considerably, avoiding bad elements altogether. This fact can

[1]http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/
[2]http://range.informatik.uni-stuttgart.de/htdocs/html/
[3]VERDICT – http://cubit.sandia.gov/verdict.html

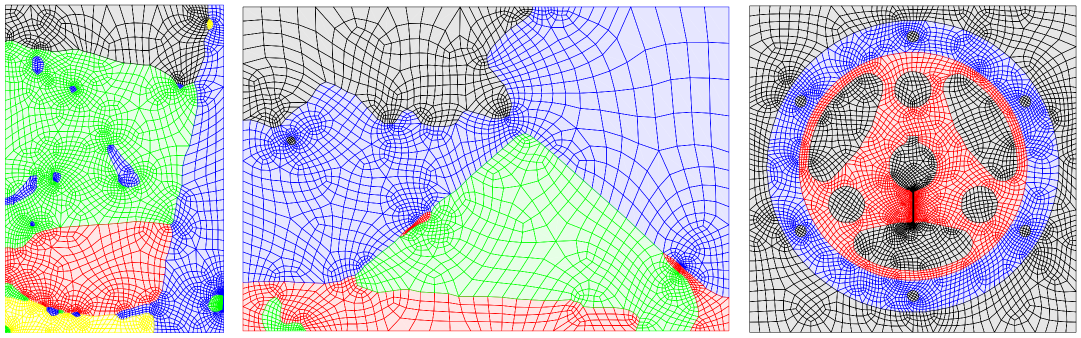
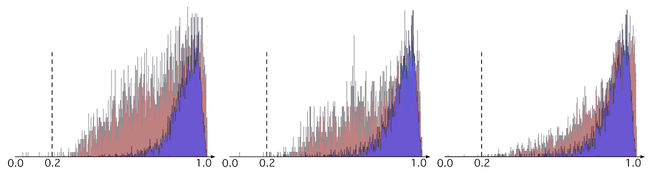

(a) Lake (b) Pyramid (c) Hub Wheel

Figure 6. Quad meshes resulting from the images in Figure 5. Mesh segmentation is directly obtained, avoiding any pos-processing step.

also be observed from the fifth column of Table I, which brings the quality of the worst element for the quad meshes in Figure 4 and Figure 6. Sixth and seventh columns confirm the effectiveness of the optimization mechanism, showing that, on average, the quality of quad elements is above 0.8.

Table I
NUMBER OF ELEMENTS, COMPUTATIONAL TIMES, AND QUALITY MEASURE FOR MESHES SHOWN IN FIGURES 4 AND 6.

	# vert	# cells	Remeshing (in seconds)	worst	mean	variance
Cashew	865	816	6.1	0.288	0.837	0.011
Pyramid	3197	3112	34.5	0.233	0.874	0.008
Lake	5171	5028	57.6	0.264	0.878	0.006
Hub	7307	7220	78.8	0.210	0.899	0.006



(a) Lake (b) Pyramid (c) Hub wheel

Figure 7. Quality histograms of the meshes in Figure 6. Vertical dashed lines indicate the lower bound from which a quad element is considered of bad quality. Red and blue histograms correspond to mesh quality before and after optimization respectively.

The number of vertices and quads of the meshes presented in Figures 4 and 6 are in the second and third columns of Table I. The computational times in the fourth column are quite acceptable for a quality quadrilateral meshing scheme. It is worth remarking that the optimization mechanism is indeed the most time-consuming one, representing 98% of the total computational time. The capability of representing boundaries and adapting the mesh locally are important features of the proposed remeshing method. Such features can easily been seen in Figure 8, which shows quad meshes with different refinements for the Hub Wheel image (Figure 5(c)). Even for the coarser mesh, which contains about 1.2K vertices, the proposed remeshing scheme was able to satisfactorily represent boundaries. It was also able to adapt quad sizes to capture the small chink in the lower part of the hub. It is worth mentioning that finer quad meshes may be obtained by simply adding new triangles in $\mathcal{DT}'$ before triggering the pairing process and the template mapping.

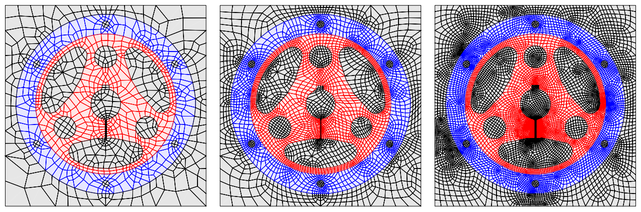
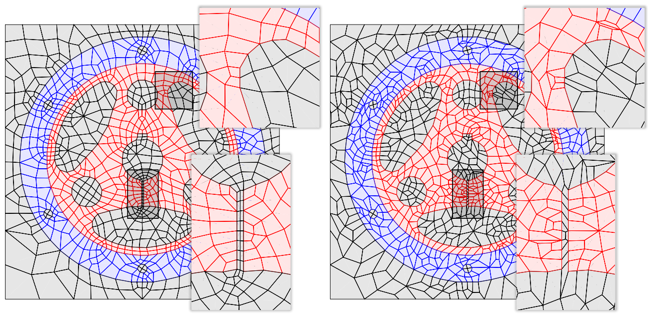

(a) 1.2K (b) 5K (c) 10K

Figure 8. Quad meshes generated from the Hub Wheel image: a) 1.2K vertices, b) 5K vertices, and c) 10K vertices.

Parameters. The level of boundary simplification needs to be carefully chosen, in order to preserves the topology of G in G'. We use as threshold for the Cashew, Lake, Pyramid and Hub Wheel, the respectively number of pixels: 15, 2.5, 3 and 4. We use the 4-subdivision templates (Figures 3(a)-(b)) in all examples, except for the Hub Wheel with 1.2K quads, where we use a 3-subdivision templates. Finally, to define the density of each mesh, we controlled the number of templates by setting Ruppert's refinement quality criterion to 20.7°, 20.7°, 20.7°, 18°, 18°, 26.5° and 30°, respectively to the Cashew, Lake, Pyramid and Hub Wheel with 1.2K, 5K, 7K and 10K quads.

We end this section showing a comparison between the proposed *Imesh* extension and the quad meshing algorithm described in [21], which generates quad meshes from polygonal curves while employing Laplacian smoothing as a post-processing step. We used the 1.2K hub wheel quad mesh (see Figure 8(a)) as a basis for comparison. Since most vertices in that mesh are constrained by the boundary curves, the control-point-based optimization step barely affects mesh quality. Even so, our proposed extension was able to produce a better quality quad mesh, as one can see in Figure 9. The computational time, in seconds, to generate each mesh shown in Figure 8 is described in the second column of Table II. For the purpose of comparisons, the third column shows the computation time for the algorithm described in [21].



(a) (b)

Figure 9. Comparison between the proposed methodology (a) and the quad meshing scheme described in [21].

Table II
COMPUTATION TIME (IN SECONDS) TO GENERATE EACH QUAD MESH FROM HUB WHEEL.

# cells	Template-based	Meshing described in [21]
1.2k	17.2	3.8
5k	53.5	20.7
10k	128.8	43.3

VII. Conclusions

This paper described a novel quad-based remeshing scheme, which can be regarded as a replacement for the triangle-based improvement step of *Imesh*. Our scheme is able to generate all-quad meshes, while preserving boundaries and partitions defined in the first steps of *Imesh*. To produce good quality meshes, our scheme is accompanied by a new optimization procedure, which moves mesh vertices around guided by changes of control points of Bézier patches defined on the mesh domain. By changing the position of mesh vertices via control points, our procedure reduces computational effort. Our experiments show evidences that our optimization procedure is also quite effective. To the best of our knowledge, no other meshing scheme generates good quality quad meshes *directly* from images. We are currently investigating how to tailor templates so as to reduce the number of extraordinary vertices (valence other than four) in the final quad mesh. We are also interested in proving lower and upper bounds for the internal angles of the resulting mesh quads. Finally, we are looking into ways of extending some of the ideas presented here to 3D meshing schemes.

Acknowledgments. This work was supported in part by grants from Fapesp-Brazil (#2008/03349-6), CNPq-NSF (#491034/2008-3), CNPq (# 307450/2009-0) and was carried out in the framework of the Instituto Nacional de Ciência e Tecnologia em Medicina Assistida por Computação Científica (CNPq, Brazil).

References

[1] M. A. S. Lizier, D. C. Martins-Jr, A. J. Cuadros-Vargas, R. M. Cesar-Jr, and L. G. Nonato, "Generating segmented meshes from textured color images," *Journal of Visual Communication and Image Representation*, vol. 20, pp. 190–203, 2009.

[2] A. J. Cuadros-Vargas, M. A. S. Lizier, R. Minghim, and L. G. Nonato, "Generating segmented quality meshes from images," *Journal of Mathematical Imaging and Vision*, vol. 33, pp. 11–23, 2009.

[3] M. Bern and P. Plassmann, "Mesh generation," in *Handbook of Comput. Geom.*, J.-R. Sack and J. Urrutia, Eds. Elsevier, 2000.

[4] J. Ruppert, "A delaunay refinement algorithm for quality 2-dimensional mesh generation," *J. of Algorithms*, vol. 18, no. 3, pp. 548–585, 1995.

[5] G. Miller, S. Pave, and N. Walkington, "When and why ruppert's algorithm works," in *Proceedings of the 12th International Meshing Roundtable (IMR)*, 2003, pp. 91–102.

[6] F. B. Atalay, S. Ramaswami, and D. Xu, "Quadrilateral meshes with bounded minimum angle," in *Proceedings of the 17th International Meshing Roundtable (IMR)*, 2008, pp. 73–91.

[7] A. Malanthara and W. Gerstle, "Comparative study of unstructured meshes made of triangles and quadrilaterals," in *Proceedings of the 6th International Meshing Roundtable (IMR)*, 1997, pp. 437–447.

[8] P. Alliez, G. Ucelli, C. Gotsman, and M. Attene, "Recent advances in remeshing of surfaces," in *Shape Analysis and Structuring*, ser. Mathematics and Visualization, L. D. Floriani and M. Spagnuolo, Eds. Springer Berlin Heidelberg, 2008.

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

[10] 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*, 1999, pp. 168–170.

[11] S. Coleman and B. Scotney, "Mesh modeling for sparse image data set," in *IEEE ICIP*. IEEE Computer Society, 2005, pp. 1342–1345.

[12] P. Kocharoen, K. Ahmed, R. Rajatheva, and W. Fernando, "Adaptive mesh generation for mesh-based image coding using node elimination approach," in *IEEE ICIP*, 2005, pp. 2052–2056.

[13] Y. Yang, M. Wernick, and J. Brankov, "A fast approach for accurate content-adaptive mesh generation," *IEEE Trans. on Image Processing*, vol. 12, no. 8, pp. 866–881, 2003.

[14] J. Cebral and R. Lohner, "From medical images to cfd meshes," in *Proc. of the 8th Intern. Meshing Roundtable*, 1999, pp. 321–331.

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

[16] 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*, 2003, pp. 286–291.

[17] J.-D. Boissonnat, J.-P. Pons, and M. Yvinec, "From segmented images to good quality meshes using delaunay refinement," *Lecture Notes in Computer Science*, vol. 5416, pp. 13–37, 2009.

[18] O. Skrinjar and A. Bistoquet, "Generation of myocardial wall surface meshes from segmented mri," *International Journal of Biomedical Imaging*, vol. 2009, 2009.

[19] J. S. B.P. Johnston and A. Kwasnik, "Automatic conversion of triangular finite meshes to quadrilateral meshes," *International Journal for Numerical Methods in Engineering*, vol. 31, no. 1, pp. 67–84, 1991.

[20] S. Owen, M. Staten, S. Cannan, and S. Saigal, "Q-morph: An indirect approach to advancing front quad meshing," *Intern. J. for Num. Meth. Eng.*, vol. 9, no. 44, pp. 1317–1340, 1999.

[21] S. Ramaswami, M. Siqueira, T. Sundaram, J. Gallier, and J. Gee, "Constrained quadrilateral meshes of bounded size," *Intern. J. of Comput. Geometry & Applications*, vol. 15, no. 1, pp. 55–98, 2005.

[22] R. Schneiders, "Refining quadrilateral and hexahedral element meshes," in *Proc. 5th Intern. Conf. on Num. Grid Generation in Computational Field Simulations*, 1996, pp. 679–688.

[23] G. Herman, *Geometry of Digital Spaces*. Birkhäuser: Boston, MA, USA, 1998.

[24] J. Shewchuk, "What is a good linear element? interpolation, conditioning, and quality measures," in *Proceeding of the 11th International Meshing Roundtable*, 2002, pp. 115–126.

[25] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a line or its caricature," *The Canadian Cartographer*, vol. 10, no. 2, pp. 122–122, 1973.

[26] L. Velho and D. Zorin, "4-8 subdivision," *Computer Aided Geometric Design*, vol. 18, no. 5, pp. 397–427, 2001.

[27] P. M. Knupp, "Algebraic mesh quality metrics," *SIAM Journal on Scientific Computing*, vol. 23, no. 1, pp. 193–218, 2001.

[28] M. J. D. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The Computer Journal*, vol. 7, no. 2, pp. 155–162, 1964.