PERGAMON

Computers & Graphics 26 (2002) 907–918

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

A modeling methodology for finite element mesh generation of multi-region models with parametric surfaces[☆]

William M. Lira[a], Paulo Roma Cavalcanti[b], Luiz C.G. Coelho[a], Luiz F. Martha[a,*]

[a] *Department of Civil Engineering and Computer Graphics Technology Group (Tecgraf), Pontifical Catholic University of Rio de Janeiro (PUC-Rio) 22453-900, Rua Marquês de São Vicente, 225, Rio de Janeiro, RJ, Brazil*

[b] *Department of Computer Science, Federal University of Rio de Janeiro (UFRJ) 21945-970, Cidade Universitária, Ilha do Fundão, Rio de Janeiro, RJ, Brazil*

Abstract

This paper presents a description of the reorganization of a geometric modeler, MG, designed to support new capabilities of a topological module (CGC) that allows the detection of closed-off solid regions described by surface patches in non-manifold geometric models defined by NURBS. These patches are interactively created by the user by means of the modeler's graphics interface, and may result from parametric–surface intersection in which existing surface meshes are used as a support for a discrete definition of intersection curves. The geometry of realistic engineering objects is intrinsically complex, usually composed by several materials and regions. Therefore, automatic and adaptive meshing algorithms have become quite useful to increase the reliability of the procedures of a FEM numerical analysis. The present approach is concerned with two aspects of 3D FEM simulation: geometric modeling, with automatic multi-region detection, and support to automatic finite-element mesh generation.
© 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Multi-region non-manifold modeling; Parametric surfaces; Finite-element mesh generation; Object-oriented programming

1. Introduction

Finite-element analysis [1,2] and geometric modeling of solids [3,4] are important items in the process of simulating engineering problems, especially when an analytic solution is unknown or difficult to obtain. See, for example, the turbine disk model shown in the Section 6 of this article.

Generally, the finite-element method (FEM) is based on a numerical model obtained from the refinement or "discretization" of the problem's domain, combined with additional information necessary for the complete definition of the physical problem. Such information consists of a set of parameters, called *simulation attributes* [5,6]. The discretization, denominated finite-element mesh, consists of a group of nodes or vertices (points with coordinates) and a group of cells, called finite elements, with a predefined topology (triangular, quadrilateral or tetrahedral, for example). The elements are defined by a list of node connectivity (the sequence of vertices that belong to each element). A finite-element model is the association of a finite-element mesh with a set of simulation attributes.

One important aspect of three-dimensional finite-element simulation is mesh generation. It is a research area that has been active since the creation of the method. In general, the mesh-generation process is time-consuming and quite tiresome, apart from demanding a

[☆] Based on "An OOP approach for mesh generation of multi-region models with NURBS" by William M. Lira, Paulo Roma Cavalcanti, Luiz C. G. Coelho and Luiz F. Martha, which appeared in Proceedings of SIBGRAPI 2001 (XIV Brazilian Symposium on Computer Graphics and Image Processing) © 2001 IEEE.

*Corresponding author. Fax: 55-21-3114-1195.

E-mail addresses: william@tecgraf.puc-rio.br (W.M. Lira), roma@lcg.ufrj.br (P.R. Cavalcanti), lula@tecgraf.puc-rio.br (L.C.G. Coelho), lfm@tecgraf.puc-rio.br (L.F. Martha).

0097-8493/02/$ - see front matter © 2002 Elsevier Science Ltd. All rights reserved.
PII: S0097-8493(02)00179-6

certain degree of experience from the professional responsible for this task. In this context, automatic and adaptive meshing algorithms have revealed themselves quite useful to increase the reliability of the procedures in a FEM numerical analysis [1].

Another important aspect in 3D FEM simulation is the creation of the geometric model. There are several issues involved in this task, ranging from user-interface strategies to data-representation schemes. To accomplish this, it is necessary to use special programs, called modelers, which can digitally reproduce common geometric forms of the target objects [4]. The geometry and shape of realistic engineering objects are intrinsically complex, usually composed by several materials and regions.

The modeling methodology discussed in this paper is concerned with two aspects of 3D FEM simulation:

- geometric modeling, with automatic multi-region detection, and
- support to automatic finite-element mesh generation.

This methodology could be implemented using currently available solid-modeling libraries, such as ACIS [7], Parasolids [8] and Pro/ENGINEER API Toolkit [9]. These libraries provide topological and geometric representation as well as application program interface (API) functions, which are necessary for this type of modeling. However, they are expensive, include a large number of classes, long APIs, and, generally, the model requires a pre-processing stage before being adequate as input for 3D mesh generation. To tackle these issues, the authors of the present paper have been involved for the last decade in the development of modeling tools that may also provide an appropriate environment for the implementation of the target methodology. One key aspect in this methodology is the integration of solid modeling and automatic/adaptive finite-element mesh generation. This integration provides a consistent conversion between the solid-model and the finite-element representation, and allows a fast prototyping of new concepts using relatively small pieces of software.

The proposed modeling methodology was implemented in an existing finite-element modeler called MG, which is a finite-element pre-processor that was originally devised for the generation of surface (shell) finite-element models [10]. Later on, the system was extended to also consider solid meshes [11]. MG's modeling capabilities address two important issues in 3D finite-element modeling. The first is related to user-interface and interactive-graphics procedures to generate surface meshes [12], and the second is the intersection between surface finite-element meshes, such as the ones shown in Fig. 1.

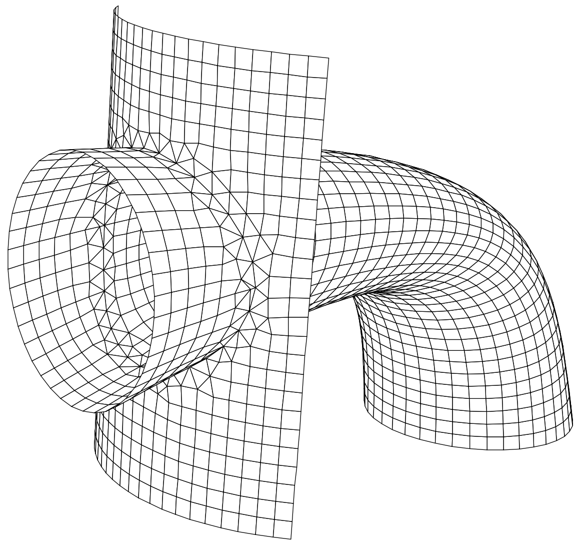
Fig. 1. Example of intersection of surface meshes.

The algorithm for intersecting surface meshes [10] is actually a scheme for parametric–surface intersection in which existing surface meshes are used as a support for the definition of intersection curves. These curves have representations in the Cartesian space and in the parametric space of each intersecting parametric surface (trimming curves). These geometric representations consist of B-splines defined by interpolation points that result from the existing support surface meshes. Only these interpolation points are considered in the geometric solution of the intersection problem. A numerical iterative scheme defines the location of an interpolation point according to the parametric representations of the intersecting surfaces. In this sense, intersection curves are defined in a discrete fashion, as opposed to represent them analytically. This discrete scheme avoids problems experienced by most modelers, such as inconsistencies between parametric representations of a point in the intersection of two or more surface patches. The searches required for computing the intersection curves and for surface re-meshing are supported by an auxiliary topological data structure whose main feature is that topological entities are stored in spatial-indexing trees, instead of linked lists. These spatial-indexing structures play a major role in the overall efficiency of the algorithm. The auxiliary data structure is defined in the parametric space of each intersecting surface.

The original version of MG is powerful in model representation, and has a relatively simple and efficient interface. However, its data structure was not based on any formal geometric-modeling concept. Therefore, in many situations, the geometric consistency of a model relies on user intervention. For example, there is no capability to automatically detect when a volume in space is enclosed off by a set of surface patches. The user must explicitly indicate this, which may be a hard task in a realistic engineering model. This capability is particularly important for finite-element mesh generation, as sometimes it is desirable to have several regions, each one meshed by a different algorithm.

The formalism necessary for modeling capabilities that allow automatic recognition of created solid regions is beyond the scope of this paper. The methodology adopted here has been previously devised by the authors [13] and is based on a complete topological representation of a space subdivision, called complete geometry complex (CGC), which is summarized in Section 2.

This paper describes in Sections 3–5 a class organization, in the context of Object-Oriented Programming (OOP), of a new version of the MG modeler that, still keeping the simple and efficient user-interface characteristics, provides capabilities for automatic region detection with surface patches represented by non-uniform rational B-splines (NURBS) [14]. To achieve this, a hybrid approach was adopted in which a full CGC representation of the model is not maintained at every step of the modeling process. In this approach, surface intersection is accomplished using an algorithm implemented in the MG modeler [10], while CGC module is just responsible for multi-region recognition. The previous MG data structure was extended so that a CGC model can be created at any moment when the user requires region detection. Three-dimensional geometric modeling is a difficult task that might involve a series of intermediate steps in which automatic intersection computation is not desirable. Depending on the model's complexity, maintaining consistency between geometry and topology after every step of the modeling process might be very inefficient. It would be very difficult to achieve a reasonable degree of user-interface efficiency if this consistency were enforced after each user-interface task.

The new class organization also provides support for automatic surface- and solid-mesh generation. It is not the purpose here to describe meshing algorithms. It suffices to say that surface-mesh generation is performed in the parametric space of each surface and that the algorithms provided in MG are described elsewhere [10,11,15]. The support for automatic mesh generation is accomplished by the creation of two new topological entities (*Segment* and *Patch2d*) and by the concept of *use* of a topological entity by a surface.

2. Complete geometric complex representations

Usually, there are two main strategies for the representation of a three-dimensional geometric model: a constructive scheme and a boundary scheme. The most common constructive scheme is constructive solid geometry (CSG) [16], in which the target object is constructed by a set of Boolean operations applied to primitive objects. In the boundary scheme, the geometry of an object is defined by its boundary elements, such as vertices, edges and surface patches, which may be created through an interactive graphics interface. However, the complete definition of the solid model requires combinatorial relationships among the several surface patches, which will result in the definition of the interior region, model boundaries and other topological information. This type of solid-model representation is called boundary representation (B-REP) [3,7].

Traditional CSG and B-REP techniques apply to objects that decompose space into three parts: interior, exterior and boundary. This class of objects is referred to as manifold objects, because their boundaries are two-manifold sets in three-dimension [3]. This means that the original techniques could not model multi-region objects.

Many applications in Engineering need to model multi-region objects or objects that present other non-2D closed-manifold features, such as dangling faces or edges. For this reason, many works in the literature have proposed non-manifold modeling schemes [7,13,17–23].

The generation of a consistent non-manifold B-REP model from a set of surface patches is not a simple task and may involve surface intersection and region detection. Considering the user-interface problems related to this task, it is desirable that the creation of a B-REP model from a set of surface patches be automated. The ideal environment for the user would be to create these patches with no explicit relationship, except for the use of already created geometry information for the generation of a new patch. The modeler would automatically generate surface-intersection curves and topologically link these entities to the geometric definition.

A previous work by the present research group [13] proposed a non-manifold approach for modeling multi-region objects. A general methodology for creating and manipulating spatial subdivision in cells of arbitrary shape and geometry was developed. A spatial subdivision may be created by means of inserting planar surface patches one by one, allowing the insertion of new patches in real time. The object resulting from this decomposition is classified as CGC, as it is a special case of geometric complex [22] that occupies the entire three-dimensional space (the unlimited outer region is also represented in the subdivision).

Several works have presented methods to represent spatial subdivisions. Rossignac and O'Connor [22] addressed the general problem of representing *n*-dimensional objects, possibly with internal structures. Some data structures used in non-manifold solid modeling [7,17–19,21] represent, in a general way, the adjacency relationships of three-dimensional objects not necessarily homogeneous in dimension. In the present

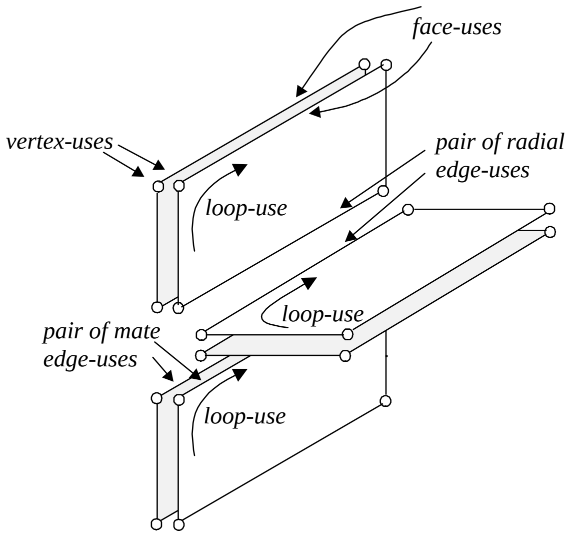

Fig. 2. *Uses* of topological elements in RED data structure.

implementation, the radial-edge data structure (RED), proposed by Weiler [20,21], is adopted.

This data structure is known as radial-edge because it explicitly stores the list of faces radially ordered around an edge (Fig. 2). RED was conceived for non-manifold modeling and Weiler has proven its completeness, which means that any adjacency relationship can be extracted from this representation.

In order to describe the topology of a spatial subdivision, the radial-edge representation employs the concept of *use* of a topological element. A *use* can be seen as the occurrence of a topological element in an adjacency relationship related to an element of higher dimension. Thus, the radial-edge structure explicitly stores the two *uses* (sides) of a *face* by the two *regions* (not necessarily distinct) that share that *face*. Each *face-use* is bounded by one or more *loop-uses*, which are composed by an alternating sequence of *edge-uses* and *vertex-uses* (Fig. 2). *Vertex-uses* are necessary to store non-manifold conditions at vertices.

The RED structure is a hierarchical description of a spatial subdivision, starting in higher dimension levels (*regions*) and reaching the lower levels (*vertices*) (Fig. 3). The topological elements are kept in doubly linked circular lists and have pointers to their attributes.

Topological data structures are complex and should not be directly manipulated. Weiler has introduced a set of operators that provide a high-level method to access the radial-edge structure. These operators are divided in two groups. The first group has operators that act on faces of a spatial subdivision and are analogous to the (two-manifold) operators presented by Mäntylä [7]. The second group has operators that are capable of creating wireframes and adding faces, which are attached to specified edges or wireframes. They are referred to as

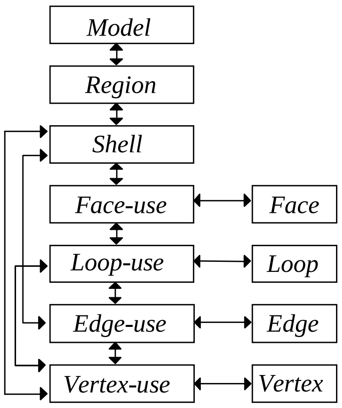

Fig. 3. Hierarchy of topological entities in RED data structure.

non-manifold operators. Considerations about a minimal set of operators can be found in Wu's work [24].

The present CGC modeling was implemented as a library of OOP classes, which provide a set of high-level operators that manipulate a spatial subdivision. These operators receive as input geometric information on the surface patches that are inserted in the spatial subdivision. This geometric information is automatically translated into topological information required by Weiler's non-manifold and manifold operators.

The CGC modeling capability is one of the key aspects of the finite-element modeling scheme proposed in this work. However, up to now the implementation of this methodology could only treat planar (polygonal) surface patches. One of the goals of this work is to extend this methodology to consider parametric curved-surface geometries using NURBS.

3. Hybrid modeling approach

As previously mentioned, the proposed data-representation scheme, which was adopted in the new version of the MG modeler, is based on a hybrid approach. The basic idea is to have two separate representations of the same model. One representation is stored in the modeler's data structure. This is the representation that is actually maintained in computer memory during the modeling process. The modeler's data structure is also permanently stored in disk when a model is saved. The other representation is a temporary conversion of the modeler's data structure into a CGC representation. In this conversion, the topology of the model is determined such that different regions can be distinguished. In an intermediate stage of this conversion, the algorithm for intersecting surfaces is used to compute the curves and surface patches resulting from intersection. The topological entities in the modeler's data structure are also updated to reflect possible changes from surface intersection.

The main advantage of this approach is that it combines the meshing and intersection capability of MG with the topological power of representation and robustness of CGC. There is a two-way communication between these representations, as shown in Fig. 4. CGC may be seen as a "topological engine" that generates topological information that is consistent with the geometry of a model. The topological entities identified by the CGC representation are passed back to the MG representation, which includes automatically detected regions. In Fig. 4, it can also be observed that the geometric description of the topological entities is common to both representations. This geometric description is stored in a separate module based on NURBS representation.

The following steps summarize the adopted hybrid modeling methodology:

(1) the user generates single surface patches that may intersect each other,
(2) MG modeler computes intersections generating curves, curve segments, and surface patches,
(3) the user selects surface patches that will be used in the final model, removing undesirable parts,
(4) CGC module identifies the closed regions and returns this information to MG data structure, and
(5) MG modeler computes the final mesh integrating individual patches and solids.

The three modules shown in Fig. 4 are implemented as OOP classes. The CGC class organization is described by Mello and Cavalcanti [25] and MG's class organization is described in Section 4. The class organization of the NURBS module is shown in Fig. 5. The geometric representation of curves and surfaces stored in the objects of this module uses a public-domain OOP library [26]. This library provides the representation of several types of curves and surfaces, including the conic and quadric forms. There are specific sub-classes for certain geometry types (for example, the *Arc* class geometrically describes a circle arc, and the *Gordon* class, a Gordon surface). The methods in these classes manipulate the corresponding geometric information. For example, there is a method that, given the parametric coordinates of a surface point, obtains the corresponding 3D Cartesian coordinates.

Surface intersections are performed prior to region detection. New curve segments and surface patches may arise from this step. The basic information passed from the modeler's data structure to the CGC data structure consists of a set of surface patches defined by the user by means of MG's graphics interface or resulting from surface intersection. For each surface patch inserted in the CGC representation, a radial-edge topological face is generated. In the CGC representation, the patches only intersect each other at their boundaries. Here, a surface patch is represented by its boundary curves and by its NURBS geometric description. These parameters are sufficient to determine an input surface patch to the CGC module.

This module processes the surface patches passed by the MG module and generates a consistent topological model with multi-region detection, which is then converted back to the MG representation, updating all the topological relationships in the MG data structure. This conversion is performed, basically, by traversing all *regions* and *faces* generated by the CGC module and transforming them into entities of the MG representation. Each *face* in the CGC representation corresponds to a surface patch in the MG representation. Two patches may belong to a same geometric surface in situations where an intersecting trimming curve has subdivided the initial patch. Similarly, each *edge* in the CGC representation corresponds to a curve segment in the modeler's representation, and these segments may share the same geometric curve if they were originated from a curve split. Finally, each *region* in the CGC representation will generate a solid in the MG representation.

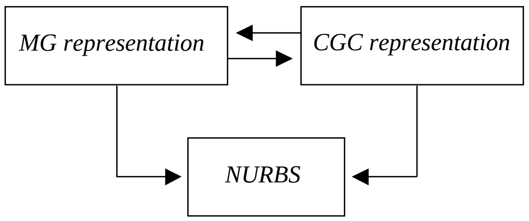

Fig. 4. General modules in proposed modeling approach.

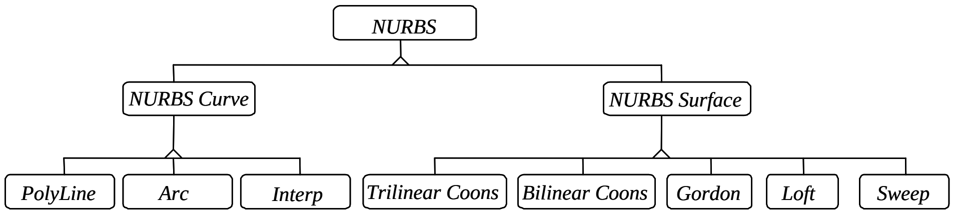

Fig. 5. The class organization of the NURBS module.

In situations where a geometric curve has two or more segments associated to it, the MG modeler does not allow manipulation of curve control points to avoid geometric inconsistencies. The same constraint is applied to surface control points.

4. Modeler's class organization

The main data structure of MG was conceived with the purpose of supporting both surface (shells) and solid finite-element modeling. In this work, MG's data structure was reorganized to consider the distinction between geometric and topological entities, which was not made in its original version. The radial-edge scheme could have been adopted to redesign MG's data structure. However, this would require enforcing consistency between geometry and topology at each modeling step. In addition, MG's data-structure class organization is simpler and lighter than the radial-edge data structure. In spite of this, MG's data structure maintains the required adjacency among topological entities needed to represent a non-manifold multi-region model. Fig. 6 shows the OOP class organization in the MG module. The class diagram shown in this figure, as well as in other figures in this paper, follows the object modeling technique (OMT) nomenclature [27].

The *Entity* class is subdivided in two sub-classes. The first, *Geometry*, refers to the geometric entities of the model. The second, *Topology*, represents the entities that contain topological information.

In the MG data structure, a vertex has a topological instance, which is represented by an object of the *VtxTop* class, and a geometric instance, which is represented by an object of the *Point* class. A *VtxTop* object contains a list of references to adjacent curve segments (*Segment* objects) and a reference to the corresponding *Point* object. An object of the *Point* class stores the geometric position of a vertex in the 3D Euclidian space and has a reference to the corresponding *VtxTop* object.

Similarly, a *Segment* object represents the topology of a portion (segment) of a geometric curve. An object in this class stores the topological information (pointers to adjacent vertices, for example) and a reference to its geometric description (*Curve* object). Fig. 7 illustrates the relationship between a curve (geometry) and its two segments (topology). In this case, curve $c1$ has a list of references to *Segment* objects that belong to it ($s1$ and $s2$). These *Segment* objects contain references to their end vertices, $(v1, v3)$ and $(v3, v2)$, respectively, and to curve $c1$. A *Curve* object must have at least one *Segment* object. Curve $c1$ also has a reference (see Fig. 8) to its geometric representation, which consists in an object of the NURBS module. The sub-classes of *Curve* class shown in Fig. 6 are necessary to implement methods for creation of curves in the modeler.

Another important class in the MG data structure is class *Patch2d*, which represents the topology of a portion of a surface. An object in the *Patch2d* class stores references to its boundary curves and to its surface geometric description, which is represented by

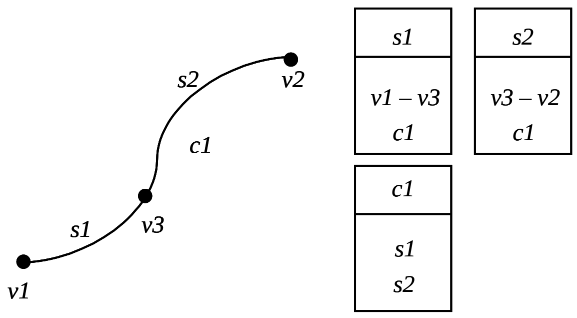

Fig. 7. Relationship *curve-segment*.

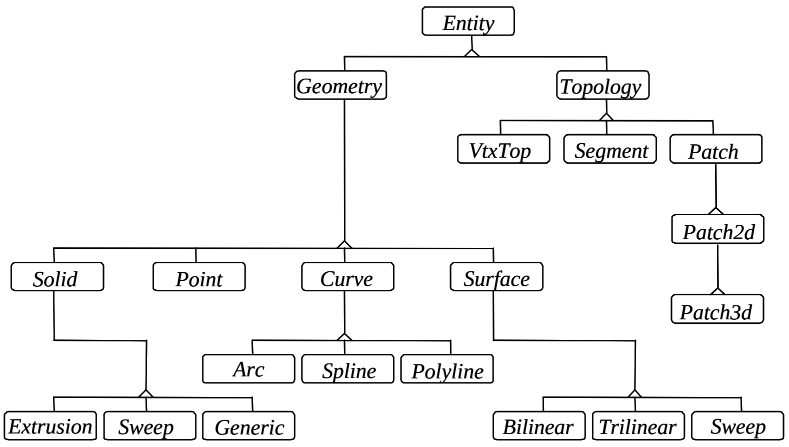

Fig. 6. Modeler's general OOP class organization.

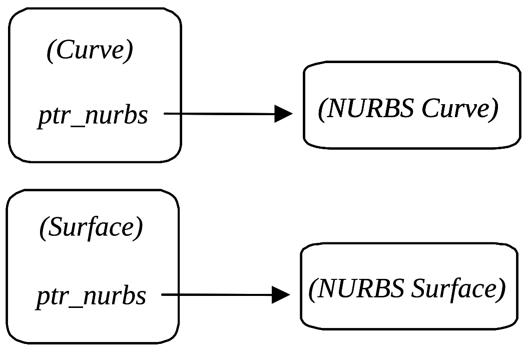

Fig. 8. Relationship between the *curve* and *surface* classes of the MG module and the NURBS module.

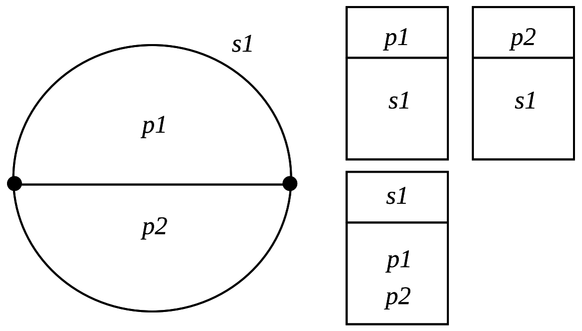

Fig. 9. Relationship *Surface-Patch2d.*

an object of class *Surface*. A *Surface* object contains a reference to its geometric representation, i.e., a pointer to a NURBS object (see Fig. 8), and a list of references to *Patch2d* objects that belong to it. Fig. 9 illustrates the relationship between surface *s1* and two corresponding *Patch2d* objects, *p1* and *p2*. Each *Patch2d* object contains topological information, which mainly consists of a list of *Segment* objects on its boundaries. In addition, a *Patch2d* object stores a surface finite-element mesh that is eventually generated on it. A *Patch2d* object also stores a list of interior surface patches. This allows, for example, the representation of interior loops in a surface patch, which may occur from surface intersection.

An object of the *Surface* class must have at least one *Patch2d* object. A *Surface* object also contains geometric information related to the process that was used for its interactive generation. The sub-classes of *Surface* class shown in Fig. 6 contain methods for surface generation. Currently, a surface may be generated from boundary curves by bilinear mapping, trilinear mapping or sweep. Therefore, each *Surface* object contains information about its creation method and a list of references to its generating curves (curves that were used for its creation).

A *Patch2d* object also contains references to adjacent *Patch3d* objects. An object in the *Patch3d* class is the topological representation of a solid region. A *Patch3d* object has a list of references to *Patch2d* objects that form its boundary and a pointer to the corresponding *Solid* object. In addition, a *Patch3d* object stores a solid finite-element mesh that is eventually generated in it.

The geometric version of the topological *Patch3d* class is the *Solid* class. A *Solid* object contains geometric information about the method used for its generation. Currently, there are three methods for solid generation: extrusion, sweep or generic generation [11]. A generic generation of a solid may be performed either explicitly by the user (by simply selecting surface patches on the boundary of the target solid) or automatically by the CGC representation when a solid region is detected. Therefore, each *Solid* object contains information about its creation method and a list of references to its generating curves and surfaces (curves and surfaces that were used for its creation).

An important issue addressed by this data-representation scheme is the support for automatic and adaptive mesh generation. As previously mentioned, finite-element mesh generation is an area of active research and is being treated by the authors elsewhere [10,11,15,28,29]. One of the goals of the present research group is the development of a complete system for geometric modeling and 3D finite-element adaptive simulation. The methodology for adaptive mesh generation has been tested in two dimensions [30] and is based on the refinement of each topological entity in its own parametric space. For example, mesh generation of a surface patch (*Patch2d* object), which is performed in its parametric space, is based on a previous refinement of its boundary segments in their own parametric spaces. Similarly, solid-mesh generation of a 3D region requires the previous refinement of its boundary surface patches.

The implementation of this mesh-generation methodology is accomplished in the present data structure by means of the concept of *use* of a topological entity by a surface. The concept of *use* in the MG data structure is quite different from its concept in the radial-edge data structure. Here, *use* simply means the geometric information of a certain entity in the parametric space of a surface, while in the radial-edge data structure *uses* are responsible for the main topological relationships. In MG's data structure, the above-described topological entities hold the adjacency relationships and maintain *use* entities to hold parametric coordinates. For example, a vertex that belongs to two adjacent surfaces has two *uses*. Fig. 10 illustrates this example. The single *use* of topological vertex $v1$ refers to surface $s1$, with parametric coordinates $u = 0$ and $v = 0$. Topological vertex $v5$ has two *uses*, one to $s1$, with parametric coordinates $u = 1$ and $v = 1$, and the other to $s2$, with parametric coordinates $u = 1$ and $v = 0$. The same idea

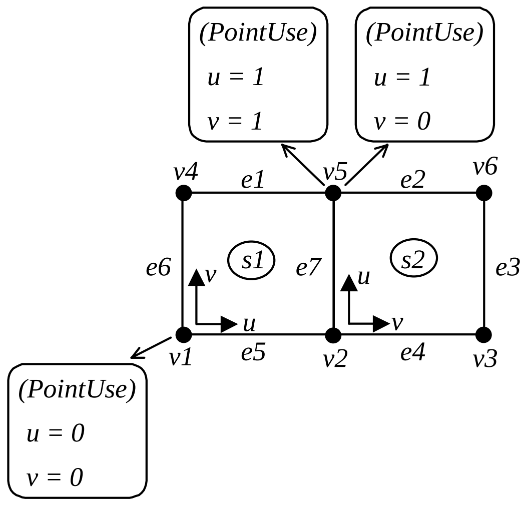

Fig. 10. *Uses* of vertices in two adjacent surfaces.

is adopted for segments. In this case, each *use* of a segment has the parametric coordinates of its end points.

Therefore, a *VtxTop* object has a list of *uses* associated to it in addition to the other previously described topological information. The *use* of a vertex is an object of the *VtxTopUse* class. A *VtxTopUse* object has a reference to its corresponding *VtxTop* object and another reference to the corresponding *Surface* object. A *VtxTopUse* object also contains two variables to store the parametric coordinates of a point on a surface.

Similarly, a *Segment* object has a list of *uses*, which are objects of class *SegmentUse*. A *SegmentUse* object has a reference to the corresponding *Segment* object and another reference to the corresponding *Surface* object. It also contains variables to hold parametric coordinates of a curve segment on a surface.

5. Implementation of NURBS in the CGC representation

The original CGC implementation treats only planar (polygonal) surface patches. However, one of the goals of this work is to consider CGC models with curved geometries, which can be achieved by incorporating NURBS's geometry.

The CGC module in this work is just used for region detection and receives as input data surface patches that only intersect each other at their boundaries. Therefore, the extension of the CGC module was relatively simple, because the CGC data structure does not depend on the underlying geometry, which means that it does not make any difference if an edge is considered as a straight line or a curve.

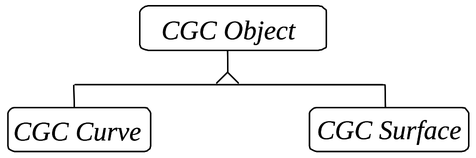

Fig. 11. New classes implemented in the CGC representation.

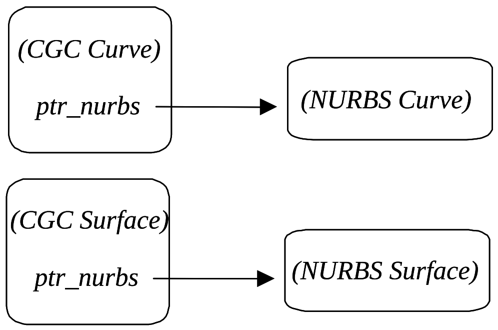

Fig. 12. Relationship between the *curve* and *surface* classes of the CGC representation and the NURBS library.

In the CGC representation, objects responsible for the definition of topological entities (*regions*, *faces*, *edges* and *vertices*) point to objects that describe their associated geometry and attributes (objects of classes *RegionAttr*, *FaceAttr*, *EdgeAttr* and *VertexAttr*). In addition, CGC implements another class called *GeomPck*, that contains all geometric algorithms necessary for region detection [25]. For example, the *GeomPck* class has algorithms for calculating the volume of a certain region, computing the location of a point with respect to a region, or computing the tangent vector at a face boundary. These methods deal with geometric information associated to each topological entity. Therefore, modifications caused by the addition of a new geometry type in the CGC implementation are limited to a small group of classes that define and manipulate the geometry of a model.

FaceAttr and *EdgeAttr* objects hold the geometric description of *faces* and *edges*. This description is now stored in two new classes, *CGC Surface* and *CGC Curve*, as can be seen in Fig. 11. Since the previous implementation of the CGC module treated only planar-polygonal faces, there was no need for these classes. In the new version, the methods in these classes obtain geometric information, from the NURBS module, that are necessary for region detection. The connection between the CGC representation and the NURBS module is illustrated in Fig. 12. *CGC Surface* and *CGC Curve* objects contain references to their corresponding NURBS objects.

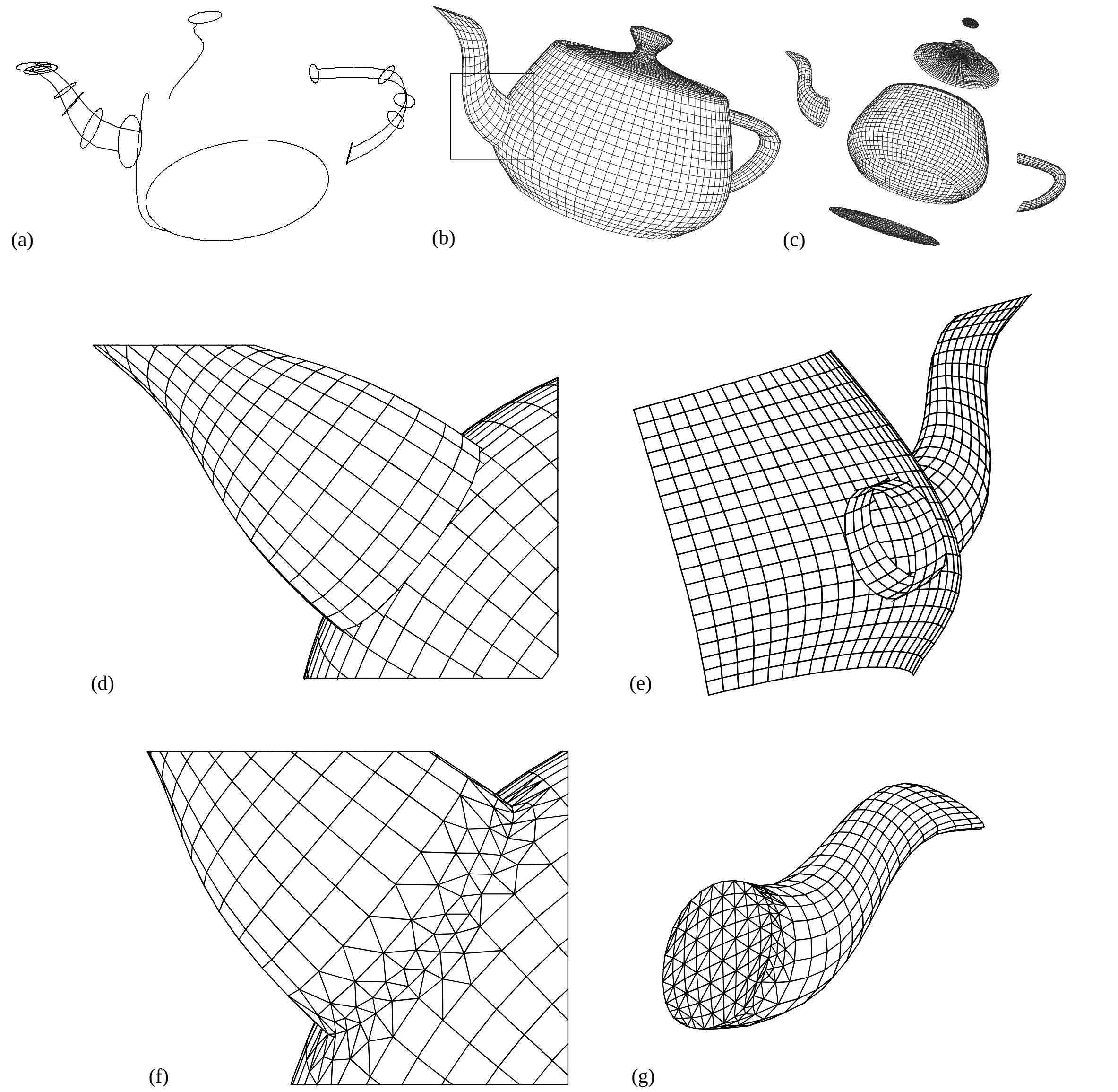

Fig. 13. Teapot modeling: (a) primitive curves of the teapot model; (b) surface patches of the teapot model; (c) exploded view of the teapot model; (d) detail of the surface finite element mesh of the spout and the body of the teapot model (before treating surface intersection); (e) detail of the original spout surface and teapot body; (f) detail of the surface finite element mesh resulting from the intersection of the spout and the body of the teapot model; (g) automatically detected region of the teapot spout after intersection with the body.

6. Application examples

This Section describes two application examples that demonstrate the efficiency of the proposed modeling methodology. The first one is the so-called Utah teapot and the second is a twelve-bladed turbine disk in which a full finite-element stress analysis is performed.

Fig. 13a shows the primitive curves that are used to generate the initial surface patches of the teapot model. These surface patches are shown in Fig. 13b. Fig. 13c depicts these patches in an exploded view. The surface patches of the teapot's body and cap are generated by rotational sweeping. The spout and the handle surfaces are modeled using Gordon patches. Planar taps are used to close off the model at the top and bottom. In fact, Figs. 13b and c show the initial surface finite-element meshes that are generated on the surface patches.

Figs. 13d–g are used to exemplify the surface-intersection and region-detection capabilities of the new version of the MG modeler. Figs. 13d and e show a

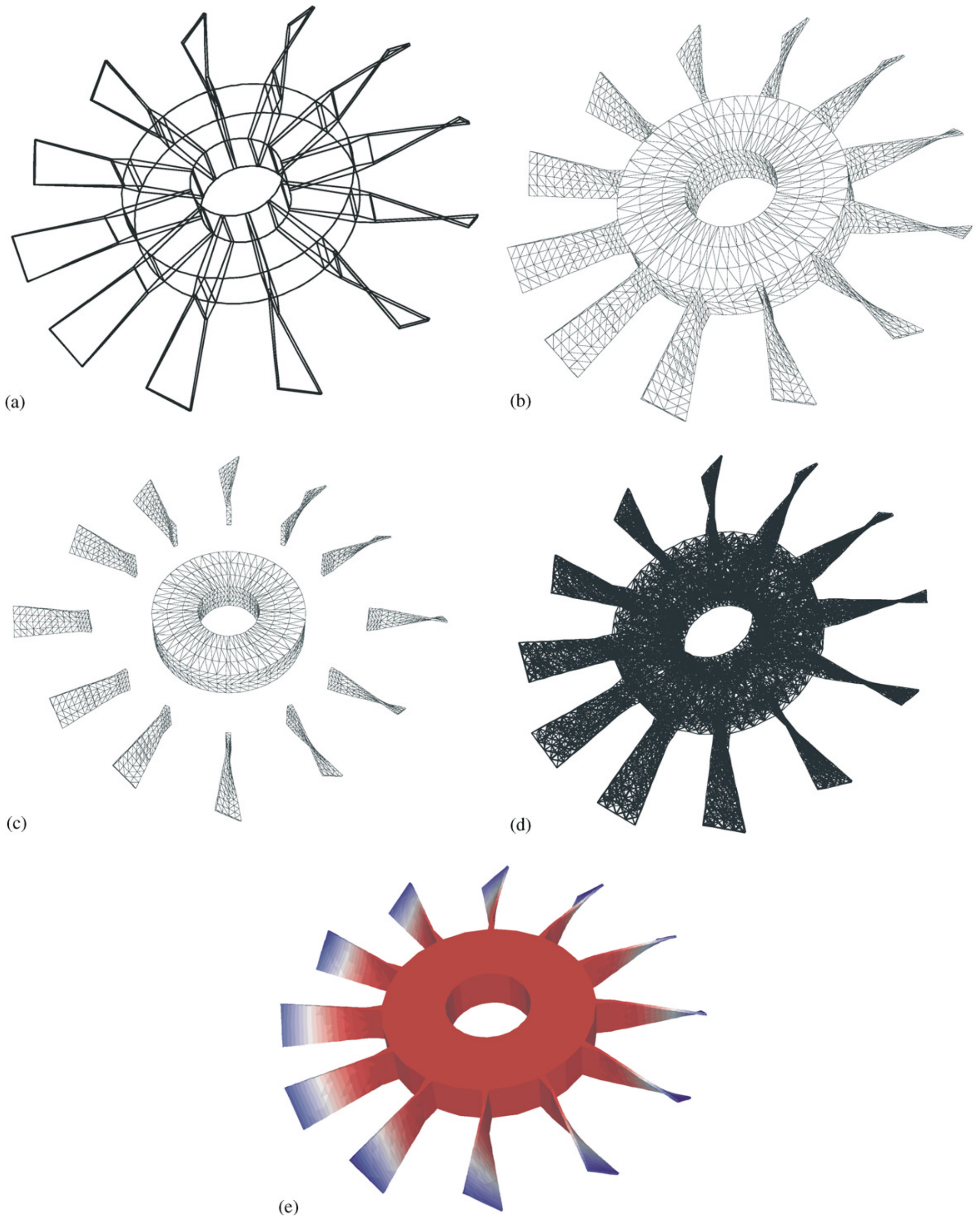

Fig. 14. Turbine disk modeling: (a) primitive curves of the turbine disk model; (b) surface patches of the turbine disk model; (c) exploded view of the turbine disk model; (d) solid finite element mesh of the turbine disk model; (e) contour plot of a stress component of the turbine disk model.

detail of the connection between the spout and the body of the teapot before treating surface intersection. In Fig. 13e, one can observe that the original spout surface goes inside the teapot's body.

Fig. 13f depicts a detail of the surface finite-element mesh resulting from the intersection of the teapot's spout and body. In this case, a local remeshing was performed, that is, only the elements close to the intersection curve are affected by the intersection. New nodes are always generated in the parametric space of each surface, which means that they lay exactly on the surfaces, including the nodes of the intersection curve.

The automatically detected region of the teapot's spout after intersection with the body is shown in Fig. 13g. One may observe that the portion of the spout's surface that was inside the body is eliminated. This is a trivial task in the current modeling environment because this portion is automatically detected as an independent patch. It may also be seen in Fig. 13g that the surface patch of the body that was closed off by the spout's surface was optionally meshed as a whole, instead of having its mesh changed locally.

The second example is an engineering problem of a finite-element stress analysis of a twelve-bladed turbine disk [31]. Fig. 14a shows the curves that are used to generate the initial surface patches. These surfaces, which are shown in Fig. 14b and c (exploded view), were modeled by bilinear Coons patches. All the blade regions were automatically detected. From this information, MG modeler generates a solid finite-element mesh independently in each region using J-Mesh algorithm [11]. The MG modeler converts the meshes associated to each region into a consistent unique mesh used as input data for a finite-element analysis program. A contour plot of a stress component resulting from the finite-element analysis is shown in Fig. 14d.

7. Conclusions

This work has proposed a modeling methodology focused on two aspects of the 3D finite-element method simulation. The first aspect refers to non-manifold geometric modeling with automatic region detection. In this context, a multi-region representation, called CGC representation [13] was generated by the insertion of curved surface patches. The second aspect is to provide support for automatic finite-element mesh generation. This methodology has been implemented in an existing finite-element modeler called MG [10].

This paper has described an OOP class organization of a new version of MG that, while maintaining its simple and efficient user interface, provides capabilities for automatic region detection and finite-element mesh generation for models with curved surface patches represented by NURBS [26].

A hybrid approach was adopted in which a full CGC representation of the model is not maintained in each step of the geometric modeling. Rather, the previous MG data structure was extended so that a CGC model could be created at any moment, when the user requires region detection (however, the CGC module requires that surface intersections be performed in a previous stage).

Surface intersection, which was inherited from the previous version of MG, is based on a procedure in which existing surface meshes are converted into an auxiliary topological data structure that allows the efficient construction of trimming curves [10]. This auxiliary topological information is also used to locally remesh the intersecting surface meshes. The surface-intersection algorithm generates elements resulting from intersections with improved geometric quality, suitable for finite-element analysis [10].

In addition to the local remeshing resulting from surface intersection, MG's new modeling methodology allows an overall remeshing of the model in a consistent way. The CGC "topological engine" provides consistency between topological and geometric entities of a non-manifold multi-region model. The complete topological and geometrical information allows the mesh generation of any surface patch or solid region independently or globally.

Solid-mesh generation is based on an a priori refinement of curves and surfaces [11]. Surface-mesh generation is performed in the parametric space of each surface. The implementation of this mesh-generation methodology was accomplished by the creation of two new topological entities, *Segment* and *Patch2d*, and by means of the concept of *use* of a topological entity by a surface. The *use* entities were added to the MG data structure to hold parametric coordinates of a vertex or a curve segment (which might correspond to a trimming curve) on a surface.

Application examples have demonstrated the efficiency of the proposed modeling methodology. Current developments are related to incorporating quadrilateral surface-mesh and hexahedral solid-mesh generation algorithms for arbitrary domains. The final goal is to develop a system for geometric modeling and adaptive 3D FEM simulation.

Acknowledgements

The first author acknowledges a doctoral fellowship provided by CNPq. The authors acknowledge the financial support provided by agency CNPq/CTPetro, process no. 468447/2000-8, and by agency Finep/CTPetro, processes nos. 65999045400 and 650003600. The authors are grateful to Carolina Alfaro for the manuscript's proofreading. The present work has been developed in Tecgraf/PUC-Rio (Computer Graphics Technology Group).

References

[1] Zienkiewicz OC, Taylor RL. The finite element method, 5th ed., vols. 1, 2. Butterworth-Heinemann, 2000.

[2] Bathe KJ. Finite element procedures. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[3] Hoffmann CM. Geometric, and solid modeling: an introduction. Los Altos, CA: Morgan Kaufmann, 1989.

[4] Mäntylä M. An introduction to solid modeling computer. Rockville: Science Press, 1988.

[5] Shephard MS. The specification of physical attribute information for engineering analysis. Engineering with Computers 1988;4:145–55.

[6] Shephard MS, Finnigan PM. Integration of geometric modeling and advanced finite element pre-processing. Finite Element Analysis and Design 1988;4:147–62.

[7] ACIS 3D Geometric Modeler. http://www.spatial.com/products/3D/modeling/ACIS.html

[8] Parasolid—powering the digital enterprise. http://www.plmsolutions-eds.com/products/parasolid

[9] Pro/ENGINEER API Toolkit. http://www.ptc.com/products/proe/app_toolkit.htm

[10] Coelho LCG, Gattass M, Figueiredo LH. Intersecting and trimming parametric meshes on finite-element shells. International Journal for Numerical Methods in Engineering 2000;47(4):777–800.

[11] Cavalcante Neto JB, Wawrzynek PA, Carvalho MTM, Martha LF, Ingraffea AR. An Algorithm for three-dimensional mesh generation for arbitrary regions with cracks. Engineering with Computers 2001;17(1):75–91.

[12] Coelho LCG, Souza CS. Problems communication and geometric solutions in a 3D interface. Proceedings of the VIII Brazilian Symposium on Computer Graphics and Image Processing, São Carlos, Brazil, October 1995. p. 183–90.

[13] Cavalcanti PR, P, Carvalho, Martha LF. Non-manifold Modeling: an approach based on spatial subdivision. Computer-Aided Design 1997;29(3):209–20.

[14] Piegl L, Tiller W. The nurbs book, 2nd ed. Ottawa: Springer, 1999.

[15] Miranda ACO, Cavalcante Neto JB, Martha LF. An algorithm for two-dimensional mesh generation for arbitrary regions with cracks. Proceedings of the XII Brazilian Symposium on Computer Graphics and Image Processing, Campinas, Brazil, October 1999. p. 29–38.

[16] Requicha AG. Representations of solid objects: theory, methods and systems. ACM Computing Surveys 1980;12(4):437–64.

[17] Dobkins DP, Laszlo MJ. Primitives for the manipulation of three-dimensional subdvisions. Proceedings of the Third ACM Symposium on Computational Geometry, Waterloo, 1987. p. 86–99.

[18] Laszlo MJ. A data structure for manipulating three-dimensional subdivisions. Ph.D. thesis 1987, Department of Computer Science, Princeton University, New Jersey, USA.

[19] Lienhardt P. Extension of the notion of map subdivisions of a three-dimensional space. In: STACS'88 Proceedings of the Cinquième Symposium sur les Aspects Thèoriques de L'Informatique, Bordeaux, France, 1988.

[20] Weiler K. Topological structures for geometric modeling, Ph.D. thesis, 1986, Rensselear Polytechnic Instiitute, Troy, New York, USA.

[21] Weiler K. The radial-edge structure: a topological representation for non-manifold geometric boundary representations. Geometric Modeling for CAD Applications 1988;3–36.

[22] Rossignac JR, O'Connor MA. A dimensional-independent model for pointsets with internal structures and incomplete boundaries. Geometric Modeling for Product Engineering 1990;145–80.

[23] Rossignac JR, Requicha AG. Constructive non-regularized geometry. Computer Aided Design 1991;23(1): 21–32.

[24] Wu ST. A new combinatorial data structure for boundary representations. Computer and Graphics 1989;13(4): 477–86.

[25] Mello UT, Cavalcanti PR. A topologically based framework for simulating complex geological processes, IBM T.J. Watson Research Center Research Report {RC21339} 1998. Yorktown Heights, New York. p. 1–12.

[26] Lavoie P. NURBS++: the NURBS package—user's reference manual http://yukon.genie.uottawa.ca/~lavoie/software/nurbs

[27] Rumbaugh J. Object-oriented modeling and design. Englewood Cliffs, NJ: Prentice-Hall, 1991.

[28] Cavalcanti PR, Mello UT. Three-dimensional constrained {delaunay} triangulation: a minimalist approach. Proceedings of the Eighth International Meshing Roundtable, South Lake Tahoe, California, USA, 10–13 October, 1999. California: Sandia National Laboratories, 1999. p. 119–29.

[29] Mello UT, Cavalcanti PR. A point creation strategy for mesh generation using crystal lattices as templates. Proceedings of the Ninth International Meshing Roundtable, New Orleans, LA, USA, 2–5 October, 2000. Loiusiana: Sandia National Laboratories, 2000. p. 253–61.

[30] Paulino GH, Menezes IFM, Cavalcante Neto JB, Martha LF. A methodology for adaptive finite element analysis: towards an integrated computational environment. Computational Mechanics 1999;23(5/6):361–88.

[31] Hsieh SH, Paulino GH, Abel JF. Evaluation of automatic domain partitioning algorithms, for parallel finite element analysis. International Journal for Numerical Methods in Engineering 1997;40:1025–51.