

## A B-Rep data structure and object GUI programming to implement 2D boundary elements

Gilberto Gomes<sup>a</sup> , Marcos A. M. Noronha<sup>b</sup>

<sup>a</sup>Civil Engineering Department, University of Brasilia, Campus Darcy Ribeiro, 70910-900, Brasilia - DF Brazil

<sup>b</sup>Civil Engineering Department, Federal University of Santa Catarina  
Campus Universitário, 88010-970, Florianópolis - SC Brasil

### ABSTRACT

One of the main advantages of the Boundary Elements Methods (BEM) is the inherent feature of a Boundary Representation scheme (B-Rep), since it represents the geometry and physical properties of a model using just its boundary. This characteristic makes the representation and manipulation of the model geometry an easy task, also allowing the boundary element mesh generation in a straightforward and simple way. Even though a simple pre-processor program may suffice for generating most two-dimensional boundary element models, in some special cases the modeling procedure represents a cumbersome task, requiring pre-processor programs with enhanced characteristics. The special cases considered in this work are non-trivial models, such as geometries with infinite or semi-infinite domains, multiple regions, disconnected regions, cracks and holes. To handle with the modeling task of these special models, the present development proposes a scheme based on the well-known Half-Edge data structure (HED). It also considers the use of Object-Oriented Programming (OOP) to implement a graphical user interface (GUI) with the proposed data structure. The results of step-by-step generation of some special two-dimensional models in the implemented GUI illustrate the main characteristics of the proposed scheme.

**Keywords:** Geometric Modeling, HED Data Structure, BEM, OOP, GUIs.

© 2012 Darbose. All rights reserved.

### 1. Introduction

Currently, the numerical methods used in Computational Mechanics to analyze physical and engineering problems are quite well established. On the other hand, the task of representing and manipulating a generic geometric model with arbitrary boundary conditions and physical attributes is one of the most critical in practical engineering analysis.

In recent years, among the various existing numerical methods, the Boundary Element Method [2] (BEM) has emerged as a very effective tool to solve a range of engineering problems. The BEM can be applied in almost all types of analysis involving linear elliptic differential equations, providing results with good accuracy. Among the many applications of the BEM are, for example, problems with stress concentration, cracks, or whose domain is infinite or semi-infinite, as required in some geotechnical applications.

The great advantage of the BEM is the representation of the model geometry using only a discretization of its boundary. This facilitates the implementation of the algorithms for geometric modeling and mesh generation, also providing a greater versatility to them. Another feature is the possibility of using discontinuous elements, which can also further improve the discretization process of the models.

With the accelerated development of computer graphics technology in the last two decades, geometric modeling has emerged as an important tool in engineering analysis. In the geometric modeling literature, there are several

---

*Email addresses:* [ggomes@unb.br](mailto:ggomes@unb.br) (Gilberto Gomes), [marcos.noronha@ufsc.br](mailto:marcos.noronha@ufsc.br) (Marcos A. M. Noronha)

models of data structures for the geometric representation and manipulating of arbitrary objects. These data structures are responsible for storing topological relationships in an efficient and low cost manner. The three main data structures used in geometric modeling are the winged-edge [1], the half-edge (HED) [6] and the radial-edge [10, 11] representations.

This work describes the adaptation and implementation of the HED data structure to store and represent the dynamic modeling and the generic manipulation of two-dimensional models of boundary elements containing special features such as infinite or semi-infinite regions, multiple regions and disjointed, disconnected edges and holes (Fig. 1). The proposed framework represents the geometric and topological information in a efficient, organized and simplified way, while guaranteeing the consistency of the step-by-step generation of the models. The proposed scheme is based on the HED data structure developed by Mantyla, but with some modifications in order to make the topological relationships more consistent with the properties of planar models.

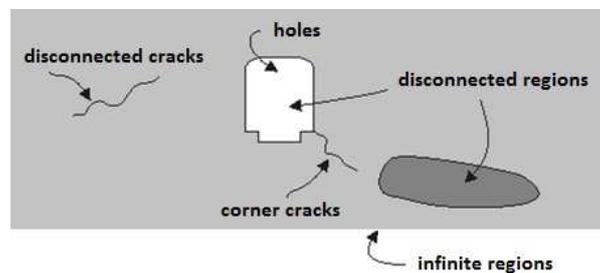


Figure 1: Special characteristics of the 2D models considered in this work.

## 2. Geometric Modeling and BEM Models

The technology of geometric modeling to represent three-dimensional objects has evolved from the wireframe representation to the polyhedral representation and to the most recent techniques of solid representation. In fact, these representations may be further developed into different modeling techniques as, for example, the reduction-decomposition models, the Constructive Solid Geometry (CSG) and the Boundary Representation model (B-Rep). Each one of these representations involves large amounts of information about the model geometry and topology, adding different levels of functionality and flexibility. Due to their particular features, each representation has a more appropriate application.

Among the aforementioned representation models, the B-Rep model stands as one of the most widely used tool [5]. In this model the surfaces of the model boundary (with its edges and vertices) are explicitly represented, so that the outline of a three-dimensional solid is based on a two-dimensional surface represented as a collection of faces. On the other hand, the outline of these faces is based on the one-dimensional curves that define its contours. B-Rep models usually represent faces in terms of an ordered graph data structure with a set of nodes and links, which enable many alternatives to represent the geometry and topology of a model boundary.

Most solid modeling systems based on a Boundary Representation model are an orientable “2-manifold“. A “2-manifold“ is a surface where every point is topologically equivalent to an open disk. Considering that boundary element models are 2-manifold surfaces by definition [9], their generation process can benefit from a set of available high-level operators which enable the interactive step-by-step construction procedure. These operators also allow the automatic identification of some special features of the model such as cracks, holes, infinite regions, etc. Upon completion of the geometric modeling, modeling of a boundary element mesh consists of two steps:

1. Definition of physical properties (material properties, tractions and supports);
2. BEM discretization:

- number of divisions of each model edge;
- element type;
- discontinuous elements;
- crack elements;
- number of internal points.

In order to achieve the desired model representation, an incremental approach based on the set of operators for the selected data structure (HED) can be used to build it in a step-by-step manner. In this case, checking the consistency of the geometric model represents an important procedure in the modeling process. Basically, this check consists of testing the intersection between edges and vertices of the model, and of removing the redundancies of vertices sharing the same coordinates.

### 3. Topology of Planar Graphs

The topological description of geometric models requires the representation of the adjacency relationships of their primitive entities. In general, the topology of planar models is formed from the combination of relationships between three primitive topological entities: vertices (V), edges (E) and faces (F) (Fig. 2a). The combination of these three primitives comprises nine fundamental relationships (Fig. 2b), describing the adjacency and arrangement of a specific group of entities in relation to a given entity.

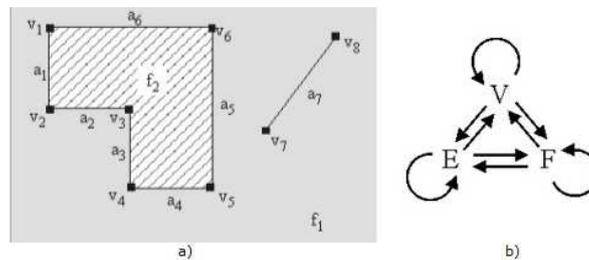


Figure 2: a) A planar graph and its topological elements; b) adjacency relations between the topological elements.

In addition to the geometric information contained in the vertices, edges and faces, a suitable data structure to represent the boundary of a model should consider its topological information. This information can be sourced from the explicit storage of some of the nine adjacency relationships (VV, VE, VF, EV, EE, EF, FV, FE, FF). On the other hand, the creation and manipulation of the desired models can be achieved through operations that consider the topological and geometric properties of arbitrary planar models. According to the theory of manipulation of planar models, a few set of operations is sufficient to describe and manipulate most of the planar models of practical interest. When handling B-Rep models, these operations are usually translated by entities called Euler Operators, according to the nomenclature introduced by Baumgart [1]. In the geometric modeling literature, the convention is usually to identify the different Euler Operators using mnemonic codes (Table 1) which indicate the actions and the objects involved in each operation.

Table 2 shows the basic set of Euler Operators sufficient for handling planar models. These operations allow the construction of arbitrary planar models, and for each operation there is a corresponding inverse operation.

Among the properties of the planar models, the concept of duality is very useful. Any model plan can be associated with a model in which dual plane faces and vertices of the original model are now considered as dual vertices and faces, respectively (Fig. 3). Therefore, if two faces are adjacent in the original model, the corresponding vertices in the dual model should be linked by a dual edge.

Table 1: Mnemonic Codes

Action	Object
	V - vertices
M - make	E - edge
K - kill	F - face
S - split	S - model
J - join	H - hole
	R - cycle

Table 2: Euler operators

Direct	Inverse
MVFS	KVFS
MEV	KEV
MEF	KEF
KEMR	MEKR
KFMRH	MFKRH

The model theory guarantees that all the properties of a model are preserved in its dual model. Furthermore, the duality implies that faces and vertices of a model have similar topological properties. Also, the edges of a model must also relate to its vertices and faces in a similar manner. These observations were the basis for the establishment of the data structure used in this work, which will be presented in the following section

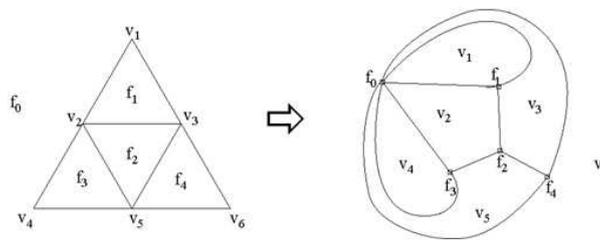


Figure 3: A planar graph and its dual.

#### 4. The Proposed Data Structure

The geometric and topological modeling considered in this paper is based on the data structure known as Half-Edges (HED) proposed by Mantyla [6] and which can be classified as a B-Rep approach. The HED data assigns two half-edges for each edge of the original model. The half-edges allow the representation of the topology of planar models in a convenient form, and the half-edges are chained together in a doubly linked list, so that the identification of the outline of a face becomes a trivial procedure. In this data structure, each half-edge has five links (Fig. 4a): a pointer to the corresponding edge; a pointer to the cycle of the selected face; a pointer to the vertex from which the half-edge originates; a pointer to the next half-edge in the face; and a pointer to the previous half-edge in the selected face. The last two pointers determine the sequence of half-edges that define the cycle of the selected face. On the other hand, the planar representation of the model is accomplished through the storage of a hierarchical list of vertices, half-edges, loops and faces of a model (Fig. 4b). Each of these entities is stored in a doubly linked list.

The data structure of Figure 4 was further developed by Gomes [3], with a subtle difference in the representation and storage of the faces. Namely, the proposed scheme has a global list of loops, for the convenience of dealing directly with the loops in a planar model instead of a list of global faces where each face contains a list of their local loops. Figure 5 illustrates the scheme developed by Gomes, which makes use of loops and pointers to edges.

In addition to the aforementioned difference, the present development also considered other modifications to the original structure of half-edge. These changes were based on the concept of duality, in order to provide similar relationships for the faces and vertices of a model. Two modifications were introduced in the original data structure, namely:

1. Elimination of the cycles in the model hierarchy, resulting in a reduced data structure (Fig. 6a). As a

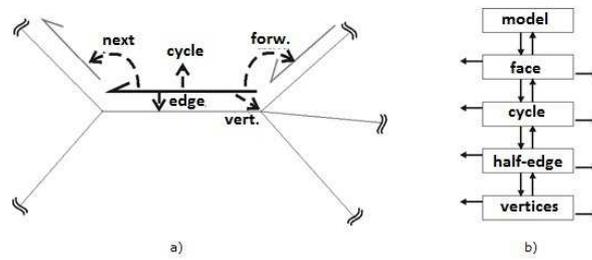


Figure 4: a) The HED data structure proposed by Mäntylä; b) hierarchical list of the entities of a model.

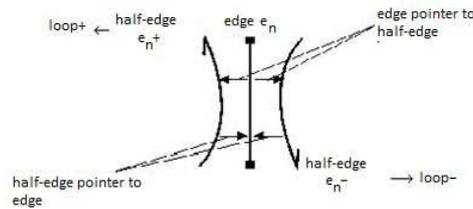


Figure 5: Scheme proposed by Gomes.

consequence, the analogy between the relationships of faces and vertices can be verified, since both entities are related directly with half-edges. On the other hand, it was necessary to use special edges, called nil edge, for the convenient representation of faces containing holes (Fig. 6b). Although these edges play a role similar to the cycles of the original HED data structure, their vertices are not identified. For implementation purposes, all nil edges of a face are stored immediately after the first half-edge that defines the face in question.

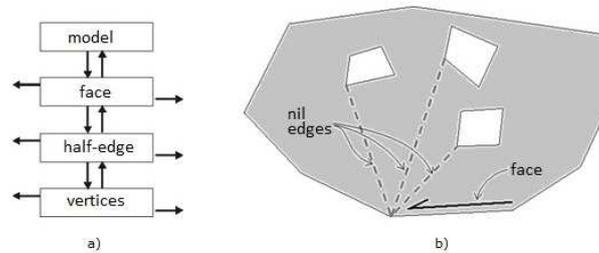


Figure 6: a) proposed hierarchical list of the entities of a model; b) use of nil edges to represent faces containing holes

2. Symmetrical relationship between half-edges. In the original model, each half-edge points to the next and previous half-edges in the cycle that defines the selected face. In this case, the concept of duality is not preserved, since the half-edges are directly related to cycles that define the outline of the faces. According to the concept of duality, there should be a similar relationship with respect to the vertices of the model. Alternatively, the proposed data structure relates each half-edge to the next half-edge of the cycle of a face and to the next half-edge of the cycle of half-edges connected to their common vertex (Fig. 7). As a consequence, the proposed data structure preserves the concept of duality, recognizing the entities directly adjacent to a vertex or to a face in a straightforward manner. The original data structure only recognizes entities directly adjacent to a face, and the entities adjacent to a vertex have to be obtained indirectly.

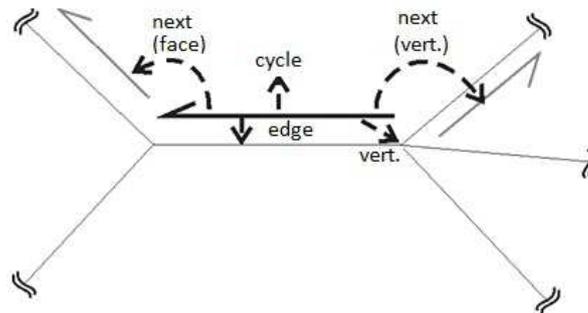


Figure 7: Proposed HED data structure.

## 5. Implementation

To implement the proposed data structure and its operators, the present work considered the development of a GUI for pre-processing based on the object-oriented programming (OOP). It also considered some basic computational geometry algorithms that allow to check, to validate and to manipulate the proposed data structure. As a programming tool that supports OOP, the implementation used the C++ language and an integrated development environment (IDE) based on Windows. The program developed was organized into five hierarchical levels of operation, as illustrated in Figure 8.

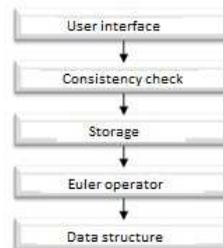


Figure 8: Hierarchical levels of operation of the program.

The first level allows the user to add vertices and edges, which are possible candidates to be introduced into the model. Before introducing the candidate vertices and edges, the second hierarchical level of the program (consistency check) performs the following tasks:

1. Verify whether the candidate already exists in the model;
2. Check for possible intersections with the primitives already existing in the model.

After the consistency check, the primitives can be stored on their lists (third level). The next operation level activates the topological Euler operators and some auxiliary topological functions. Finally, the lowest operation level of the program handles with the topological data structure and the attributes of each element.

### 5.1 GUI

The Graphical User Interface (GUI) developed in this work aims to allow the generation of two-dimensional models from a sequence of user-driven actions. The GUI consists of a graphical program, where actions are controlled by menus, toolbar buttons, a keyboard and a mouse. In this interface, the user provides basic elements

through the mouse. For each new added element, the model is updated to ensure geometric and topological consistency of the entire data structure. Figure 9 shows a sequential step to properly update a model when a new edge is added to it.

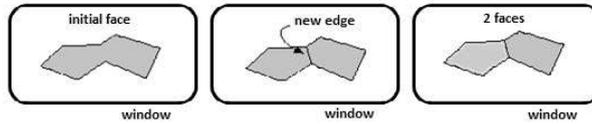


Figure 9: Example of updating a sequential model.

The main features of the proposed GUI are:

- incremental generation of two-dimensional models;
- automatic identification of the topology of the model (vertices, edges, faces and their relationships);
- manipulation of the model through common editing commands (add, select, remove, etc.);
- assign physical attributes to the geometry (boundary conditions and material properties);
- BEM mesh generation in a straightforward manner.

### 5.1.1 Event Handling

The actions on the proposed GUI are handled with the concept of events and listeners. An action requires an object of a given class and an action-event-firing component such as the toolbar or mouse buttons. All actions are implemented in the class View of the proposed GUI implementation (CBEMOO\_GIView), representing one of the most fundamental operating classes. Also, each action-event-firing button is associated with a triggering method, which registers the target object and defines the number and the type of actions to be performed. All these triggering methods must have the same format, an argument of class CPoint and a return parameter of type "void". The sequence of actions is controlled by an object of class CBEMOO\_GIView which executes the chain of actions "IdCurAction". Each time a tool button is activated, the processing method initializes the identifier of the current action. In the case of mouse events, all actions are performed by LButtonDown() object from class CBEMOO\_GIView. Figure 10 illustrates the Event handling scheme implemented in the proposed GUI.

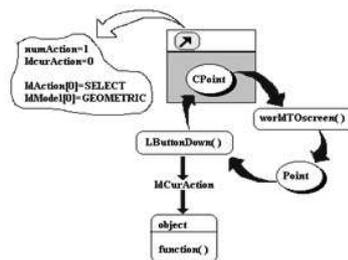


Figure 10: Event handling scheme of the proposed GUI.

### 5.2 OOP features

The proposed GUI was based on the concepts of Object Oriented Programming (OOP) and on the programming language C++ [4]. Table 3 presents the main OOP features and their respective concepts.

Table 3: Key Concepts of OOP

<i>Terms</i>	<i>Definitions</i>
<b>Object</b>	an instance of a particular class
<b>Class</b>	a template for objects to encapsulate data, functionality and behavior
<b>Member</b>	a variable or method (operation, service) within an object
<b>Instance</b>	an object of a particular class
<b>Encapsulation</b>	the binding of data and methods into an object
<b>Inheritance</b>	derivation of data and methods from a base class to a subclass, possibly redefining or adding new data and methods and creating a hierarchy of classes
<b>Polymorphism</b>	the ability of classes in a hierarchy to share names for methods that behave appropriately to the particular class for which they were designed
<b>Constructor/destructor</b>	methods to create/eliminate objects

Figure 11 illustrates the class model of the proposed data structure and GUI of the present work. A more detailed view of this model, with the main classes and methods names written in C++, can be seen in the Appendix.

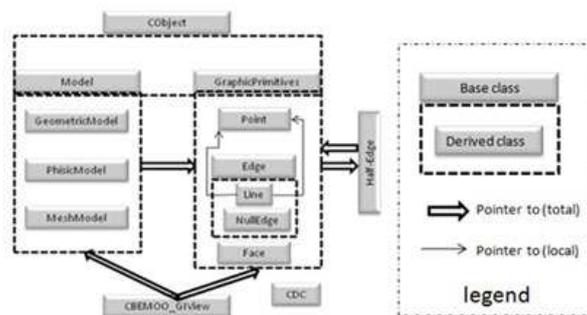


Figure 11: Class model for the proposed data structure and GUI.

### 5.3 Storage of Graphics Primitives

The storage of the set of graphics primitives (vertices, edges and faces) used circular doubly linked lists (CDLL). Basically, each node in a CDLL points to the next node and to the previous node of the list. Insertion or removal of nodes in a CDLL requires simple changes of the links of a given node.

The C++ language has a special type of data, the so-called pointer, which can be used as the concept of a link to connect the nodes of a CDLL. Also, the first (or the last) node of a CDLL corresponds to the head, the entity which allows to access the list. Figure 12a illustrates the representation of a CDLL and its head (its point of access). Adding a new node to end of a CDLL is a relatively simple task, as Figure 12b illustrates.

Figure 13 illustrates the removal of the third element of a CDLL.

### 5.4 Graphics Algorithms

The graphics algorithms used to develop the proposed GUI are intended to ensure the geometrical model consistency and to allow the selection and manipulation of geometric primitives for editing purposes. Table 4 contains the main algorithms and their purpose.

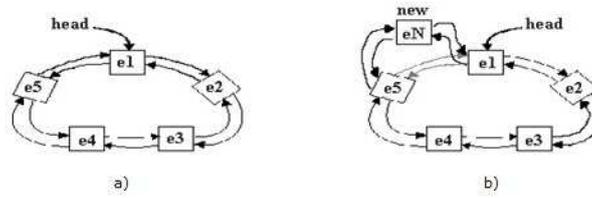


Figure 12: a) circular doubly linked list; b) adding a new element to the list.

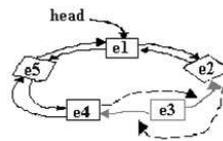


Figure 13: Removing the third element of the list.

## 6. EXAMPLES

The following examples validate the proposed data structure and the main features of the GUI, highlighting the pre-processing and interaction procedures to the incremental generation of a BEM model, namely:

- a) Generation of geometric models: Figure 14a depicts the main elements of a geometric model with special features, as disconnected regions and edges, representing the holes and cracks of a model, respectively;
- b) Verification of consistency after intersection of elements: Figure 14b depicts the updating process of the model when a new edge intersects it, creating a new face. The new faces are represented in dark color

The GUI allows the fast generation of a model (Fig. 16a). Due to its selection tools, an edge can be selected, opening a dialog window to define Neumann boundary conditions. In this case, the user can define the direction and value of a load (Fig. 16b).

For the Dirichlet boundary conditions, the GUI allows selection of nodes or edges and the user can assign prescribed displacements (Fig. 17a). For the mesh generation, the user has to select an edge or face to define the elements type and number (Fig. 17b).

Next, the user can define the internal points manually or automatically (Fig. 18a). At any moment, the user can select a region to define its material properties using a dialog window to define the material color, Young's modulus and Poisson's ratio (Fig. 18b).

Finally, a neutral file can be created (in ASCII format) to store all data for later use by a BEM analysis program, as illustrated in Figure 19.

Table 4: Main algorithms used to implement the proposed GUI.

<i>Algorithm</i>	<i>Purpose</i>
<b>Area attraction</b>	perform the selection of primitive vertices and edges
<b>Intersection thread [7]</b>	perform intersection primitives like edges
<b>Ray crossing [8]</b>	check intersection of primitive vertices and faces
<b>Sector area</b>	supervisor to perform the half-edges of each face
<b>Angular sorting</b>	perform multiple edges connected to a vertex

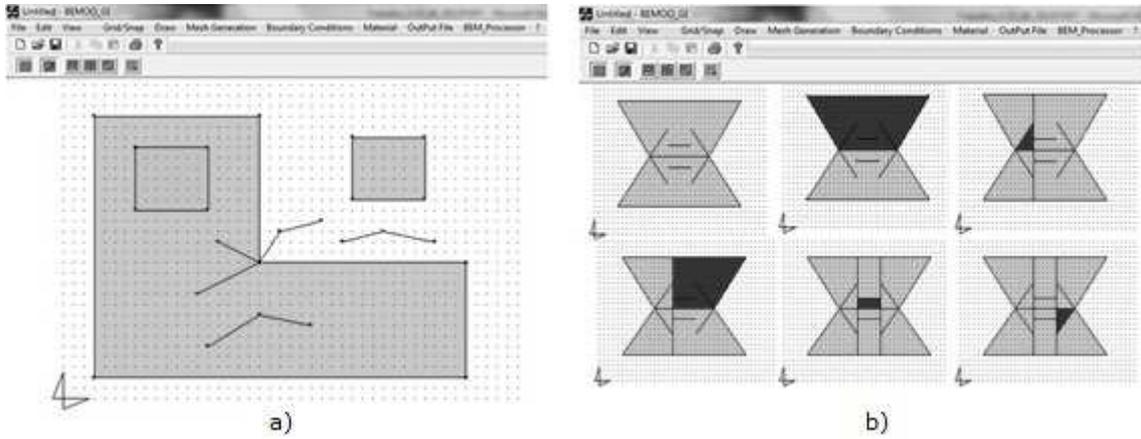


Figure 14: a) Main features of the model; b) Intersections and creation of new faces.

c) Application: Figure 15 illustrates an irregular plate ( $a = 50$  mm,  $b = 20$  mm) loaded at the top ( $t = 110$  MPa), with Young's modulus  $E = 200$  GPa and Poisson's ratio  $\nu = 0.25$ .

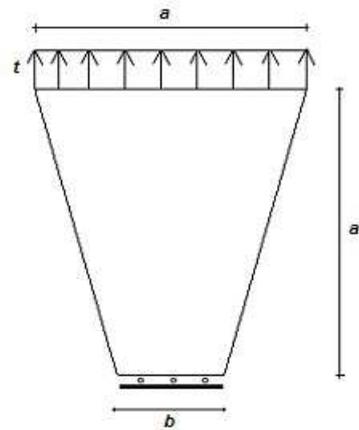


Figure 15: Physical model.

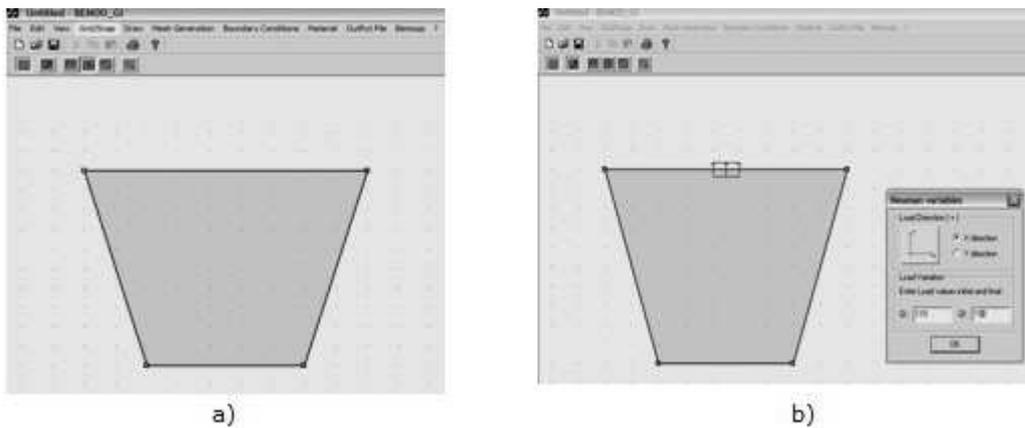


Figure 16: a) Generation of geometric model, b) defining Neumann boundary conditions.

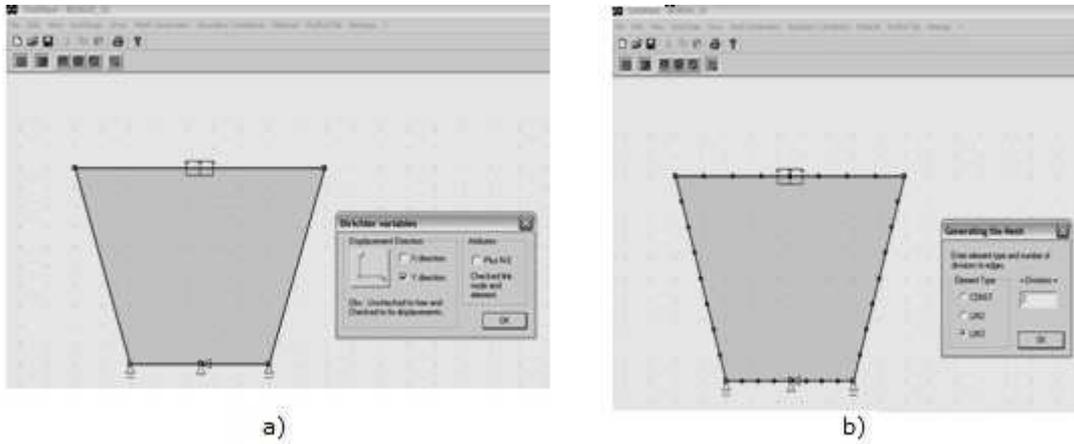


Figure 17: a) Defining Dirichlet boundary conditions; b) BEM mesh generation.

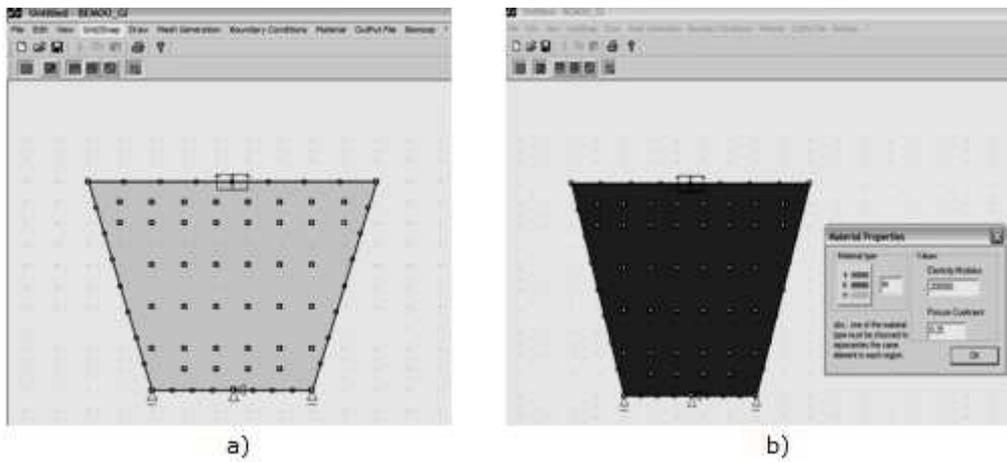


Figure 18: a) Generation of internal points; b) application of material properties.

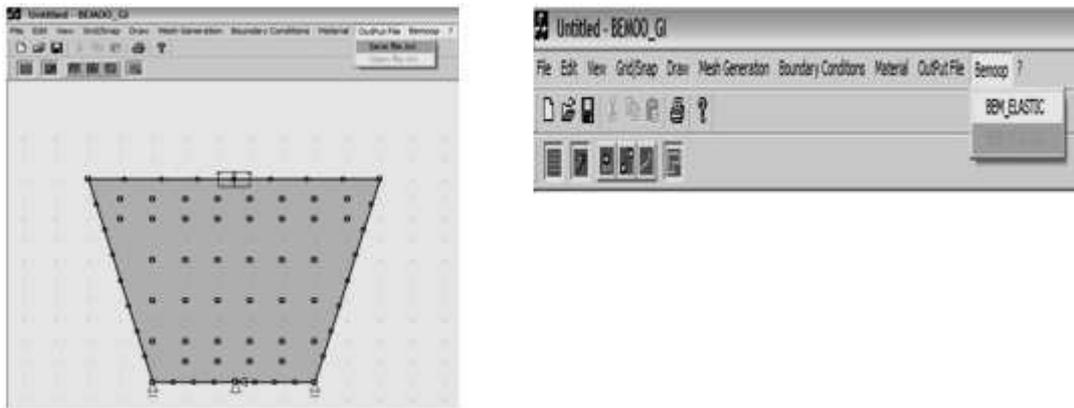


Figure 19: Generating an output data file.

## 7. Final Remark

This work showed the use of an alternative data structure based on the concept of duality of planar models for BEM modeling of two-dimensional problems. The proposed framework proved to be consistent with the dual character of planar models, providing topological information for vertices and faces in a symmetric fashion.

Unlike the conventional half-edge data structure proposed by Mantyla, the proposed structure allows a simpler face inclusion, as it only requires pointers to the next half-edge of the face. In addition, the proposed nil edges also add more functionality to the data structure, since they are similar to the cycles of a face. Remarkably, a specific entity for storing them was not necessary.

Regarding the storage of the graphics primitives, the use of circular doubly linked lists proved advantageous, since these lists do not require a pre-defined size, being limited only by the available computer memory.

The implementation of the proposed data structure and GUI with an OOP approach provided a successful platform for the dynamic and incremental features required by the generation of the BEM models. Another advantageous feature of the OOP is the reusability of existing content. This enhances the functionality of a program in a straightforward manner. For example, curved edges could be easily implemented by simply creating a class derived from class Edge.

The applicability of GUIs with robust data structures has proved important not only for numerical analysis of engineering problems with the BEM, but also for various other applications. Therefore, the proposed framework is likely to be helpful and lead to performance improvements in areas where advanced analysis tools are required.

## References

- [1] Baumgart, B. G., *A Polyhedron Representation for Computer Vision*, AFIPS Proc., 1975.
- [2] Brebbia, C. A., Dominguez J., *The Boundary Elements: an Introduction Course*, Computational Mechanics Publications, Southampton Boston, 1989.
- [3] Gomes, G., *A data structure for representing two dimensional boundary element models*, Masters thesis, University of Brasilia, Brazil, (2000). (In Portuguese)
- [4] Leinecker, R. C., T. Archer, *Windows 98 Programming Bible*, News Book, Portland, OR, 1997.
- [5] Lienhardt, P., *Topological Models for Boundary Representation: a Comparison with n-Dimensional Generalized Maps*, Research Report R90-10 Department d'Informatique, Université Louis Pasteur, Strasbourg, France, 1990.
- [6] Mäntylä, M., *An Introduction to Solid Modeling*, Computer Science Press, Rockville, Maryland, 1988.
- [7] O'Rourke, J., *Computational Geometry in C*, Cambridge University Press, 1998.
- [8] Preparata, F.P. & Shamos, M.I., *Computational Geometry: an Introduction*, New York, Springer-Verlag, 1985.

- [9] Wawrzynek, P. A., *Discrete Modeling of Crack Propagation: Theoretical Aspects and Implementation Issues in Two and Three Dimensions*, Ph.D. Dissertation, Cornell University, 1990.
- [10] Weiler, K. J., *The Radial-Edge Structure: a Topological Representation for Non-Manifold Geometric Boundary Representations*, Geometric Modeling for CAD Applications, North Holland, pp. 3-36, 1988.
- [11] Weiler, K. J., *Topological Structures for Geometric Modeling*, Ph.D. Dissertation, Rensselaer Polytechnic Institute, Troy, NY, Univ. Microfilms Intl., Ann Arbor, MI, 1986.

## Appendix

```

class GraphicPrimitives : public CObject
{
public:
    virtual void Draw(CView*)=0;
    virtual void Select(AuxPoint)=0;
    GraphicPrimitives *next, *prev;
    int selected;
    GraphicPrimitives();
    virtual ~(){}GraphicPrimitives();
};

class Edge : public GraphicPrimitives
{
public:
    virtual AuxPoint
        Intersection(Point *fp,Point *sp,int *flag)=0;
    HalfEdge *pEHE1;
    HalfEdge *pEHE2;
    Edge();
    virtual ~(){}Edge();
};

class Faces : public GraphicPrimitives
{
public:
    HalfEdge *pFHE;
    Faces();
    virtual ~(){}Faces();
    virtual void Draw(CView *pDV);
    virtual void Select(AuxPoint pt);
};
}

class Point : public GraphicPrimitives
{
public:
    double x, y;
    HalfEdge *pVHE;
    Point(const Point &paux);
    Point & operator=(const Point &paux);
    Point();
    virtual ~(){}Point();
    virtual void Draw(CView *pDV);
    virtual void Select(AuxPoint pt);
};

class Line : public Edge
{
public:
    Point *ps, *pe;
    Line();
    virtual ~(){}Line();
    virtual void Draw(CView *pDV);
    virtual void Select(AuxPoint pt);
    virtual AuxPoint
        Intersection(Point *fp,Point *sp,int *flag);
};

class HalfEdge
{
public:
    HalfEdge * Mate();
    Edge * pEdg;
    HalfEdge * pNHEF;
    HalfEdge * pNHEV;
    Faces * pHEF;
    Point * pHEV;
    HalfEdge();
    virtual ~(){}HalfEdge();
};
}

```