Development of a Python Application Aiming at the Teaching-learning Process of the Half-Edge Data Structure

Danilo S. Bomfim[1], Rodrigo L. Soares[1], Luiz F. Bez[2], Pedro C. F. Lopes[2], André M. B. Pereira[2], Luiz F. Martha[1]

[1]*Dept. of Civil and Evironmental Engineering, Pontifical Catholic University of Rio de Janeiro*
St. Marquês de São Viciente, 225 - Gávea, 22541-041, RJ / Rio de Janeiro, Brazil
dsbomfim2@hotmail.com, rodrigolucassoares@gmail.com, lfm@tecgraf.puc-rio.br
[2]*Institute of Computing, Fluminense Federal University*
Av. Gal. Milton Tavares de Souza, s/n - São Domingos, 24210-310, RJ/Niterói Brazil
luizf.bez@gmail.com, pedrocortez@id.uff.br, andremaues@gmail.com

Abstract. Solids modeling is an area of research and development with diverse applications spanning several fields such as mathematics, computer science and engineering. All these applications require the computational representation of physical objects as well as the storage of information related to the geometry and physical properties of these objects. In all these cases, solid modeling aims to produce geometric models that can be used to perform analyzes of physical phenomena. These geometric models store information describing the position, dimension, and shape of a physical object. In this way, geometric modeling aims to create computational representations of real objects so that it is possible to address all geometric and topological issues in a simple, fast, and efficient way. The representation and manipulation of geometric models properly requires the use of a data structure that allows to manage all the information necessary to describe such models. This data structure must have characteristics such as efficiency and economy in the use of memory Therefore, understanding how the data structure works is essential for researchers and students in the field. With this premise in mind, a Python application called Hetool was developed to help understand the Half-Edge data structure that is commonly used in many applications. The developed application aims to help teachers, researchers, and students during the teaching-learning process. Furthermore, the code developed in the application will be open to users.

Keywords: Half-Edge, Topological data structure, Solid modeling, Python educational software

1 Introduction

Solid modeling is a central area of research, widely used for simulations and analysis of physical phenomena, with numerous applications such as product design and manufacturing, electronic prototyping and robot programming. All these applications require the computational representation of real objects as well as the storage of information related to the geometry and physical properties of these objects. Various applications and analyzes are only possible due to the geometric modeling system that is present in many areas such as film production, automotive, aerospace and shipbuilding.

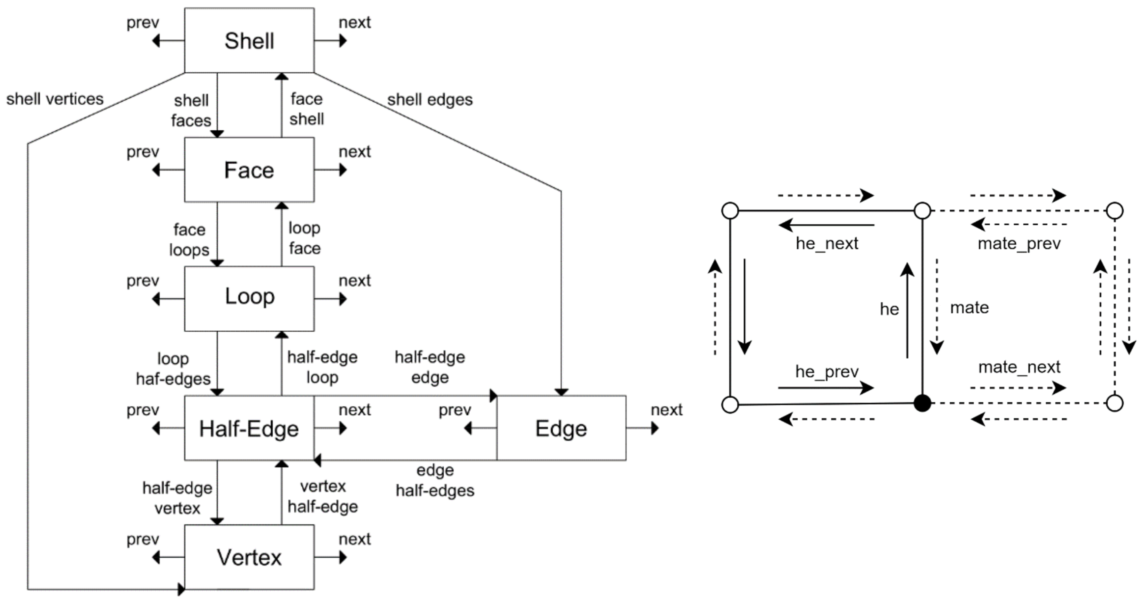
There are different ways to represent information from a geometric model. The most commonly used types of solid representation are boundary representation (B-Rep), cell decomposition and constructive solid geometry. A boundary representation (B-Rep) model describes the surface of a solid as a data structure composed of vertices, edges, and faces. B-Rep models can be built based on polygons, vertices or edges. The edge-based B-Rep model represents a face in terms of a closed sequence of edges, this sequence is usually called a loop. This edge-based model is widely used due to its simplicity and the advantages it offers regarding the handling and storage of topological information, especially information related to the adjacency of geometric elements.

The representation and manipulation of geometric models in an adequate and efficient way requires the use of a data structure that makes it possible to manage all the information necessary to describe such models. This data structure must have certain characteristics such as efficiency and economy in memory usage. The efficiency of a data structure is related to the effort required and the ability to interpret the received data properly. The memory use of a solid modeler must be one of the main factors to be considered in the elaboration of the data structure algorithm, as the indiscriminate use of memory can affect the efficiency of this modeler.

There are several topological data structures that have been developed for creating edge-based B-Rep models, some examples are: Winged-edge [1], Half-edge [2], Quad-edge [3], Radial Edge [4] and Dual Half-Edge [5]. The application developed in this work uses a data structure based on Half-Edge. In recent years, several works have been published, such as [6-9], using or optimizing topological data structures to solve problems in various fields of science. Therefore, understanding the functioning and logic of these topological structures is an essential process for advancing in this area and consequently directly influences the improvement of other research areas that use solid modeling.

1.1 The Half-Edge data structure

The Half-Edge data structure, developed by Mäntylä [3], is a variation of the Wing-edge data structure that allows for greater flexibility and uniqueness in the representation of the topology of B-Rep models. This structure essentially works through an abstract entity called a half-edge that stores most of the topological information present in this data structure. The half-edge entity originates from the subdivision of an edge into two half-edges that are oriented within a loop of a face with opposite directions. Most topological queries are performed using information stored in such an entity. The Half-Edge data structure has the following topological elements: shells (S), faces (F), loops (L), edges (E), half-edges (HE) and vertices (V). Figure 1.a presents the hierarchy of all these topological elements that are connected through doubly linked lists with identifiers for the previous and posterior element with respect to the reference entity.



(a) (b)

Figure 1. Half-Edge data structure

The Half-Edge data structure uses operations that modify it at the level of topological entities. These operations are performed by Euler operators which, according to Hoffman [10], can be defined as creators and modifiers of the topological data structure that have the ability to create or modify closed surfaces by adding or deleting faces, edges and vertices. By historical convention, Euler operations are traditionally named by a term of the form *mxky*, where *m* represents the expression *make* that is associated with the creation of a certain topological entity and *k* represents the expression *kill* that is associated with the destruction of a certain topological entity. In addition, other terms may appear such as *j* which stands for *join* and *s* which denotes *split*. The table below shows the operators implemented in the data structure developed, the meaning and the changes that each operator is able to make in the data structure. The inverse Euler operators exchange the signs of each numeric term in Table 1.

Table 1. Euler Operators

Operators	Meaning	S	F	L	E	V	HE	Inverse
MVFS	make a vertex, a face, and a shell	+1	+1	+2		+1	+1	KVFS
MVR	make a vertex ring			+1		+1	+1	KVR
MEV	make an edge and a vertex				+1	+1	+2	KEV
MEF	make an edge and a face		+1		+1		+2	KEF
MEKR	make an edge kill a vertex			-1	+1		+2	KEMR
MVSE	make a vertex split an edge				+1	+1	+2	KVJE

1.2 Objectives

This work proposes an open-source implementation of a two-dimensional solid modeler that uses a Half-Edge dynamic data structure. The developed tool aims, in a dynamic, friendly and intuitive way, to improve the teaching/learning process of the Half-Edge data structure, which is a well-known data structure that can be used in numerous applications involving solid modeling.

2 Methodology

The application code was written in Python following the guidelines of object-oriented programming. For the development of the graphic interface of the solid modeler, the Qt library was used. The representation and visualization of the models was performed through the use of the graphic library OpenGL (Open Graphics Library). The Half-Edge data structure was developed following the Model – View – Controller (MVC) pattern, as illustrated in Fig.2. The MVC pattern subdivides data structures into three main systems: a *controller* that manages user interactions with the data structure, a *model* that stores the data structure information, and a *view* that collects the model's information and transmits that data so that the modeled solid can be displayed on the screen. According to Ramnath [11], in contrast to other systems where these three functionalities are not grouped together, the MVC pattern helps to produce highly cohesive and more flexible modules. In the data structure developed, these three systems received the prefix He.

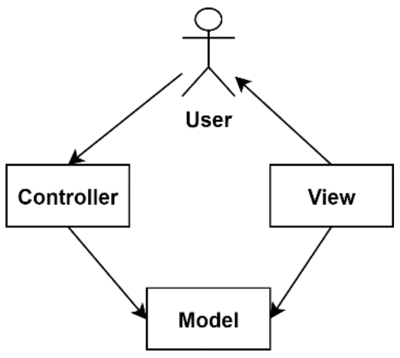


Figure 2. MVC pattern

2.1 Model

The model (*HeModel*) presents the passive function of receiving the data processed by the controller (*HeController*) and storing them. Thus, *HeModel* is the class responsible for storing and managing all topological and geometric information, allowing quick access to this data through lists. The model consists of a solid (*Shell*), which stores all topological information, and three lists that allow you to efficiently obtain all the geometric entities related to this solid. Thus, the model presents a direct relationship with the set of non-abstract topological elements, present in the data structure, which is formed by the *Shell*, *Face*, *Edge* and *Vertex* classes. In addition, *HeModel* is also related to the set of geometric entities of the data structure that is constituted by the *Patch*, *Point* and *Segment* classes. Each geometric element is associated with its respective topological entity. Figure 3 shows a summary of the *HeModel* class demonstrating the relationship of this class with the non-abstract topological entities and with the geometric elements present in the data structure.

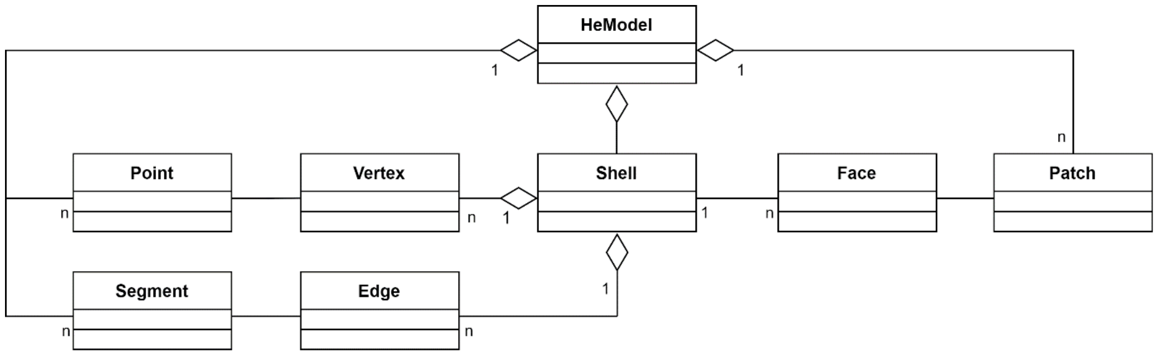


Figure 3. HeModel class diagram

2.2 View

The view (*HeView*) gets the model information in a specific format to be displayed on the screen as a graphical response to user interaction. In this way, the *HeView* class presents the role of collecting information about the geometric entities stored in *HeModel* and transmitting this collected data to *Canvas* (class created through the OpenGL graphics library) which renders this information so that the user has a visual feedback of your interaction with the app. *HeView* is only accessed by *Canvas*, so this class works as a bridge between the model and the class responsible for rendering the solid information on the screen.

2.3 Controller

The controller (*HeController*) performs all the user interaction processing with the data previously inserted in the model, modifying when necessary the information present in the *HeModel* class. The methods, from the *HeController* class, which are used to insert points and segments in the data structure, perform geometric and topological checks. These checks are necessary for the proper use of Euler operators. These methods were classified as high-level functions, as they perform global checks on the data structure.

Euler operators were implemented in a way that they act locally. These operators only perform necessary modifications in the data structure to add or remove new topological entities and because of that these operators were classified as low-level functions. In addition, the insertion or removal of topological and geometric entities in the model is performed through other operators called auxiliary operators that are created in high-level functions. Fig.4 illustrates a summary of the methods used to add a point or segment to the data structure.

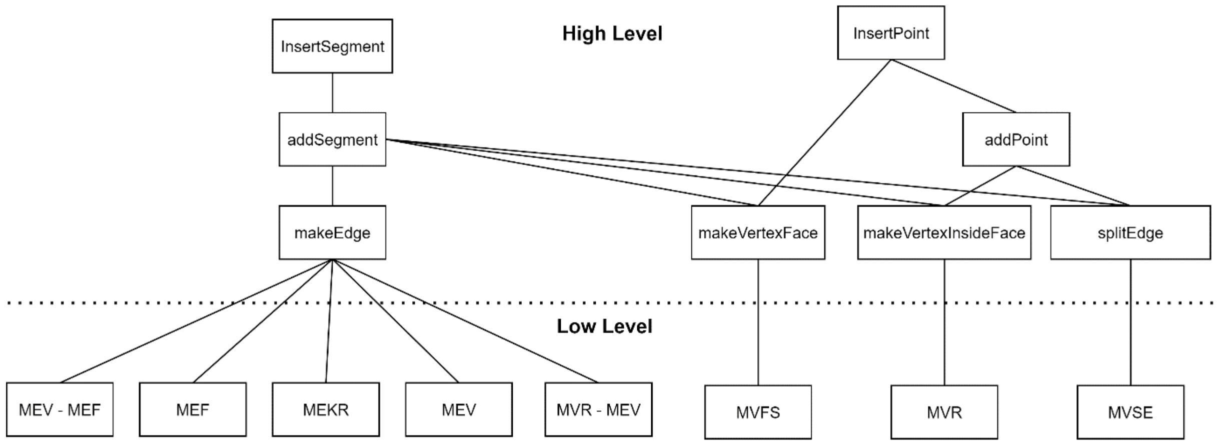


Figure 4. High-level and low-level functions

Each Euler operator is a class that has two methods in addition to its constructor. These methods were called *execute* and *unexecute*. The first method has the function of executing the operation and the last one has the role of undoing everything that was done by the first method, starting with the creation and execution of its respective inverse operator. This pattern was adopted to allow the user to make or undo any commands quickly and efficiently. Therefore, all operators as well as auxiliary operators were implemented following this criterion.

Auxiliary operators are a set of classes that fulfill specific functions within the data structure. For example, they have the ability to add or remove some entity from the model, transform a face into a hole or a hole into a face, and other functions that contribute to the proper functioning of the data structure. In addition, these auxiliary operators provide greater flexibility to the data structure by making it possible to undo or redo a varied set of user interactions with the application.

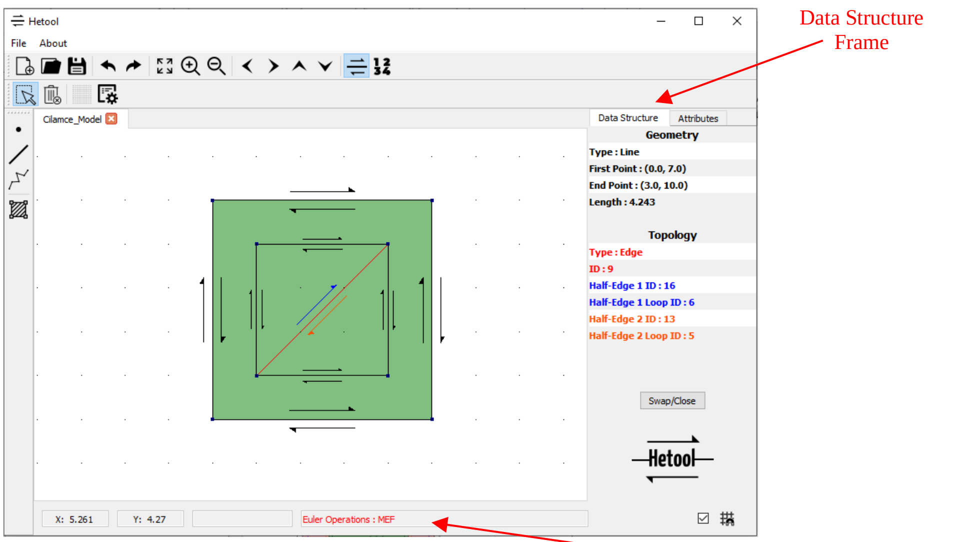
Every operation performed by the user is stored in lists within a class called *UndoRedo* that manages this information. One of the lists is made up of operations that can be undone and the other list is made up of operations that can be redone. Thus, all user interaction is stored in the form of operations in these two lists, making it possible to undo or redo commands during the modeling process.

3 Hetool Program

The developed application aims to improve the teaching-learning process of the Half-Edge data structure, making this learning more dynamic, friendly and attractive to the user. Hetool allows you to model several solids simultaneously, through multiple windows that can be easily added or removed, and save the information of these solids in a *json* file. The program's interface can be seen in Fig. 5, where all the functionalities offered by the application are presented, such as managing the window visualization limits, creating and deleting entities, saving and reading models. As can be seen in this figure, during the solid modeling process it is possible to visualize the half-edges of each edge, which enables the dynamic learning of the basic functioning of the data structure.

In addition, it is possible to select one of the entities (face, edge or vertex) and obtain the geometric and topological information about this element as can be seen in the bar, on the left in Fig. 5, named as data structure. Each topological element was assigned an ID as a form of identification.

An edge of the modeled solid was selected in Fig.5.a, where you can get the information about this entity that has been identified as edge 9. In the data structure bar, geometric information of this edge is presented, such as the coordinates of the starting and ending points and the length of this element. This bar also allows obtaining relevant topological information for a better understanding of the data structure, like the identification of the two half-edges belonging to the selected edge and the identification of the loops of these half-edges.

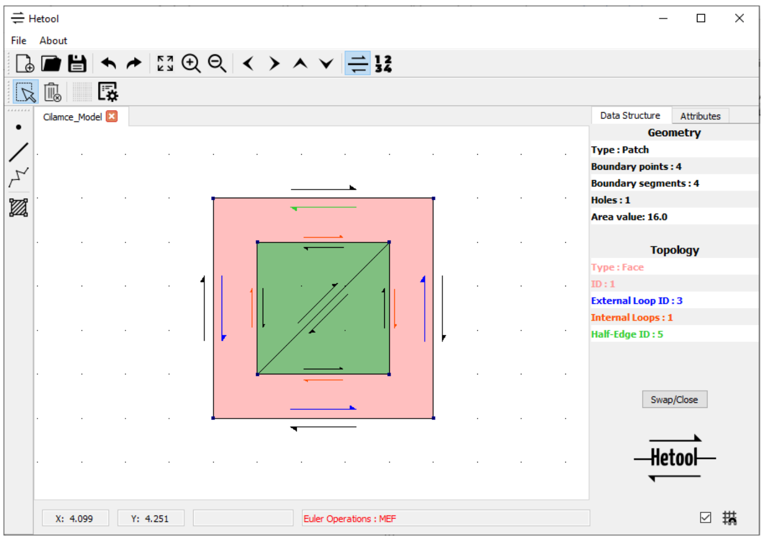


Geometry
Type : Line
First Point : (0.0, 7.0)
End Point : (3.0, 10.0)
Length : 4.243

Topology
Type : Edge
ID : 9
Half-Edge 1 ID : 16
Half-Edge 1 Loop ID : 6
Half-Edge 2 ID : 13
Half-Edge 2 Loop ID : 5

X: 5.261 Y: 4.27 Euler Operations : MEF

(a)

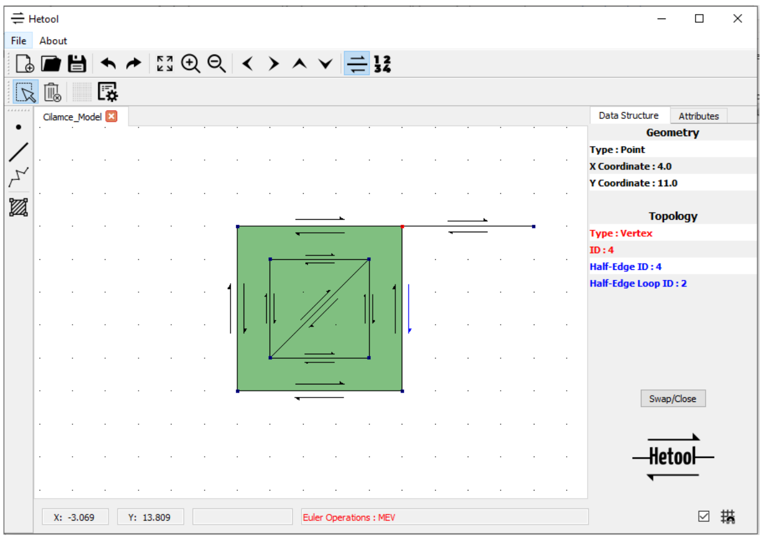


Geometry
Type : Patch
Boundary points : 4
Boundary segments : 4
Holes : 1
Area value: 16.0

Topology
Type : Face
ID : 1
External Loop ID : 3
Internal Loops : 1
Half-Edge ID : 5

X: 4.099 Y: 4.251 Euler Operations : MEF

(b)



Geometry
Type : Point
X Coordinate : 4.0
Y Coordinate : 11.0

Topology
Type : Vertex
ID : 4
Half-Edge ID : 4
Half-Edge Loop ID : 2

X: -3.069 Y: 13.809 Euler Operations : MEV

(c)

Figure 5. Hetool program

In Figure 5.b, it is possible to visualize the information about a selected face that was identified as face 1. In the data structure bar certain geometric information is presented, such as the face area, number of segments and points present in the contour external of the selected face. Also in this bar, the user can obtain topological data identifying the face's outer loop (highlighted in blue), number of inner loops (highlighted in orange) and the half-edge (highlighted in green) that is stored by face 1 and from where the navigation through the half-edges of this face begins.

Figure 5.c presents the information of a selected vertex that was identified as vertex 4. In the bar located to the left of the modeled solid, the user has access to the coordinates of the selected vertex and the topological information of the half-edge that is stored by the vertex (highlighted in blue).

In the lower corner of the application interface, there is a frame called Euler Operations that displays all the Euler operations used during the solid modeling process. This frame enhances the teaching/learning process of the basic concepts of Euler operators that are a fundamental part of the half-edge data structure. In Figures 5.a and 5.b, the frame shows the MEF operation that was the result of adding edge 9 (the selected edge in Fig 5.a) to the data structure. In figure 5.c, this same frame shows the MEV operation that was the result of adding a new edge connected to vertex 4 (vertex selected in this figure). In addition, all other Euler operators mentioned in Table 1 can also be displayed by this frame.

4 Conclusions

This work demonstrated the use of a Half-Edge data structure, which is still being improved, in an educational application. The Hetool application allows you to study/teach in a friendly, sequential and interactive way the basic concepts present in this data structure. The program makes it possible to immediately visualize the layout of the half-edges as well as the information of each element of the data structure. In addition, the platform organizes the topological information of each entity through a numerical identification, enabling the user to verify the various hierarchical relationships between the topological entities present in the data structure.

Acknowledgements. This work was carried out with the support of the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Financing Code 001.

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] Baumgart, B.G., "A polyhedron representation for computer vision". *Proceedings of the National Computer Conference (AFIPS '75),* Anaheim, CA, USA, 19–22 May 1975
[2] MÄNTYLÄ, M. *An Introduction to Solid Modeling Computer*. Science Press, Rockville, Maryland, 1988.
[3] GUIBAS, Leonidas; STOLFI, Jorge. "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams". *ACM Transactions on Graphics*, v. 4, n. 2, pp. 74-123, 1985.
[4] WEILER, K. *Topological Structures for Geometric Modeling*. Ph.D. Thesis, Rensselear Polytechnic Institute, Troy, N.Y., 1986.
[5] Boguslawski, P.; Gold, C. "The Dual Half-Edge- A Topological Primal/Dual Data Structure and Construction Operators for Modelling and Manipulating Cell Complexes". *ISPRS Int. J. Geo-Inf*, v. 19, n. 2, 2016.
[6] JANSSON, Erick Sven Vasconcelos. *Half-Edge Mesh Data Structure*. Linköping University, Swed, 2018.
[7] BRUNO, Hugo B. S. et al. "Interpretation of Density-Based Topology Optimization Results by Means of a Topological Data Structure". *Proceedings of the VI International Symposium on Solid Mechanics*, Joinville - Brazil, 2017
[8] KARIM, Hairi et al. The Potential of the 3D Dual Half-Edge (DHE) Data Structure for Integrated 2D-Space and Scale Modelling: A Review. *Advances in 3D Geoinformation: Lecture Notes in Geoinformation and Cartography*. Springer, Cham, 2017.
[9] CHATZIVASILEIADI, Aikaterini et al. "Characteristics of 3D Solid Modeling Software Libraries for Non-Manifold Modeling". *Computer-Aided Design and Applications*, v. 16, n. 3, pp. 496-518, 2019.
[10] HOFFMANN, C. M. *Geometric & Solid Modeling: An Introduction*. Purdue University, indiana, 1989.
[11] RAMNATH, Sarnath; DATHAN, Brahma. *Object-Oriented Analysis and Design*. Springer, London, 2011.