Journal of Biomedical Informatics 63 (2016) 195–211

Contents lists available at ScienceDirect

Journal of Biomedical Informatics

journal homepage: www.elsevier.com/locate/yjbin

Comparing efficient data structures to represent geometric models for three-dimensional virtual medical training

Helton H. Bíscaro *, Fátima L.S. Nunes, Jéssica dos Santos Oliveira, Gustavo R. Pereira

School of Arts, Sciences and Humanities, University of São Paulo, Av. Arlindo Bttio, 1000, São Paulo CEP: 03828-000 Brazil

ARTICLE INFO

Article history:
Received 25 November 2015
Revised 5 July 2016
Accepted 8 August 2016
Available online 24 August 2016

Keywords:
Data structures
Compact Half-Edge (CHE)
Mate-Face (MF)
Three-dimensional objects
Neighborhood relationships

ABSTRACT

Data structures have been explored for several domains of computer applications in order to ensure efficiency in the data store and retrieval. However, data structures can present different behavior depending on applications that they are being used. Three-dimensional interactive environments offered by techniques of Virtual Reality require operations of loading and manipulating objects in real time, where realism and response time are two important requirements. Efficient representation of geometrical models plays an important part so that the simulation may become real. In this paper, we present the implementation and the comparison of two topologically efficient data structures – Compact Half-Edge and Mate-Face – for the representation of objects for three-dimensional interactive environments. The structures have been tested at different conditions of processors and RAM memories. The results show that both these structures can be used in an efficient manner. Mate-Face structure has shown itself to be more efficient for the manipulation of neighborhood relationships and the Compact Half-Edge was more efficient for loading of the geometric models. We also evaluated the data structures embedded in applications of biopsy simulation using virtual reality, considering a deformation simulation method applied in virtual human organs. The results showed that their use allows the building of applications considering objects with high resolutions (number of vertices), without significant impact in the time spent in the simulation. Therefore, their use contributes for the construction of more realistic simulators.

1. Introduction

Data structures are widely explored, developed and improved to solve several problems in computer applications. Systems that demand processing large volumes of data in a short time are interesting problems to apply efficient data structures. One of these applications include three-dimensional (3D) medical training using Virtual Reality (VR) technology, which provides interactive and immersible systems involving users in a real-time computational simulation [1].

VR attempts to immerse the users senses so that the virtual representation of life is as close to reality as possible. The use of three-dimensionality and interaction in real time makes VR an even more attractive proposition for training and simulations in health area, once it gives the user the possibility of exploring and repeating several different procedures without any wear and material maintenance costs. Obtaining realism in this type of system requires the use of models with appropriate colors, textures and lighting, as well as high resolution.

These objects usually are composed by a large number of vertices and cells that are manipulated in real time, for example, to detect collisions and simulate deformations arising from user interaction. Methods for these and other functionalities work with meshes by manipulating sets of vertices, changing their positions within the Virtual Environment (VE), so that they can simulate changes that occurred in the objects considering their properties [2]. The computational costs of these iterative methods are normally high, especially in terms of processing time.

Considering the paradox between precision and computational cost, as well as the need for feedback in real time (essential in the case of interactive 3D environments applications), the study of Data Structures (DS) to enable the efficient storage and recovery of data for representing flexible objects is a key factor. When the subject is executing simple operations on static meshes there are efficient alternatives [3,4]. However, the problem complexity increases as we consider real time interaction, which is the problem that we propose to contribute in this article. In medical training applications where we need to perform deformation in

* Corresponding author.
E-mail addresses: heltonhb@usp.br (H.H. Bíscaro), fatima.nunes@usp.br (F.L.S. Nunes), jessica.santos.oliveira@usp.br (J. dos Santos Oliveira), gustavo.pereira@usp.br (G.R. Pereira).
URL: http://www5.each.usp.br (G.R. Pereira).

http://dx.doi.org/10.1016/j.jbi.2016.08.014
1532-0464/

3D objects, it can be necessary the access to several ring neighborhood in order to simulate the procedure with realism. Thus, it is not only a problem that an additional representation of a vertex's neighborhood could solve. This search is recursive and it can involve many vertices.

Within this context the purpose of this work is to compare the Data Structures known as *Mate-Face* [5] and *Compact Half-Edge* [6], considering as basic parameter the processing time, and analyzing their behavior within and out 3D VEs for medical training. These DS were integrated to a framework that allows the generation of three-dimensional interactive medical training applications [7].

Considering the presented scenario, our focus is to compare the performance of the DS considering virtual medical training specificities, mainly related to visual realism (which requires objects with high volume of vertices and faces) and responses in real time (which requires fast access to vertices and faces). VR applications for medical training usually include several tasks to process simultaneously (e.g., devices management and rendering). Even if we consider the use of graphic cards, constant interaction from users make more difficult to reach the requirement of real time feedback. Thus, the access to the mesh should not be an additional factor for delaying the response time. We can state that the simple structures (for example, simple arrays of vertices and faces without any optimization) can decrease the performance, as we will show in our case study. Thus, our main contribution is show that the use of faster intelligent data structures can provide the real time response in this type of applications.

Although these structures were first presented in the literature in the last decade, they are still well explored and analyzed in recent studies. Mate-face has been applied in areas like virtual medical training [8–10], volume simulations [11], and Computational Fluid Dynamics simulation [12]. Compact Half-Edge has been used to represent models mesh in 3D music visualization [13,14] and it was extended to answer topological queries while accept hybrid models composed by different types of cells [15]. The Computational Geometry Algorithms Library (CGAL) [16], which is an open source software library that provides several reliable implementations of efficient geometric algorithms, makes use of a half-edge data structure in a great number of its implementations. The CGAL applications include mesh generation [17], point set surfaces operations [18], and graph applications [19] among others.

Advances of computing resources has contributed to allow modern computers process an increasing amount of information in a shorter time. Even considering new hardware features, such as graphics cards, VR applications still have delays in response times if they are not well-designed. The visual quality of applications as well as interaction demands have increased, and users have become more demanding in terms of both visual and haptic realism. Some authors have discussed the GPU use particularly in VR applications for medical training [20,21]. Although one cannot overlook the contribution of technological advances in the hardware area, it does not mean that one can neglect the time of data access, especially in applications that require adequate response times so that the users does not notice delays that decrease their focus attention and the motivation for using the application.

In this work we did not consider any special computer architecture (such as GPU) that could improve the performance in order to not limit the application of our proposal and implementation. Although we did not consider special architectures, we tested different computer configurations in order to evaluate which factor (memory or processor speed) had more influence in the results. We also proposed an implementation using hash-table which allowed a significant increase in the performance of our algorithms.

From the analysis we conducted it is possible to supply an efficient way to manipulate vertices and cells that represent synthetic objects. Therefore, functions as collision detection and deformation of objects can be improved and contribute to generate applications with greater realism.

This work is established as follows: Section 2 shows the concepts about the representation of topological meshes. Section 3 presents a historical overview and related works that address the issue of efficient data structures. The development of the DS appraised is shown in Section 4. The results obtained with experiments, considering both raw structures (out of VR applications) and their inclusion in VR medical training applications, as well as discussions about them are made available in Section 5. Finally, Section 6 presents the conclusions of the work.

2. Meshes representation and VR applications

Considering the definitions as proposed by Cunha [5] and Ferreira [22], the present work takes into account the concepts presented hereunder, for the implementation of Data Structures.

A **simplex** σ of dimension k in the Euclidian space $\mathbb{R}^m$ ($k \leqslant m$), also known as a k-simplex, which is a generalization of the notion of a triangle to arbitrary dimensions, is defined by the convex hull involving $k+1$ points $p_0, p_1, \ldots, p_k \in \mathbb{R}^m$, in general position. The points are placed in such a way that the vectors formed by subsequent points $\overrightarrow{p_1 - p_0}, \overrightarrow{p_2 - p_0}, \ldots, \overrightarrow{p_k - p_0}$ are linearly independent.

This means to say that a k-simplex σ can also be defined by the set shown in Eq. (1).

$$\sigma_k = \{p \in \mathbb{R}^m : \quad p = \sum_{i=0}^{k} \lambda_i p_i, \quad \sum_{i=0}^{k} \lambda_i = 1, \quad \text{with,} \quad \lambda_i \in \mathbb{R} \quad \lambda_i \geqslant 0 \quad \text{for} \quad i = 0, \ldots, k\} \tag{1}$$

where σ_k is a simplex of dimension k.

When the context is understood, we shall say that the simplices of dimensions $0, 1, 2$ and 3 are respectively a *vertex*, *edge*, *triangle* and *tetrahedron*. Similarly, the points $p_0, p_1, \ldots, p_k$ belonging to a k-simplex σ are called *vertices* of σ. A sub-cell, also known as a **face of a simplex** σ is the convex hull of a subset of vertices of σ and is, by definition, also a simplex. A simplex is said to be adjacent to (or in the neighbor of) another simplex if they both have a common sub-cell.

A **simplicial complex**, also known as a **mesh**, is a finite collection K of simplices, where the following conditions hold:

1. If γ is a face of a simplex of K, then $\gamma \in K$;
2. The intersection of two simplices of K is either empty or a sub-cell of one of the simplices;

Simplicial complexes are a discrete representation of mathematical objects that we call manifold. In the particular case where the dimension of the manifold is 2, it is called a surface. Fig. 1 shows examples of meshes that represent manifolds of dimension 2 (on the left) and 3 (on the right).

For a simplicial complex K of dimension n, a $(n-1)$-simplex is an **inside simplex** when it is shared by two n-simplices; should this not be the case, it is a **boundary simplex**. The simplices with dimensions of less than $(n-1)$ contained in boundary simplices are also considered boundary simplices. The **boundary** of K is a sub-complex $S \subset K$ comprising all the boundary p-simplices, for $p = 0, 1, \ldots, n-1$ of K.

We also define, for a vertex v, a **first-ring** neighbors of a vertex as the set of all cells that contain v. The **second-ring** neighbors as the set of all cells that intersect the first-ring, and recursively, **the** k_{th} **ring** neighbors as the set of all cells that intersect the $(k-1)_{th}$ ring.

A simplicial complex is a topological space of a certain kind, constructed by the union of points, line segments, triangles, and their n-dimensional counterparts. For this reason, data structures

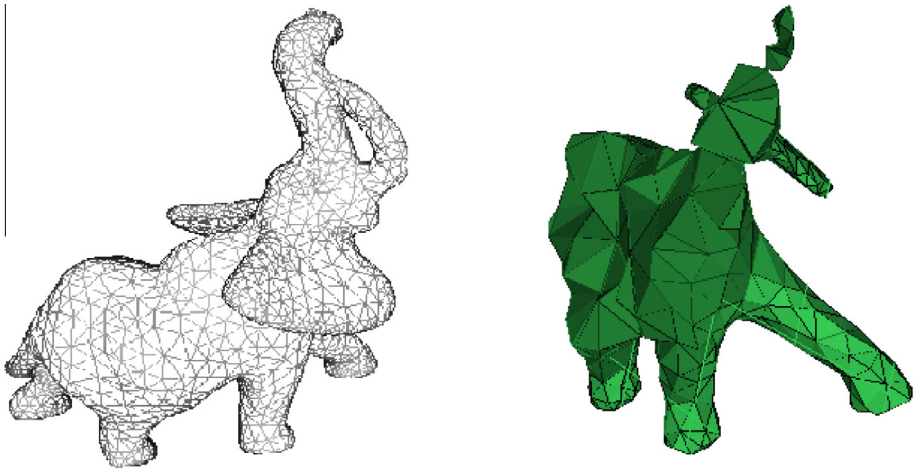


Fig. 1. On the left, a triangular mesh of one surface, and on the right a tetrahedral mesh of one volume.

that aims to index the elements of the mesh are known as **Topological Data Structures**.

Topological Data Structure organize the mesh information in a way that it represents the relations of incidence and adjacency among the different elements and also to make it easier to recover the information. The method in which the structures are stored may be implemented explicitly (in vector form, for example) or implicitly (by recovering the information through arithmetic operations). The advantage of using implicit forms is the low memory consumption, as the explicit forms enable greater access and less recovery time.

2.1. Virtual reality applications for medical training

VR applications for medical training require manipulate 3D meshes in real time when an interaction occurs in the VE. This manipulation generates actions such as changing colors and scale of the object, rotating or moving the object within the VE, detecting collision among objects, and deforming elastic objects. From these, collision detection and deformation are those operations that require search of vertices and faces in the 3D meshes.

Fig. 2 shows a VE for breast biopsy simulation. The breast is represented by a mesh composed by triangles. When the virtual medical device collides with the virtual breast, a deformation occurs. This deformation is performed by moving the position of the vertices located around the collision local. The first step is to locate the nearest vertex of the collision. Then, an iterative procedure is triggered in order to solve a system of equations involving forces and springs, where the force, and hence the displacement at each vertex, depend directly on the neighbor relations that should be accessed as quickly as possible in each time step.

3. Related work

Studies of topological DS started in the 1970s. Besides the comprehension of works which had previously defined DS, there are also comparative projects about topological DS.

In 1975 there the first significant structure was introduced, developed by Baumgart [24], called Winged-Edge. Since then, other experts have based their works on this theory to create other structures over time. Fig. 3 shows the sequence of DS that emerged.

Winged-Edge was one of the first work projects proposed to represent surfaces in $\mathbb{R}^3$. It uses edges to access the data of a mesh. Each edge keeps the vertices of its extremities, the left and right faces and also the preceding and succeeding edges in relation to the left face [24]. In 1983 Guibas and Stolfi [25] presented a generalization of *Winged-Edge*, called *Quad-Edge*, which is able to represent non-orientable surfaces. Then there is *Half-Edge*, as proposed by Mantyla in 1988 [26], which also uses edges to access the data of a mesh; each edge is divided into two *half-edges*, one for the left-hand face and the other for the right-hand face, with opposite directions (Fig. 4). Each *half-edge* stores one vertex and one incident face, featured in Fig. 4, while for each vertex and face their respective *half-edges* are then stored.

In 1996, Lopes [27] presented the *Handle-Edge* which is just an extension of the *Half-Edge*, with the difference that here there is explicit representation of boundary curves and also the use of nodes for representing the mesh elements.

The first structure to use the concept of scalability was *Directed-Edges*, presented by Campagna et al. in 1998 [28]. In 2001, Rossignac and their collaborators [29] proposed the *Corner-Table*, a structure which represents triangular meshes using the concept of corners to represent the association of a triangle with its vertices. The triangles are implicitly stored using the equation $t = \lfloor c/3 \rfloor$ (where t is the number of the triangle and c is the *corner*), while the neighborhood among triangles is defined by a corner c and its opposite corner $O[c]$ which has the same opposite edge. Fig. 5 shows a tetrahedron and its elements using the *Corner-Table* structure.

The work of Lewiner et al. [30] introduced two types of primitive operators on the underlying meshes. According to the authors, there are operators that change the topological characteristic of the mesh and operators that only modify its combinatorial structure. They demonstrated that those operators provide a complete and coherent set of elementary operations for mesh construction and editing.

Opposite-Face (OF), presented by Lizier in 2004 [31] is based on the *Corner-Table* DS. It explicitly represents the vertices and the cells of a mesh, while the edges and the faces (only in three-dimensional cases) are stored implicitly.

All the Topological Data Structures presented are used to represent different objects, whether in two or three dimensions, which means that each and every DS uses its own technique. For greater clarity, Table 1 shows the relationships among the DS and the objects thus represented.

There has been continuous research conducted aimed at finding efficient DS for the representation of different kinds of objects. Many of these investigations explain how each one operates in terms of storage, neighborhood relationships and also comparisons between different algorithms of the same DS, as shown next.

Chen et al. [32] presented a versatile data structure known as *Edge-Based*, based on central edges. These researchers also proposed an efficient algorithm which calculates the least distance

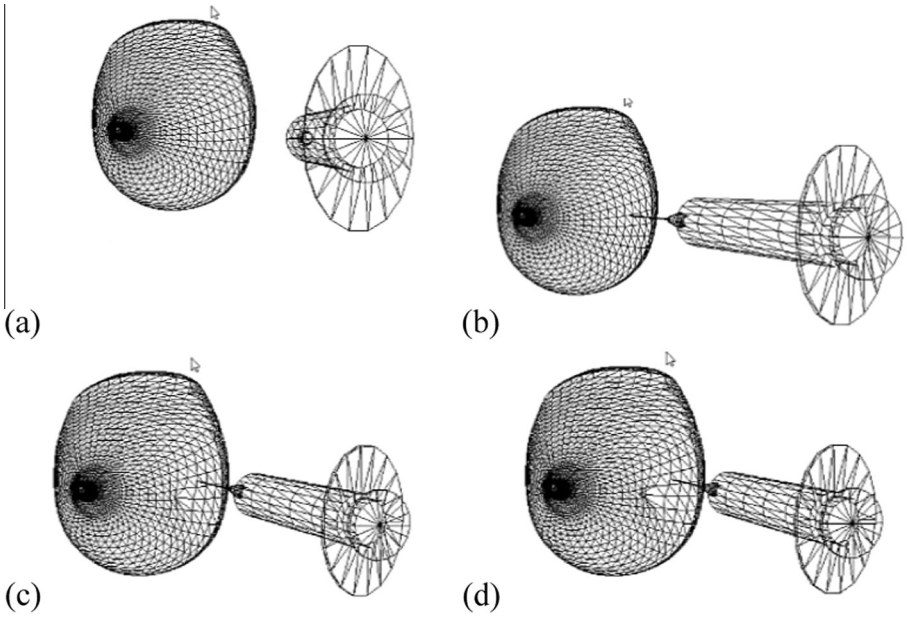


Fig. 2. Example of deformation in a 3D mesh over time. Extracted from Vimet application [23].

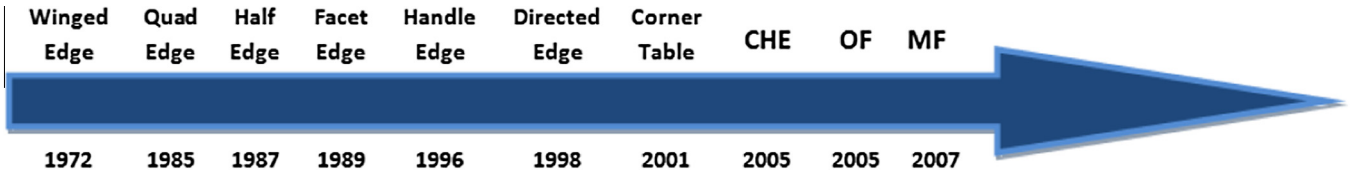


Fig. 3. Time line showing the DS thus created.

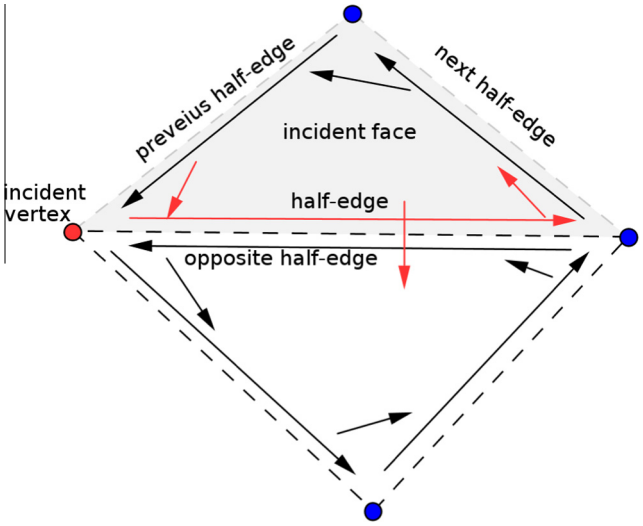


Fig. 4. *Half-edge* and its associations.

between two vertices. The algorithm was analyzed using speed tests and memory consumption. Fan [33] described another versatile data structure based on edge-symmetry, which can be applied to represent three-dimensional objects. Performance was also analyzed based on time and space complexity.

Weiler [34] compared four different data structures (*winged-edge, modified winged-edge, vertex-edge and face-edge*), all of which are based on edges to obtain a relationship of adjacency, with representations aimed at solid objects with curvatures. Also De Floriani and Hui [35] compared topological DS of different representations, which are classified based on the dimensions of the objects worked on (two or three dimensions) and their own methods of representation.

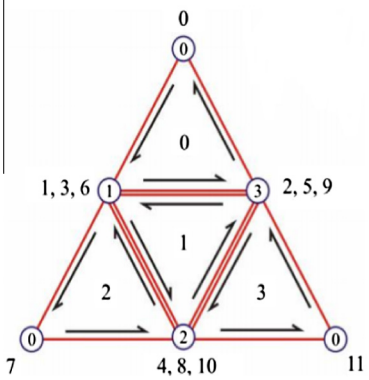
In our work we consider the need for realism to represent objects in three-dimensional interactive environments. With this purpose in mind, this work implements and compares the performances of two DS: *Compact Half-Edge* (CHE) and *Mate-Face*, in order to efficiently represent two-dimensional geometric models of human organs.

4. Data structures development

In this work we have implemented two data structures: *Mate-Face* (MF) [5] and *Compact Half-Edge* (CHE) [6]. CHE has the advantage of being a scalable structure able to balance performance and memory. If there is any availability, the structure allows the use of additional memory to improve performance and, for example, the border edges can be stored in a separate vector, so that there is no need to conduct a search to obtain the border of a given surface. On the other hand, MF has an interface which is both simple and efficient, which can also represent mixed meshes, making it easy to integrate with any application using meshes. Both DS were implemented in Java programming language. The codes developed consider structures that represent triangular and two-dimensional meshes.

4.1. Mate-Face data structure

The *Mate-Face*, developed by Cunha [5], is a flexible structure created with the capacity to represent simplicial complexes in



Corner	Vertex	Triangle	Opposite
0	0	0	2
1	1	0	2
2	3	0	2
3	1	1	0
4	2	1	0
5	3	1	0
6	1	2	3
7	0	2	3
8	2	2	3
9	3	3	1
10	2	3	1
11	0	3	1

Fig. 5. Open tetrahedron and its elements (adapted from Cunha [5]).

Table 1
Data structures and objects that each is able to represent.

Data structure	Objects	Method of indexation
Winged-Edge	OS without boundary[a]	By Edges
Quad-Edge	NOS without boundary[b]	By Edges
Half-Edge	OS without boundary	By *Half-Edges*
Handle-Edge	OS with boundary	By *Half-Edges*
Directed-Edge	OS without boundary	By *Half-Edges*
Corner-Table	OSs without boundary	By *corners*
Opposite-Face	NOS without boundary	By *corners*

[a] OS = Orientable surfaces.
[b] NOS = Non-orientable surfaces.

two and three dimensions, as well as meshes with other types of polygons, such as quadrilaterals. The structure was based on the *Opposite-Face* structure [31], and the main differences are the possibility of representing edges and faces in explicit form and also the method of indexing neighboring cells.

The *MF* consists of a vector of vertices which stores the respective coordinates, as well as a reference to the last incident cell; a vector of cells which stores their respective vertices and a reference to the adjacent cells, and a mesh that stores the cells and vertices. This structure may also contain a vector of edges, but in this work this possibility was not implemented in its first version, with the edges being implicitly represented. Fig. 6 shows the composition of the *MF*.

In Fig. 6 the position 0 of the vertex vector represents the vertex 0, which is defined by the coordinates x_0, y_0 and z_0, and which is incident to cell 0. We also see that the position 1 of the cell vector represents cell 1 (the same index), which is formed by vertices 2, 1 and 3, and its neighboring cells numbered 2 and 0. Following these rules for storing data, we can represent the whole mesh through these two vectors.

The neighborhood relationship in *MF* is not always obtained through corners. As is the case with the *OF* structure, it can also be obtained through *half-edges*, depending on the type of mesh considered. In other words, the neighborhood relationship depends on the type of mesh used, and not just on the opposite vertices. In the cases of triangles and tetrahedra, the adjacent cells continue to be indexed based on opposite vertices.

In the present paper the neighboring cells are established after the vectors of vertices and cells have been loaded. The cell vector is examined to identify cells that have two common adjacent vertices, e.g., which have one common edge. For this, is necessary to reach the vertices of one cell based on a specific incident vertex v, using Eqs. (2) and (3), where $next(i)$ and $prev(i)$ produces the local index of the next and the previous vertex of v respectively, and the symbol "%" is the division remainder.

$$next(i) = (i+1)\%3 \tag{2}$$

$$prev(i) = (i+2)\%3 \tag{3}$$

Fig. 7 shows the neighborhood relationships for each cell constructed in relation to the index of opposite vertices. We can see the neighborhood relationship being constructed through the local l indices of the cell in question. This means that cell 1 is the neighbor of cell 3 through the vertex opposite 1 with local index equal 0. Likewise, we see that this same cell 1 is also the neighbor of cell 2 through the vertex opposite vertex 2 with local index $l = 1$, and also for cell 0 through the vertex opposite vertex 3 with local index $l = 2$.

Though the storage of these indexes is reliable, it is a lengthy process as its complexity is quadratic relative to the number of faces. The processing time was reduced by creating a second method to construct the neighborhoods, making use of a *hash* table as an auxiliary structure to optimize the process.

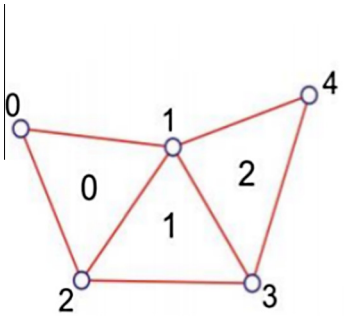
4.2. Compact Half-Edge data structure

The *Compact Half-Edge* structure was presented by Lages et al. [6]. This is a structure for the representation of triangles divided into 4 levels, in order to enable changes to the amount of data stored, improving efficiency. In the present work levels 0 and 1, and part of level 2 were implemented (we can say that we have implemented up to level 1.5), sufficient to represent the structures in the context of this work.

At **Level 0** (known as the **Triangle Soup**) only the basic information of the mesh is stored: the vertices with their coordinates and also the cells that contain them. The coordinates of the vertices are stored in vector *G*. This level is only for viewing the mesh, and does not represent the relationships of adjacency. Each triangle is implicitly represented by 3 *half-edges*. The index *i* of the *half-edge* is represented by Eq. (4), where *t* is the index of the cell and *k* is the index of the *half-edge he* in the cell. In other words, the index of the *half-edge* is 3 times the index of the cell plus the local index of the *he* in the cell. Cell 30, for example, has the *half-edges* 90, 91 and 92.

$$i = k + 3 * t \tag{4}$$

The *he's* are represented by a vector *V* of integers, in which the position *he* shows the index of the vertex of origin, known as the "foot".

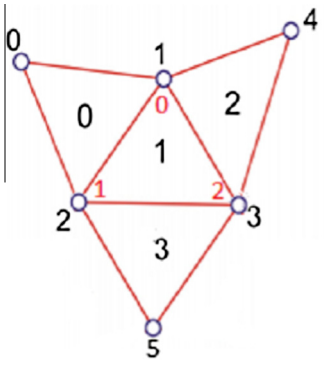


Mesh vectors representation

id	Vertex Coords			Cell
0	X_0	Y_0	Z_0	0
1	X_1	Y_1	Z_1	1
2	X_2	Y_2	Z_2	1
3	X_3	Y_3	Z_3	2
4	X_4	Y_4	Z_4	2

id	Cells V_0	V_1	V_2	Neighbors		
0	0	1	2	1	-1	-1
1	2	1	3	2	-1	0
2	3	1	4	-1	-1	1

Fig. 6. Representation of the mesh and also of vertices and cell vectors, by the MF structure (adapted from Cunha [5]).



Cell id	Opposite of cell 1		
	3	2	0
	0	1	2

Fig. 7. Neighborhood relationship by opposite vertex (adapted from Cunha [5]).

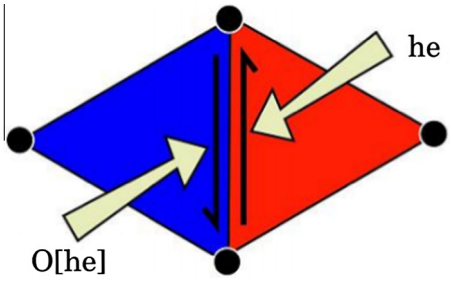


Fig. 8. Representation of *half-edge* and opposite *half-edge* on *CHE* data structure (extracted from Lopes [27]).

Eqs. (5) and (6) re used to obtain the previous *he* and the next *he* within a cell.

$$next(he) = 3 * \lfloor he/3 \rfloor + (he + 1)\%3 \quad (5)$$

$$prev(he) = 3 * \lfloor he/3 \rfloor + (he + 2)\%3 \quad (6)$$

Level 1 deals with the connection among the triangles, known as **Adjacency among Triangle**. For this purpose, the concept of opposite half-edge is used. As each edge is only attached to two vertices, these vertices are therefore adjacent. We therefore look for half-edges that form the same edge but considering the opposite directions, as shown in Fig. 8.

The adjacencies are stored in a vector *O*. The index of the position *he* for this vector contains the index of its opposite half-edge. If the *he* is a border, which means it does not have an opposite edge, then the index stored is −1. Like in the *MF* structure, a second method was created in order to add adjacencies using a hash table as an auxiliary tool to optimize the algorithm.

In **Level 2**, which is known as **Representation of Cells**, two new vectors are created, *Edge Map* (HE) and *Vertex Half-Edge* (VH). The vector *HE* contains the edges of the mesh. The edge is represented by one of its *he's*, as the other can be recovered using the vector *O*. For the vector *HE*, the vertices that make up the edge are also stored, recovered using its constituent *he's*.

The vector *VH* stores, for each vertex index, an incident *half-edge*. This vector is important for ring neighborhood operations on the vertex (explained below), in which there is a need to pass through all the incident cells of a certain vertex. As is the case for the *CHE* structure, the process of searching for the cells starts with the opposite cells. This means that it is necessary to know one edge that is incident on that vertex.

At the last level (**Level 3**, called **Representation of Boundary Curves**), a structure is created to explicitly represent the border curves of the surface. Each border curve is represented by one of its incident *half-edges* and all the others can be found by other element vectors. To store the curves, a vector designated *CH* is created.

Fig. 9 shows how a *CHE* is composed with all its levels. Besides the aforementioned structures and methods, a method has been created for scanning the ring neighborhood based on a given *half-edge*, in a manner similar to that of the *MF* structure.

4.3. Retrieving the ring-neighbors of a Vertex

A relevant operation in a DS is the *ring neighborhood of a Vertex* traversal, which passes through all the incident cells attached to a given vertex. This operation is extremely important in 3D interactive environments when operations such as deformation must be executed.

The operation is performed based on a cell containing the starting vertex. In the example of Fig. 10a, the starting cell is cell 0. In this example, this is the last cell to be stored which contains the vertex highlighted in red. After this, there is the passage through neighboring cells with this same vertex, in a cycle, until it returns to the starting cell. In the case of a border vertex, as there are no more neighboring cells in that direction, the direction is then inverted in order to pass through all the cells until the other extremity, as shown in Fig. 10a. In Fig. 10b we see a complete Star cycle in the first ring of neighbors, which means only those which have cells in common with the central vertex (the vertices of those cells are highlighted in blue) and the second ring which contains the cells in common with the first ring (the vertices of those cells are highlighted in green in Fig. 10c).

The algorithms that traverse the ring neighborhood of a vertex are slightly different in the two structures, as seen in Algorithm 1 (for the MF structure) and Algorithm 2 (for the CHE structure). Both are based on the assumption that the mesh is coherently orientated. Notice that in lines 3–8 of Algorithm 1 there is a local search for the position of a given vertex within the cell that contains it, which does not exist in Algorithm 2.

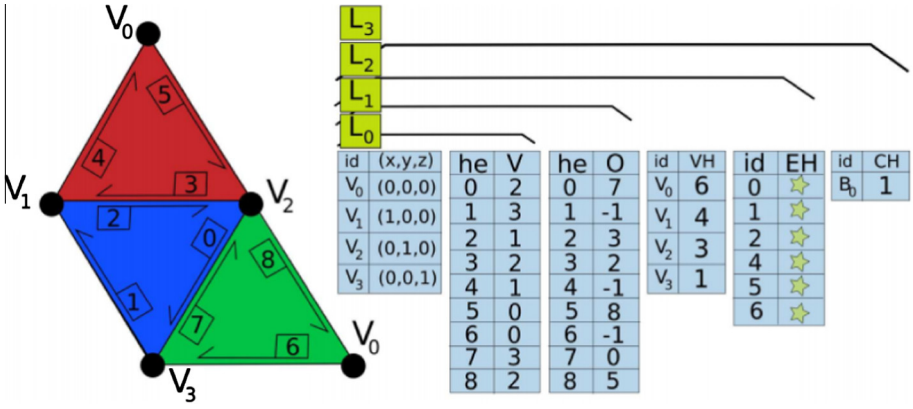


Fig. 9. Representation of *half-edge* and opposite *half-edge* in the CHE data structure. (extracted from Lopes [27]).

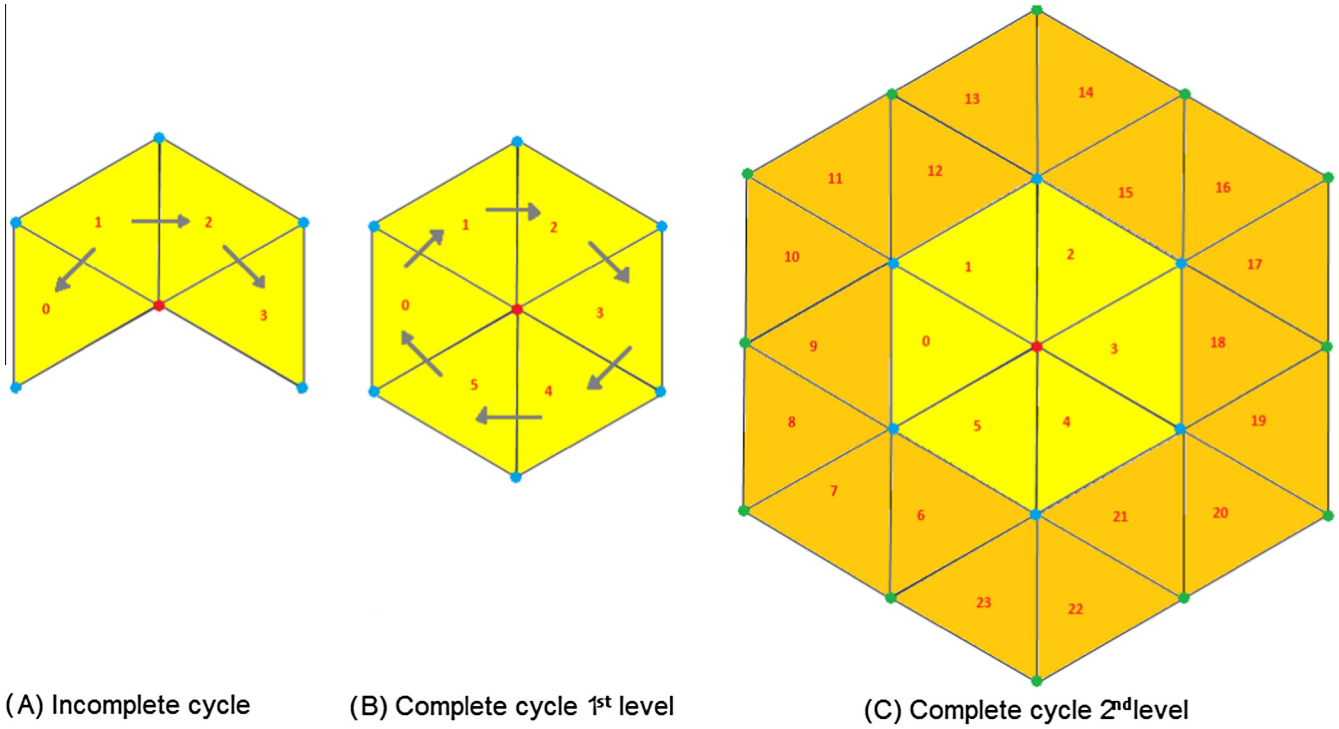


Fig. 10. Ring neighborhood formats of the vertex, in MF: incomplete, complete, and multiple levels.

Algorithm 1. Algorithm to scan the first level of the ring neighborhood of a vertex in the MF structure

Input: A vertex v_i
Output: A list with the vertices neighboring v_i
Let c_i be a cell that contains v_i;
Let L be a list of vertices neighboring v_i;
Find i the position of v_i in c_i;
$v_{n_i} \leftarrow$ next(i) in c_i;
Add v_{n_i} in L;
$c_o \leftarrow$ the opposite cell to c_i through the vertex v_{n_i};
while $c_o \neq c_i$ **do**
  Find i the position of v_i in c_o;
  $v_{n_i} \leftarrow$ next(i) in c_o;
  Add v_{n_i} in L;
  $c_o \leftarrow$ the opposite cell to c_o through the vertex v_{n_i};
return L

Algorithm 2. Algorithm to scan the first level of the ring neighborhood of a vertex in the CHE structure

Input: A vertex v_i
Output: A list with the vertices neighboring v_i
Let he_i a half-edge that contains v_i;
Let L be a list of vertices neighboring v_i;
$he_{n_i} \leftarrow$ next(he_i) ;
$v_{n_i} \leftarrow$ vertex "foot "of he_{n_i} ;
Add v_{n_i} in L;
$he_o \leftarrow$ the opposite half-edge to he_{n_i} ;
while $he_o \neq he_i$ **do**
  $he_{n_i} \leftarrow$ next(he_o) ;
  $v_{n_i} \leftarrow$ vertex "foot "of he_{n_i} ;
  Add v_{n_i} in L;
  $he_o \leftarrow$ the opposite half-edge to he_{n_i};
return L

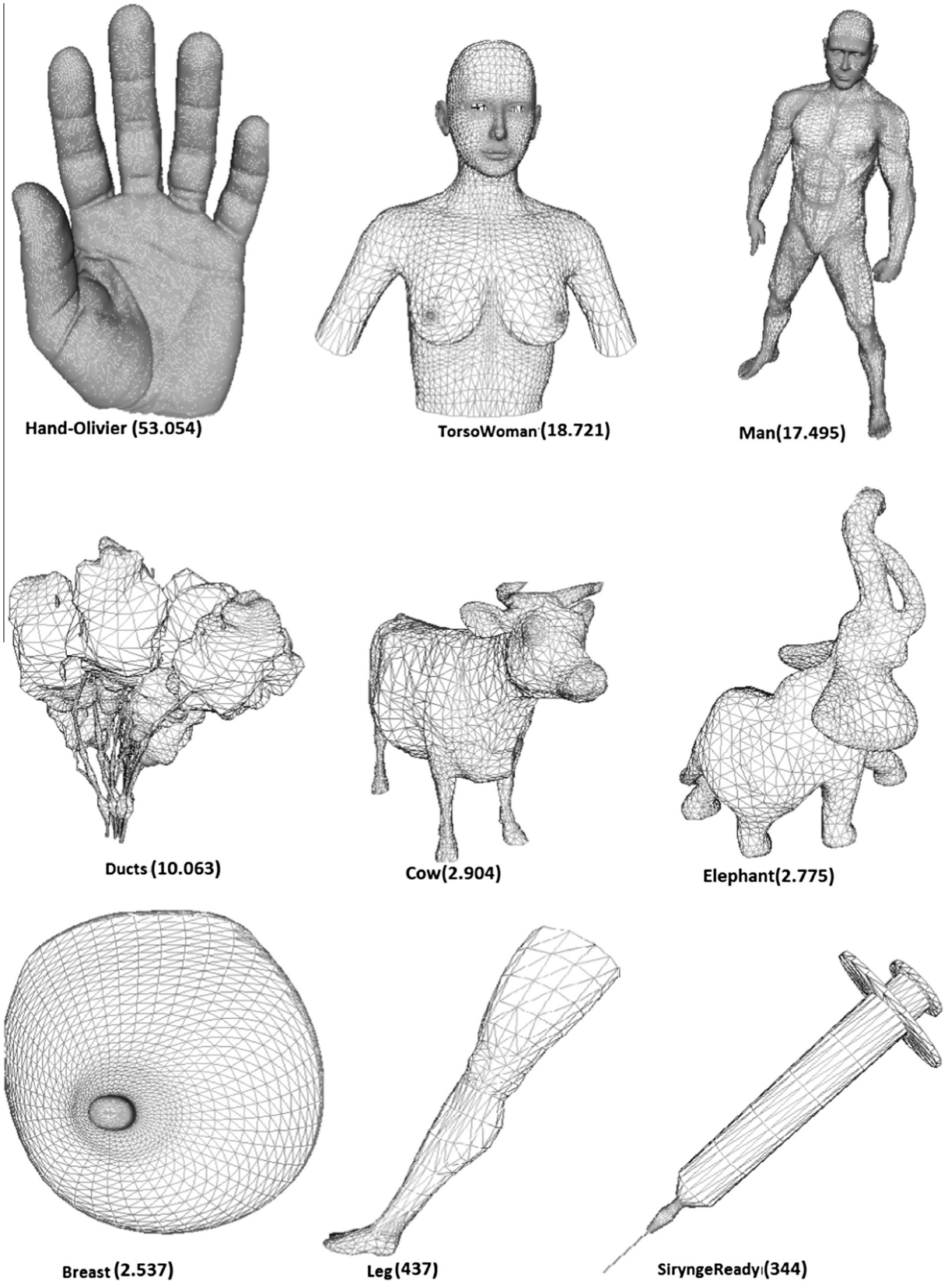


Fig. 11. Meshes used in the tests, with the respective amount of vertices.

Table 2
Configuration of the computers used in the experiments.

Resource	PC1	PC2	PC3
Processor	Ci3 U380 @1.33 GHz	Ci5 M430 @2.27 GHz	Ci5 661 @3.33 GHz
Memory (RAM)	4 Gb	3 Gb	4 GB
OS	Wins7-64 bits	Win7-64 bits	Win7-32 bits

5. Results and discussions

To compare the structures implemented, we conducted tests using meshes with different number of vertices and different levels of complexity, considering two approaches. Firstly, to evaluate the general behavior of CHE and MF structures, we verified the processing time for both loading and accessing tasks, considering an implementation of "raw" structures, i.e., outside VR applications and, thus, without the influence of other aspects (such as interaction and rendering of objects) (Section 5.1). Secondly, we implemented the two DS in a framework which generate applications for biopsy exams simulation, and conducted modifications in the resultant applications to minimize the effects of some VR aspects (Section 5.2).

5.1. Results with raw structures

Fig. 11 presents the meshes used in our tests. Three computers with different configurations were used to carry out the test (Table 2). Evaluating the performance of meshes using different computer configurations allows composing a discussion about how the processor and memory influenced the results.

In order to minimize the influence of the computer conditions during the tests, we made sure the computers did not keep other programs running, with only the programs related to the Operational System remaining active. We first evaluated the loading mesh times in the implemented DS in order to verify the influence of the mesh size on the performance of the structures. Each mesh was read from an ASCII file and its data were loaded onto the structures shown in Section 4. The test was executed 30 times using the implementation with *hash* table and 30 times without the *hash* table. Considering each previously cited equipment, we processed each mesh 180 times (total of 1620 executions) and recorded the time of each execution.

5.1.1. Processing time for loading meshes

Usually the use of hash table allows performing the retrieval in a time almost constant if the hash function is adequately defined for this. Therefore, its use is recommended even for lower amount of vertices. The challenge here is define an efficient hash function as well as an adequate table size, since the performance depends on both the hash function as the percentage of the table that will be filled. In our implementation we use Eq. (7), where $e = (v_1, v_2)$ is an edge and v_1 is the vertex with the smallest index in e, v_2 is the vertex with the biggest index in e. In this equation, t is defined as three times the number of cells and % indicates the rest of the division operation. From our results we realized that the benefits of the hash table is lower when we use objects with few vertices; however, the retrieval time remains almost constant as the amount of vertices increases, which characterizes a notable benefit for VR applications that require objects with big amount of vertices (see Figs. 12–15).

$$h(e) = (10 * v_1 + v2)\%t \tag{7}$$

The four graphs show that the processing time is proportional to the number of vertices. The confirmation of this fact was somewhat expected, once the formation of the explicit memory structures (vectors) is directly dependent on the number of data units (vertices). This supports the hypothesis that efficient structures are needed to represent these objects in 3D interactive environments, as it requires precise modeling of the objects and consequently requires structures with many vertices to provide a realistic sensation.

In relation to the influence of the machines configuration on the results obtained, we see that PC1, which has a slower processor than the others, had the worst performance in almost all time

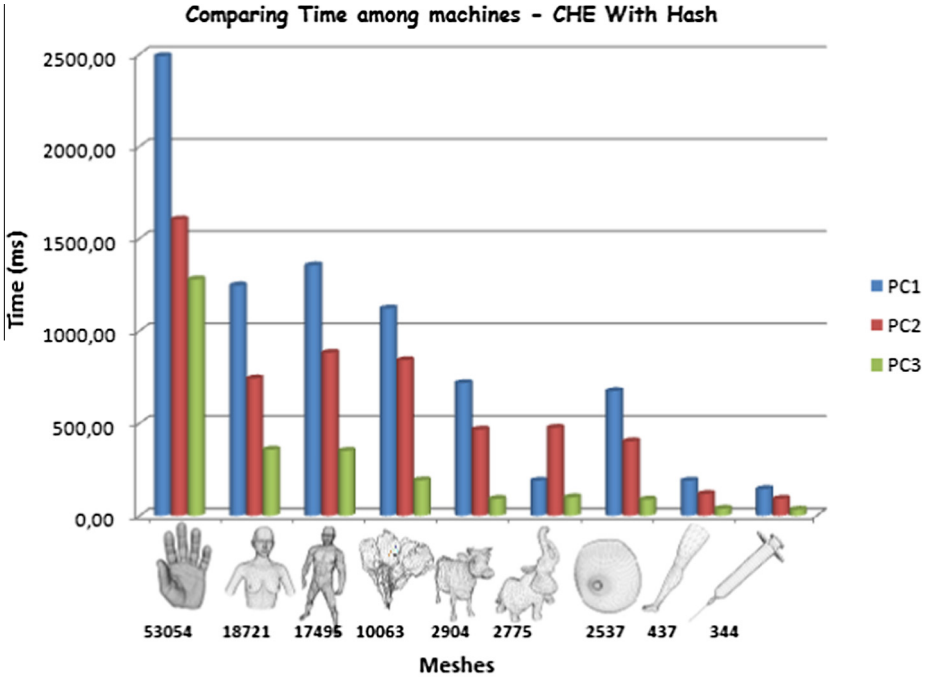


Fig. 12. Comparison of processing time – *CHE* with *hash* table.

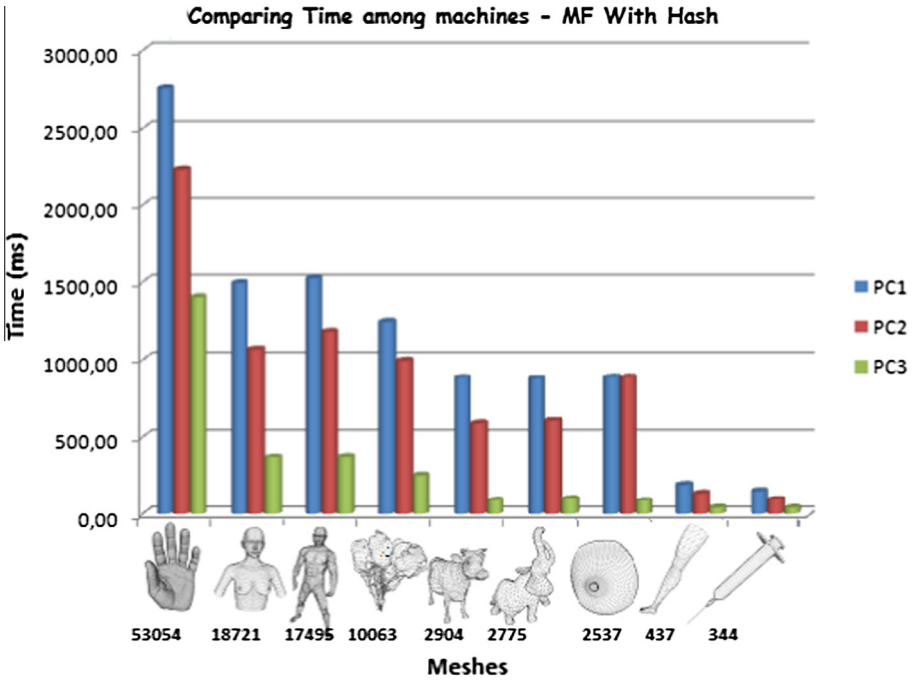


Fig. 13. Comparison of processing time – *MF* with *hash* table.

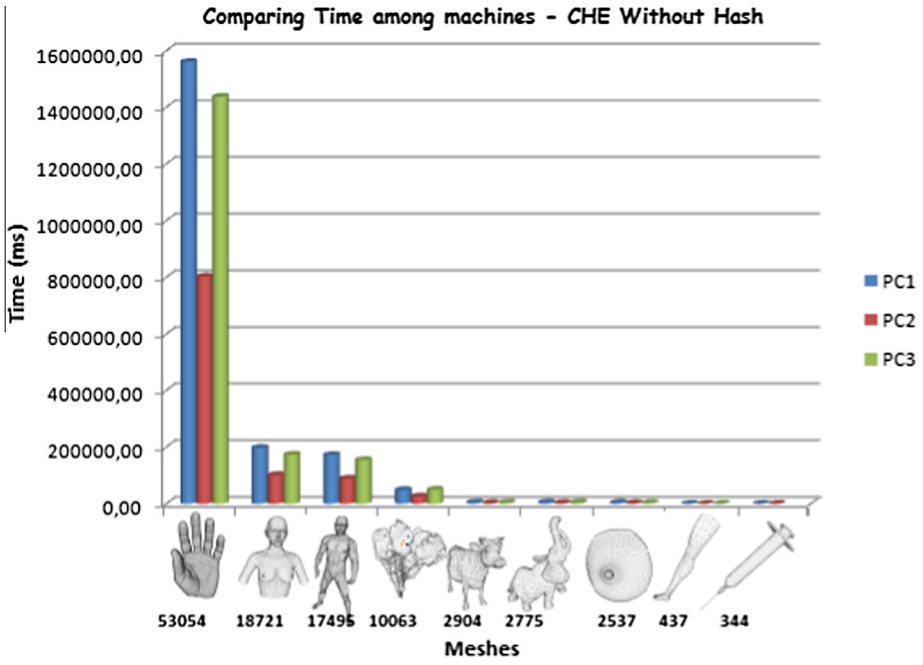


Fig. 14. Comparative graph between machines – *CHE* without *hash* table.

comparisons. This indicates that within the context of our experiments, the processor had significant influence for the performance of the system. We also see that compared to PC2 this machine has the same memory capacity and the same memory capacity as PC3. Even so, in general the processor capacity showed to be the most decisive factor in the tests conducted.

Comparing the processing times obtained with each structure (Tables 3 and 4), we see a better performance for the CHE structure, shown by the models with more vertices. The loading of the Hand-Olivier model, for example, took 1790 ms in the version with hash table for the CHE structure and 2134 ms in the same version for the MF structure, a processing time difference of about 19% (Fig. 17). Considering the same conditions for the second model (Torso-Woman, with 18,721 vertices), the time was 781 ms with hash table using CHE and 977 ms with hash table using MF, which means that there is a 25% time addition. This rate undergoes a variation, for other models, that ranges between 0% and 112%, and the tests conducted show that the processing time addition rate is not proportional to the number of vertices.

The use of the concepts of *half-edge* and opposite *half-edges* is more efficient than the use of corners and opposite cells. This result is extremely important for the context of the present work, considering the high number of vertices demonstrated by the models that represent objects in 3D interactive environments.

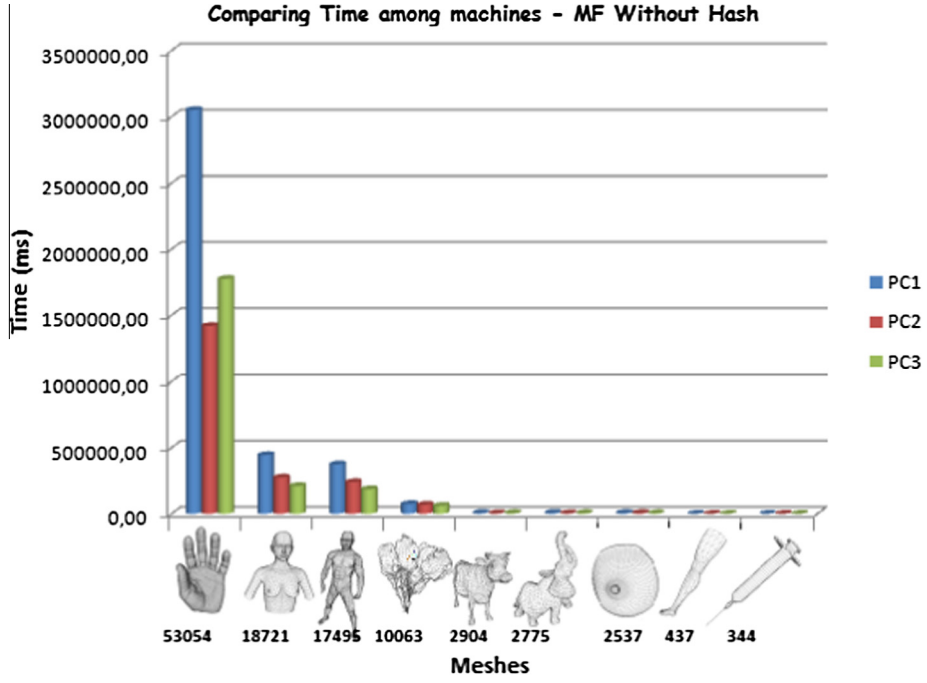


Fig. 15. Comparative graph between machines – *MF* without *hash*.

Table 3
Processing time (in seconds) for the meshes without *hash table* in both structures.

Mesh	*CHE*	*MF*	Percent (%)
Comparing time without hash – CHE versus MF			
Hand-Olivier (53,054 vertices)	1266.87	2081.11	64
TorsoWoman (18,721 vertices)	156.99	308.10	96
Man (17,495 vertices)	138.12	265.31	92
Ducts (10,063 vertices)	41.42	66.54	60
Cow (2094 vertices)	3.79	5.00	32
Elephant (2775 vertices)	4.01	4.99	24
Breast (2537 vertices)	2.91	5.24	80
Leg (437 vertices)	0.19	0.22	16
SiryngeReady (344 vertices)	0.13	0.16	23

Table 4
Processing time (in seconds) for the meshes with *hash table* in both structures.

Mesh	*CHE*	*MF*	Percent (%)
Comparing Time with hash – CHE versus MF			
Hand-Olivier (53,054 vertices)	1.79	2.13	19
TorsoWoman (18,721 vertices)	0.78	0.98	25
Man (17,495 vertices)	0.86	1.03	20
Ducts (10,063 vertices)	0.72	0.83	15
Cow (2094 vertices)	0.42	0.52	24
Elephant (2775 vertices)	0.25	0.53	112
Breast (2537 vertices)	0.39	0.62	59
Leg (437 vertices)	0.11	0.12	9
SiryngReadt 344 vertices)	0.09	0.09	0

5.1.2. Access time

To test the access time for the mesh data, tests with the ring neighborhood were conducted. We conducted tests considering 30 random vertices from each mesh. The accesses to the ring neighborhood of these vertices were performed from the first to the fourth ring of each model tested. For each mesh, as well as for the previous tests, the process was repeated 30 times on each vertex and the average of the values obtained were computed. Fig. 16 shows the results of these tests for the 4 meshes that produced the most significant results. Table 5 shows the number of vertices at each level for these meshes.

MF was faster than *CHE* in almost all cases. There was only one case in which the *CHE* structure was faster than the *MF* (the second level of the *cow* mesh, where difference was 0.04 ms). The access time difference was of 15 ms. *Breast* mesh, for example, showed a time of 1.26 ms for the *MF* structure at the fourth level, and 4.23 ms for the *CHE* structure at the same level. At the second level, the same mesh also showed a small time difference in the comparison of the structures, with 0.08 ms execution time *MF* and 0.04 ms with *CHE*.

For three specific meshes – *Elephant, Man and Ducts* – the *CHE* structure had a peak in execution time, showing a greater difference when compared with *MF*. This is the case with the *Man* mesh: with the *MF* structure the access to its fourth level took 0.94 ms while for the *CHE* structure the time was 15.88 ms.

5.2. Results with VR applications

In this section we present results of the two DS structures (and also the access without them) after their implementation in VR applications for medical training. Besides, we present the integration of the implemented DS in a VR framework that generates this type of applications (Section 5.2.1).

5.2.1. Integration of the data structures into the ViMeT Framework

ViMeT (Virtual Medical Training) framework is composed of a set of classes and a instantiation tool that allow generating applications for simulating biopsy exams. They were implemented in the Java language in order to provide a free and easy way to generate applications in this domain (available at http://www.each.usp.br/lapis/). This framework allows including devices (such as haptics and shutter glasses) in the generated applications. It also allows choosing the technique used for collision detection and tissue deformation, among others functionalities.

In the generated applications the simulations of human organs must take into account the physical properties of these objects in order to provide realistic sensations during the virtual training. Furthermore, users must receive feedback of their actions in real

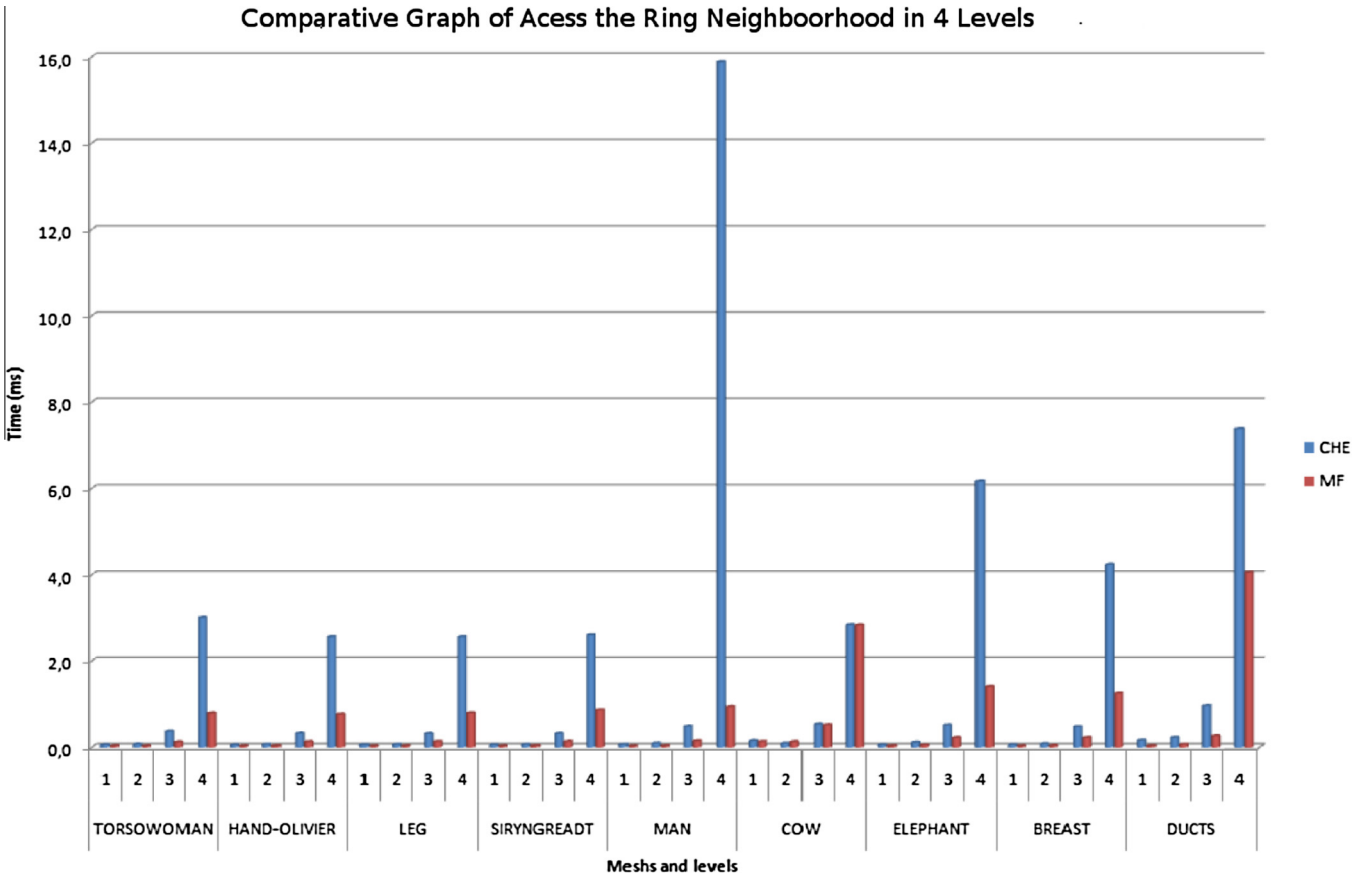


Fig. 16. Comparison of processing time to the ring neighborhood vertex operation – level 1–4.

Table 5
Number of vertices per level, for each mesh as shown on the graph of Fig. 16.

Mesh	Level 1	Level 2	Level 3	Level 4
Amount of vertices in each scanned vertex				
TorsoWoman	6	17	34	57
Hand-olivier	6	17	35	58
Leg	6	18	35	58
SiryngReadt	7	19	37	61
Man	7	19	40	69
Cow	7	20	40	68
Elephant	7	21	43	72
Breast	7	22	49	90
Ducts	9	23	47	76

time. Therefore, functionalities like collision detection and deformation require the displacement of a many vertices in a short time interval.

This integration required remodeling the instantiation tool to allow the choice of the data structure (Fig. 17) and the adjustments made into the code.

The deformation method implemented in ViMeT framework is known as mass-spring. In this method, one must calculate, for each vertex v_i, at each time instant, the resulting force f_i of the influence of surrounding vertices v_i. For this purpose, it defines fictitious springs on the edges of the mesh and determines the resulting force based on the Hook's law (Eq. (8)).

$$f = -k\delta \tag{8}$$

where k is the resistivity coefficient of the spring and δ is the displacement.

As each edge of the mesh is treated as an individual spring and thus having its own coefficient of resistivity, the connectivity of the mesh determines which vertices have influence on the calculation of the resultant force acting on a given vertex v_i (Eq. (9)).

$$f_i = \sum_{j \in N(v_i)} k_{ij}(\delta_j - \delta_i) \tag{9}$$

where $N(v_i)$ represents the first-ring neighborhood of v_i and δ_i is the displacement of the vertex v_i.

Eq. (9) makes it clear that, at every instant of time, in the simulation process, one must have access to neighborhood relations of each vertex of the mesh that represents the object being deformed. This fact justifies the implementation of data structures that store this relationship efficiently as part of efforts to optimize processes in a VR environment.

Another important part of the simulation is the collision detection, i.e., the identification of the exact local where the medical instrument touches the geometrical model. This part of the simulation process is not affected by the topological data structures and will not be subject for our discussion.

5.2.2. Processing time for accessing meshes

Although the functionalities provided by ViMeT framework provide enough resources to obtain realistic applications, we eliminate some aspects in order to minimize their influence in the results. Thus, we did not include colors and textures in the object, and we exclude real time interaction in the our applications. Therefore, our applications consider wireframe objects. (See Table 6) and execute the collision and deformation considering a fixed vertex.

The data presented following considered the average of five runs of the same test. We carried out these tests in two stages: (1) traversing only the neighborhood relations of each vertex and (2) considering the time spent until the end of the deformation

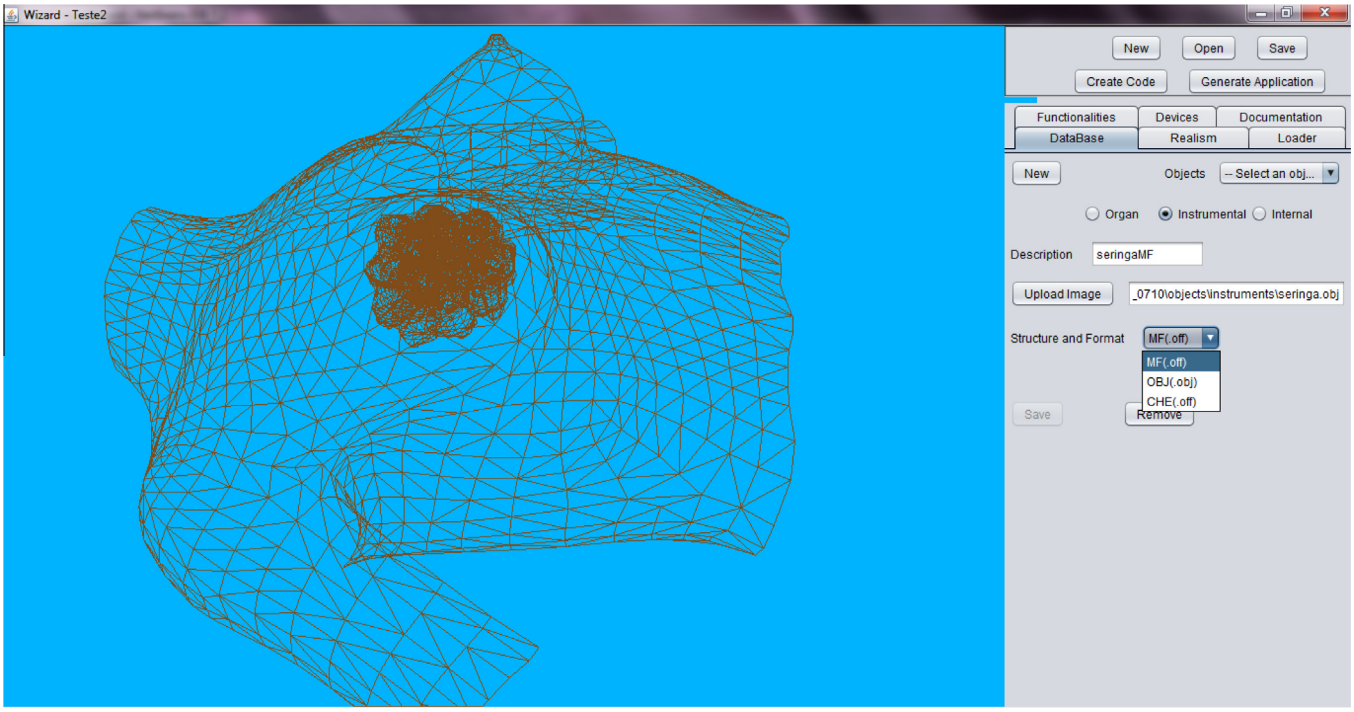
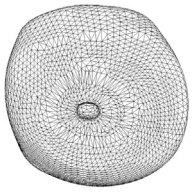
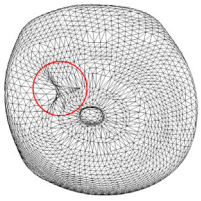
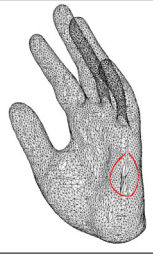


Fig. 17. New Wizard Interface of the *ViMeT Framework*, with the possibility of choosing the structure for loading.

Table 6
Original × deformed mesh.

Initial Mesh	Deformed Mesh
	
	
	
	

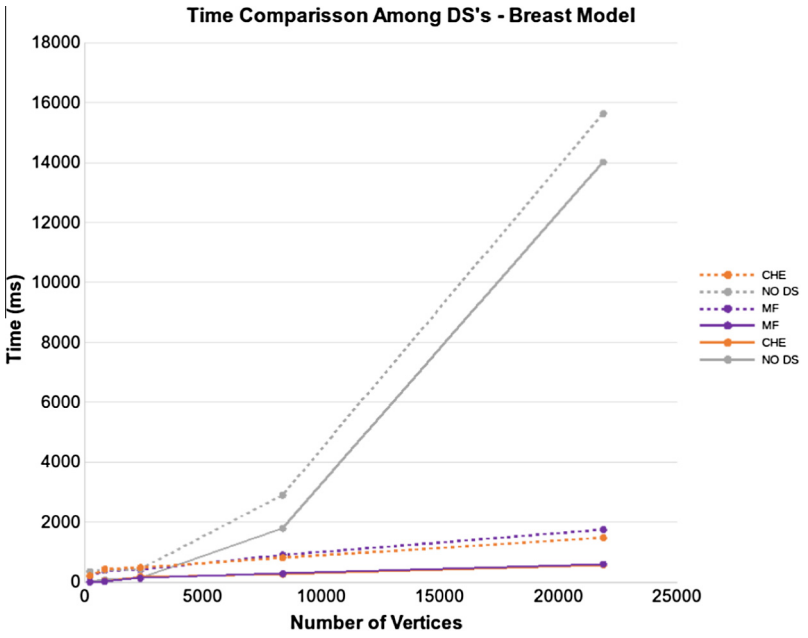


Fig. 18. Comparison among simulations times (in a breast geometry) using CHE, MF and without DS.

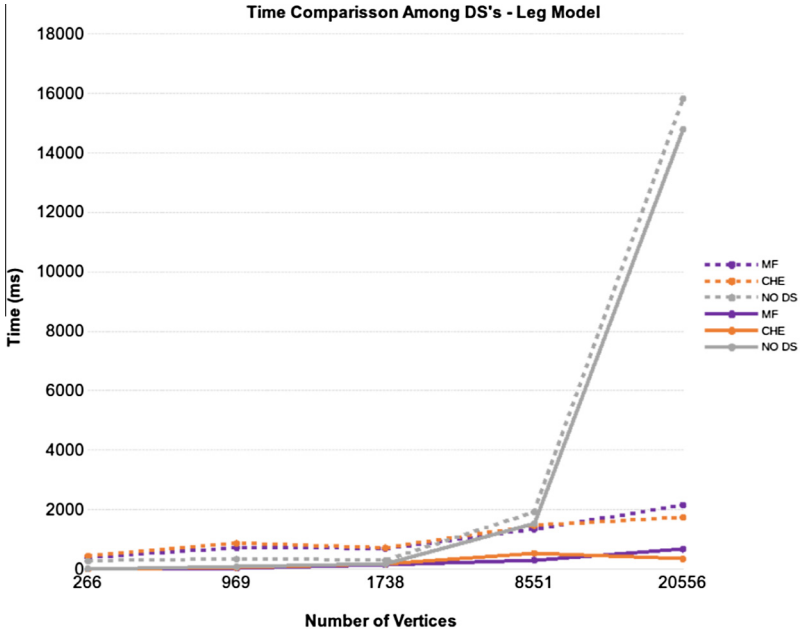


Fig. 19. Comparison among simulations times (in a leg geometry) using CHE, MF and without DS.

algorithm, which implies the convergence of solving the system of equations defined in Eq. (9). We emphasize that the time of the calculation of the deformations was computed because this is the process most affected by the immediate neighborhood of the vertices.

We considered four biopsy applications, with different objects representing different human organs. Since these objects have different geometry and different amount of vertices and faces, we can evaluate the performance of the structures under several conditions.

We used five variations for each geometric model: two models considering hundreds of vertices (between 200 and 800 vertices), two models considering thousands of vertices (between 2000 and 8000 vertices) and one model with around 20,000 vertices.

Figs. 18–21 show the time spent (in milliseconds) for the four different models, both in the calculation of the immediate neighbors of the vertex, as the deformation algorithm, which is the resolution of the linear system defined by Eq. (9). The line identified by the legends as "NO DS " is the time to run the tests without any special data structure, that is, only with the information of "triangles soup". The *X*-axis indicates the different number of vertices for a specific geometric model that represents a human organ. The continuous lines represent the time to find the first ring of neighbors for each vertex, whereas the dashed lines represent the total time spent for the calculation of deformation, i.e., including the time of convergence of the numerical method for solving the linear system.

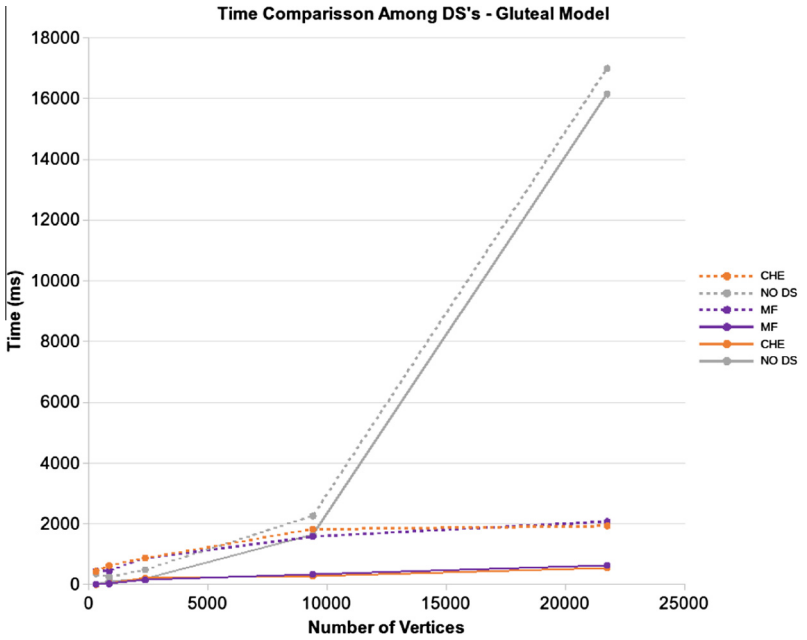


Fig. 20. Comparison among simulations times (in a gluteal geometry) using CHE, MF and without DS.

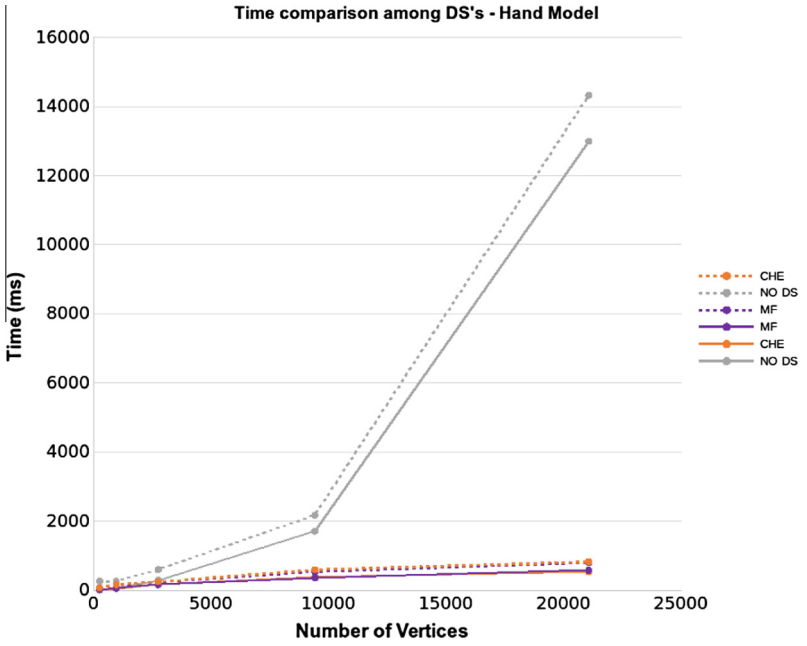


Fig. 21. Comparison among simulations times (in a Hand geometry) using CHE, MF and without DS.

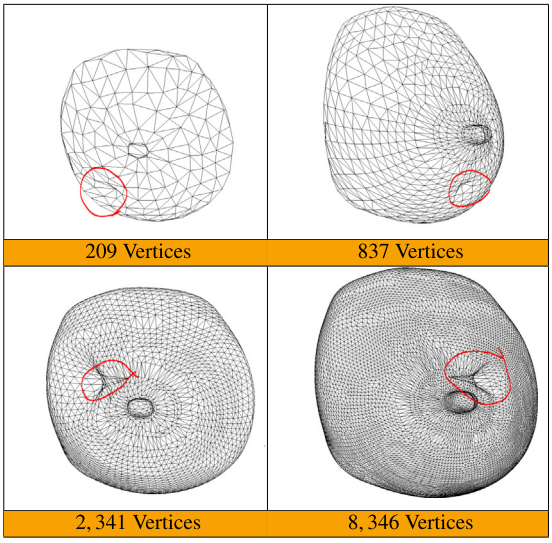
We highlight three important points from the test cases: (1) there was a balance between the calculation time of the relations of neighborhoods for both structures CHE and MF; (2) despite the increase in the number of vertices, spent time remained stable when these DS were used; and (3) the most significant time differences, when compared with the tests without the data structures, appeared in models with a number of vertices on the order of thousands.

It is worth mentioning that in models with about 20,000 vertices, without the use of fast data structures, the user could wait up 16 s for a response of the VR simulation, which means wait for the calculation of the new positions of the vertices after interaction. This precludes any notion of realism required for this type of application.

In order to illustrate this process, Table 6 shows the result of our four biopsy applications with the different objects (a gluteal, a breast, a leg and a hand). The models in the left column reflect the meshes in their initial states, whereas the models on the right side represent the same models after applying a force which causes an elastic deformation, considering the method previously presented. The highlighted area in each model of the right column represents the locus where the deformations occurred.

In order to allow a comparison of the effect of the deformation in objects with different resolutions, Table 7 shows the visual

Table 7
Visual breast deformation with different number of vertices.



result of the mass-spring deformation algorithm in the breast model with four different amount of vertices. It is important to note here, based on the graph presented in Fig. 18, that the use of topological data structures allows the user to choose models considering different levels of details, depending on the application needs. This choice does not have a significant impact on the total time spent in the simulation deformation. Consequently, the use of the DS implemented in this work – CHE and MF – allows providing more realistic objects without decreasing performance.

5.3. Limitations and future work

In this section, we explored the execution time, conducting an evaluation of both loading time and access time considering the two DSs presented. Although we think this is an important contribution to aid researchers to choose the most suitable structure to their implementations, the fact of taking account only the execution time can be a limitation, since other aspects could be explored.

Although this limitation, we must highlight that the higher time efficiency, the more effective is the application in the context of medical training using VR. VR applications must provide real time feedback, without delay, since delays can compromise the sensation of realism perceived by the user. Thus, time efficiency is a primordial requisite to reach in this type of application.

We evaluated that in terms of memory use, the two structures proved to be exactly the same, except that *CHE* may be expanded to represent a greater number of geometrical structures such as border curves and edge maps. It is obvious that this expansion will involve an additional consumption of memory. We intend to deepen this evaluation in future work, in order to consider other objects and even other implementations.

Other types of evaluation can be explored as future works, as conducting experiments with users from medical area, such as the definition of objective metrics to measure how much the user can realize delays in this type of application using different DSs, as well as explore others DSs that can be adequate for VR systems in the context presented here.

6. Conclusions

In this work we aiming at comparing two Data Structures (Mate-Face and Compact Half-Edge) for using in 3D virtual environments for medical training, where objects can present a big amount of vertices and interactive operations require moving positions of vertices.

The results obtained indicate that both structures are interesting for this type of applications which require precision in the objects representation and response to user actions in real time. With this adequacy, we can verify that each one is indicated for a type of use in these interactive environments.

Based on the comparative analysis of the loading time for the models, in all tests conducted, the conclusion was that, even with changes in memory, processor and operational systems for the machines used, the concepts of *half-edge* and opposite *half-edges* of the *CHE* structure ensure greater speed, that is 55% faster in the tests carried out.

In relation to access time, tested based on the Vertex Star using 30 random vertices in 4 levels, *MF* was usually faster. It should also be noted that the *CHE* structure, though usually slower, with regards to access time it is faster for loading meshes. Moreover, the CHE structure is easily scalable to represent other geometric structures such edge-maps and boundary curves.

Experiments with the DS embedded in a framework for medical training indicated that both CHE and MF maintain a very higher performance in relation to applications without fast data structures. Additionally, these structures allow the use of models with different resolutions (number of vertices), without significant impact on the time spent in the simulation, which contributes for the construction of more realistic simulators.

Therefore, when the application requires constant loading of objects and these objects do not frequently have their topological structure changed the *CHE* structure is recommended. Games that generally that use 3D environments are good examples of this class of applications. However, when the loading is not often required, but the objects should accurately reproduce physical properties by displacement of the vertices, the *MF* structure is more suitable. Virtual medical training represents typical examples of these applications.

Another conclusion reached by this work regards to the use of the *Hash Table*, which has shown to be extremely efficient, and proved through tests conducted on machines with different configurations. This is because the hash table significantly reduces the number of memory accesses for searching neighborhoods, empowering its use for loading the model more than 900 times faster.

Although another points can be explored in relation to these data structures presented here, this paper offers a contribution to researchers, mainly those that work with medical training using VR, to choose the best structure considering characteristics of their applications.

Acknowledgments

The authors thank to National Council for Scientific and Technological Development (CNPq) - process 559931/2010-7, São Paulo Research Foundation (Fapesp) - process 2010/15691-0, and National Institute of Science and Technology - Medicine Assisted by Scientific Computing (INCT-MACC).

References

[1] G.C. Burdea, P. Coiffet, Virtual reality technology, Comput. Animat. Virtual Worlds 16 (5) (2003) 559–560, http://dx.doi.org/10.1002/cav.v16:5.
[2] A.C.M.T.G. Oliveira, L. Pavarini, F.L.S. Nunes, L.C. Botega, D.J. Rossato, A. Bezerra, Virtual reality framework for medical training: implementation of a deformation class using Java, in: Proceedings of ACM International Conference on Virtual Reality Continuum and its Applications, vol. 1, Hong Kong, 2006.
[3] H. Hoppe, Optimization of mesh locality for transparent vertex caching, in: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999, pp. 269–276, http://dx.doi.org/10.1145/311535.311565. <http://dx.doi.org/10.1145/311535.311565> .
[4] G. Lin, T.P.Y. Yu, An improved vertex caching scheme for 3d mesh rendering, IEEE Trans. Visual. Comput. Graphics 12 (4) (2006) 640–648, http://dx.doi.org/10.1109/TVCG.2006.59. <http://dx.doi.org/10.1109/TVCG.2006.59> .
[5] I.L.L. Cunha, Estrutura de dados mate face e aplicações em geração e movimentos de malhas, Master Thesis, Instituto de Ciências Matemáticas e de Computação - ICMC-USP, São Paulo (SP), Brazil, 2009 (in Portuguese).
[6] M. Lages, T. Lewiner, H. Lopes, L. Velho, Che: A Scalable Topological Data Structure for Triangular Meshes, Tech. Rep., 2005.
[7] A.C.M.T.G. Oliveira, F.L.S. Nunes, Building a open source framework for virtual medical training, J. Digital Imaging 23 (6) (2010) 706–720.
[8] Ícaro L.L. Cunha, P.J. Xiab, L.S. Machado, T. Restivo, R.M. Moraes, A.M. Lopes, Cut and suture support on volumetric models in the cybermed framework, Proceedings of the Conference on ENTERprise Information Systems/International Conference on Health and Social Care Information Systems and Technologies, vol. 5, Elsevier Ltd., 2012, pp. 771–776.
[9] A. Oliveira, R. Tori, W. Brito, J. Dos Santos, H. Biscaro, F. Nunes, Simulation of soft tissue deformation: a new approach, in: 2013 IEEE 26th International Symposium on Computer-Based Medical Systems (CBMS), 2013, pp. 17–22, http://dx.doi.org/10.1109/CBMS.2013.6627758.
[10] A. Oliveira, R. Tori, J. Bernardes, R. Torres, W. Brito, F. Nunes, Deformation method using physical parameters composed of different tissue structures, in: 2014 IEEE 27th International Symposium on Computer-Based Medical Systems (CBMS), 2014, pp. 94–99, http://dx.doi.org/10.1109/CBMS.2014.71.
[11] F.P. Barbosa, I.R.V. Pola, A.C. Filho, Performance analysis of data structures for unstructured grid solutions on finite volume simulations, Proceedings of the 22nd International Congress of Mechanical Engineering, vol. I, ABCM, 2013, pp. 3480–3489.
[12] F.P. Barbosa, A.C. Filho, J.A. Cuminato, J.L.F. Azevedo, C. Breviglieri, A computational study of WENO schemes using a topological data structure, Proceedings of the 13th Brazilian Congress of Thermal Sciences and Engineering, vol. I, ABCM, 2010, pp. 1–9.
[13] T. Lewiner, T. Vieira, A. Bordignon, A. Cabral, C. Marques, J. Paixa o, L. Custodio, M. Lage, M. Andrade, R. Nascimento, S. de Botton, S. Pesco, H. Lopes, V. Mello, A. Peixoto, D. Martinez, Tuning manifold harmonics filters, in: 2010 23rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI), 2010, pp. 110–117, http://dx.doi.org/10.1109/SIBGRAPI.2010.23.
[14] T. Lewiner, C. Marques, J. Paixão, S. de Botton, A. Cabral, R. Nascimento, V. Mello, A. Peixoto, D.M. Morera, T. Vieira, Stereo music visualization through manifold harmonics, Visual Comput. 27 (10) (2011) 905–916, http://dx.doi.org/10.1007/s00371-011-0617-4. <http://dx.doi.org/10.1007/s00371-011-0617-4> .
[15] M. Lage, L.F. Martha, J.P.M. de Almeida, H. Lopes, IBHM: index.based data structures for 2D and 3D hybrid meshes, Eng. Comput. 1 (1) (2015) 1–18, http://dx.doi.org/10.1007/s00366-015-0395-0.
[16] E. Fogel, M. Teillaud, The computational geometry algorithms library CGAL, ACM Commun. Comput. Algebra 49 (1) (2015) 10–12, http://dx.doi.org/10.1145/2768577.2768579.
[17] G. Damiand, M. Teillaud, A generic implementation of dD combinatorial maps in {CGAL}, Procedia Eng 82 (2014) 46–58, http://dx.doi.org/10.1016/j.proeng.2014.10.372. 23rd International Meshing Roundtable (IMR23).
[18] P. Angelini, D. Eppstein, F. Frati, M. Kaufmann, S. Lazard, T. Mchedlidze, M. Teillaud, A. Wolff, Universal point sets for drawing planar graphs with circular arcs, J. Graph Algorithms Appl. 18 (3) (2014) 313–324, http://dx.doi.org/10.7155/jgaa.00324.
[19] S.G. Kobourov, M. Nöllenburg, M. Teillaud, Drawing graphs and maps with curves (Dagstuhl Seminar 13151), Dagstuhl Rep. 3 (4) (2013) 34–68, http://dx.doi.org/10.4230/DagRep.3.4.34. <http://drops.dagstuhl.de/opus/volltexte/2013/4168> .
[20] J. Mosegaard, T.S. Sorensen, Gpu accelerated surgical simulators for complex morphology, in: Virtual Reality, 2005, Proceedings, VR 2005, IEEE, 2005, pp. 147–153, http://dx.doi.org/10.1109/VR.2005.1492768.
[21] B. Kevelham, N. Magnenat-Thalmann, Virtual try on: an application in need of gpu optimization, in: Proceedings of the ATIP/A*CRC Workshop on Accelerator Technologies for High-Performance Computing: Does Asia Lead the Way?, ATIP '12, A*STAR Computational Resource Centre, Singapore, Singapore, 2012, pp. 10:1–10:9. <http://dl.acm.org/citation.cfm?id=2346696.2346709>.
[22] M.I.C. Ferreira, Estruturas de dados topológicas para variedade de dimensão 2 e 3, Master Thesis, Pontífica Universidade Católica do Rio de Janeiro - PUC-Rio, Rio de Janeiro (RJ) - Brazil, 2006 (in Portuguese).
[23] A.C.M.T.G. Oliveira, F.L.S. Nunes, An Open Source Framework for Virtual Medical Training (November 2014 (accessed November 15, 2015)). <http://www.each.usp.br/lapis/ViMeT.htm>.
[24] B.G. Baumgart, A polyhedron representation for computer vision, in: Proceedings of the May 19–22, 1975, National Computer Conference and Expositionc, ACM, New York, NY, USA, 1975, pp. 589–596.
[25] L.J. Guibas, J. Stolfi, Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams, in: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, ACM, New York, NY, USA, 1983, pp. 221–234, http://dx.doi.org/10.1145/800061.808751.
[26] M. Mantyla, An Introduction to Solid Modeling, Monography – Computer Science Press, Computer Science Press, 1988.
[27] H. Lopes, Algorithms to Build and Unbuild 2 and 3 dimensional manifolds, PhD Thesis, Pontífica Universidade Católica do Rio de Janeiro - PUC-Rio, Rio de Janeiro- Brazil, 1996.
[28] S. Campagna, L. Kobbelt, H. peter Seidel, Directed edges – a scalable representation for triangle meshes, J. Graphics Tools 3 (1998) 1–11.
[29] J. Rossignac, A. Safonova, A. Szymczak, 3d compression made simple: Edgebreaker on a corner-table, in: Shape Modeling International Conference, 2001, pp. 278–283.
[30] T. Lewiner, H. Lopes, E. Medeiros, G. Tavares, L. Velho, Topological mesh operators, Comput. Aided Geometric Des. 27 (1) (2010) 1–22, http://dx.doi.org/10.1016/j.cagd.2009.08.004.
[31] L. Nonato, A. Castelo, M. de Oliveira, M. Lizier, Topological approach for detecting objects from images, in: Electronic Imaging 2004, International Society for Optics and Photonics, 2004, pp. 62–73.
[32] G. Chen, M. Pang, J. Wang, Calculating shortest path on edge-based data structure of graph, in: Proceedings of the Second Workshop on Digital Media and its Application in Museum & Heritage, IEEE Computer Society, Washington, DC, USA, 2007, pp. 416–421. <http://dl.acm.org/citation.cfm?id=1333786.1333801> .
[33] C. Fan, A versatile data structure schema and algorithms based on edge-symmetry, 2010 International Conference on Information Management, Innovation Management and Industrial Engineering (ICIII), vol. 2, 2010, pp. 519–521.
[34] K. Weiler, Edge-based data structures for solid modeling in curved-surface environments, IEEE Comput. Graphics Appl. 5 (1) (1985) 21–40.
[35] L. De Floriani, A. Hui, Data structures for simplicial complexes: an analysis and a comparison, in: Proceedings of the third Eurographics symposium on Geometry processing, Eurographics Association, Aire-la-Ville, Switzerland, 2005. <http://dl.acm.org/citation.cfm?id=1281920.1281940> .