Topology Reconstruction for B-Rep Modeling from 3D Mesh in Reverse Engineering Applications

Roseline Bénière[a,c], Gérard Subsol[a], Gilles Gesquière[b], François Le Breton[c] and William Puech[a]

[a]LIRMM, Univ. Montpellier 2, CNRS, 161 rue Ada, 34 095, France;
[b]Aix-Marseille University, LSIS, CNRS, IUT, BP 90178, 13 637 Arles cedex, France;
[c]C4W, 219 rue Le Titien 34 000 Montpellier, France

ABSTRACT

Nowadays, most of the manufactured objects are designed using CAD (Computer-Aided Design) software. Nevertheless, for visualization, data exchange or manufacturing applications, the geometric model has to be discretized into a 3D mesh composed of a finite number of vertices and edges. But, in some cases, the initial model may be lost or unavailable. In other cases, the 3D discrete representation may be modified, for example after a numerical simulation, and does not correspond anymore to the initial model. A reverse engineering method is then required to reconstruct a 3D continuous representation from the discrete one.

In previous work,[1] we have presented a new approach for 3D geometric primitive extraction. In this paper, to complete our automatic and comprehensive reverse engineering process, we propose a method to construct the topology of the retrieved object. To reconstruct a B-Rep model, a new formalism is now introduced to define the adjacency relations. Then a new process is used to construct the boundaries of the object. The whole process is tested on 3D industrial meshes and bring a solution to recover B-Rep models.

Keywords: CAD, Reverse Engineering, 3D Mesh, B-Rep model, Topology, Adjacency Graph

1. INTRODUCTION

In reverse engineering, the aim is to reconstruct a continuous model of an object (a B-Rep model for example) from a discretized representation (as a 3D mesh). Several methods have been proposed these last years, Várady *et al* present in[2] a state of the art. These methods are often focused on the algorithm of detection of geometric primitives (as planes, spheres, cones, torus...) which allow to extract from a discretized mesh, Figure 1(a), several continuous object, as shown Figure 1(b). But, a key-point is to define the adjacency relations and the boundaries between these primitives, called *wires* and represented in black in the Figure 1(c)), in order to obtain a B-Rep representation (Figure 1(d)).

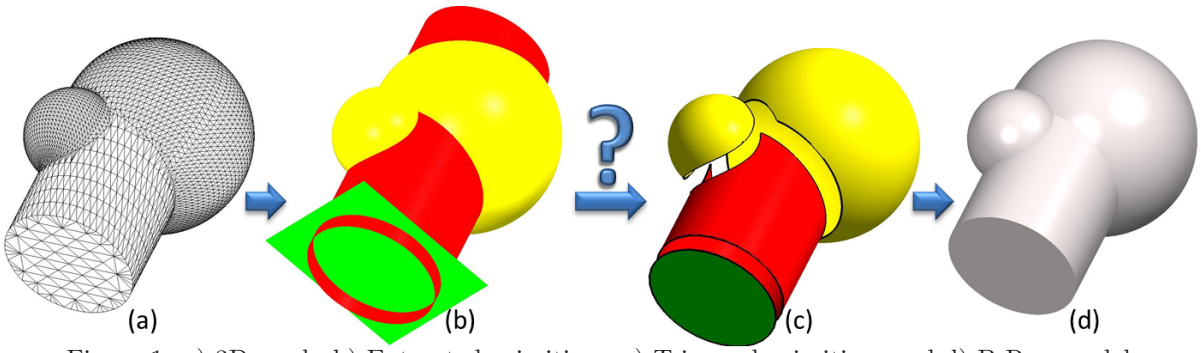


Figure 1: a) 3D mesh, b) Extracted primitives, c) Trimmed primitives and d) B-Rep model.

roseline.beniere; gerard.subsol; william.puech@lirmm.fr; gilles.gesquiere@lsis.org; flb@c4w.com

Three-Dimensional Image Processing (3DIP) and Applications II, edited by Atilla M. Baskurt, Robert Sitnik,
Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 8290, 82900M · © 2012 SPIE-IS&T
CCC code: 0277-786X/12/$18 · doi: 10.1117/12.909837

The informations stored in the B-Rep model for each surface are: the primitive type, the parameters and the references of the wires (which form the boundaries of the primitives). If two surfaces are neighbors, they must reference a same part of their wire, so the adjacency informations between the surfaces are needed. Hence after a first step to extract simple primitives,[1] a second one lead to reconstruct the topology, determining the neighbor primitives to other primitives. The intersections between the neighbor primitives define the surface boundaries and allow to reconstruct the B-Rep model. Our proposed method reconstructs the topology using both the parameterized representation of the extracted primitives and their corresponding point areas in the mesh.

In this paper, a method to reconstruct a B-Rep model with a consistent topology from a 3D mesh is presented. After a presentation of existing methods and the explication of the primitive extraction based on the previous work[1] in Section 2, the different steps are detailed in Section 3. Experimental results on simple and complex CAD objects are proposed in Section 4 and conclusion and perspectives are presented in Section 5.

2. PREVIOUS WORK

Few papers propose a comprehensive process of reverse engineering, like[3] or,[4] whereas many papers deal with one step. Furthermore methods exist, in other domains, to extract the relationship between the primitives and to construct the consistent boundaries of the object.

Chappuis *et al.*[5] use the relations between the primitives to construct the correct intersections and to be sure that after the remeshing the edges corresponding to theses intersections are not removed by this remeshing. After a computation of primitives belonging to the mesh, the adjacency relations are extracted from the sub-meshes used to compute the primitives and they are stored in an adjacency graph. Face wires which guide the remeshing are computed using this graph defined with a primitive by node and an edge if the primitive are neighbor. Even if this method is not a method of B-Rep reconstruction, its definition of the relationship between the faces can be used to extract consistent intersections and reconstruct a B-Rep model.

The difficulty for constructing the wires is to combine all the intersection curves, computed on all the pairs of geometric primitives, in consistent wires that are continuous and closed curves. This may appear very similar to the problem called "Boundary Evaluation"[6] which allows recovering a B-Rep representation from a CSG model. For example, Miller[7] computes the intersections between the primitives to define the wires. In the case of a CSG, the primitives are solid. For example, a cylinder is described by a cylindrical surface and by the two extreme circular plans. After the intersections computations, the author has a set of edge which is labeled with the information *Cross-edge* for the edge created by the intersection or *Self-edge* for the edge already existing in the volumes. The wires construction is based on the definition of a path through the edges; if several path are possible, the path using the *Cross-edge* is chosen. Thus, the wires are constructed for each faces with intersection between the volumes. Nevertheless, the Boundary Evaluation problem is much easier as we have the exact set of geometric volumes, whereas in our case we have surface, so we do not make the difference between the different edges.

In the following sections, we propose a new method to reconstruct a consistent B-Rep model with wires constructing using the relationship between the primitives using the method presented in[1] to extract the geometric primitives.

3. RECONSTRUCTING THE TOPOLOGY

3.1 Position in the reverse engineering process

The comprehensive process of reverse engineering that we have developed is decomposed into five steps (see Figure 2). The first step is the geometric extraction and is described by Bénière *et al.* in.[1] This method is based on the curvature features (principal curvatures and directions) to label the mesh vertices with a primitive type (one color by type in the Figure 2(b)) and extract *point areas*. Then, for each *point area*, a primitive is computed, using an approximation according to the type of the primitive (Figure 2(c)). The *point areas* are used to obtain the primitives, so extract only points contained in the primitive is more important than detect all points corresponding to the primitive. Thus, points corresponding to a primitive can not be present in the *point area* of its, like in the Figure 2(b).

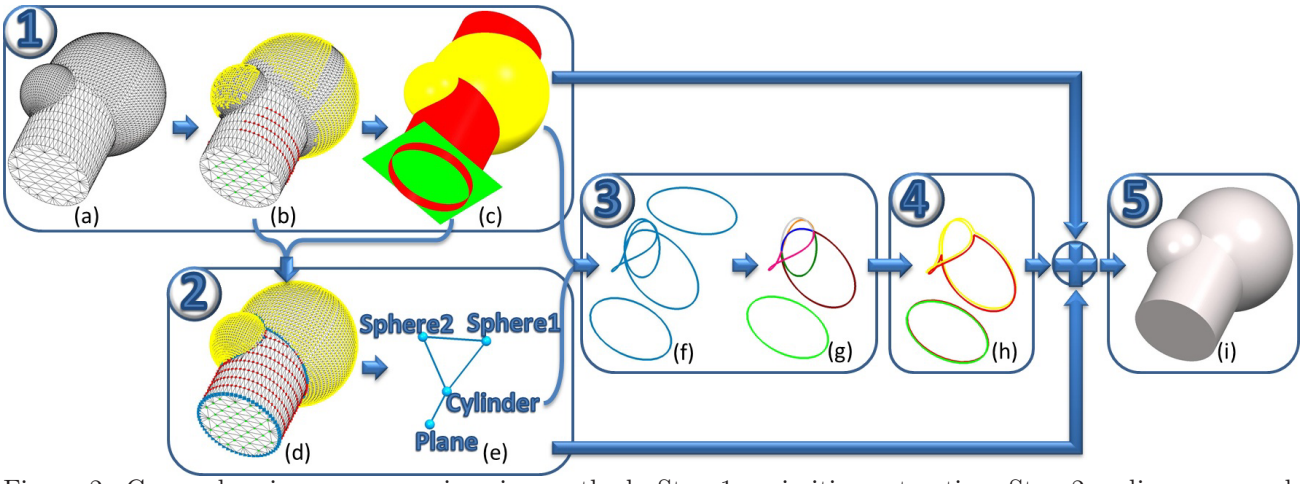


Figure 2: Comprehensive reverse engineering method: Step 1: primitive extraction, Step 2: adjacency graph determination, Step 3: edge extraction, Step 4: wire construction and Step 5: B-Rep creation.

After this primitive extraction to trim these primitives and assemble its in a consistent B-Rep model, we used four steps:

- **Step 2: Adjacency relation determination (Section 3.2):** this step defines the relation between the extracted primitives (Figure 2(e)). It is based on *common points* (in blue in the Figure 2(d)) but it requires before to extend *point areas* associated to each geometric primitive.
- **Step 3: Edge extraction (Section 3.3):** this step creates using the real intersections between the adjacent primitives (Figure 2(f)) which will be cut, the *edges* (Figure 2(g)).
- **Step 4: Wire construction (Section 3.4):** this step assembles the *edges* in consistent *wires* which are the boundaries of the object (Figure 2(h)).
- **Step 5: B-Rep creation (Section 3.5):** this step combines the informations of the previous step to construct a B-Rep model (Figure 2(i)).

3.2 Adjacency Graph Determination

In order to get a B-Rep representation, each geometric primitive has to be trimmed, according to its intersections with the other ones. To know which intersection is interesting for the wires, the adjacency relations are determined. For this purpose, an *adjacency graph* containing the relationship between the primitives is used. Each primitive corresponds to a node of the graph, and an edge is added between two nodes if the two primitives corresponding are neighbor.

The extraction of the *point areas* in the first step is based on a propagation method. A *point area* is initialized by a vertex with curvature specific and the neighbor vertices with the same curvature characteristics (according to the primitive type) are added to the area. During this construction, vertices on the primitive limits can not be added in the *point area*. Indeed, the curvature computation is based on neighborhood study for each vertex, so the vertex curvatures on the limit of primitive are disturbed by the vertices of the neighbor primitive. To obtain *point areas* containing all vertices corresponding to the primitive, an extension of these areas is then made. To extend a *point area*, the distance between adjacent vertices of the area and the corresponding primitive is computed. If the distance is lower than a given threshold, the vertex is added to the *point area* and their neighbors are studied too. Thus, using the informations containing in the primitives and in the *point areas*, the *extended areas* are obtained, see Figure 3(a). In a second time, the *extended areas* allow to define the *common points* (Figure 3(b)). For each pair of primitives, a set of *common points* is defined; if a point belongs to several *extended areas*, it is added to the *common points* of the primitive pair corresponding to the *extended areas*. To

represent the adjacency relations, an *adjacency graph* is used. It is initialized with a node by primitive. If the set of *common points* corresponding to two primitives is not empty, an edge is added in the graph between the two corresponding nodes as shown in Figure 3(c).

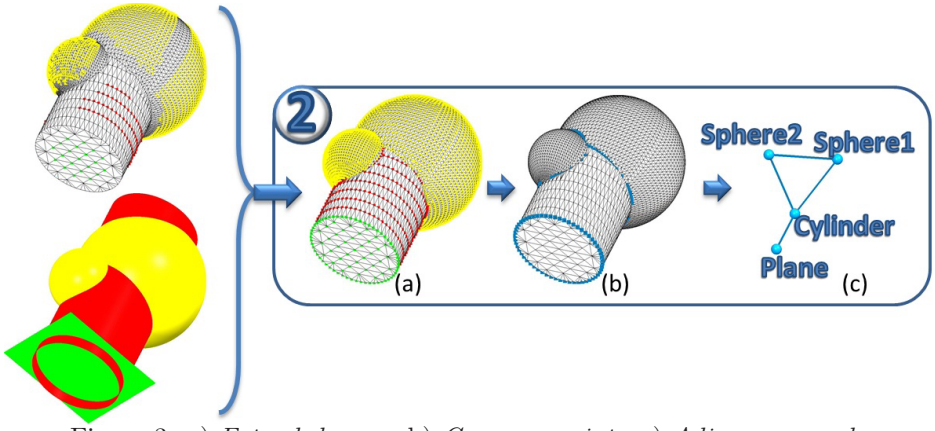


Figure 3: a) *Extended areas*, b) *Common points*, c) *Adjacency graph.*

3.3 Edge extraction

To recover the *wires* which are the boundaries trimming the geometric primitives, we first have to compute the intersection curves between the geometric primitives. For this, the edges of the *adjacency graph* which define the pairs of intersecting primitives are used. To compute the intersection curves between two primitives, many methods have been proposed (see for example[8]). In our case, the method implemented in the *Open Cascade Library*[*] is chosen. Its performs this operation and gives intersection curves as parametric curves defined by an equation according to the type (a B-Spline curve or a circle for example) and two limit points.

In Figure 4(a), the set of all intersection curves between geometric primitives are presented. But, some intersection curves are not really significant as, for instance, the one between the cylinder and the top of the sphere 1. Then, the validity of each intersection curve is checked by comparing its with the *common points* shared by the two corresponding primitives. If the distance between the *common points* and the intersection curve is larger than a given threshold, the intersection curve will not be taken into account in the reconstruction process. Then, in Figure 4(b), the red intersection curve is rejected whereas the green ones are validated.

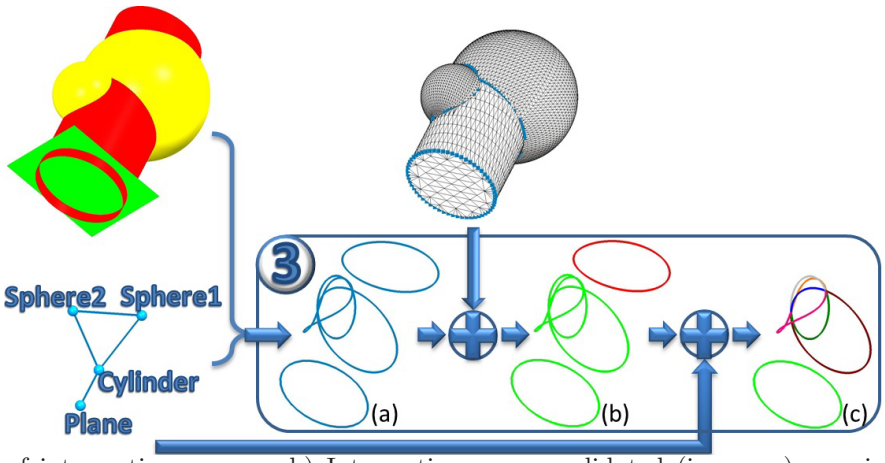
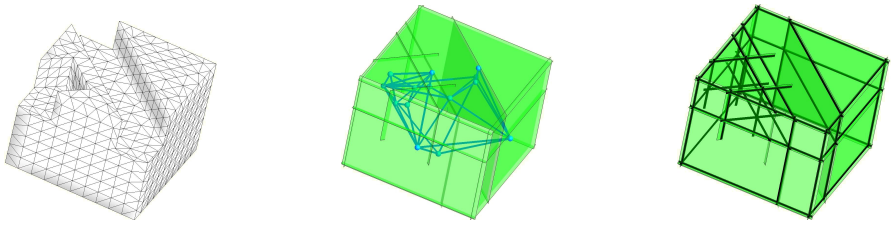


Figure 4: a) Set of intersections curves, b) Intersection curves validated (in green) or rejected (in red) and c) Intersection curves decomposed into *edges*, one by color.

[*]http://www.opencascade.org

In a second time, this validated curves are trimmed in *edges* (Figure 4(c)). The geometric primitives do not intersect only two by two. For example, the two spheres and the cylinder have common intersections. More generally, each intersection curve has to be decomposed into parts called *edges*. These *edges* are delimited by two *junctions* which correspond to the intersection points between three adjacent geometric primitives. To explain the *edge* construction, the example proposed in Figure 5 is used.



(a) (b) (c)

Figure 5: a) 3D mesh, b) Extracted geometric primitives (14 planes in this case) with the *adjacency graph* in superimposition and c) Valid intersection curves.

First, all the intersection points between two intersection curves are computed, then they are tested to check if they are *junctions* or not. A *junction* in a B-Rep model corresponds to a connection between two *edges*, so it binds three primitives. Furthermore, these three primitives have to be adjacent in the B-Rep model, thus there is a cycle in the *adjacency graph* between the three corresponding nodes. For each intersection point, two tests are effectuated (one by column in Figure 6), illustrated by three examples (one by line in Figure 6):

- In the first example, the intersection point connect four primitives, so it is not a junction;
- In the second example, the intersection point is between three primitives, but they are not bind in the *adjacency graph* by a cycle, thus this point is not a *junction*;
- In the third example, the intersection point connect three primitives which are bind in the *adjacency graph* by a cycle, so this point is a correct *junction*.

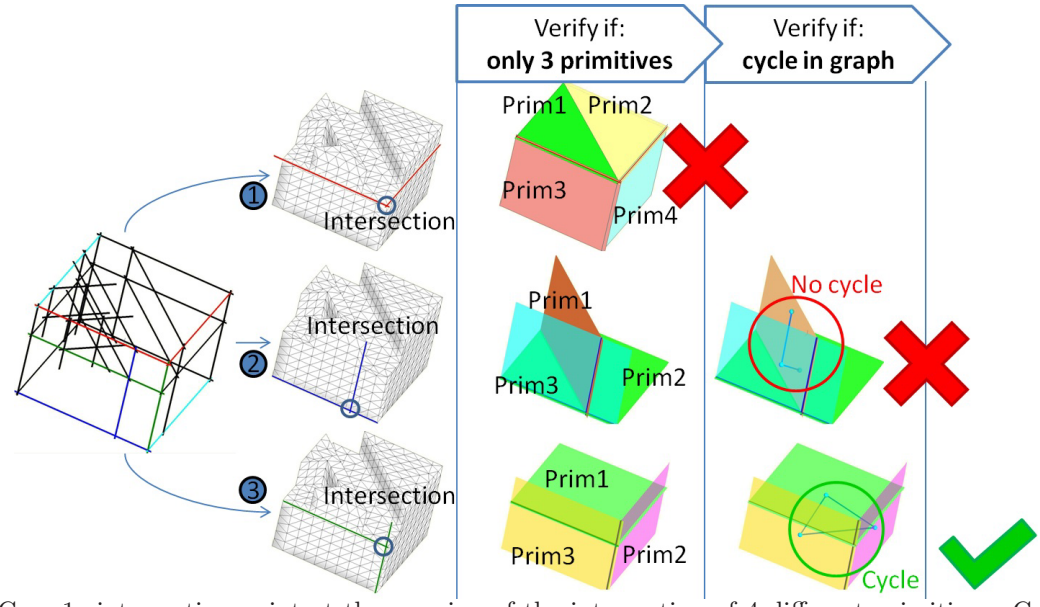


Figure 6: Case 1: intersection point at the crossing of the intersection of 4 different primitives. Case 2: intersection point at the crossing of 3 geometric primitives but not link in the *adjacency graph*. Case 3: intersection point at the crossing of 3 adjacent primitives so it is a *junction*.

As a same *junction* can be extracted from several intersection curve pairs, a fusion is performed when two *junctions* correspond to a same vertex. After the *junction* extraction and fusion, the *edges* are created by cutting the valid intersection curves. To construct the *wires*, a closed path through the *edges* is constructed; so if an *edge* has an extremity which is not connected with an another *edge* extremity, this *edge* can not belong to a closed path. All of these *edges* are removed. Thus a set of valid *edges* is obtained as shown in the Figure 7.

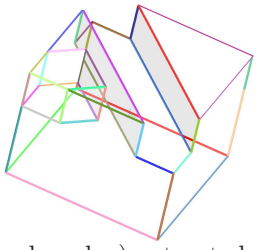


Figure 7: Valid edges (one color by edge) extracted from the 3D mesh of Figure 5.

3.4 Wire Construction

To build the *wires*, one exterior and none or several interiors for each geometric primitive, closed paths going each through a subset of *edges* have to be find. In fact, there are two cases: a *wire* can be created with only one way with the *edges* (like for the plane in the Figure 8) or several paths are possible to create a *wire* (like for the spheres in the Figure 8).

In the second case, for each *edge* a weight is attributed. This weight corresponds to the average distance between the *edge* (Figure 8(a)) and the *extended areas* (Figure 8(a)), so the best *edge* have the minimal weight. To find a path closer to the optimal, we use a simple sequential method. The *wire* construction is initialized with the *edge* having the lowest weight. Then, the connected *edges* are studied and the one with the lowest weight is selected and connected to the current *wire*. The process finish when the *wire* is closed or if there is no more *edge* to connect. The *wire* is valid and kept in the first case but rejected in the second case. Thus, the *wires* of the Figure 8(c) are constructed.

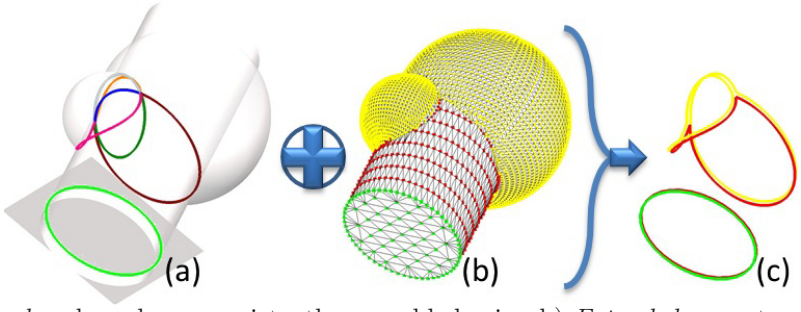


Figure 8: a) Valid *edges* have been consistently assembled using b) *Extended areas* to create c) The *wires*.

This building process ensures that all *wires* have a valid topology. They are closed and cannot self-intersect (otherwise, there will be a supplementary *junction* on the self-intersection). After the B-Rep creation, we can check the consistency of the *wire*, by assessing that the B-Rep model is closed: all the faces are entirely delimited by the edges.

3.5 B-Rep Model Creation

Once the *wires* have been constructed, they are combined with the geometric primitives and the *adjacency graph* to reconstruct the B-Rep model (see Figure 9). The B-Rep model is composed, for each geometric primitive, of its type, its parameters, and the corresponding *wires* (one outer and none or several inners). In fact, each *edge* is stored once and the *wires* reference the *edges*.

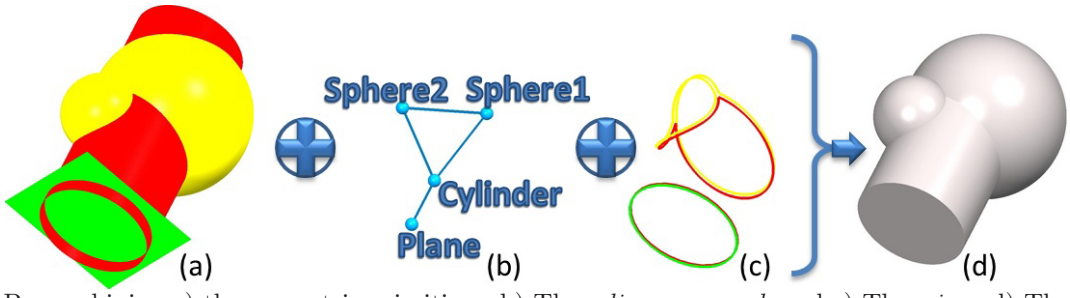


Figure 9: By combining a) the geometric primitives, b) The *adjacency graph* and c) The *wires*, d) The definitive B-Rep model is reconstructed.

The B-Rep structure can be saved easily in the STEP format. Indeed, this format has the same structure as a B-Rep model, some informations are specific to each face, like the parameters or the *wires*, and some informations are common to several faces, like the points or the *edges*. The Figure 10 presents a STEP simplified structure of our test object. Each entity is defined and identified by a number which allow, for example, to describe the *edge* 1, labeled *#6*, only once and to reference its several times, in the plane *wire*, labeled *#5*, and in a cylinder *wire*, labeled *#12*.

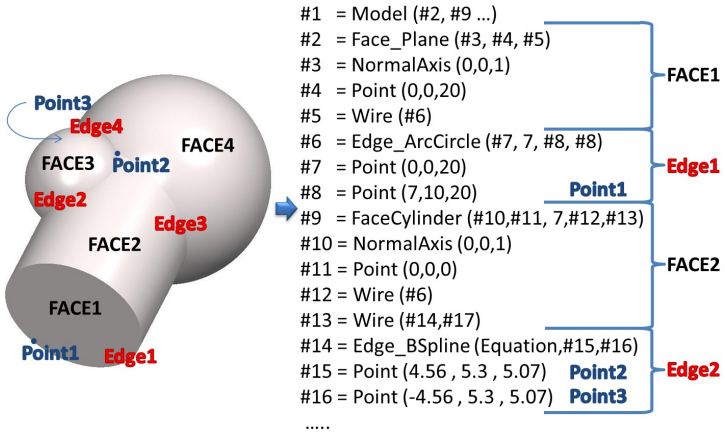


Figure 10: Presentation of a STEP structure for our test object, some informations are specific to each face and some informations are common to several faces.

4. EXPERIMENTAL RESULTS

4.1 Implementation

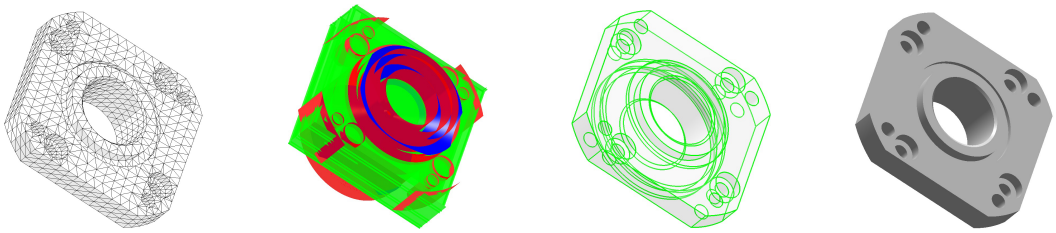
The method was implemented in the *3D Shop* software produced by *C4W*[†] company. The program only requires to tune some parameters. These parameters could be defined automatically, based on the characteristics of the mesh (maximal or mean edge length). Once the process is started, it runs entirely automatically and there are no interaction with the user.

4.2 Tests on simple 3D meshes

First tests are performed on two CAD meshes, both with a low complexity but with very different structures. The first one, Figure 11, contains planes, cylinders and cones, and the *wires* correspond essentially to its intersections between two primitives only. The second, Figure 12, was created with planes and cylinders but the *wires* are formed by the intersection of many primitives (for instance 6 planes to the cylinder of the screw head).

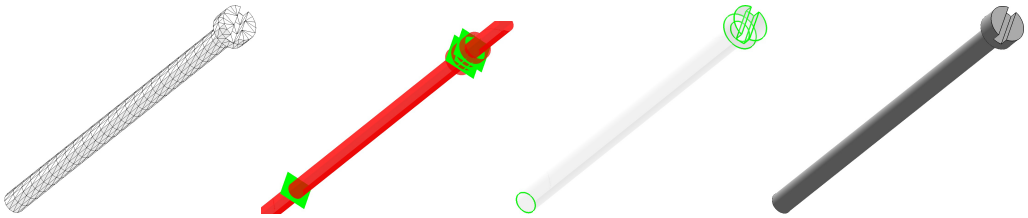
[†]www.c4w.com

In the two cases (Figures 11(b) and 12(b)), all the primitives are extracted. After the computation of *wires* (Figures 11(c) and 12(c)), using weights in the second case, the B-Rep model is reconstructed (Figures 11(d) and 12(d)).



(a) (b) (c) (d)

Figure 11: B-Rep reconstruction for *PieceMotor*: a) 3D mesh (1,746 vertices and 3,524 triangles), b) Extracted geometric primitives (2 cones, 22 cylinders and 15 planes), c) Reconstructed *wires* and d) The final B-Rep model.

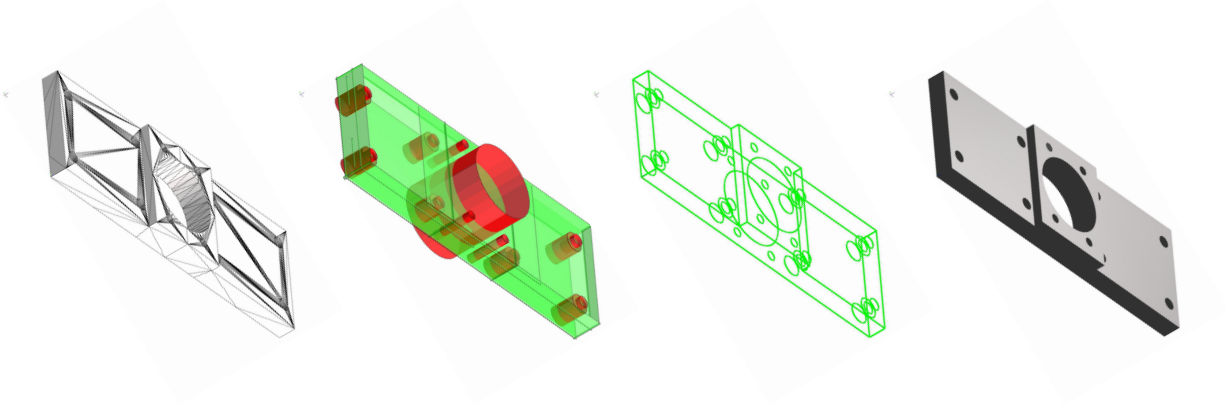


(a) (b) (c) (d)

Figure 12: B-Rep reconstruction for *Screw*: a) 3D mesh (966 vertices and 1,928 triangles), b) Extracted geometric primitives (1 cone, 6 cylinders and 71 planes), c) Reconstructed *wires* and d) The final B-Rep model.

4.3 Tests on real 3D meshes

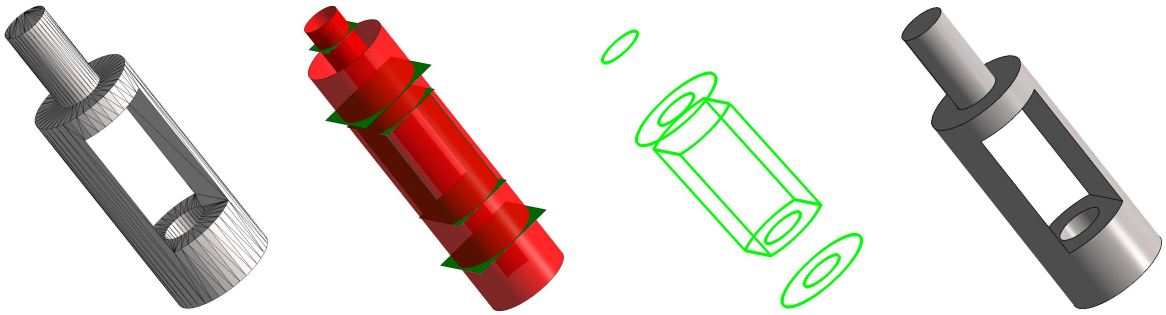
A second series of test is performed on two real CAD objects: a plate and a cylindrical adapter which are parts of the I4L parallel robot designed at LIRMM.[9] The CAD models were designed and discretized by using *Solidworks 2010*. The reconstruction method is applied to these meshes and gives the B-Rep models presented in Figures 13 and 14.



(a) (b) (c) (d)

Figure 13: B-Rep reconstruction for *Plate*: a) 3D mesh (1,585 vertices and 3,220 triangles), b) Extracted geometric primitives (21 cylinders and 18 planes), c) Reconstructed *wires* and d) The final B-Rep model.

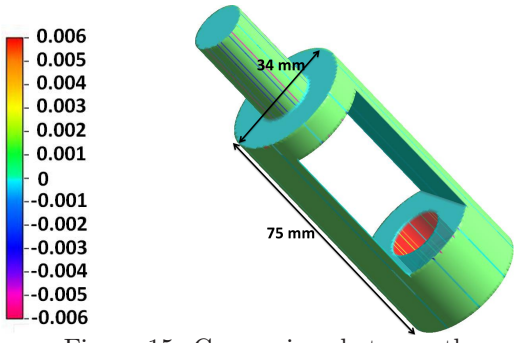
In Figure 15, a quantitative analysis of the reconstruction is presented. The distance between the initial 3D mesh and the B-Rep model is computed. For this, the resulting B-Rep model is discretized very densely and



(a) (b) (c) (d)

Figure 14: B-Rep reconstruction for *CylindricalAdapter*: a) 3D mesh (268 vertices and 540 triangles), b) Extracted geometric primitives (3 cylinders and 7 planes), c) Reconstructed *wires* and d) The final B-Rep model.

the distance between these points and the original mesh is obtained. In this example, the mean distance is very low and the maximum distance (0.052 mm) remains very small with respect to the total length of the object. Furthermore, the maximal error is located along the mesh edges. We conclude that the error is mainly due to the discretization process which generates minimal errors on planar parts and maximal error on salient parts and not to the reconstruction method itself.



Distance between the original mesh and the recovered B-Rep model.

Minimum	Mean	Maximum
0.000	≈0.000	0.052

Figure 15: Comparison between the recovered B-Rep model and the initial mesh.

4.4 Analysis of the computation time

The method was tested on a standard computer with an *Intel Core 2 Duo 2.33GHz processor and 4Gb RAM*. Computation time of the comprehensive method is presented in Table 1 for the different 3D meshes. The computation time does not only depend on the number of triangles. Thus, the reconstruction takes more than 19 seconds for the *PieceMotor* mesh which have 3,524 triangles whereas it takes only 2 seconds for the *PlaqueReconstruction* mesh which is composed of 3,220 triangles.

In fact, the computation time is related to the complexity of *wires*. For example, the reconstruction of the *Screw* mesh (854 triangles, 9 geometric primitives) takes much more time that for the *Plate* mesh (3,220 triangles and 39 geometric primitives). In particular, for each *edge*, the weight has to be computed and then the best path must be found. So the operation which influences the most the overall computation time is the creation of the topology and the construction of the *wires*.

Mesh	Nb of Triangles	Nb of Primitives	Time
PieceMotor (Figure 11)	3,524	39	19s
Screw (Figure 12)	854	9	4s
Plate (Figure 13)	3,220	39	2s
Cylinder adapter (Figure 14)	540	10	1s

Table 1: Computation time (*Intel Core 2 Duo 2.33GHz processor, 4Gb RAM*).

5. CONCLUSION

In this paper, we have presented a method to construct a consistent topology from a set of geometric primitives and the corresponding 3D mesh. This method is composed of three steps: the determination of an *adjacency graph* which represents the neighborhood relationship between the primitives; the wire construction which computes the intersection curves between the primitives in a consistent way by using the *adjacency graph* and the model creation which fuses all the results to build a B-Rep representation. This method gives good results with CAD meshes, independently of the mesh structure which can be composed of a lot of geometric primitives.

Our topology construction is quite robust, problems appear only if there are errors in the primitive extraction. Indeed, the imprecision of the primitive detection raises many problems on the accuracy of the computation of intersection curves. So, improvements have to be done in the primitive extraction, to be sure that all primitives are extracted with a low error. An idea could be to find and add many constraints, like tangency constraints, on the relationship between the primitives[10, 11] to robustify the reconstruction process.

Nevertheless, the topology reconstruction can be also improved to be faster. We have notice, that the operation taking more time is the *edges* assembling to create *wires*. Extract the best path through a valuated graph is a classical problem and several algorithms can be used to find an optimal result in a shorter time.

REFERENCES

[1] Bénière, R., Subsol, G., Gesquière, G., Le Breton, F., and Puech, W., "Recovering primitives in 3D CAD meshes," *SPIE Electronic Imaging 2011, 3D Imaging, Interaction and Measurement* **7864**, 7864 0R–1–9 (2011).

[2] Várady, T., Martin, R., and Cox, J., "Reverse engineering of geometric models–an introduction," *Computer-Aided Design* **29**(4), 255–268 (1997).

[3] Benkö, P., Martin, R., and Várady, T., "Algorithms for reverse engineering boundary representation models," *Computer-Aided Design* **33**(11), 839–851 (2001).

[4] Huang, J. and Menq, C., "Automatic CAD model reconstruction from multiple point clouds for reverse engineering," *Transactions of the ASME* **2**, 160–170 (2002).

[5] Chappuis, C., Rassineux, A., Breitkopf, P., and Villon, P., "Improving surface meshing from discrete data by feature recognition," *Engineering with Computers* **20**, 202–209 (2004).

[6] Requicha, A. and Voelcker, H., "Boolean operations in solid modeling: Boundary evaluation and merging algorithms," *Proceedings of the IEEE* **73**(1), 30–44 (1985).

[7] Miller, J. R., "Incremental boundary evaluation using inference of edge classifications," *IEEE Computer Graphics & Applications* **0272**(17), 71–78 (1993).

[8] Patrikalakis, N., Maekawa, T., and Mukundan, H., "Surface to surface intersections," *Computer Graphics and Applications, IEEE* **13**(1), 89–95 (1993).

[9] Krut, S., Company, O., Benoit, M., Ota, H., and Pierrot, F., "I4: A new parallel mechanism for scara motions," *Proc. of ICRA 2003: International Conference on Robotics and Automation, Taipei, Taiwan* , 1875–1880 (2003).

[10] Benkö, P., Kós, G., Várady, T., Andor, L., and Ralph, R. M., "Constrained fitting in reverse engineering," *Computer Aided Geometric Design* **19**(3), 173–205 (2002).

[11] Li, Y., Wu, X., Chrysathou, Y., Sharf, A., Cohen-Or, D., and Mitra, N., "Globfit: consistently fitting primitives by discovering global relations," *ACM Transactions on Graphics (TOG)* **30**(4), 52:1–52:12, ACM (2011).