Universitat Politècnica de Catalunya
Master Erasmus Mundus in Computational Mechanics

International Center for Numerical Methods in Engineering

Simplification of Surface Mesh Using Hausdorff Distance

by

Sébastien Terrana

Thesis Proposal
Advisor: Josep Sarate

Barcelona, June 2010

Abstract

Simplification of Surface Mesh Using Hausdorff Distance

Sébastien Terrana

In this Master Thesis, we implement and and improve a new method for surface mesh simplification based on the utilisation of the Hausdorff distance introduced by the article (Borouchaki and Frey, 2005). The objective is to reduce the number of triangles of an initial surface mesh while preserving the geometry represented by the initial mesh as well as the shape quality of the resulting mesh. Two tolerance parameters with respect to the reference mesh will be introduced in order to preserve the geometry of the surface throughout the simplification process. The reference mesh is then simplified and optimized so that the resulting mesh respects these tolerances parameters.
The algorithm of this simplification method will be described first, then we will highlight the main improvements that we add to the initial method of (Borouchaki and Frey, 2005). Because we implemented this method using two diferents langages, we will discuss some problem raised by that double implementation. Several examples of surface meshes will be finaly provided regarding different application areas in order to assess the efficiency of our simplification method.

Acknowledgments

I personally acknowledge my advisor the Professor Sarrate who guided me patiently throughout my thesis. He spent a lot of his time explaining me some implementation details and leading all the steps of my thesis. I specially thank him for his avaibility. I am also very grateful to Doctor Xevi Roca who helped me to overcome some mathematic and in implementations difficulties.

I acknowledge as well Doctor Frank Ledoux, professor and informatician from the french atomic energy research center, the CEA, where I spent six month to do my internship. M. Ledoux was my supervisor there and he gave me a lot of explanations and advices about the implementation in C++ using GMDS. He also allowed me to spend my working time for writing this master thesis report.

Finally I thank Cesar Rivas and Hannes Schuemann, two of my classmates who helped me a lot to learn respectively about C++ and LATEX. I am also very grateful to P. Allal who helped to create good working conditions at the CEA. I will never forget Lelia Zielonka, who has done all the administrative support that I needed throughout the two years of master time.

Contents

List of Figures

Chapter 1

Problem statement

1.1 Importance of mesh coarsening

Surfaces can be defined essentially in three different ways for geometric modeling of objects depending on the application foreseen: computer graphics, finite element computations, and physical iso-surfaces.

The first way, mostly used in CAD modelers, consists in defining a set of conformal parametric patches, a complex object then requires a large number of such patches to be accurately described. In most cases, a patch dependent algorithm will preserve the contours of each patch. For finite element computation however, it is often desirable to eliminate redundant information as well as small geometric features of the object, either from the CAD model or directly from the mesh.

The second way consists in defining the surface via a triangulation. Then a mesh is created using a reconstruction method based on a set of sampled surface points (e.g. provided by sensing devices) or on volumetric data (e.g. provided by scanning devices). Obviously, the accurate definition of a complex object requires a high-accuracy data acquisition, thus leading to an unnecessary large number of mesh elements. Typically modern data acquisition techniques lead to meshes containing of about several millions of elements. Finally, the third way consist in getting a mesh from a previous computation of an iso-surface obtained after a langrangian computation. Indeed, an iso-surface is computed from cartesian grid cutted according to the concentrations of the neighbouring cells. In that case, the elements of that iso-surface may be very flat and have a poor quality, according to the angle the interface meets the grid. Moreover, the previous step may have needed a very important number of elements, which is no longer necessary for the the modelization of the surface itself, therefore a coarsening is needed.

1.2 Main challenges

As stated above, it is usually desirable to significantly reduce the complexity of the discrete surface representation (i.e., the mesh size) while preserving, as much as possible, the geometry of the object. In other words, the challenge is to eliminate the geometric redundancies while not altering the geometric properties of the model.
Modern mesh simplification methods attempt to answer these two requirements. Actually, they are mostly based on the optimization of geometric criteria. The applications potentially concerned range from computer graphic to numerical simulations (e.g. finite element computations), including scientific visualisation, data compression, etc...
A surface mesh (where element nodes are considered to belong to the surface) is considered as geometrically suitable if the two following conditions are satisfied:

- all mesh elements should be close to the surface. That *proximity* property allows to bound the gap between the elements and the surface. This gap is the measure of the largest distance between any point of an element and the surface.
- each mesh element should be close to the tangent plane of the surface at its nodes. That second *smoothness* property ensures that the surface is locally C^1 continuous. Hence, the angular deviation between the element and the tangent plane at its vertices shall be bounded.

Moreover we can state what is an *optimal* surface mesh. It is a mesh such that the elements are close to regular or equilateral (which is an essential requisite for most numerical applications).

1.3 A new approach of mesh coarsening

The particularity of this approach is that we do not coarse a mesh with the knowledge of any underlying analytic geometry. We just know the initial mesh and we try to coarse it without having idea of the exact geometry, and then witout having any idea of how far we can be from the exact geometry during the collapse process.
This is actually an important difference, with the traditional approach because it is usually very easy to define a distance or a measurement from a mesh to an analytical geometry. Indeed once the analytical geometry is known, the distance from the mesh's nodes to the geometry is known, as the difference between the exact normal at one point and the normal of the neighbouring, then a measurement of the difference between the 2 configurations can be established. This traditionnal approach has been widely developped by (Frey and Borouchaki, 1997) and as a part of the mesh optimization process in the article (Hoppe et al., 1993).
It become more complicated on our situation when the analytical geometry is unknown. How to define and measure these differences? Moreover, this measurements have to be able to capture the

defects of any unwanted coarsed mesh (for exemple spurious variations produced by the collapse). Then how these deffects can be measured?
To answer these questions, we will use for our Master Thesis the solution presented by the paper (Borouchaki and Frey, 2005) as a theoretical basis for the implementation of a mesh simplification algorithm. The main idea of H. Borouchaki and P. Frey is that the Hausdorff distance can be used as a way to measure and control the evolution of the mesh during the simplification process. The Hausdorff distance is a *pure* geometrical way of controling the degeneration of the mesh. Several methods quite fast has been developped using geometric controls, for instance (Hamann, 1994) uses polynomials approximations neighbourhood of the triangles suppressed.

1.4 General scheme of the simplification method

In this thesis, we present a new mesh simplification method based on the discrete evaluation of the Hausdorff distance between two meshes, notably inspired by the ideas developed in (Cohen et al., 1996) and in (Borouchaki and Frey, 2005). Schematically, it consists in three stages. At first, a global tolerance envelope is defined around the surface as well as a local tolerance cone centered at each node of the reference surface mesh. Then, the edges of the initial surface mesh are iteratively analyzed and eventually removed if the resulting elements do not violate the tolerance requirements. Actually, the tolerance areas are introduced to enforce the proximity and the smoothness properties introduced hereabove. Finally, after each node or edge removal operation, the current mesh quality is optimized with respect to the element shape measure using edge flipping and point relocation procedures, provided the geometric accuracy is preserved.

Chapter 2

Description of the method

In ths part we will assume (without any lack of generality) that the initial reference mesh is only defined by the list of its nodes' coordinates and the list of its triangular elements. The mesh simplification procedure will now be described as it was exposed by (Borouchaki and Frey, 2005) in the case of smooth surfaces (i.e. with a sufficient geometric continuity order). In particular, the general scheme of the method is explained in details and the extension to the case of surfaces presenting geometric non-smooth (first spatial derivatives discontinuous) will be discussed in the chapter 3.

2.1 Control parameters

2.1.1 Importance of the control parameters

Before introducing the method itself, it is quite important to define the parameters used by our method. The proposed method enables to obtain a simplified mesh, associated with a given set of control parameters that allows to quantify, on the one hand, the desired level of geometric approximation and, on the other hand, the allowable mesh quality degradation. In order to control the progressive coarsening of the mesh, and in order to allow the creation of bigger elements, three main *control parameters* will be used:

- the authorized Hausdorff distance δ
- the angle of tolerance cone at each of the nodes θ
- the authorized degradation of the qualities of the triangles β

The basic idea of the theory developped by P. Frey and H. Borouchaki is that the elements can be collapsed as long as the suppressions satisfy the three control parameters. The parameters (δ, θ) usually are increased step by step while β remains constant.

2.1.2 Mathematical definiton of the Hausdorff distance

As explained in the previous section, the control of the simplification process should rely on a local tolerance property about the *proximity* between the current simplified mesh and the reference mesh. The main problem is then *how* to define a distance between two meshes. And the clever answer given by (Borouchaki and Frey, 2005) is to use the Hausdorff distance to evaluate that distance.
To introduce this new distance, let us first recall the definition of a distance $d(X, F)$ from a point X in $\mathbb{R}^3$ to a closed bounded set F in $\mathbb{R}^3$:

$$d(X, F) = \inf_{Y \in F} d(X, Y) \tag{2.1}$$

where $d(.,.)$ denotes the usual Euclidean distance. Let F_1 and F_2 be two closed bounded sets in $\mathbb{R}^3$ and let denote $\rho(F_1, F_2)$ the quantity:

$$\rho(F_1, F_2) = \sup_{X \in F_1} d(X, F_2)$$

then, the Hausdorff distance $d_H(F_1, F_2)$ between F_1 and F_2 will be defined as:

$$d_H(F_1, F_2) = \max\left(\rho(F_1, F_2), \rho(F_2, F_1)\right) \tag{2.2}$$

This distance is especially designed for set of points, which is a more basic structure than a real mesh, but we will see later that this distance is actually perfectly relevant for meshes as well.
Once this distance is defined, a global tolerance region can be defined around the reference surface mesh $\mathscr{T}^{\text{ref}}$ at a given distance δ on both sides of the reference surface. Thus, giving a tolerance distance δ is geometrically like giving a tolerance layer (i.e. an *enveloppe*) of width 2δ around the initial surface and centered on that surface. This distance δ can be expressed in percentage of the minimal bounding box size containing the initial mesh.
Now we can state that each triangle K resulting from any mesh modification of the *reference mesh* $\mathscr{T}^{\text{ref}}$ must belong to that proximity envelope, that is expressed by the following formula :

$$\boxed{d_H(K, \mathscr{T}^{\text{ref}}) \leq \delta} \tag{2.3}$$

This statement makes the parameter δ a *control parameter* of the proximity between $\mathscr{T}^{\text{ref}}$ and the current modified mesh $\mathscr{T}$.
We shall notice that the exact computation of the Hausdorff distance $d_H(K, \mathscr{T}^{\text{ref}})$; is extremely costly. Indeed, the Hausdorff distance needs the computations of all the distances between the nodes of K and all the nodes of the mesh $\mathscr{T}^{\text{ref}}$, which is very expensive. We will see in the section 2.4 that a very cheap local approximation (actually an upper bound) of this quantity can be computed.

The figure 2.1 is an attempt to visualize geometrically the Hausdorff distance. The figure (a) represents what would be the Hausdorff distance $d_H(M_1, M_2)$ between two simple meshes : M_1 and M_2. One can see that both meshes are quite flat and parallel, but M_2 has something like a peak. The maximum distance between the two meshes will be then the distance between a node on the peak of M_2 and its closest node on the mesh M_1. The figure (b) represents the Hausdorff tolerance layer of thickness 2δ around a mesh. Note that the layer is of course centered on that mesh.

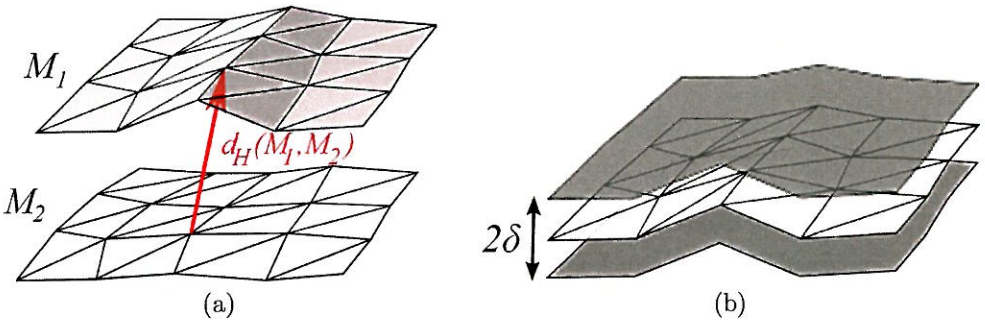


Figure 2.1: (a) is a representation of the Hausdorff distance between two meshes and (b) is a representation of the tolerance layer around an initial mesh.

2.1.3 Advantages and drawbacks of the Hausdorff distance

The use of a method using the Hausdorff distance is particularly appropriated to the mesh coarsening. Indeed, the Hausdorff distance can be computed iteratively quite easily, which allows the users to follow the mesh degeneration during the process and to bound the coarsed mesh between a layer of thickness 2δ centered on the initial mesh. Nevertheless, the Hausdorff enveloppe is unable to prevent our algorithm to produce some crappy situation with the elements placed like *stairs*. Indeed, the Hausdorff criterion is a pure distance mesure, and it does not impose any constraint on the orientation of the elements, which then can be completely discontinuous. This dificulty is illustrated by figure 2.2. On this 1D-mesh example, the initial configuration on the left hand side is drawn with the Hausdorff envelop. On the middle, a mesh collapsed (and relocalized) whith elements inside the Hausdorff layer and with some smoothness for the direction of the elements. On the right hand side, another mesh collapsed also inside the Hausdorff layer but with irregularities in the direction of the elements. This last mesh is, obviously, unacceptable, but it satisfies the Hausdorff criterion. This leads to the necessity of another criterion on controlling the orientation of the elements of the mesh. This control is the tolerance angle θ described in the following section.

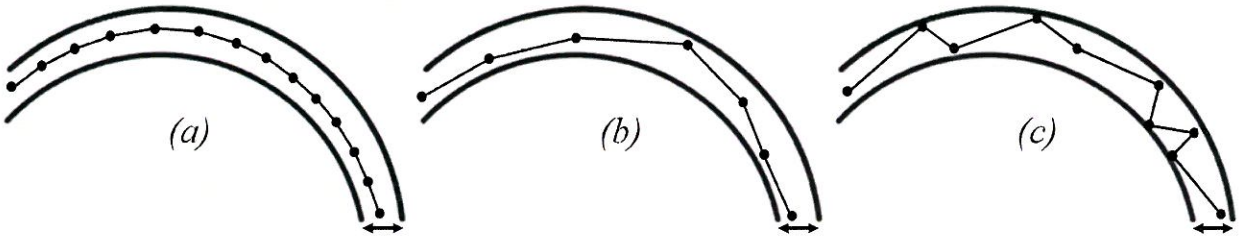


Figure 2.2: (a) Mesh before collapse; (b) a smooth collapsed mesh; (c) a sharp collapsed mesh.

2.1.4 Tolerance angle

The angular tolerance is specially designed to enforce the *smoothness*, or the *regularity* property of the mesh. Geometrically, this tolerance angle can be represented by local cones centered at each node of the mesh and with an aperture θ as drawn in figure 2.1.4. The normals of the nodes, called *pseudo-normals* are computed as described in section 2.4.4 from the normals of the adjacent triangles. That pseudo-normal is computed in the initial configuration, and defines the principal axis of this cone. The aperture angle θ of that cone is given by the user.

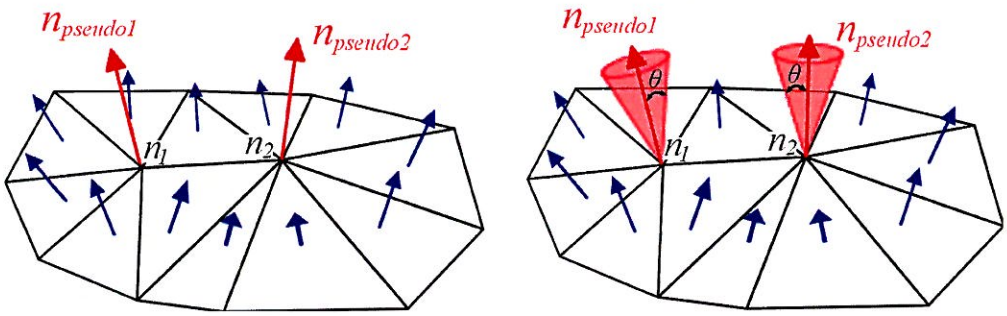


Figure 2.3: Tolerance cones around the pseudo-normals. The normals of the triangles are represented in blue.

The pseudo-normal represents the normal of the geometry at the considered node, and therefore this is a representation of the tangent plane of the geometry at that node. This angular control ensures that the modified triangles remain close to the tangent planes at each of its three vertices. The mathematical expression of that constraint on all the triangles K of the mesh is:

$$\boxed{\langle n_{pseudo-k}(K), n(K)\rangle \geq \cos\theta \qquad \forall k} \tag{2.4}$$

Where $n(K)$ denotes the unit normal to triangle K (drawn in blue on the figure) and $n_{pseudo-k}(K)$ denotes the pseudo-normal to the surface at the k^{th} node of K. ($\langle .,. \rangle$ is the standard scalar product in $\mathbb{R}^3$).

2.1.5 Authorized degradation

The control parameter β which controls the authorized degradation of the mesh's triangles is not a *pure* geometrical control of the modified mesh unlike δ and θ are. It is more a parameter that controls the shape quality of the triangles, independently from the geometry of the initial mesh. The expression of quality $q(.)$ choosen in this thesis is a monotonous shape quality measure whose expression, for a triangle K is given by:

$$q(K) = c\frac{S(K)}{\sum_{e(K)} l^2(e(K))} \tag{2.5}$$

With $S(K)$ being the measure of the surface of K, $e(K)$ is an edge of K, $l(e(K))$ is the euclidean length of edge $e(K)$ and $c = 4\sqrt{3}$ is a coefficient designed to have a quality equal to one (resp. zero) for an equilateral (resp. flat) triangle. With this mesure of the quality, we will have $q(K) \in [0;1]$ with the maximum obtained for an equilateral confugation. Figure 2.1.5 represents some examples of triangular shapes and the quality related.

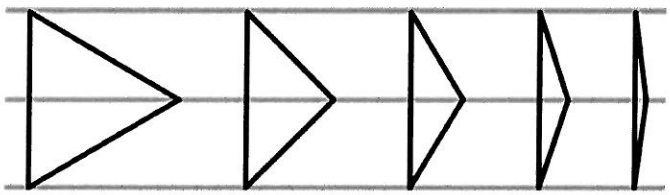


Figure 2.4: From left to right, the related qualities are $q = 1$, $q = 0.7$, $q = 0.5$, $q = 0.2$ and $q = 0.1$

The idea of controling the authorized degradation is that the new triangles created or modified during the modification of the mesh should keep a quite good quality compared too the triangles of the reference mesh K_{ref}. That can be translated by the condition on the quality of the newly created triangle $q(K_{new})$:

$$q(K_{new}) \leq \beta q(K_{ref}) \qquad \forall K_{new} \tag{2.6}$$

This condition will be implemented and developped with more details by the section 2.4.2.

2.1.6 Tolerance incrementation

To summarize, the two tolerance parameters δ and θ, define two tolerance regions (cone and layer) which allow to control the *proximity* of the surfaces between the initial mesh and the simplified mesh as well as to ensure the geometric *regularity* (or smoothness) of the resulting mesh. The last tolerance parameter β keeps the quality in a good shape.
Thus, the nodes are suppressed as long as the two first conditions on the mesh 2.3 and 2.4 are satisfied. That means that, any triangle K resulting from the modification of $\mathscr{T}^{\text{ref}}$ must comply with the two following relations throughout all the modifications:

$$\begin{cases} d_H(K, \mathscr{T}^{\text{ref}}) \leq \delta \\ \langle n_k(K), n(K) \rangle \geq \cos\theta \qquad \forall k \end{cases} \tag{2.7}$$

With $n_k(K)$ is the light notation for the pseudo-normal $n_{pseudok}$ computed at the node k^{th} of the triangle K. ($\langle ., . \rangle$ is the usual standard scalar product in $\mathbb{R}^3$).
The suppression/modification of the mesh continues as long as the triangles can be suppressed while respecting the two precedent conditions. Once the modifications can no longer occur, the tolerances are relaxed and the control parameters increase : δ becomes $\delta + \Delta\delta$ and θ becomes $\theta + \Delta\theta$. A new step of suppression and modification occurs and then the tolerance parameters are incremented again etc... until the maximum values δ_{max} and θ_{max} are reached.
Now we have the main idea that constitutes the skeleton of the algorithm of our programm which is presented in the following section.

2.2 Overview of the general algorithm

2.2.1 General algorithm

From a practical point of view, we consider a relaxation of the first two parameters and we carry out the mesh modifications in an iterative manner. The mesh simplification algorithm can be described as following:

The proposed method mainly consists in iteratively removing and optimizing all mesh edges until the modifications can no longer occur. To this end, two local mesh modification operations are involved: the node identification (the two endpoints of an edge are merged together) and the edge flip. The first procedure, while simple to describe, is rather efficient to carry out on mesh edges, as the two properties (i.e., proximity and regularity) can be explicitly checked a priori, without actually doing the modification on the mesh structure. During the procedure of removal, a mesh edge is removed and a new mesh (known a priori) is locally constructed. The operation is carried out if, on the one hand, the new mesh preserves the geometry of the surface and, on the other hand, if the

Algorithm 2.1 Global Algorithm of the method

1: **procedure** `Algorithm`(β,δ_{max},θ_{max},N_{iter})
2: Set consistancy in the orientation of all the triangles of $\mathscr{T}^{\text{ref}}$ ▷ Pretreatment
3: Get boundary nodes and sharp nodes. ▷ Pretreatment
4: Edge swap and node relocation. ▷ Pretreatment
5: Set $\Delta\delta = \delta_{max}/N_{iter}$ and $\Delta\theta = \theta_{max}/N_{iter}$. Set $\delta = \Delta\delta$ and $\theta = \Delta\theta$.
6: **for** $i = 1$ to N_{iter} **do** ▷ N_{iter} is choosen by the user
7: **while** Elements of $\mathscr{T}$ are suppressed **do**
8: **if** the geometry (δ, θ) and the degradation β are preserved **then**
9: Remove and optimize all edges in $\mathscr{T}$
10: **end if**
11: Optimize the edges using the edge swap.
12: **end while**
13: **if** the geometry (δ, θ) is preserved **then**
14: Relocate the nodes of $\mathscr{T}$
15: **end if**
16: Increment $\delta = \delta + \Delta\delta$ and $\theta = \theta + \Delta\theta$
17: **end for**
18: **end procedure**

shape quality of the mesh elements is not too much degraded. When all the mesh edges have been removed, the second procedure (edge swap) is checked with each mesh edge and eventually applied if the mesh quality is improved. These steps (edge collapse and edge swap) are finally followed by a node relocation procedure to optimize the elements shape quality.
The user gives only the parameters δ_{max}, θ_{max}, β and the number of iterations N_{iter} to the program. Then at each iteration the tolerance parameters δ and θ are incremented (line 16) until they reach δ_{max} and θ_{max}

2.2.2 Order of the operations

Even if it is not formally required, it is very usefull to sort all the edges according to their quality so that the worst edges, in term of quality are deleted first. Note that the *quality* of an edge is simply given by the minimum quality of the two triangles sharing that edge, because an edge itself cannot have a quality shape.
The edge removal (or edge suppression) is made first. During that part of the algorithm, the quality of the triangles surrounding the suppressed edges are very deteriorated. The edge swap and the node relocation are made after that step of suppression in order to finally increase the quality shape. We have choosed to perform the collapse and the swap separately (the swap start once the collapse procedure is finished) and not at the same time as it is suggested by (Borouchaki and Frey, 2005). Indeed, if the two operations are performed simultaneously, the computational cost is decreased but then some swap that would be necessary are not performed because they are not in the scope of the local operation of collapse.

2.3 Edge collapse

The Edge Collapse procedure is the key part of the Algorithm because in this part is done why the whole program is made for: suppressing edges. But here is also the most sensitive part because the removable edges have to be choosen carefully in order to keep the mesh as close to the initial geometry as possible.

2.3.1 Overview of the edge collapse principles

In this section, we will deal only with *smooth* surfaces (non-smooth geometry will be treated in section 3.2), i.e a surface such that the tangent plane at each point is continuous. Typically, at each point of the surface, an unique normal vector is defined, that characterizes the tangent plane to the surface at this point. Let first consider an initial reference mesh $\mathscr{T}^{\text{ref}}$ of such a surface. We are going to simplify $\mathscr{T}^{\text{ref}}$ using the algorithm defined above.

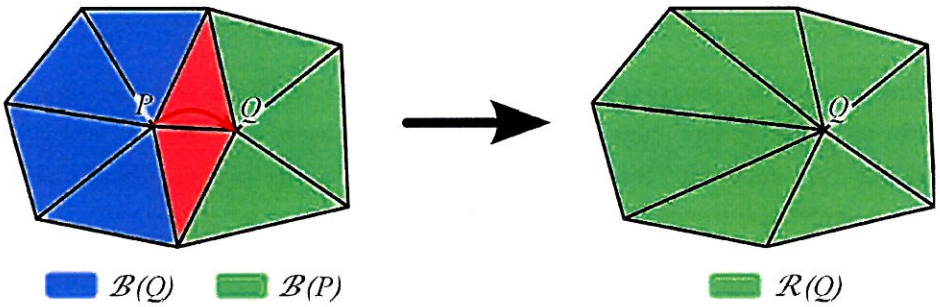


Figure 2.5: On the left, the balls of triangles $\mathscr{B}(P)$ and $\mathscr{B}(Q)$ before the collapse and on the right, the region $\mathscr{R}(Q)$ after P collapsed.

Let us follow the local modifications of the mesh after en edge collapse with figure 2.3.1. Let PQ be an edge in the modified mesh $\mathscr{T}$ between two nodes P and Q. Let $\mathscr{B}(P) = \bigcup_{K \ni P} K$ be the ball of triangles K sharing the vertex P. The collapse of P towards Q (that means the removal of the edge PQ) consists in replacing the triangles $\bigcup_{K \ni P} K$ connected to P by a new set of triangles $\bigcup_{K' \ni Q} K'$ defining the new region $\mathscr{R}(Q)$ around the node Q. By doing the identification of P with Q, the geometry is preserved if each newly created triangle $K' \in \mathscr{R}(Q)$ complies the two following properties inequalities, introduced by the equation 2.7. These inequalities translated in this *local* situation, the *local* reference mesh $\mathscr{T}^{\text{ref}}$ becomes the union of $\mathscr{B}(P)$ and $\mathscr{B}(Q)$, and the new *local* simplified mesh $\mathscr{T}$ becomes $\mathscr{R}(Q)$. In this local situation, the equations 2.7 lead to :

$$\begin{cases} d_H(K', \mathscr{B}(P) \cup \mathscr{B}(Q)) \leq \delta & \forall K' \in \mathscr{R}(Q) \\ \langle n_k(K'), n(K') \rangle \geq \cos\theta & \forall k \in \mathscr{R}(Q) \end{cases} \tag{2.8}$$

We owe some explanations about how the first line of 2.7 becomes the first line of 2.8. If we follow exactly the conditions of 2.7, it is actually very difficult to compute $d_H(K', \mathscr{T}^{\text{ref}})$ because in order to do that, all the distances between the nodes of K' and the full set of nodes of $\mathscr{T}^{\text{ref}}$ should be computed, which is extremely expensive in term of computation, especially if the mesh is big.
The first idea is then to compare the distances between K' and the vertexes of its local neighbouring elements in the reference configuration $\mathscr{T}^{\text{ref}}$. This seems to solve partially the problem because we reduce then the global computation of d_H to a local one, but then two problems appear:

- After several iterations, as the triangles K' may have changed a lot, it become then irrelevant to compare the distances between K' and the local reference patch of $\mathscr{T}^{\text{ref}}$. It becomes even very difficult to find the part of the reference mesh $\mathscr{T}^{\text{ref}}$ which is in the vicinity of a new triangle K' after a significant number of collapses and relocations.

- If we want to compare the distances with the reference mesh, that implies the storage in the memory of the datas of $\mathscr{T}^{\text{ref}}$, which may be undesirable.

To solve this problem, it can be choosen to make the comparaison between the triangles $K' \in \mathscr{R}(Q)$ and the region of $\mathscr{T}$ around K' in the configaration just before the collapse. What is this region? It is actually the union $\mathscr{B}(P) \cup \mathscr{B}(Q)$ which is the ball of triangles around the edge $[PQ]$ before the collapse happens. That gives to us the explanation of the first line of 2.8.
However, a last improvement needs to be implemented. Indeed, if we use the method described above, then the Hausdorff distance is computed only with respect to the immediate *precedent* configuration, and not with respect to the *initial* reference mesh. Therefore the Hausdorff distance of an element of $\mathscr{T}$ should be computed and then stored. And then each time a new Hausdorff distance is computed with our method (i.e with respect to the precedent configuration) it is added with the old distance stored for the corresponding elements to find the real total Hausdorff distance.
Hence, knowing the Hausdorff distances of the triangles created at iteration j to the reference mesh, it is possible to *know*, or at least *bound* the distances of the new triangles, created at iteration $j+1$. With each triangle K in $\mathscr{T}$ is associated an approximation (actualy an upper bound) $h(K)$ of its Hausdorff distance to the reference mesh $\mathscr{T}^{\text{ref}}$). For any newly created triangle K' of the mesh, it can be proved (see (Borouchaki and Frey, 2005), p.4869) that we have:

$$h(K') = d_H(K', \mathscr{B}(P)) + \max_{K \in \mathscr{B}(P)} h(K) \tag{2.9}$$

This relation, which is very important, allows us to compute practically the new approximation of the Hausdorff distance, knowing the one at the previous iteration, and computing a real Hausdorff distance only with respect to the local triangles. Thus, with that relation, the computation of Hd is local instead of being global, and computed with respect to the last previous step instead of the

reference configuration. Then the set of inequalities 2.8 finaly becomes:

$$\begin{cases} h(K') \leq \delta \\ \langle v_k(K'), v(K') \rangle \geq \cos\theta \qquad \forall k \end{cases} \tag{2.10}$$

2.3.2 Edge collapse algorithm

Above is the detailed algorithm of the Edge Collapse procedure:
We add some precisions about this algorithm for a better comprehension:

- The lines 10, 12, 14, 24, 26 and 28 are **if** conditions. Of course, if just one of these conditions is not satisfied, the corresponding node is not collapsed and then another node is considered.
- The order of the condition checked is not made randomly. The condition the most likely to be rejected (quality degradation) are checked first and then the more plausible criterion are checked.
- The lines 20 to 33 are the repetition of the lines 6 to 19 but this time collapsing Q instead of P. Indeed, there is two possibilities to collapse an edge $[PQ]$: sending $P \to Q$ or $Q \to P$
- Then the algorithm chooses which node to collapse if both nodes are able to be collapsed (lines 35 to 42).
- This algorithm is basically the same whatever the language of programmation used (MATLAB or C++). But an important change happens lines 39 and 41 because the way the nodes/edges/-triangles are suppressed depends on how the datas are stored.

In this algorithm, we will not give more details on how exactly the elements of the mesh are suppressed. It will be explained more precisely in section 4.3.1. But in the following subsection, we will get into the problem of updating the elements of the mesh surrounding the suppressed elements.

Algorithm 2.2 Algorithm of the edge collapse

```
1: procedure CollapseEdge(δ, θ)                                   ▷ Comment No1
2:     for j=1 to number of edges do
3:         if the j^th Edge has not been already suppressed then
4:             Compute the Balls of triangles : ℬ(P) and ℬ(Q)
5:             Remove the 2 common triangles
6:             if P is not on boundary or(P and Q) on boundary then
7:                 P is collapsed virtually
8:                 Compute: old q(ℬ(P)), new q(ℛ(Q))
9:                 Check Orientations of the new triangles
10:                if q(ℛ(Q)) ≤ βq(ℬ(P)) and the orientation is good then
11:                    Check Conditions on the Normals
12:                    if the conditions on the Normals are okay then
13:                        Check Conditions on H_d
14:                        if the H_d condition is satisfied then
15:                            bool IsPremovable = true
16:                        end if
17:                    end if
18:                end if
19:            end if
20:            if Q is not on boundary or(P and Q) on boundary then
21:                Q is collapsed virtually
22:                Compute: old q(ℬ(Q)), new q(ℛ(P))
23:                Check Orientations of the new triangles
24:                if q(ℛ(P)) ≤ βq(ℬ(Q)) and the orientation is good then
25:                    Check Conditions on the Normals
26:                    if the conditions on the Normals are okay then
27:                        Check Conditions on H_d
28:                        if the H_d condition is satisfied then
29:                            bool IsQremovable = true
30:                        end if
31:                    end if
32:                end if
33:            end if
34:            If the two config are possible, we choose the one that gives the best quality :
35:            if IsPremovable = true and IsQremovable = true then
36:                q_max = min q(ℛ(Q)), q(ℛ(P))
37:                if q_max ≥ βq_old then
38:                    if q_max = q(ℛ(Q)) then
39:                        P is really collapsed, colnode=P and fixnode=P
40:                    else
41:                        Q is really collapsed, colnode=Q and fixnode=P
42:                    end if
43:                end if
44:            end if
45:        end if
46:    end for
47: end procedure
```

2.3.3 How to update the region after an edge suppression?

For more explanations about the organization of the datas and the links between the differents elements of the mesh, please consult the section 4.2. Let us focus on the situation described by figure 2.3.3. On these pictures, you can observ the edge e which will be suppressed is drawn in red, the two adjacents triangles Tr_1 and Tr_2, the two corresponding fixed edges e_1 and e_2, the collapsed edges e_{col1} and e_{col2}, and finally the two nodes of e : n_{col} and n_{fix}.
For a better visualization of the situation, please see figure 2.3.3 which is a fragmented view of the precedent figure. You can remark the opposite nodes $oppnode_1$ and $oppnode_2$ (they are *opposite* because they are opposed to the edge e) and you can also observ the two triangles Tr_{new1} and Tr_{new2} which share the edges e_{col1} and e_{col2} with Tr_1 and Tr_2.

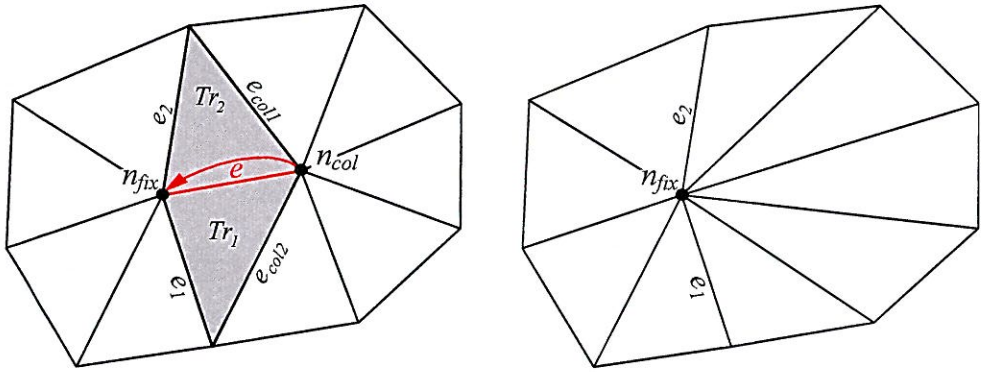


Figure 2.6: The edge e is collapsed: the node n_{col} is sent on the node n_{fix}

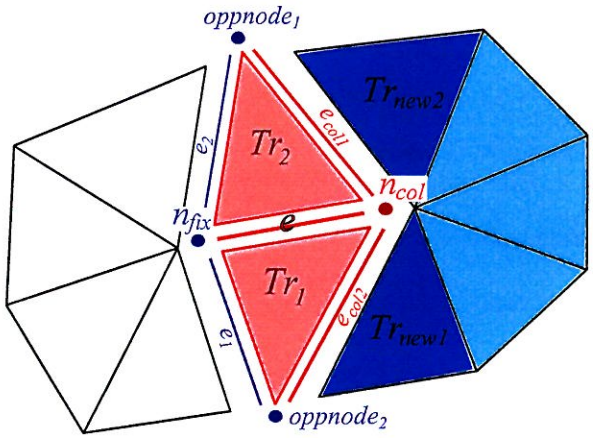


Figure 2.7: The edge e is collapsed: the node n_{col} is sent on the node n_{fix}

The collapse process described in 2.3.3 will lead to the suppression of the following geometric objects drawn in red in figure 2.3.3:

- the node n_{col};
- the edge e and the two lateral edges e_{col1} and e_{col2};
- the two triangles Tr_1 and Tr_2.

The collapse will also lead to some updates in the geometrical objects around the suppressed edge e. These objects (drawn in blue in figure 2.3.3) are not suppressed but their neighbours will change during the collapse process, so the program has to *inform* these objects that they have new neighbours in the following way :

- the node n_{fix} will replace the triangles Tr_1 and Tr_2 which was its neighbours by its new neighbours Tr_{new1} and Tr_{new2};
- the node $oppnode_1$ will remove Tr_1 as a neighbour and $oppnode_2$ will remove Tr_2;
- the edge e_1 will replace its former neighbour Tr_1 for Tr_{new1} and similarly, e_2 will change its neighbour Tr_2 for Tr_{new2};
- the two triangles Tr_{new1} will change its edge $e_{col1} \rightarrow e_1$ and the triangle Tr_{new2} will change $e_{col2} \rightarrow e_2$;
- all the triangles drawn in dark or clear blue will change their vertex $n_{col} \rightarrow n_{fix}$;
- the node n_{col} will also have all the clear blue triangles as new neighbours.

2.4 Conditions for edge collapsing

The set and the order of conditions to be tested before collapsing an edge are very important. Indeed, the edge collapse is the main cause of degeneration of the mesh because some geometrical informations are suppressed during the collapsed. Then the aim of these conditions is to decide which of the nodes can be suppressed without loosing to much usefull information on the mesh. Moreover these conditions should preserve the mesh from overlapping of elements, inversion of orientation, degradation of quality, apparition of sharp geometries etc. in order to obtain a mesh usefull for a possible utilisation in a numerical simulation.

2.4.1 Condition of geometric compatibility

This condition should prevent the formation of overlapped elements, and the change of orientation of the triangles of the mesh. Indeed all the triangles of the reference mesh $\mathscr{T}^{\text{ref}}$) are oriented the same

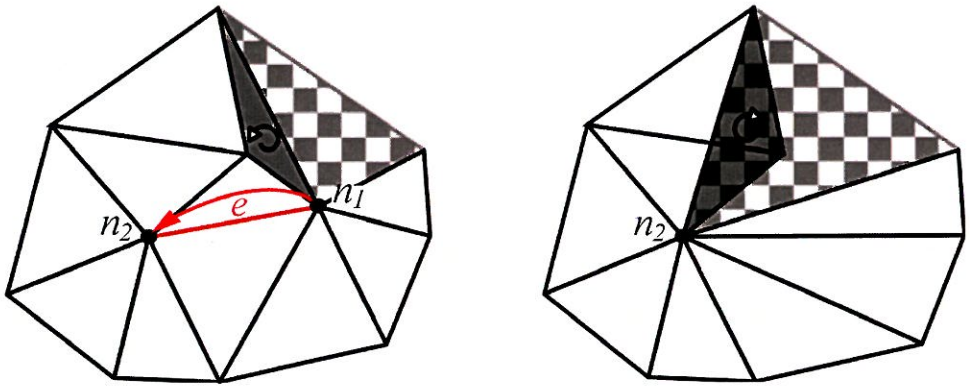


Figure 2.8: Collapsing of the node n_1 onto n_2 producing an overlap and an inversion of orientation.

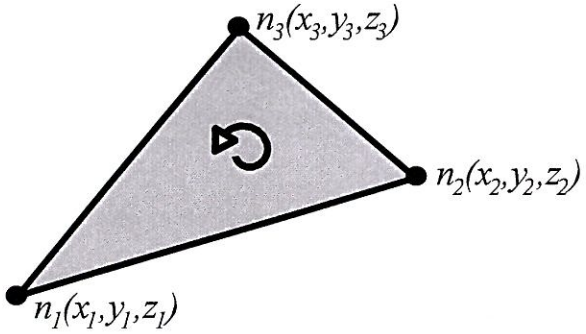


Figure 2.9: Triangle with its coordinates.

way i.e for each triangles, the 3 nodes are stored in an order such that the normal of the triangle computed from these 3 nodes is an outward normal. We have impemented a pretreatment that ensure that all the elements are consistently well oriented. It is very important to keep consistant this orientation all along the process of coarsening because the orientation of the normal is usefull for both our programm and also for the use of the surface mesh to compute fluxes in a numerical simulation. Figure 2.8 represents a case of edge collapse that produces an inversion of orientation of the element painted in grey, and moreover, this element overlaps its neighbor painted with a grid. This configuration should be avoided, using a cheap criterion on the orientation of the newly created triangles. Thus the following condition is used :

For each of the elements surrounding the edge e, after the collapse of that edge, a matrix of principal vectors is computed. Figure 2.9 represents the current triangle whose matrix will be calculated. M

is defined such as :

$$M = \begin{pmatrix} x_2 - x_1 & x_3 - x_2 & 1 \\ y_2 - y_1 & y_3 - y_2 & 1 \\ z_2 - z_1 & z_3 - z_2 & 1 \end{pmatrix} \tag{2.11}$$

Then the determinent $\det(M)$ is computed and its sign gives us the orientation of the triangle. If all the determinants of the triangles around e have the same sign (i.e. all the triangles have the same orientation) then the condition is satisfied.

2.4.2 Condition of reasonable degradation of element's qualities

With this tool to evaluate the elements quality, their degeneration can be controlled during the edge collapse. Even if the procedures of edge swapp and node relocation will later increase the quality of the elements, it is good to have such a control on the quality degradation *inside* the edge collapse procedure. Indeed, the collapse of one edge usually reduces the quality of the neighbouring elements, therefore, after several edge collapses in the same neighbourood, the quality in this area become unwelcoming poor. It is asked then, before each edge colapse, to check if the qualities obtained after an edge collapse will be not too much degenerated. How? The minimum quality of the set ball $\mathscr{B}(P)$ (before edge collpase) should not be to much degenerated compared with the minimum quality of the ball $\mathscr{R}(Q)$ (after edge collpase), which means formally that:

$$\min_{K' \in \mathscr{R}(Q)} q(K') \geq \beta \min_{K \in \mathscr{B}(P)} q(K) \tag{2.12}$$

The coefficient β is the coefficient of quality degradation. $\beta \in [0; 1]$ and on a practical point of vue a relevant value is $\beta \approx 0.85$. By decreasing this coefficient, we allow the Edge removal procedure to supress more elements in one iteration, but with the risk to obtain worse quality elements. Usually this coefficient should not be modified during the whole computation. The choice of β has to be made very carefully: for instance in the case of quite plane areas initially populated with a very fine (and then unusefull) mesh, one can be tempted to choose a very small value for β in order to supress a big number of elements during each of the iterations (and then reducing the number of iterations). But this choice can be catastrophical for the MATLAB implementation because then some of the nodes will have a huge number of neighbours (more than 15) and the matrix of neighbouring elements may become overfilled. I have introduced a method which can handle such problems, sending the extra neighbours in another lines of that matrix, but this method can not handle the situation if there are too much nodes with a large number of neighbouring triangles.

2.4.3 Condition on the Hausdorff envelop

This condition is the key of the Hausdorff Simplification method. The distance between the initial and the coarsened meshes is computed only here. The basic idea is to associate a value $h(T)$ to each of the triangle T of the mesh. This value will represent a bound of the Hausdorff distance of that triangle to the initial mesh. Of course at the beginning, each of these Hausdorff values are zero, but throughout the simplifications, they will be updated according to the equation 2.9.
Of course the node collapse must be prevented in the case the elements modified by the collapsed are (at least partially) outside the Hausdorff tolerance layer. In case they are not, the collapse is allowed and the Hausdorff values are just updated. The node collapse should satisfy the Hausdorff criterion which has been implemented in a separated function that returns either **TRUE** or **FALSE**. Therefore that criterion is implemented that way:
Notice that $\mathscr{B}(n)$ is the ball of elements around the node n *before* the collapse whereas $\mathscr{B}_{new}(n)$ is

Algorithm 2.3 Algorithm of the Hausdorff crterion

procedure `CheckHausdorff`(δ) ▷ δ is the maximum distance authorized
  set $h_{max} = \max_{T \in \mathscr{B}(n)} h(T)$ ▷ Initialization of h_{max}
  for all triangles $T \in \mathscr{B}_{new}(n)$ **do**
    Compute the euclidian distances $d(n,T)$, $d(n_{fix},T)$ and $d(n,n_{fix})$
    Compute $d_H(T,\mathscr{B}(n)) = \max(d(n,T), d(n_{fix},T), d(n,n_{fix}))$
    Compute $h(T) = d_H(T,\mathscr{B}(n)) + \max_{T \in \mathscr{B}(n)} h(T)$
    if $h(T) \geq \delta$ **then** ▷ Triangle outside tolerance
      return FALSE
    end if
  end for
  Update $h(T)$ for each $T \in \mathscr{B}_{new}(n)$
  return TRUE
end procedure

the ball around n *after* the virtual collapse of that node on another node named n_{fix}.

2.4.4 Condition on the normals

This condition is very important and it is also the last to be verified. Indeed, it is the most expensive condition, and therefore the last to be checked in order to save computational time. To understand the method used to check the conformity of the normal, I have to introduce the so called *pseudo-normal*. A pseudo-normal is a normal computed at one node using a weighted association of the normals of the triangles surrounding this point. On figure 2.10 the pseudo normal drawn at the node n in red is computed from a weighted sum of the normals of the surrounding elements of the node n. There is an unique way to compute the normal of a triangle, however there are actually several ways to build the pseudo normal according to the way the weight are defined. We will use

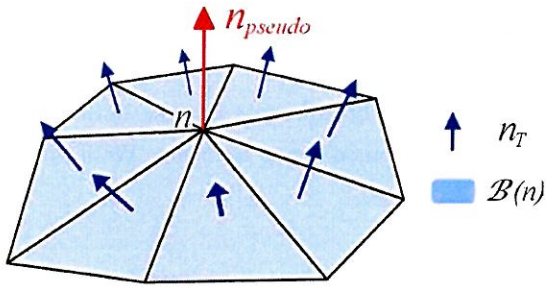


Figure 2.10: In red, pseudo-normal at the node n, computed from the normals in blue

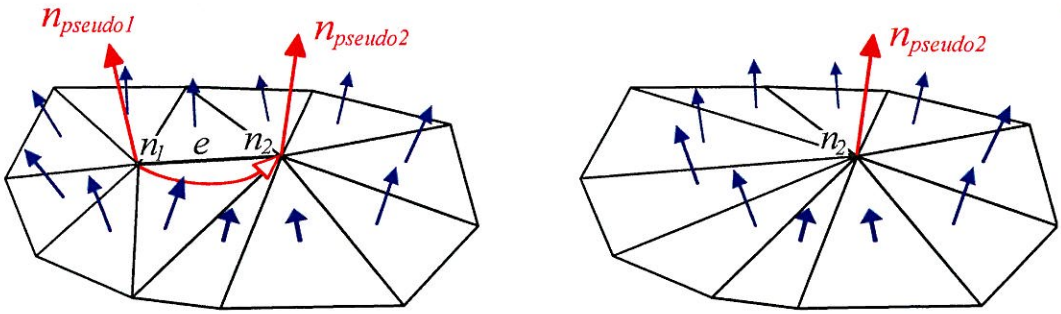


Figure 2.11: Normals and pseudo-normals before and after a collapse.

the method defined in (Roca and Sarrate, 2005) which is probably the most commonly used method in the mesh community. This method defines the pseudo-normal $\vec{n}_{pseudo}$ computed at the node n from the triangles T belonging to $\mathscr{B}(n)$ ($\mathscr{B}(n)$ is the set, or the *ball* of triangles sharing the node n) using the following formula:

$$\vec{n}_{pseudo} = \sum_{T \in \mathscr{B}(n)} A_T \vec{N}_T \tag{2.13}$$

With A_T the area of the triangle T, and $\vec{N}_T$ the normal of the same triangle. Of course that resulting pseudo-normal has to be normalized in order to be compared with the other surrounding normals.

Now, let us introduce the normal criterion. Basicaly, it consists in comparing the angle between the pseudo-normal and the normals of each of the triangle of $\mathscr{B}(n)$. If just one of the normals is not inside the tolerence cone of the pseudo-normal, the condition is not satisfied and the collapse is not performed.

Remark : there are two different ways to implement that for the edge collapse as it is described in the figure 2.4.4:

- The first way is to compare two series of normals : the pseudo-normals $\vec{n}_{pseudo1}$ and $\vec{n}_{pseudo2}$ with the normals of their respective balls. All that *before* the collapse of the edge e.

- The second way is to make a provisory collapse of the edge e and then compute the differences between the normals. Therefore the test is made *after* the edge has been collapsed.

These two methods lead to results slightly differents but the both can prevent a suppression in a sharp area and the formation of unwelcomed sharp elements. We have chosen the first one in order to avoid the provisory collapse.

2.5 Edge swap

The Edge Swap is the simplest and the cheapest way to increase the qualities of the triangles of the mesh. The basic Idea is to loop on all the edges of the mesh, getting the two neighbouring triangles and checking if the swap of that edge would increase or not the quality of these two elements.

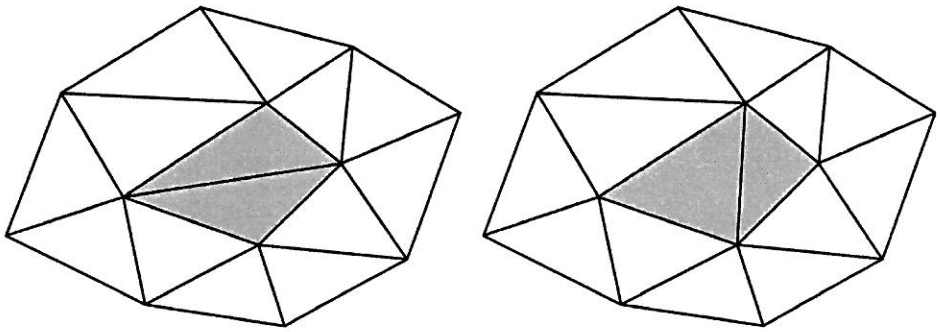


Figure 2.12: Example of edge swap

2.5.1 Enforcing a standard local configuration

Before swapping the edge, it is very important to enforce a standard local configuration, in order to make the operations as general as possible. Let us consider a set of two triangle which need to be swapped. In order to keep the modification local, we have to wonder which of the objects of the mesh will be updated. Actually, the choice of our datas structure is such that the neighbouring triangles won't be affected, which is a good thing in terms of computational cost and in term of readability of the code.

So let us consider the following local mesh around an edge swap on Figure 2.12. Let us concentrate on the objects of the mesh direcly related with these two triangles (we will see in section 4.2 that the triangles do not *know* the triangles that surround them).
Let us draw a fragmentation of these objects in order to see what need to be updated on Figure 2.13. It appears that, after the edge swapp, the following objects will be updated :

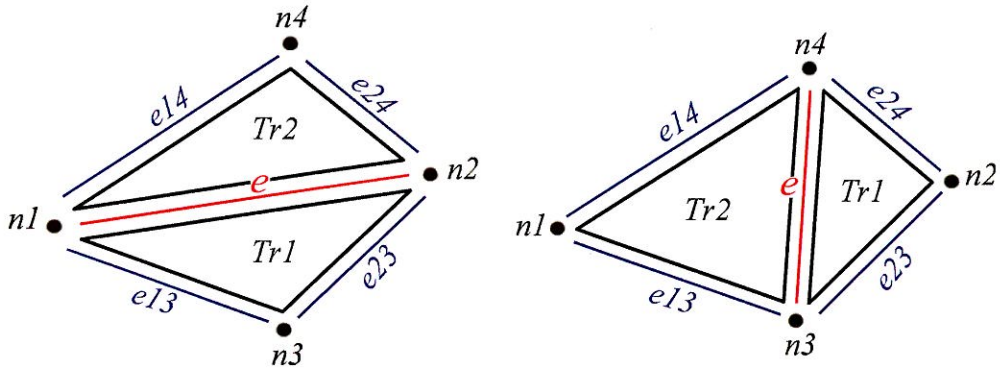


Figure 2.13: Elements of the mesh which will be updated after the swap

- the edge e which will change its extremity nodes from n_1 and n_2 to n_3 and n_4
- the triangles Tr_1 and Tr_2 will have new edges and new nodes
- the edges e_{13} and e_{24} will change of neighbouring triangle (notice that $e14$ and $e23$ are not updated)
- the nodes n_1, n_2, n_3 and n_4 will have some changes in their set of neighbour triangles

Note that, with this method for the swapping, there is not any creation or destruction of any object of the mesh. We just have to update the relations between the objects. This is a very good point for the use of the memory of our program, as well for the computational time.

2.5.2 Overview of the edge swap algorithm

As for the edge collapse procedure, we need to satisfy some conditions before swapping the edge. If you see our algorithm 2.4 of edge swap on the next page, you will see that the boundary edges are not treated (line 7) and the swapped edges should satisfy two conditions :

- The quality of the new configuration q_{new} should be greater that the quality of the old configuration q_{old} (line 16) else there is no interest in swapping the edge.
- The angle between the normals $\vec{n_1}$ and $\vec{n_2}$ should not be greater than a certain tolerance tol. That tolerance could be the same that the parameter θ or something smaller. But it is a nonsense to choose $tol > \theta$.
- Finally the orientation of the newly created triangles must be checked. Actually, most of the time this condition is satisfied. Only in case the mesh is very degenerated this condition can be usefull. So for a quite regular mesh, this condition can be forgotten.

Algorithm 2.4 Edge Swap

procedure `EdgeSwap`$(X, T, DatasEdges)$
  Sort the Edges according to the quality
  Initialize n_{loop} and define the tolerance angle tol
  while $n_{loop} < Nmax_{loop}$ **or** a significant nb of node is swapped **do**
    for $i = 1$ to $NbEdges$ **do**
      the current edge Edge(i) has two neighbouring triangles: `tri1` and `tri2`
      if Edge(i) is not on the boundary **then**
        Check the current configuration of the triangles
        if The configuration is not standard **then**
          Re-number the triangles in a standard configuration
        end if
        Compute q_{old} the old quality of Edge(i)
        Compute q_{new} the new quality as if Edge(i) was swapped
        Compute n_1 and n_2 the normals of `tri1` and `tri2`
        Compute the new orientation as if Edge(i) was swapped
        if $angle(\vec{n_1}, \vec{n_2}) > tol$ **and** $q_{new} > q_{old}$ **and** the orientation is conserved **then**
          Do the Swapping, which contains the following operations:
          Update the datas of the swapped edges in $DatasEdges$
          Update the verteces and the connectivities of `tri1` and `tri2`
          Update the datas of all the other edges of `tri1` and `tri2`
        end if
      end if
    end for
    $n_{loop} = n_{loop} + 1$
  end while
end procedure

2.6 Node relocation

2.6.1 Overview of the method

After the suppression of edges, the quality of the elements has globaly decreased and the mesh has become sharper. Then, after the edge swapping, the node relocation is the second part of the process of improvement of the quality mesh.

Moving a mesh node P consists in relocating this node towards an *optimal* point location. Such a point corresponds to a configuration of optimally shaped (equilateral) triangles. To this end, at each mesh node, the surface can be locally approached by a quadric function passing at best through all neighboring nodes following an idea suggested by (Hamann, 1993) and explicited by (Borouchaki and Frey, 2005). Once the equation of that quadric is known, a position P^* that maximizes the quality of the neighbouring triangles is computed. Finally the point P^* is projected on the quadric and that projection is the new optimal point location for P. We will now start describing how to compute the quadric approximation.

2.6.2 Method of local quadric approximation

Let P be a mesh vertex, $v(P)$ be the unit normal to the surface at P and let $(\tau_1(P), \tau_2(P))$ be an orthonormal basis in the tangent plane to the surface at P, such that $(\tau_1(P), \tau_2(P), v(P))$ forms a direct basis in $\mathbb{R}^3$. This is then a local basis at the point P. Let consider the frame $(P, \tau_1(P), \tau_2(P), v(P))$ in $\mathbb{R}^3$ and let define in this frame the quadric surface that locally approaches the surface at P. This quadric surface is then centered at $P = (0, 0, 0)$ in the local coordinates, thus any point (x, y, z) of $\mathscr{Q}(P)$ is such that:

$$F(x, y, z) = z - ax^2 - 2bxy - cy^2 = 0 \tag{2.14}$$

where a, b and c are the quadric coefficients that have to be numerically determined. The true surface is locally approached by a quadric surface, thus meaning that the neighboring vertices $P_i = (x_i, y_i, z_i)$ of P shall be long at best to the quadric. This is equivalent to minimizing the squared distance of these points to the quadric surface. The coeficients a, b and c are then solutions of the following minimisation problem:

$$\min \sum_i \left(ax_i^2 + 2bx_iy_i + cy_i^2 - z_i\right)^2 \tag{2.15}$$

This problem has the same solution as the following linear system:

$$\begin{pmatrix} \Sigma_i x_i^4 & \Sigma_i 2x_i^3 y_i & \Sigma_i x_i^2 y_i^2 \\ \Sigma_i 2x_i^3 y_i & \Sigma_i 4x_i^2 y_i^2 & \Sigma_i 2x_i y_i^3 \\ \Sigma_i x_i^2 y_i^2 & \Sigma_i 2x_i y_i^3 & \Sigma_i y_i^4 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} \Sigma_i x_i^2 z_i \\ \Sigma_i 2x_i y_i z_i \\ \Sigma_i y_i^2 z_i \end{pmatrix} \tag{2.16}$$

The quadric surface $\mathscr{Q}(P)$ locally defined at vertex P will be used as geometric support to find the *optimal* position of P.

2.6.3 Computation of the point P^*

The quadric is computed to keep the geometry's information througout the relocation process. Now we are going to introduce the point P^* which is an optimal configuration with respect to the shape quality of the triangles of $\mathscr{B}(P)$ (the ball of triangles surounding P). The computation of P^* introduces the notion of *best quality* into the relocation, that is why the way that computation is done is very important.

The basic idea is that P^* is the average location of the optimal points corresponding to equilateral triangles based on the boundary edges of $\mathscr{B}(P)$. The differents steps of that computation are explained by Fgure 2.14. Let us consider the point P with its ball of adjacent triangles $\mathscr{B}(P)$ (picture (a)). Now, on each of these triangles, we consider the edge opposite to P and we build an equilateral triangle based on that edge in the plane of the original triangle. These new triangles are drawn in blue on picture (b) and the new nodes created are the P_i. Finally, P^* is the average

location of these points P_i as represented on picture (c).
Now the last step on the relocation is to project this optimal point P^* on the local approximation of the geometry.

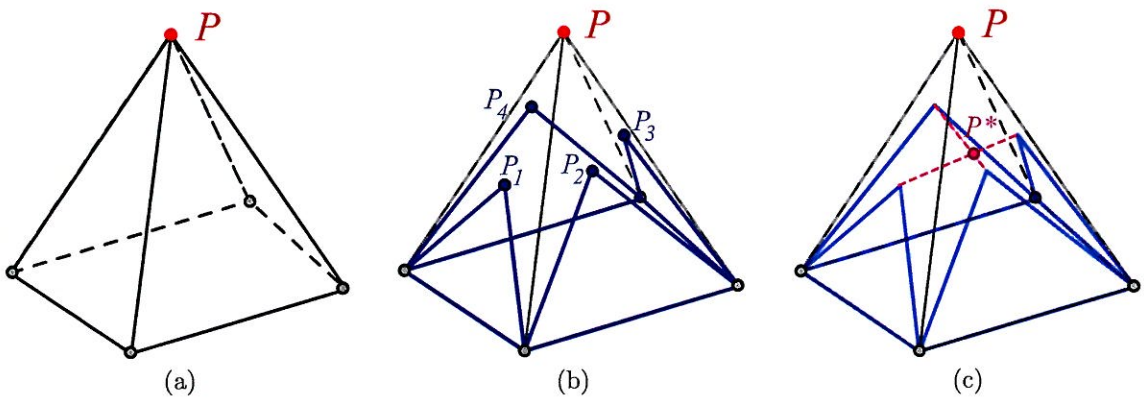


Figure 2.14: Steps of the P^* computation.

2.6.4 Computation of the optimal position by projection on the quadric

The projection of P^* on the quadric surface $\mathcal{Q}(P)$ is the solution of the optimisation problem:

$$\text{find} \min_{X \in \mathcal{Q}(P)} \left\| \vec{XP^*} \right\|^2 \tag{2.17}$$

That equation can be reduced to the solving a polynomial of order 5. This equation always has a real solution and, among the real solutions, we consider the closest to P^*. From a practical point of view, we use the Newton method for finding an approximation of the projection of P^* on the quadric surface. The corresponding algorithm can be written as :

At the end of that algorithm, the projection of P^* on the quadric surface is then the point U_{k+1}. An important task is to update the Hausdorff values associated to each of the triangles connected to the relocated point P. Of course, these values has to be computed as in the section 2.4.3. In particular, the approximation of the Hausdorff distance can be applied according to the relation 2.9 where Q represents the new position (after the point has moved) of P.

2.6.5 Possible improvements of that relocation

We can either decide that the node relocation is fundamentally acceptable or that there should be some criterion that controls if the relocation is desirable or not. We will develop this improvement later in my internship.

Algorithm 2.5 Approximation of the projection of P^* on the quadric surface

procedure `Projection`$(arg1, arg2)$ ▷ The projection of P^* on the quadric
  for all triangles **do**
    set U_0 to P^* ▷ Initialization
    while $\|U_{k+1} - U_k\|$ *is not sufficiently small* **do**
      Apply the newton step to the function $F(U_k + t\nabla F(U_k))$:
      $U_{k+1} = U_k - \frac{F(U_k)}{\|\nabla F(U_k)\|^2}\nabla F(U_k)$
      $k = k + 1$
    end while
    return U_{k+1} b ▷ Coordinates of the projected Point
  end for
end procedure

Another possible improvement is the way the local approximation of the surface is computed. Following (Borouchaki and Frey, 2005) we have choosen a quadric representation of the vicinity that needs the computation of only 3 coefficients. The quadric is a quite cheap modelization. Moreover it is actually very well adapted to the relocation of a node with a very few number of triangles neighbours (3 or 4) or if the geometry is very smooth. For a more complex geometry, or for an higher number of connexions, it becomes then relevant to choose a representation with an higher number of coefficients or with an higher polynomial degree. The following modelization for instance would add two coefficients d and e, and could be used for a better representation of the local surface:

$$F(x, y, z) = z - ax^2 - 2bxy - cy^2 + dx + ey = 0 \tag{2.18}$$

Chapter 3

Improvements of the method

The extension of the method to the surfaces presenting geometric discontinuities consists in taking into account the interface curves (between two smooth pieces) as well as curves traced onto the surface. Notice that, in the first case, the variation of the normal to the surface is not continuous throughout the segments of interface curves. These critical curves are known in a discrete manner and are composed of edges of the reference mesh. To enforce the proximity and regularity properties of the mesh with respect to the critical curves, two other tolerance regions are added to the already defined regions. The first region is a cylinder related to the global proximity, its principal axis corresponds to the critical curve and its radius is equal to a prescribed Hausdorff distance δ. The second property is enforced using another cone of regularity centered at each vertex of the critical curve in the reference mesh, its principal axis is given by the principal normal to the critical curve at this vertex and of (prescribed) aperture angle θ. The deletion of an edge located along a critical curve Γ is *geometrically* validated if:

- the resulting configuration (i.e., the set of triangles) is at a Hausdorff distance d from the reference mesh and the new critical edge (along Γ) is at a Hausdorff distance δ from Γ;
- the normal to the new triangles are contained within the regularity cones at the vertices and the normal to the critical edge is contained within the cones associated with the edge endpoints.

Techniques close to the one described in the previous section can also be used to compute the Hausdorff distances with respect to a critical curve. Relocating a vertex along a critical curve consists in moving this vertex, step by step, toward an *optimal* point location, i.e., resulting in a configuration of optimal triangles. The new location is computed using a local approximation of the curve with a quadric curve passing at best through the adjacent vertices. As usual, the node is moved if the geometric approximation is preserved.

3.1 Open surfaces: special treatment for the boundaries

3.1.1 Suppressions allowed on the boundary

An *open surface* is a surface with one or sveral boundary line(s). The suppression of the nodes on the boundary lines should be made very carefully in order to preserve the geometry of the boundary lines. The figure 3.1 shows what kind of suppression are allowed and what is not allowed around a boundary. On these figures, the boundary is drawn in blue. We can make several remarks:

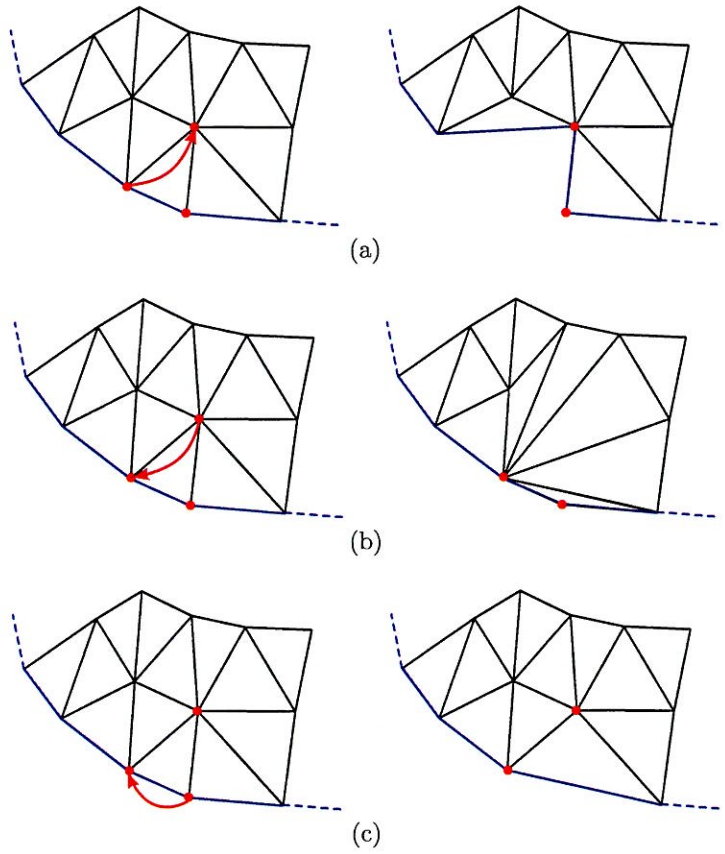


Figure 3.1: The collapses (b) and (c) are allowed, but (a) is not allowed

- for an edge whom both nodes do not belong to the boundary, the boundary is not affected, therefore this edge is treated as usual.
- for an edge whom only one among the two nodes belongs to the boundary, that edge can accept the collapse of the non-boundary node on the boundary node (figure 3.1:(b)). But the other direction of the collapse (boundary node towards non-boundary node) is not accepted because

(see the case (a)) the boundary line is then considerably modified.

- finally, for an edge which is fully on the boundary line, both nodes can be collapsed, as it is drawn in the case (c). Therefore the edges have to be marked to enter in one of these three categories in order to deal with these edges appropriately.

Please notice that for a collapse like (b), the usual conditions defined in the section 2.3 have to be checked. But for the collapses of the second case : like (c), another specific condition for the boundary has to be checked, and that is what we are going to introduce.

3.1.2 Special condition for collapsing on the boundary

This condition has to be checked before to collapse a boundary node n_{col} on another boundary node n_{fix}. This condition does not replace the other criterions defined in the section 2.3 but it is added to them and it is the last to be checked. So the algorithm of the edge collapse has to be slightly modified and the new algorithm obtained is given in the first annex.
The main idea of that special condition is to compare the direction of two vectors $\vec{V_{CF}}$ and $\vec{V_{LC}}$ of the current boundary edge e with the vector obtained after the collapse $\vec{V_{LF}}$. For a best understanding, see Figure 3.2. Therefore an angular tolerance θ_{boundary} has to be defined, which is different from the parameter θ described in the precedent chapter. We can either choose a value constant throughout the simplification process or a value incremented the same way that θ and δ. Practically we choosed a constant $\theta_{\text{boundary}} \in [10; 15]$. Thus the edge collapse is accepted if the two following relations are satisfied:

$$\begin{cases} \cos\left(\vec{V_{CF}}, \vec{V_{LF}}\right) \geq \cos(\theta_{\text{boundary}}) \\ \cos\left(\vec{V_{LC}}, \vec{V_{LF}}\right) \geq \cos(\theta_{\text{boundary}}) \end{cases} \tag{3.1}$$

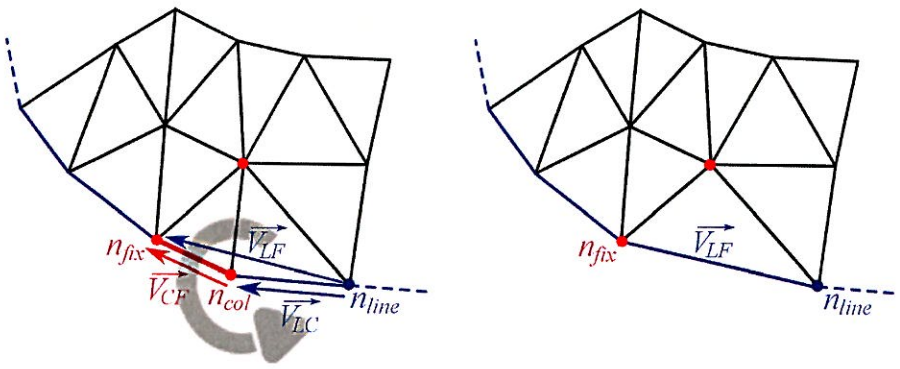


Figure 3.2: Condition for collapsing a boundary node.

3.1.3 Relocation of the nodes on the boundary

The main idea of the relocation is to make an approximation of the local boundary line using a circular curve, and then move the point we want to relocate along that curve towards the optimal position. Figure 3.3 is a representaton of the relocation of the node we want to move n_{moved}. That node has two neighbours on the boundary curve: n_{line1} and n_{line2}. From these three points a unique circular curve can be computed. Once that curve is known, it is divided into small arcs and the node n_{moved} is virtually placed in each of these small arcs and for each position the quality shapes are computed. The position that leads to the best quality shapes is then the optimal position for n_{moved}.

Another possibility for the node relocation would be something similar to the relocation described in section 2.6 with the curve replacing the quadric. But then the problem comes from the projection of an optimal point P^* on the curve, which is a little bit tricky.

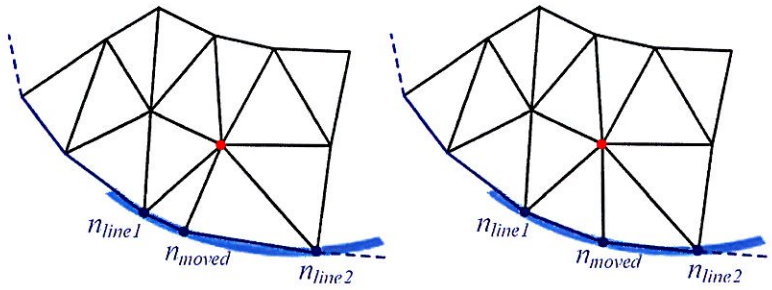


Figure 3.3: Condition for collapsing a boundary node.

3.2 Special treatment for the critical curves

3.2.1 Definitions of the critical curves

The critical curves are special lines of the geometry that are particularly important for the geometry or that are delimitations of two surfaces or they represent the sharp lines of the geometry. An example is represented on figure 3.4 with the critical curve drawn in blue. These curves can be defined and given by the user, or they can be found automatically by a preprocessing that finds automatically the sharp lines. The automatic recognition works quite well, but the problem is that the problem may mark some spurious irregularies of the initial mesh as critical curves. And then, these spurious artefacts would be conserved instead of being cleaned by the program.

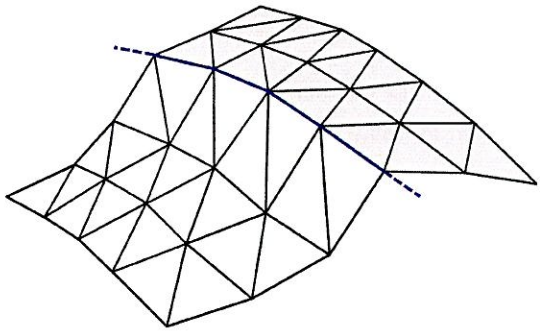


Figure 3.4: Condition for collapsing a boundary node.

3.2.2 Suppressions allowed on the critical curves

Again, as it was defined for the boundary line, only a special kind of collapse is allowed. The basic idea is that a node of the critical curve can only be collapsed on another node of the critical curve. Thus, as it is shown in the figure 3.5, the configuration (b) is allowed but (a) is not.

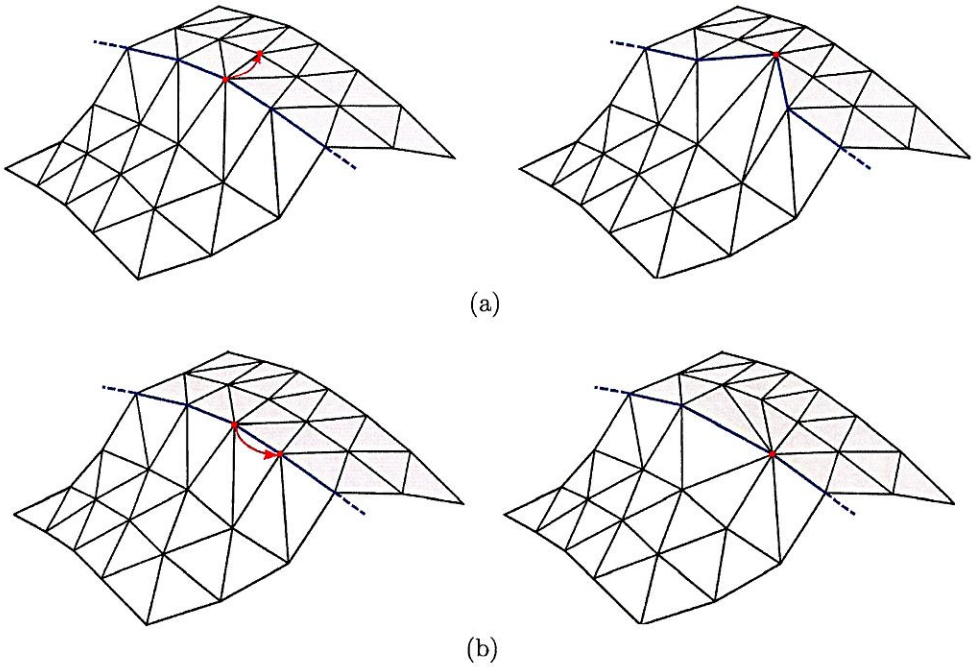


Figure 3.5: The collapse (b) is allowed, but (a) is not allowed

We can transpose the remarks that we have done for the boundary lines :

- for an edge whom both nodes do not belong to the critical curve, the curve is not affected, therefore this edge is treated as usual.

- for an edge whom only one among the two nodes belongs to the curve, that edge can accept the collapse of the non-curve node on the boundary node (figure 3.1:(b)). But the other direction of the collapse (curve node towards non-curve node) is not accepted because the boundary line is then considerably modified (see the case (a)).

- finally, for an edge which is fully on the boundary line, both nodes can be collapsed, as it is drawn in the case (b).

- in the case a node belongs to both a critical curve and a boundary line, or belongs to two diferents critical curves, then this point is neither suppressed or moved. Therefore the edges has to be marked, with another mark that the boundary one, in order to deal with the edges appropriately.

3.2.3 Special condition for collapsing on the critical curve

This condition has to be checked before to collapse a curve node n_{col} on another curve node n_{fix}. This condition does not replace the other criterions of the section 2.3 but it these conditions will be modified. Finally, added to these conditions and it is the last to be checked.
The main idea is to compare the direction of the vector of the current boundary edge e with the two neighbouring vectors directions.

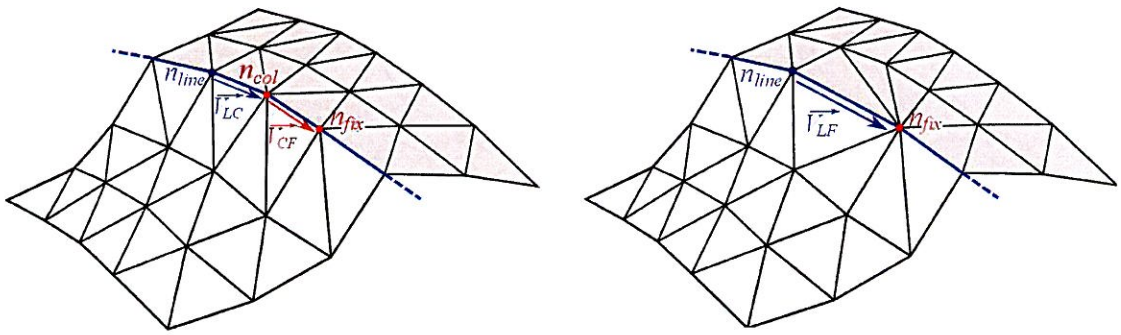


Figure 3.6: Condition for collapsing a curve node.

3.2.4 Modifications of the normals criterion

As we have explained above, when an edge belongs to a critical curve, the criterions defined in the section 2.3 remain the same, but the nomal criterion hat to be slightly modified.
The basic idea is that the critical curve splits the mesh in two parts. On each of these parts, the usual condition on the normals of the section 2.3 will be checked separately, and if *all* the criterions on the both parts are satisfied, the collapse can happen.
Therefore the ball of triangles around n_{col} has to be divided in two balls $\mathscr{B}1(n_{col})$ and $\mathscr{B}2(n_{col})$ separated by the critical curve. This configuration is represented on the figure 3.7 :

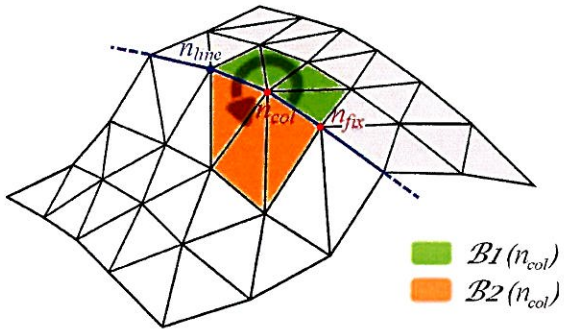


Figure 3.7: Balls of triangles around n_{col}

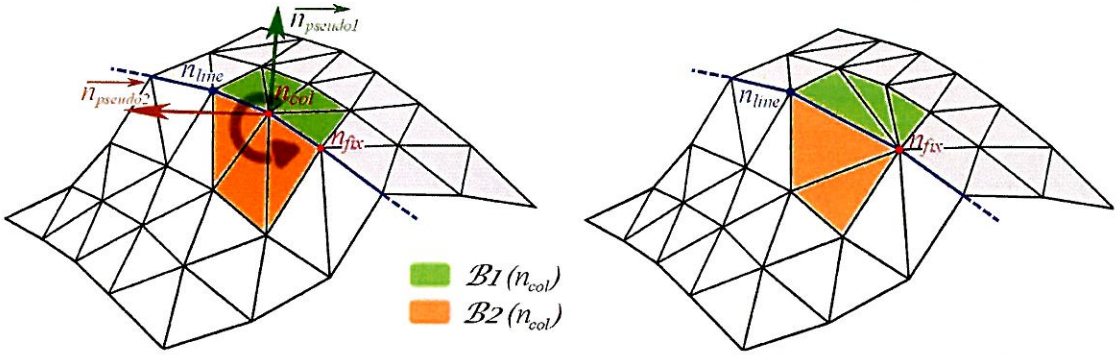


Figure 3.8: Condition for collapsing a curve node.

For that purpose, we need first to create a program `GetHalfBalls` that builds the two balls of triangles and that finds the following node on the sharp line n_{line}. In order to do that, the program will turn around the node n_{col} and first store the triangles inside $\mathscr{B}1(n_{col})$. Then, when the program meets a node on the sharp line, it calls this node n_{line} and store the following triangles in the

second half-ball until all the triangles arround n_{col} are parsed. This procedure is developped by the algorithm 3.1. Once these two half-balls are created we can compute the two related pseudo-normals which are computed at the node n_{col} : $\vec{n}_{pseudo1}$ and $\vec{n}_{pseudo2}$. Then the normal criterion as defined in section 2.4.4 will be checked two times : once for $\mathscr{B}1$ using $\vec{n}_{pseudo1}$, and another time for $\mathscr{B}2$ with $\vec{n}_{pseudo2}$. That configuration is represented on the figure 3.8.

Algorithm 3.1 Algorithm for the creation of the 2 balls

procedure `GetHalfBalls`(n_{col}, n_{fix}) ▷ Coordinates and Connectivities
  Initialize $\mathscr{B}1$,$\mathscr{B}2$ and set $Nb_{balls} = 1$
  Get the triangle T whih has n_{col} and n_{fix} as verteces
  Get the third node of T : n_{next}
  Store this triangle in $\mathscr{B}1$
  Set $\mathscr{B}_{current} = \mathscr{B}1$
  while $n_{next} \neq n_{col}$ **do**
    $n_{current} = n_{next}$
    Get the triangle T whih has n_{fix} and $n_{current}$ as verteces
    Get the third node of T : n_{next}
    if n_{next} is marked as sharp **then**
      if $Nb_{balls} \geq 2$ **then** ▷ If n_{col} is the junction of 2 lines
        ReSet $\mathscr{B}_{current}$ as empty and **Break** ▷ The collapse is stopped
      end if
      $n_{line} = n_{next}$
      $\mathscr{B}_{current} = \mathscr{B}2$
      $Nb_{balls} = Nb_{balls} + 1$
    end if
    Store the triangle T in $\mathscr{B}_{current}$
  end while
  return $\mathscr{B}1$, $\mathscr{B}2$ and n_{line}.
end procedure

3.2.5 Relocation of nodes on a sharp line

The relocation of the nodes belonging to a critical curve follows the same pattern that the one defined for the boundary nodes in the section 3.1.3. The only diference is that the procedure of relocation is automatically stopped if the node belongs to two diferents sharp-lines.

Chapter 4

Implementation Details

4.1 Introduction

For the implementation, I had the chance to use two completely differents environments: Matlab and C++. If the algorithms themselves that has been described in the precedent parts of this thesis was the same, the way to implement them was completely differents according to the language used.

4.2 Relations between the cells of the mesh

Nevertheless, the both will use the same pattern for the data. Indeed the relations of *knowledge* between the cells is the same. By *cells* we designate the objects of 0, 1, 2 or 3 dimentions that form together a mesh. In our case, the *cells* are divided in 3 types : the triangles, the edges and the nodes. Of course, we won't have 3D cells like tetrahedrons or hexahedrons, because we just deal with surface meshes. Each of these cells have a *knowledge* of some other cells around it. For instance the triangles *know* obviously the 3 nodes that form their 3 verteces, they may also know (but it is not compulsory) their egdes or the triangles that surround them. Equally, the edges have to know the nodes of their extremities, but they may know or not the adjacents triangles.
So it is clear that the organization of the data is a choice from the programmer, according to the kind of local operation needed in the treatment of the mesh. So the problem is how to choose the relations between the cells. The basic idea for this choice is that the program should never loop on all the cells of the mesh to find a neighbouring cell. All the neighbouring cells have to be accessible very quickly. But we also have to keep in mind that our meshes may be very big, and then the storage of too much relations may become really unmanageable. The Figure 4.1 represent our final choice, which is of course a compromise between the speed and the memory used.
On this figure, the arrow $\rightarrow$ means *know* or *have access to*. For instance the relation Edges$\rightarrow$Nodes

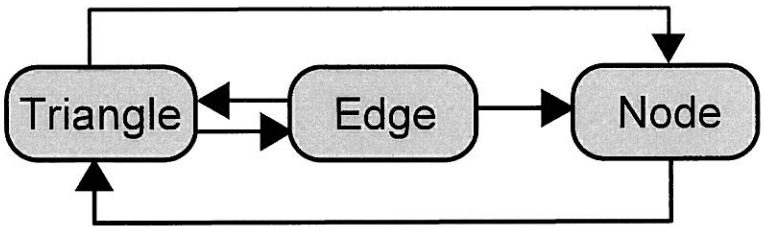


Figure 4.1: Elements of the mesh which will be updated after the swap

means that for any Edge, the two nodes of its extremities are stored.
This choice is also questionable: we could replace the relation Nodes→Triangle by the relation Nodes→Edge. In one hand, this change would increase the operations of node relocation but in another hand, it would slow down the computation of the pseudo-normal. We could also add some relations wich would increase considerably the speed of some operations. For instance the relation Nodes→Nodes is an improvement for the relocation, but it is also a huge amount of extra information (usually the nodes are connected to more than 3 nodes) that has to be stored and updated.

4.3 Static memory implementation - matlab

All MATLAB built-in datatypes are passed by value. Therefore, all input parameters are usually copied to be used in the functions and subfunctions of the program. That can often saturate the memory aviable in the case the mesh is big. Thus, with an usual personal computer, a mesh with more than 3000 nodes become very slow to be treated.

4.3.1 Why the MATLAB static memory is a problem?

All the variables in MATLAB are static arrays, or matrices. In our code, there is a lot of suppressions of elements, that means that all the matrices of connectivities (relations) has to be resized regularly. That implies that the static arrays have to be resized, which is a big problem because in that case we have to create a completly new array, with a smaller size and copy all the data in the new array. Below the equation 4.1 represents a suppression of the i^{th} triangle in the connectivity matrix

Triangles→Nodes.

$$\begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ \vdots & \vdots & \vdots \\ \boxed{n_{i1}} & \boxed{n_{i2}} & \boxed{n_{i3}} \\ n_{(i+1)1} & n_{(i+1)2} & n_{(i+1)3} \\ \vdots & \vdots & \vdots \\ n_{m1} & n_{m2} & n_{m3} \end{bmatrix} \xrightarrow{\text{Suppression of line } i} \begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ \vdots & \vdots & \vdots \\ n_{(i+1)1} & n_{(i+1)2} & n_{(i+1)3} \\ \vdots & \vdots & \vdots \\ n_{m1} & n_{m2} & n_{m3} \end{bmatrix} \tag{4.1}$$

4.3.2 Solution to this problem: use of pointers

A solution for this problem is to update all the matrices of connectivities once for all at the end of the programm. Therefore during all the programm, not even one node will be suppressed, but a vector of redirection pX, which is not formally a pointer, will send the collapsed node on the fixed node, another vector of redirection pE will send the collapsed edges on the corresponding fixed ones, and another vector pT will take te value -1 for the elements that are suppressed. Let us come back to the standard configuration for the collapse of edges drawn on the figure 4.2:

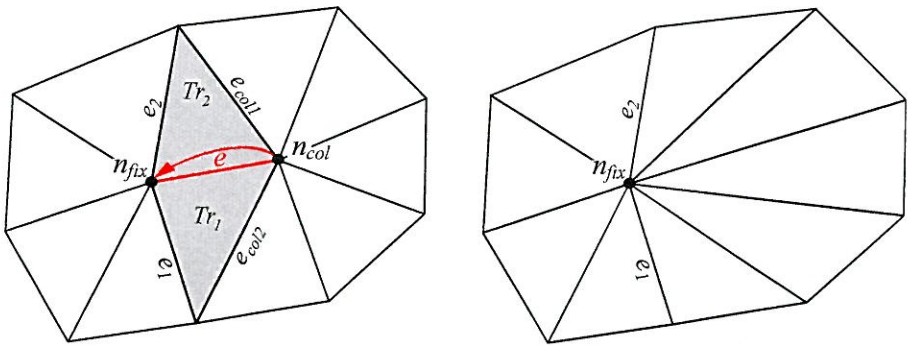


Figure 4.2: Elements of the mesh which will be updated after the swap

As we already seen in the section **??**, the edges e, e_{col1} and e_{col2} will be suppressed, as well as the node n_{col} and the edge e and the triangles Tr_1 and Tr_2. Thus, an operation like the one described on 4.1 should be performed six times for each edge collapse. This is of course too much costly, so the other solution is to keep untouched the data themselves, but we will use vectors of redirection as filters to redirect the elements collapsed and to mark the elements suppressed.

- redirect node n_col on n_fix using the vector pX
- redirect edges e_{col1} and e_{col2} respectivelly on edges e_{fix1} and e_{fix2} using the vector pE
- mark edge e as 'to be suppressed' using the vector pE

• mark triangles Tr_1 and Tr_2 as 'to be suppressed' using the vector pTr.

$$pX = \begin{bmatrix} n_1 \\ \vdots \\ \boxed{n_{col}} \\ \vdots \\ n_{fix} \\ \vdots \\ n_m \end{bmatrix} \longrightarrow \begin{bmatrix} n_1 \\ \vdots \\ \boxed{n_{fix}} \\ \vdots \\ n_{fix} \\ \vdots \\ n_m \end{bmatrix}; \quad pE = \begin{bmatrix} e_1 \\ \vdots \\ \boxed{e_{col1}} \\ \vdots \\ \boxed{e} \\ \vdots \\ \boxed{e_{col2}} \\ \vdots \\ e_m \end{bmatrix} \longrightarrow \begin{bmatrix} e_1 \\ \vdots \\ \boxed{e_{fix1}} \\ \vdots \\ \boxed{-1} \\ \vdots \\ \boxed{e_{fix2}} \\ \vdots \\ e_m \end{bmatrix}; \quad pTr = \begin{bmatrix} Tr_1 \\ \vdots \\ \boxed{Tr_1} \\ \vdots \\ \boxed{Tr_2} \\ \vdots \\ Tr_m \end{bmatrix} \longrightarrow \begin{bmatrix} Tr_1 \\ \vdots \\ \boxed{-1} \\ \vdots \\ \boxed{-1} \\ \vdots \\ Tr_m \end{bmatrix}; \tag{4.2}$$

4.3.3 Suppression of the data at the end of the programm

At the end of the program, all the data will be completely updated, that means that all the cells which have been suppressed by the algorithm, or the one. That means in practical terms that for all the pointers, we check : $if\ pX(i) \neq n_i \Rightarrow n_i$ is suppressed

4.3.4 Recursivity for updating of the data structure

Let us imagine the following situation represented by the figure 4.3 :

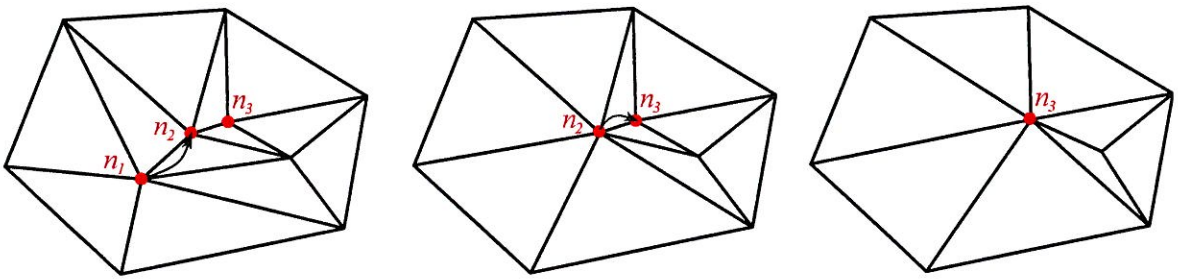


Figure 4.3: Two successives suppression of nodes

The problem is that first, we do the redirection of $n_1 \rightarrow n_2$, that means that $pX(n_1) = n_2$ and then we do the second redirection $n_2 \rightarrow n_3$ wich is concretized by $pX(n_2) = n_3$. That means that, each time someone will want to have access to n_2, he will find n_3, but if someone wants to have access to n_1, he will find... n_2 which no longer exists! The problem is then to track the differents redirections to finally redirect all the precedents nodes to the final node. Let us imagine the following scheme of

node suppression (operations of redirections are performed from the left to the right):

$$\left.\begin{array}{lll} n_1 & \rightarrow & n_2 \\ n_3 & \rightarrow & n_4 \\ & & n_5 \end{array}\right\} \rightarrow n_6 \rightarrow n_7 \tag{4.3}$$

It is obvious that when the node n_6 is sent on n_7, all the nodes $n_1, n_2, \ldots, n_6$ should be redirected towards n_7 as well. To do that, we have to build an antecedent matrix as follow : for each collapse of a node n_i on the node n_j, a node n_i is added at the j^{th} line of the antecedent matrix. By the way, *antecedent Matrix* means that for each line l, all the antecedents of the nodes n_l appear. Therefore, the scheme 4.3 will lead to the following antecedents matrix :

$$\begin{array}{c|ccc} n_1 & & & \\ n_2 & n_1 & & \\ n_3 & & & \\ n_4 & n_3 & & \\ n_5 & & & \\ n_6 & n_2 & n_4 & n_5 \\ n_7 & n_6 & & \\ \vdots & \vdots & \vdots & \vdots \end{array} \tag{4.4}$$

Therefore the idea is to parse all the modes that have been modified in order to redirect them towards the good node. As each of the antecedents can have themselves one or several nodes antecedants, the best way to deal with this situation is using a recursive approach as explicited on the algorithm above. Of course, the same method can be applied for the redirection vector pE with its appropriate antecedents Matrix.

Algorithm 4.1 Alogrithm : updating the pointer by recursivity

```
procedure UpdatepX(M_ant, pX, n_col, n_fix)        ▷ fixed and collapsed nodes
    change pX(n_col) = n_fix
    M_ant(l, end) = n_col
    Set line l as the line of n_col
    for all the nodes n_i of the line l of the matrix M_ant do
        n_col = n_i
        UpdatepX (M_ant, pX, n_col, n_fix)          ▷ Recursivity
    end for
end procedure
```

4.3.5 Post-processing requirements

At the end of the whole suppresion/redirection process, we have to really delete all the elements. In that purpose, we have to loop on all the nodes, and then on all the elements in order to finally get the untouched cells and free the memory.

4.3.6 MATLAB advantages

But using MATLAB has also advantages. The point using MATLAB is the easiness to deal with vectors and matrices operations. All the tools of linear algebra are already implemented and are quite fast. MATLAB is particularly fast in inverting the matrices and solving the linear systems. Moreover it has interesting features in storing and handling sparse matrices, which is quite usefull for storing the antecedent matrices.

4.4 Dynamic memory implementation with C++

The dynamic memory used in C++, (vectors and co) is much more relevant for our problem that the MATLAB approach because of the problems that we raised before. In this present section, we will consider the mesh structures developed with C++ from a general point of view. Therefore the following notions have to be introduced :

- a **cell** is an abstract geometrical object which can be 1D (node), 2D (triangle, quadrilateral etc...) or even 3D (hexahedron, tetrahedron etc...). A cell is actually an abstract class which is the mother of all the other classes of geometrical objects used with an object-oriented programm.
- a **face** is an abstract geometrical object which is 2D (triangle, quadrilateral etc...). The class face is actually an abstract class which is the mother of all the other classes of geometrical 2D as the triangles that we used for our implementation.
- a **region** is an abstract geometrical object which is 3D (tetrahedrons, hexahedrons, pyramids). The class region is actually an abstract class which is the mother of all the other classes of geometrical 3D. Because in the context of our problem, the mesh we deal with is always surfacic, then we will not deal with any region.

The relations between these classes (and some other ones) will be explained below. For an overview you can see the figure 4.4.
The development of an appropriated mesh data structure is a decisive point that needs to be rightly balanced between memory consumption and speed performances. For the C++ programmation, we had to follow the structure developped by the CEA that we are going to present in the following section.

4.4.1 Presentation of the mesh data structure GMDS

Several years ago, the CEA has decided to create their own structure named GMDS for *Generic Mesh Data Structure* which is able to fit with diferents application field. This structure has been presented in depth by the article (Ledoux et al., 2009) and this section is just an overview of that paper.

The compatibility of the data structure to a large range of mesh and mesh algorithm is very important for the CEA. Indeed, the underlying mesh data structure must handle and must provide different kinds of cells and connections. For instance, a simple geometric smoothing algorithm may just need to know the elements and the nodes composing the mesh and the connections from the elements to their nodes while a 3D advancing-front algorithm generating a tetrahedron mesh from a boundary surface may require to know the mesh faces and the connections from the tetrahedron to the faces and vice-versa. In fact, there always exists a particular mesh data structure that better fits the requirements of an algorithm, and GMDS is designed to produce always the best data structure.

The word Generic in GMDS has actually two meanings:

- first, this data structure can handle any mesh model and any cell type(quadrilateral, triangles, polygons, hexahedra, tetrahedral, pyramids, etc)
- it is based on the generic programming . The use of generic programming in order to optimize memory consumption and not to increase speed performances. to define generic containers or generic algorithms which are independent of the objects they deal with. In numerical simulation codes, generic programming is also used to improve speed performances by performing mathematical computation at compile-time. But in our case, the generic programming is mainly used to get a tailored memory occupation for a mesh model.

4.4.2 The cellular mesh components of GMDS

As it was explained previously, the priority is given to the memory consumption and ease of use. The figure 4.4 shows a simplified class diagram, which gives a structural view of the cellular mesh component. Classes can be split into three categories:

- **The API interfaces**[1] are `Node`, `Edge`, `Face`, `Region`, `Cell` and `Mesh` classes. They give a user-friendly access to mesh concepts without dealing with the template parameters (except for the mesh class) ;
- **The internal mesh classes** are `TNode`, `TEdge`, `TFace`, `TRegion` and `TCell` classes. They use generic programming to optimize memory consumption ;

[1] An interface is an abstract clas providing an interface made of abstract functions and having no attributes.

- **The allocation classes** which are not represented in figure 4.4 but are essential. They manage memory allocation to improve memory allocation performances.

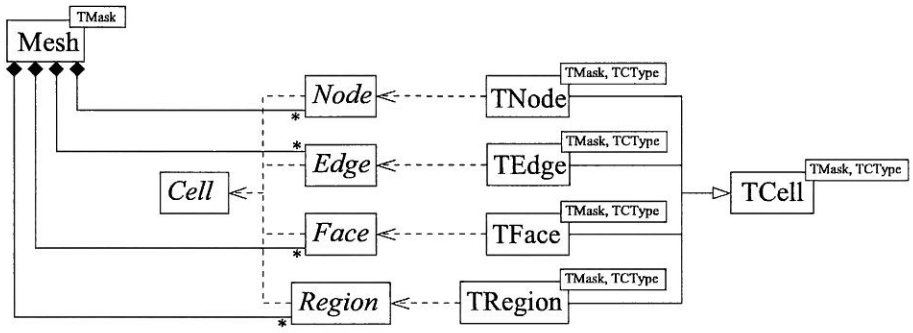


Figure 4.4: Class Diagramm of the GMDS cellular mesh module.

A very important notion introduced to generalise the mesh structure is the so called *model mask*. A model mask defines the available cells and connections in the mesh model. In our design, every internal mesh class has a corresponding parameter `TMask`. For instance, the mask of the model $M1$ (see figure 4.5) is `Dim3|N|R|R2N|N2R`. It means it is a 3-dimentional mesh (`Dim3`) represented by nodes (`N`) and regions (`R`) where connections are Regions→Nodes (`R2N`) and Nodes→Regions (`N2R`) are stored. Internal classes but Mesh class have also an extra parameter defining the type of cell they represent. For instance, for a face, it values `GMDS_QUAD`, `GMDS_TRIANGLE` or `GMDS_POLYGON`. These parameters are used to specialize memory consumption and cell behavior. Each internal cell class implements an API interface and inherits from the TCell class. From a developer/user point of view, internal classes and allocation classes are hidden. Developers only use API classes and the Mesh class. This design provides flexibility and a developer-friendly interface.
The figure 4.5 provides a selection of several model masks used by the CEA for differents purposes (mesh generation, mesh optimization, mesch refinement etc...). The mask $M8$ framed in red is the one that we used for our simplification algorithm. It is the same that the diagram of figure 4.1 but with the general notation *Face* instead of *Triangle*. Then the corresponding mask is `Dim3|F|E|N|F2N|E2N|F2E|E2F|N2F`.

4.5 Main differences in programming between MATLAB and C++GMDS

Even if we will not mention here the differences of syntax and the unavoidable differences between an object-oriented programm and a MATLAB, there are some other specificity that we have to highlight:

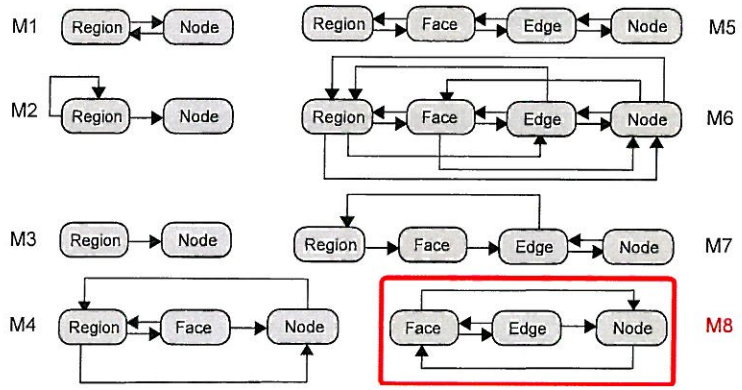


Figure 4.5: Samples of mesh models.

- The C++ language does not have a ready-made class for vector and matrices in the mathematic sense of that terms. GMDS has a templated geometric vector class named `Vector.tpp` which provides a tailored process for building a vector and doing some mathematical operations. But the problem with this template structure is that the size of the the vectors should be known at the compilation. But with our algorithm, mainly in the relocation part of the algorithm, we create vectors whose size is not known *a priori* during the compilation. Therefore the existing classes of GMDS are unusefull. That is why we decided to create new classes of dynamic vectors `DynVector.cpp` and dynamic matrices `DynMatrix.cpp`. In these classes we had to define as well all the kind of mathematical operations between scalars, vectors and matrices.

- Our simplification alorithm was itself a class `AlgoSimplification.cpp` and the differents subfunctions of that algorithm (edge removal, relocalization, edge swap, pretreatment etc...) are implemented as methods of that class. Note that the model mask and the vector H_d that stores for each trangles the distance between a triangle and the initial referance mesh are private attributes. The only argument given to the constructor of `AlgoSimplification` is a pointer to the reference mesh $\mathscr{T}^{\text{ref}}$ itself.

- We have built another class `ComputeHausdorff.cpp` which computes the Hausdorff distances between the initial mesh and the final mesh after all the process of simplification in order to verify the result given by the simplification algorithm. These distances are computed with the method developed at the end of the article (Borouchaki and Frey, 2005) This distance can not be computed by a method in `AlgoSimplification` because it needs the data of the initial mesh wich are progressively modified by `AlgoSimplification`. Indeed, constructor of `ComputeHausdorff` needs not only the final mesh given by `AlgoSimplification` but also $\mathscr{T}^{\text{ref}}$ and with a lighter model mask. Therefore the algorithm of verification of the Hausdorff

distance needs to be in a separated class.

- For the implementation of the C++ model, unlike the MATLAB implementation, there is no need of post-treatment of the data in the sense of the section 4.3.5. Indeed, the elements are suppressed in the memory instantaeously and there is not any redircetion of a cell to another one. Therefore the GMDS code is faster.

Chapter 5

Results and numerical exemples

5.1 Results for some standard tests

The following tests have been performed in order to highlight the effect of several particular points of our algorithm. Because these tests involve a quite small number of elements, they have been performed with MATLAB.

5.1.1 The *peak* test

This first test has been designed to show the accuracy of the pseudo-normals' computation. This mesh is a regular plane triangular mesh mapped by the *peak* function :

$$z(x,y) = \exp\left[-\alpha\left(x^2+y^2\right)\right] \tag{5.1}$$

With α a positive coefficient. The idea of this test is to compute at each node of the mesh the pseudo-normal, and to compare that pseudo-normal with the exact normal of the surface at this point. Indeed we have access to the exact normal $\vec{n}(x,y,z)$ at that point which is given by the partial derivatives of the function $z(x,y)$ in that way :

$$\vec{n}(x,y,z) = \begin{pmatrix} 2\alpha xz \\ 2\alpha yz \\ 1 \end{pmatrix} \tag{5.2}$$

Then for each node of the mesh, once $\vec{n}(x,y,z)$ is computed and normalized, it is compared with the pseudo normal $\vec{n}_{pseudo}(x,y,z)$ and the error e is computed that way :

$$e(x,y,z) = \|\vec{n}(x,y,z) - \vec{n}_{pseudo}(x,y,z)\| \tag{5.3}$$

It is then very interesting to plot that error according of (x, y) and we can see if the computation of the pseudo-normal is accurate or not. In the figures 5.1 below, the same peak is meshed two times with a fine mesh (a1) and a coarser mesh (b1). The pictures (a2) and (b2) represents the corresponding errors. As expected, the error is bigger for the coarse mesh, but it remains reasonnable, always less than 7% even on the regions surrounding the peak where the curvature is the most complex. It appears then that the way we have defined the pseudo-normals is quite reliable.

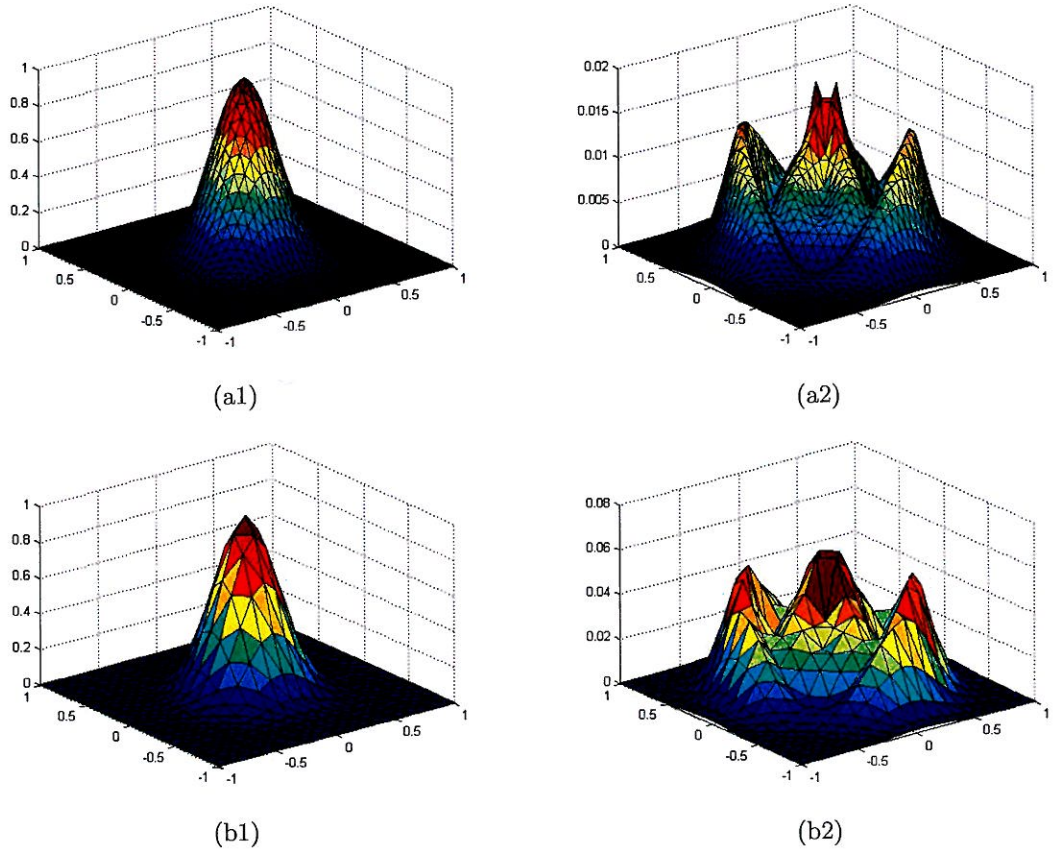
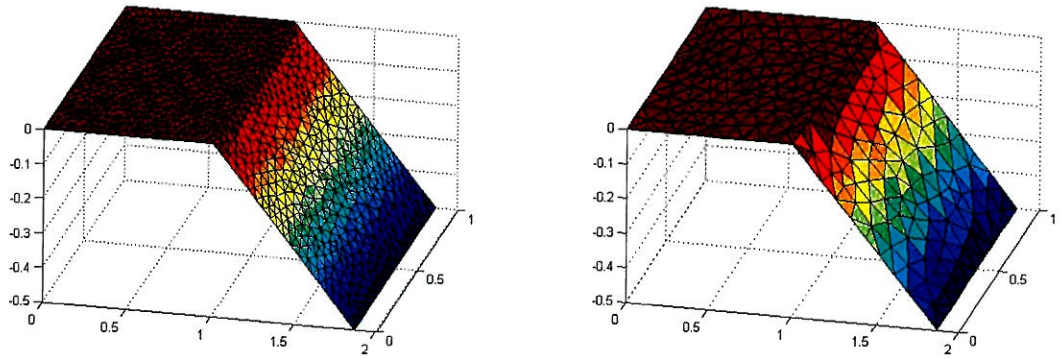


Figure 5.1: Peak test

5.1.2 The *two planes* test

The next test has been implemented in order to check the behaviour of the tolerance angle criterion. The idea is very simple : two planes with an inclination of 30 degrees are meshed and that mesh is simplified. The simplification occurs with a progressively increasing tolerance angle θ. As expected, while $\theta \leq 30$ the common line between the two planes is preserved, but when $\theta > 30$, some collapses happen across this line and the geometry of th intersection is no longer respected. Note that the recognition of sharp edges has been desactivated and the Hausdorff tolerance has been relaxed in order to isolate the effect of the angular criterion. The result below shows that our code has passed the test successfully.



Before Simplification After Simplification

Figure 5.2: *two planes* Test

5.1.3 The *spiders* test

This test has been designed to show the necessity of refining the boundaries. Indeed, in the case the boundary are not simplified, all the boundary nodes join together with some interiors nodes and that makes some *spiders* nodes which in the sense that they have a lot of neighbours elements. Then of course the elements qualities are quite bad and the propagation of the suppression of elements inside the surface is stopped.

The following figure 5.3 shows the interest of the boundary coarsening. The picture (a) is the initial mesh composed of 1682 triangles which will be coarsened. The picture (b) is a simplification that does not deal with boundary nodes. 350 nodes remains after the 3 iterations of simplification. The pictures (c) and (d) shows the results if the boundary nodes are treated as described in section 3.1. (c) is the resulting mesh after one iteration, and (d) is the mesh after 3 iterations. In that last case, only 213 elements remain.

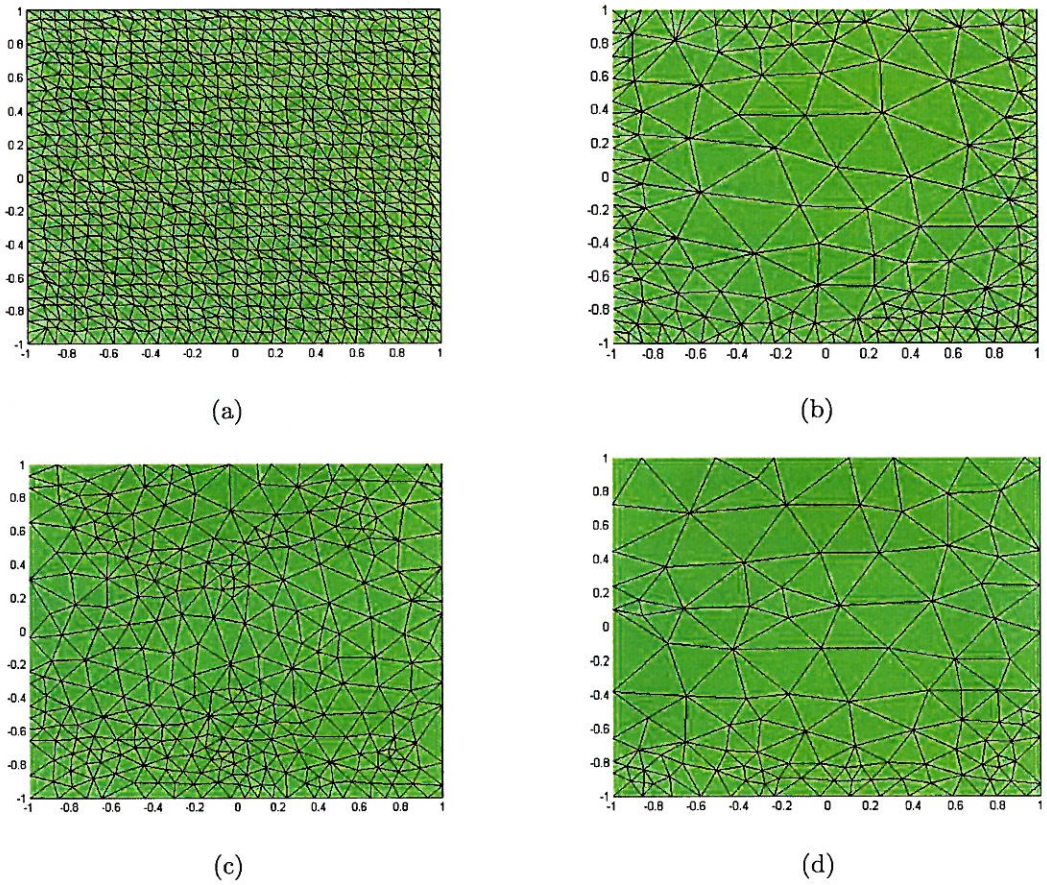


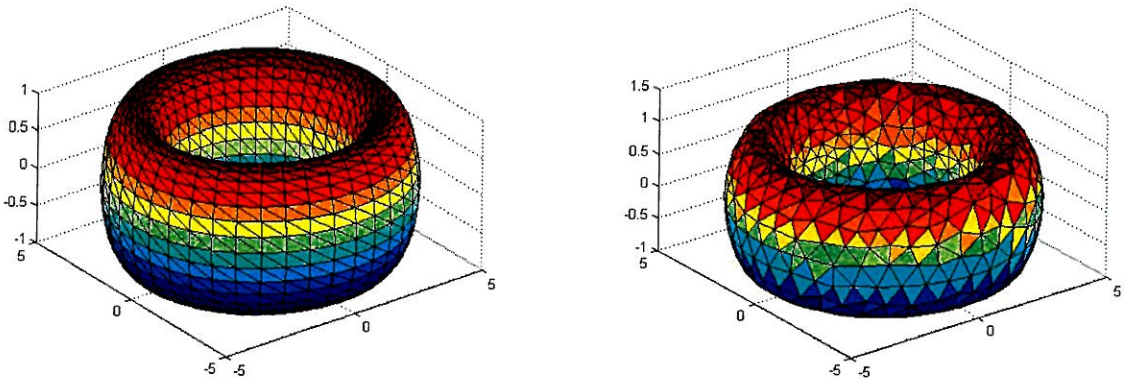
Figure 5.3: Peak test

5.2 Some results obtained with MATLAB

Before exposing the most complex examples performed with our C++ implementation, we will describe some results obtained with our MATLAB code.

5.2.1 Donut shape

The first set of exemples are the cases of closed surfaces, which are quite simple because they do not have boundary lines or sharp lines. The pictures above represents the results obtained after a simplification of a Donut shape. The number of elements has been reduced from 2200 to 825, that means a rate of suppression of 62.5%. The average quality has also been increased from 0.76 to 0.87.



Before simplification

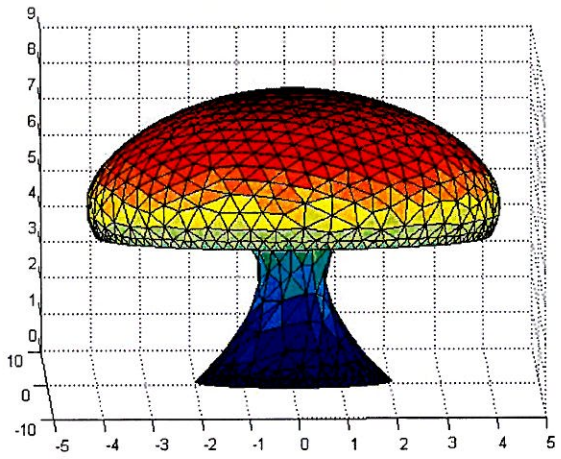
After simplification

Figure 5.4: Results for a Donut shape.

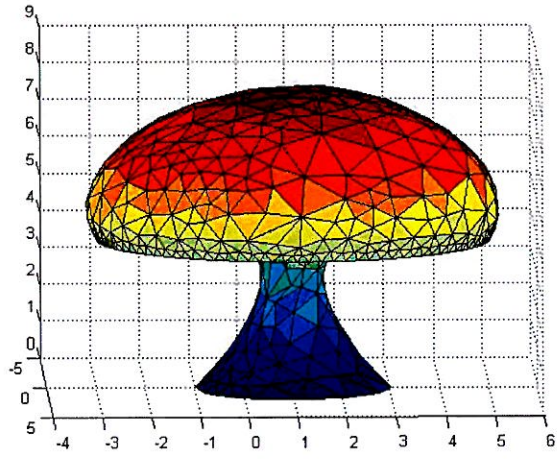
5.2.2 Mushroom shape

This exemple which is much more interesting is a mesh representing a mushroom which shows the interest in refining the sharp line. We will focus our attention on the boundary line (at the basis of the mushroom) and on the sharp line as well. The first picture of 5.5 is the initial mesh (1450 triangles). The second one is the simplified mesh obtained without the special procedure described in the section 3.2. Therefore the nodes of the sharp line (which is actually a circle) have not been suppressed, contrary to the boundary nodes. That left a region around the mushroom's ring, which is needlessly finely meshed and the suppression rate reaches only 35%. The third picture is the result with the suppression of sharp edges. The result is obviously more homogeneous and more elements has been suppressed (around 60%).

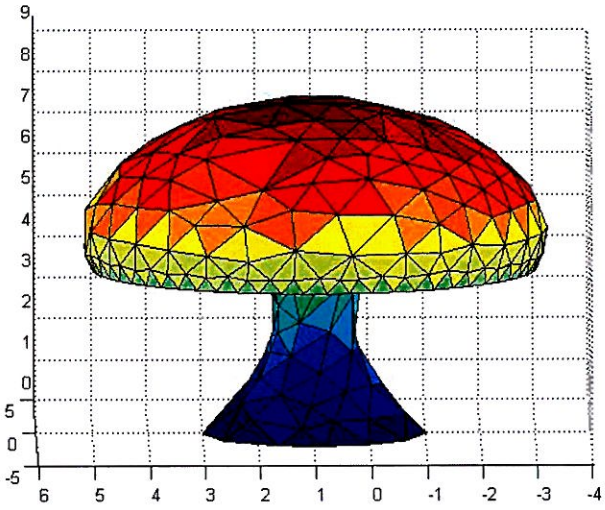
It can be noticed that the nodes of the critical line have been sent on other nodes of the same critical line. Therefore the sharp curve has been preserved. After the suppression, the relocation of the nodes has made their distribution more regular on the sharp line.



Mesh before simplification



Result without the sharp edges treatment



Result with the sharp edges treatment

Figure 5.5: Results for a mushroom shape.

5.3 Results for some open surfaces

The main interest of the open surfaces is to observe how the program works on the surface's boundaries. Some of the following tests are applied on surface meshes designed by the INRIA (french National Research Institute in Computational Sciences and Automatic). These meshes can be downloaded from their webpage : `http://www-roc.inria.fr/gamma/gamma/download/download.php`.

5.3.1 Mesh of a human face

Below is the result obtained after the traitment of the surface mesh of a face (nose and mouth). This mesh contains 11903 nodes. After the coarsening, 4923 elements will be suppressed. The parameters given by the user are: $(\delta_{max} = 10\%, \theta_{max} = 25, \beta = 0.85)$ and the number of iterations $Nb_{iter} = 5$.

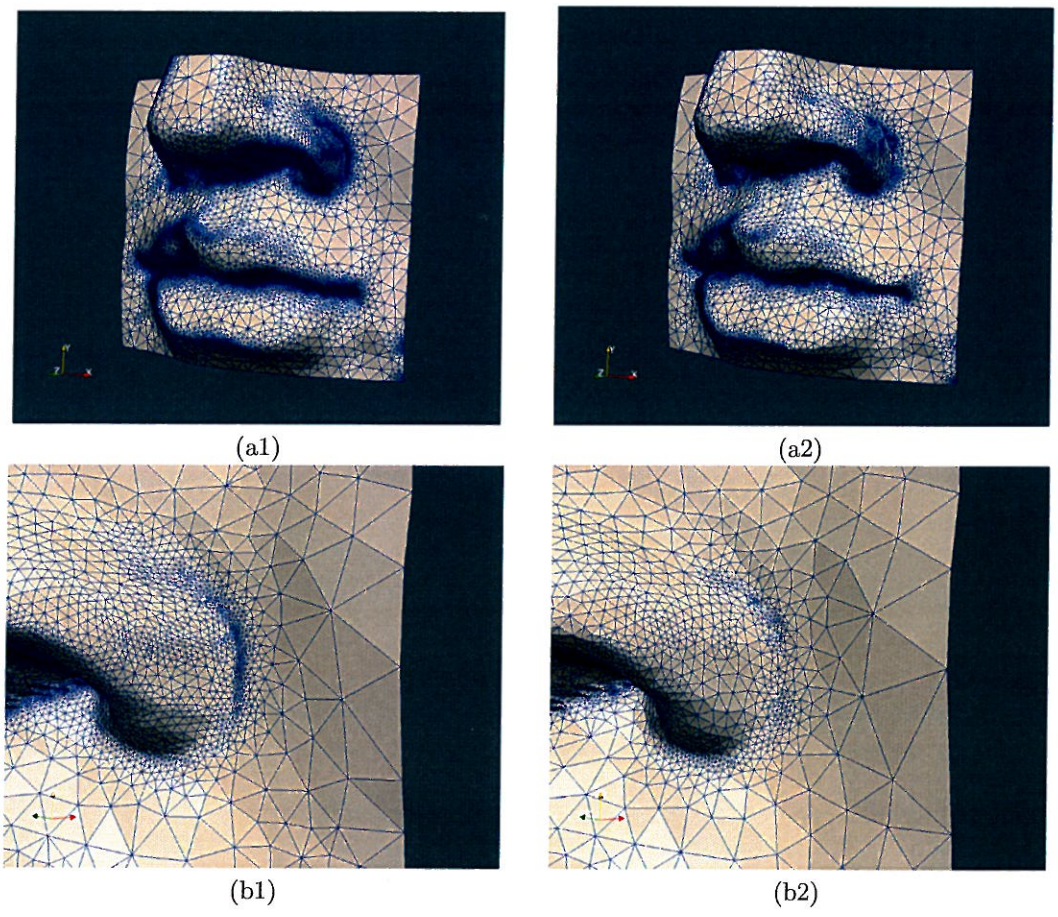
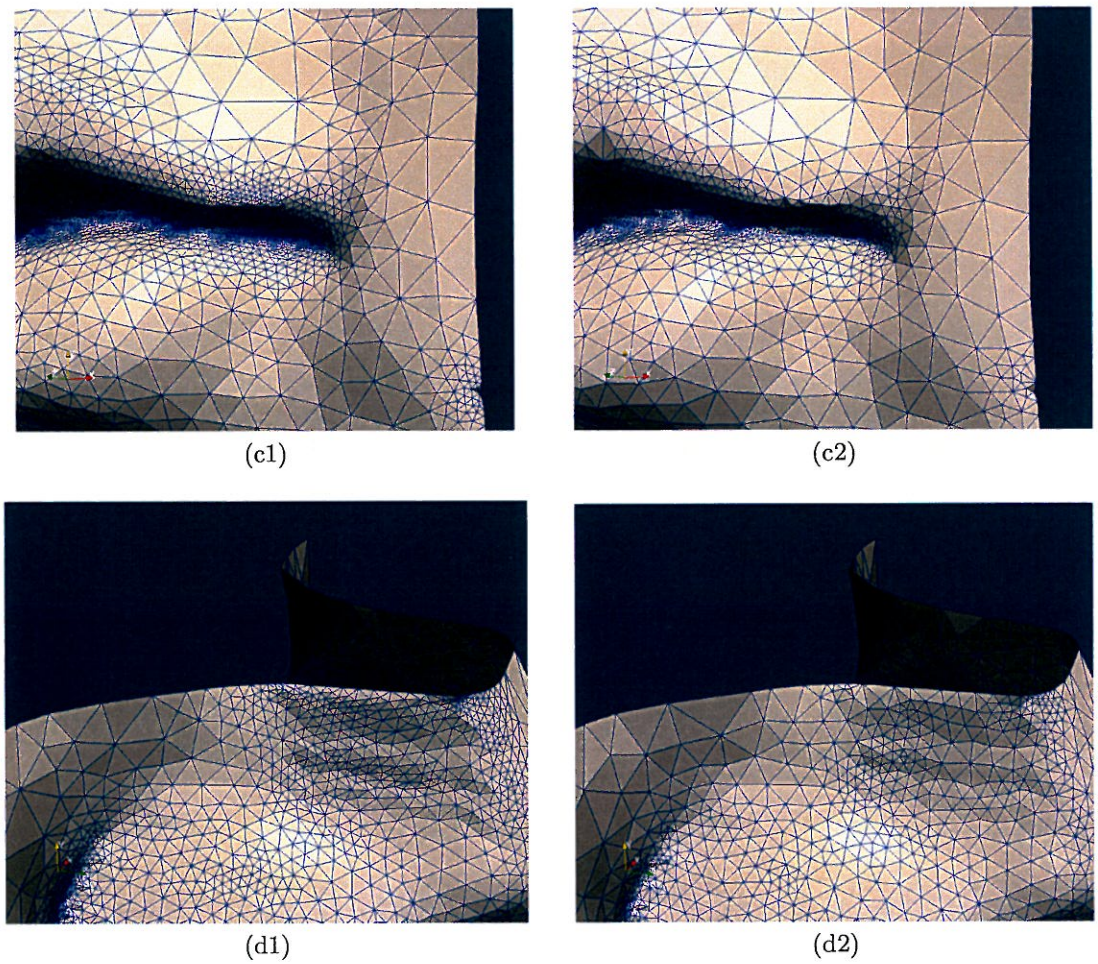


Figure 5.6: Nose and a zoom on the right region. On the left, the pictures before simplification. On the right, after simplification.

With these parameters, the rate of suppression is near 41.4%. On the left hand side are the pictures from the initial mesh, and on the right hand side the pictures of the mesh after simplification. On the figures 5.6 (a1) and (a2), the effects of the coarsening is not obvious, but we can see that the mesh is particularly refined in some regions (then mouth corners and the mostrils). The pictures (b1) and (b2) are a zoom in the nostril. Then the high rate of suppression can be easily observed in this area.

On the figure 5.7 it is interresting to zoom in the mouth corner area. Due to the curvature of the geometry at that point, there is a huge concentration of elements in this area in the initial mesh (c1). The coarsening can be noticed on the picture (c2). Finally the pictures (d1) and (d2) point the coarsening process at the upper boundary of the mesh. The preservation of the boundary is quite clear on these pictures.



(c1) (c2)

(d1) (d2)

Figure 5.7: Zooms on the regions of the lips and on the upper boundary.

5.3.2 Shape coming from an iso-surface

The example treated below is a typical mesh used by the CEA, obtained by an identification of an iso-surface after cutting a cartesian grid. That kind of meshes has very poor quality and are usually over-rafined. Therefore the coarsening process has a double interest for these meshes. Our example mesh is initially made of 356 nodes and 708 triangles (see figure 5.8 (a)). The picture (b) shows the same mesh after the relocation procedure, without any suppression of node. The gain of quality is obvious : the average measure of quality increases from 0.42 to 0.74. After a first iteration of coarsening, only 283 nodes remain (see picture (c)). The suppression rate is then 20.5% with the parameters ($\delta_{max} = 5\%, \theta_{max} = 10, \beta = 0.85$). Most of the bad quality elements have been suppressed, therefore the average quality of the mesh have been increased to 0.89.

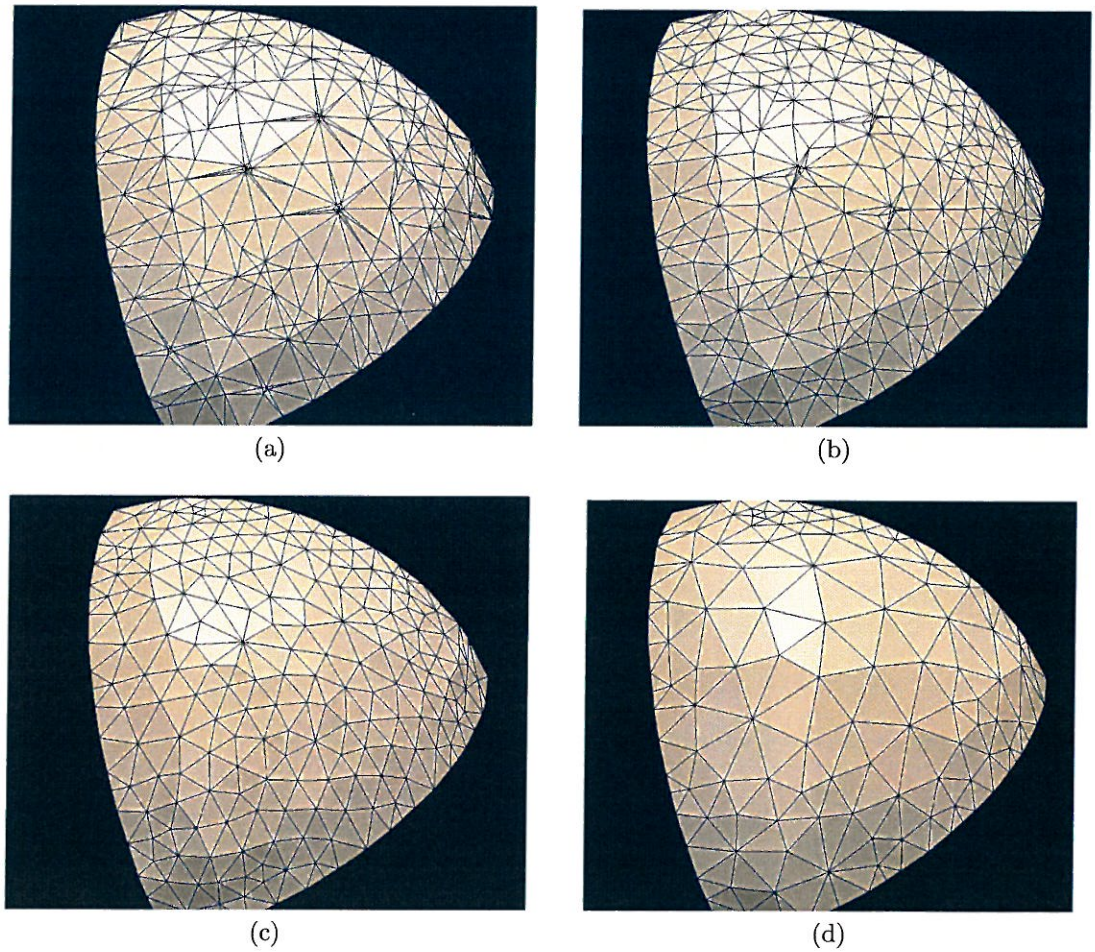


Figure 5.8: Coarsening of a quarter of a sphere obtained from an iso-surface.

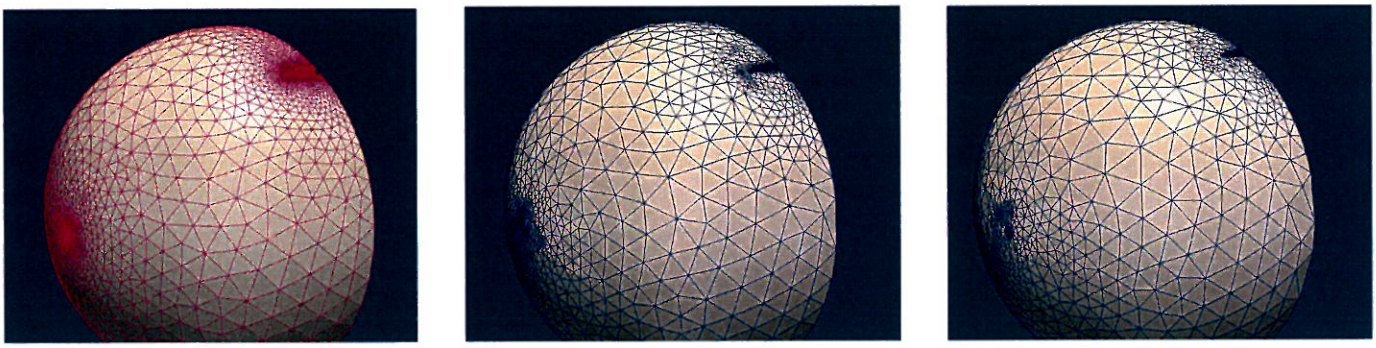
The picture (d) represents the resulting mesh after 3 iterations. At that stage the tolerance parameters have reached : $(\delta_{max} = 15\%, \theta_{max} = 30, \beta = 0.85)$. With these parameters, the rate of suppression is 50.1%. We can see with this exemple that our *code works* quite well *even for meshes with very bad aspect ratios.* (i.e. triangles very longs are adjacents to very small triangles) and for very flat triangles. So the initial quality of our mesh has no impact on the code efficiency.

5.4 Results some close surfaces

For the closed surfaces, there is no boundary lines by definition. But there are still the sharp edges and the sharp curves on which our algorithm can be checked. We will then mainly focuss on the treatment of the initial mesh's geometrical features in this part.

5.4.1 Mesh coming from a CAD file

Below is the result obtained after simplifying a mesh obtained from a CAD file. This mesh represents a geometry of a half-ball drilled with an small hole and with a notch on its side. It is made of 19779 nodes. This mesh presents two points of interest : the area around the hole and the one around the notch. We will then present the results of the global mesh, and then we will zoom in the two areas of interest. You will find on the left hand side the pictures of the reference mesh drawn in maroon, on the center the pictures of the simplified mesh after two iterations and on the right hand side, the same mesh after 4 iterations.
After two iterations, 51% of the elements are suppressed, and using tolerance parameters $(\delta_{max} = 10\%, \theta_{max} = 10, \beta = 0.82)$. After 4 iterations, 79.7% of the elements are suppressed. The parameters given by the user are: $(\delta_{max} = 20\%, \theta_{max} = 20, \beta = 0.82)$.



(a1) (a2) (a3)

Figure 5.9: CAD model refined

On the first row of pictures, one can see that only some areas of the mesh are refined : precisely the ones around the two iregularities mentionned above. The areas made of big triangles remains untouched.

Now, let us zoom in the area of the hole. It is first very interesting to notice that the decimation rate in this region is extremely high, even for a global rate of 51% elements suppressed (b2). Even if a large number of triangles are suppressed, the geometry of the hole is preserved (b2) and (b3) and the triangles sharing an edge with the hole remain quite small. This is due to the special treatment of the sharp edges.

If we zoom in the area of the notch, we will see that the geometry of that notch is quite well respected at the beginning, but if the anular tolerance is to high, then the geometry is no longer respected.

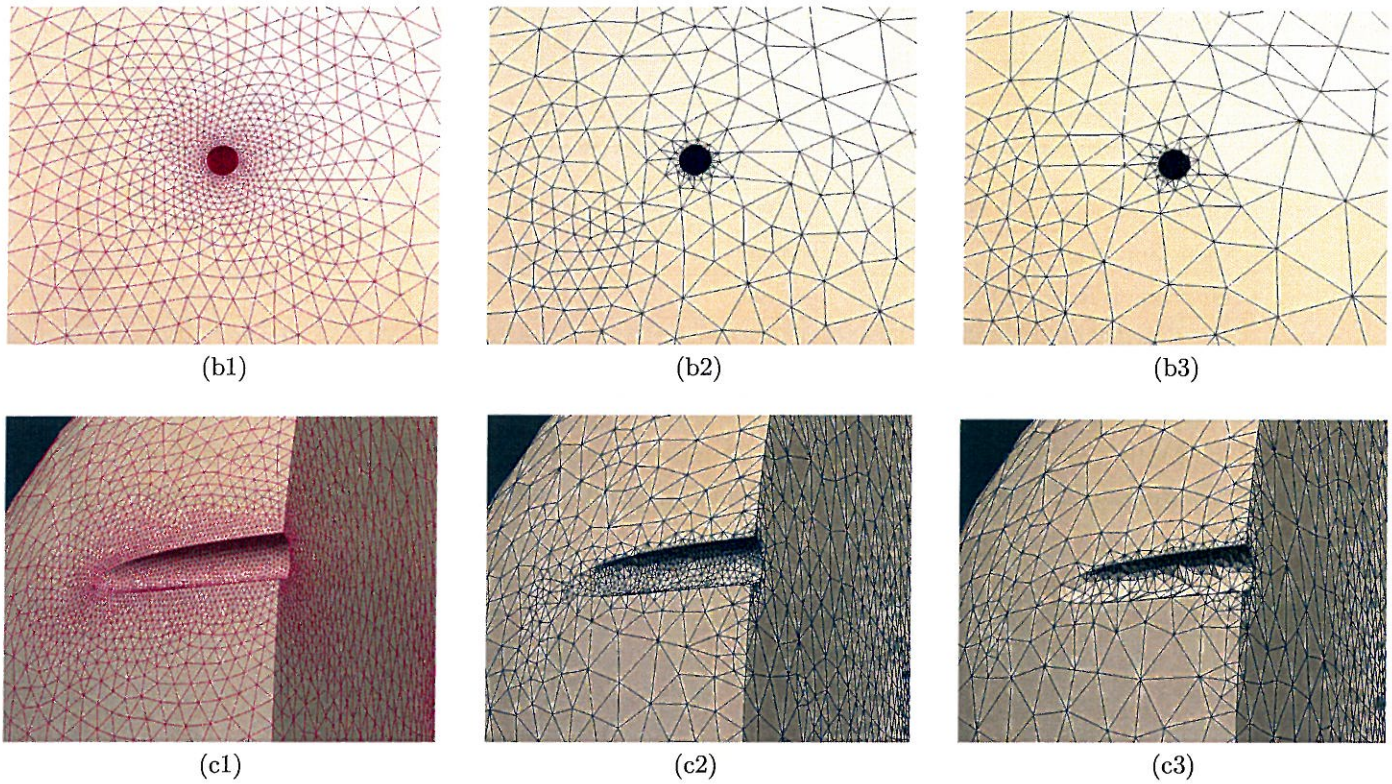


Figure 5.10: CAD model refined

5.4.2 Test with an Adam Kraft statue

In this subsection, we give the results of the most important test that we have performed with our algorithm. The test mesh contains 70072 nodes and 140140 triangles. It represents a statue of Moses made by the great German gothic sculptor Adam Kraft (1455-1508). This mesh has been choosen for the complexity of its geometry and for the high number of elements that allow a quite high suppression rate without damaging to much the geometry.

Three different simplifications has been performed using the parameters reported by the table below with a maximal simplification of 58.5% elements for 10 iterations.

	Reference Mesh	Simplified Mesh 1	Simplified Mesh 2	Simplified Mesh 3
δ_{max}	-	5%	20%	25%
θ_{max}	-	10	25	35
β	-	0.85	0.85	0.85
N_{iter}	0	2	5	10
Nb Sharp nodes	27906	27906	27906	5410
Nodes Suppressed	0	9945	31504	40957
Suppression Rate	0%	14.2%	45%	58.5%

Three different simplifications has been performed using the parameters reported by that table with a maximal simplification of 58.5% elements for 10 iterations. A preliminary remark is that, in order to enforce a greater suppression rate, the treatment of the sharp edges has been relaxed for the simplified mesh 3. Thereore for this mesh, the pre-treatment has recognised a smaller number of Sharp Nodes (5410 instead of 27906) and then the nodes that are no longer marked as *sharp* could have been suppressed more easily.

In all the following set of pictures, the initial reference mesh will be printed in maroon and the simplified ones in blue.

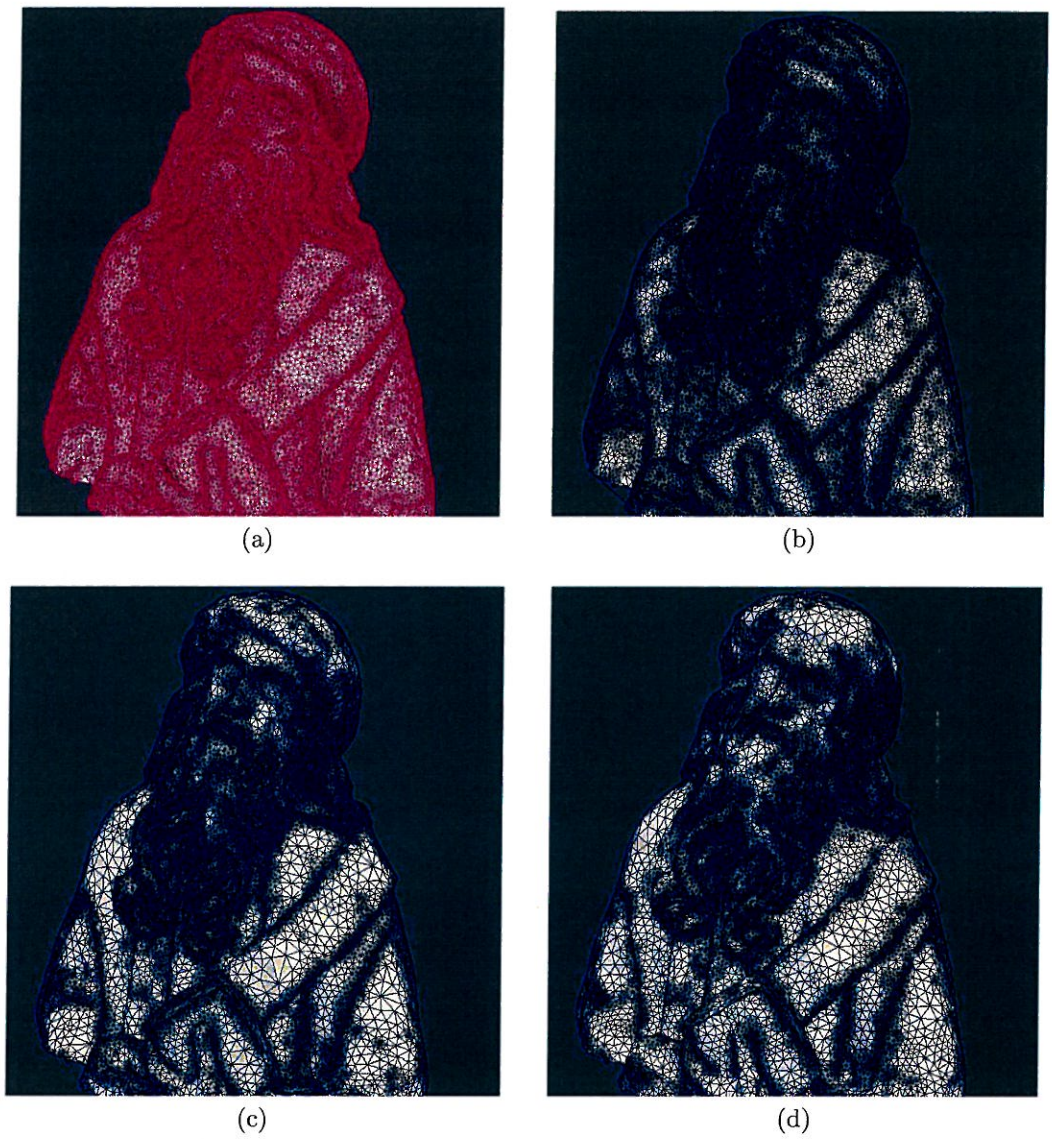


Figure 5.11: General view of Adam Kraft's statue.

On this global view of the statue, it is interesting to notice which regions has been specialy coarsened by the program. It can be remarked that the forehead, the hat, the shoulders and the scarf have been particularly simplified whereas some more iregular regions like the bear or the hair remain very refined.

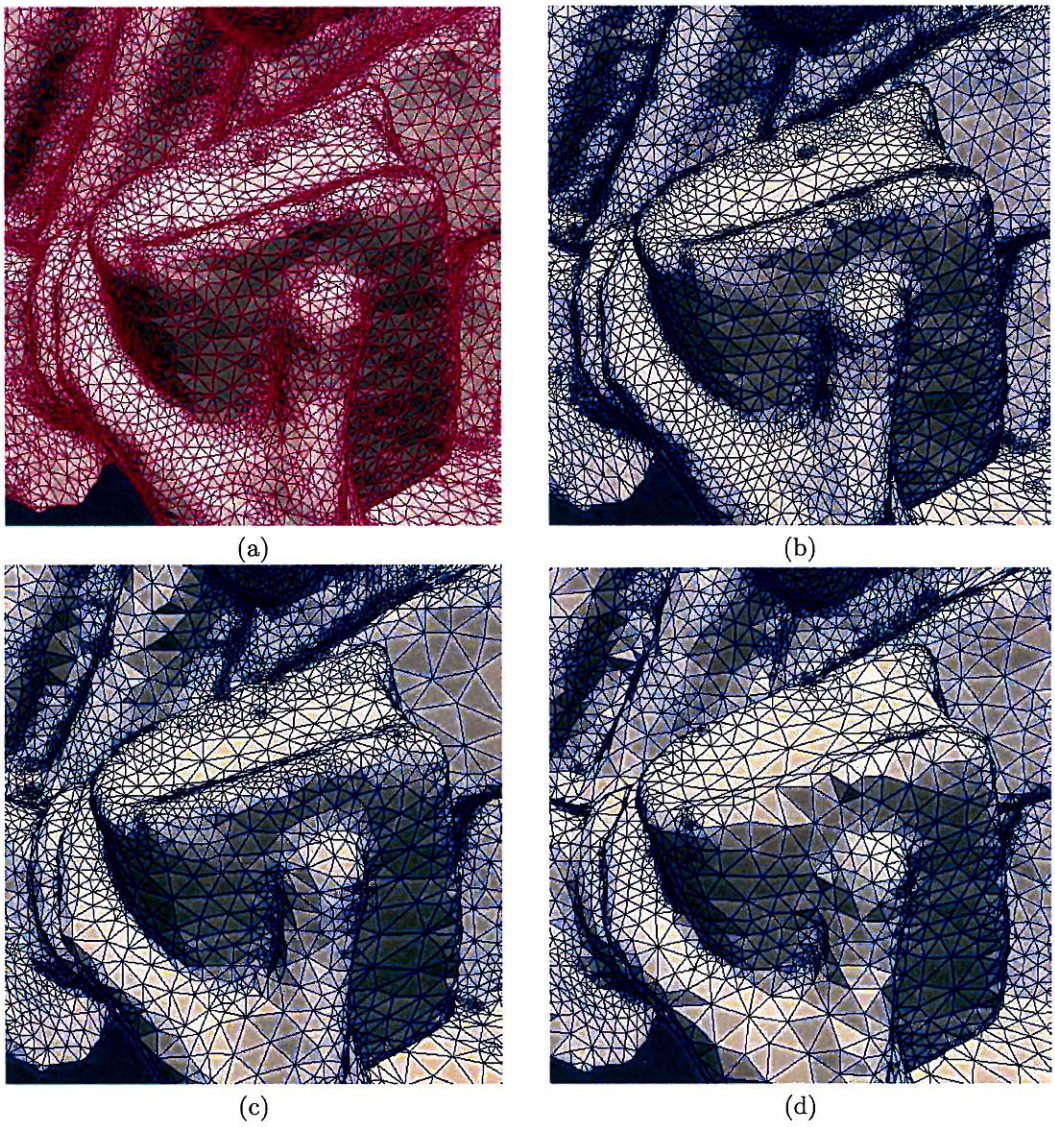


Figure 5.12: Zoom on the book.

The figures 5.12 are a zoom in the area of the book hold by the statue. It is interesting to watch the preservation of the book's cover. If this cover is quite well respected by the simplifications 2 and 3 (pictures (b) and (c)), it is not the case for the third simplification (picture (d)). Indeed in this exemple, the effect of the laxity in the treatment of the sharp lines/edges/nodes is particularly obvious : the program no longer recognize the boundaries of the book's cover as a special geometrical feature to be preserved. Then the consequence is the *melting* of the book's cover and the *fusion* of the Moses'thumb with the book.

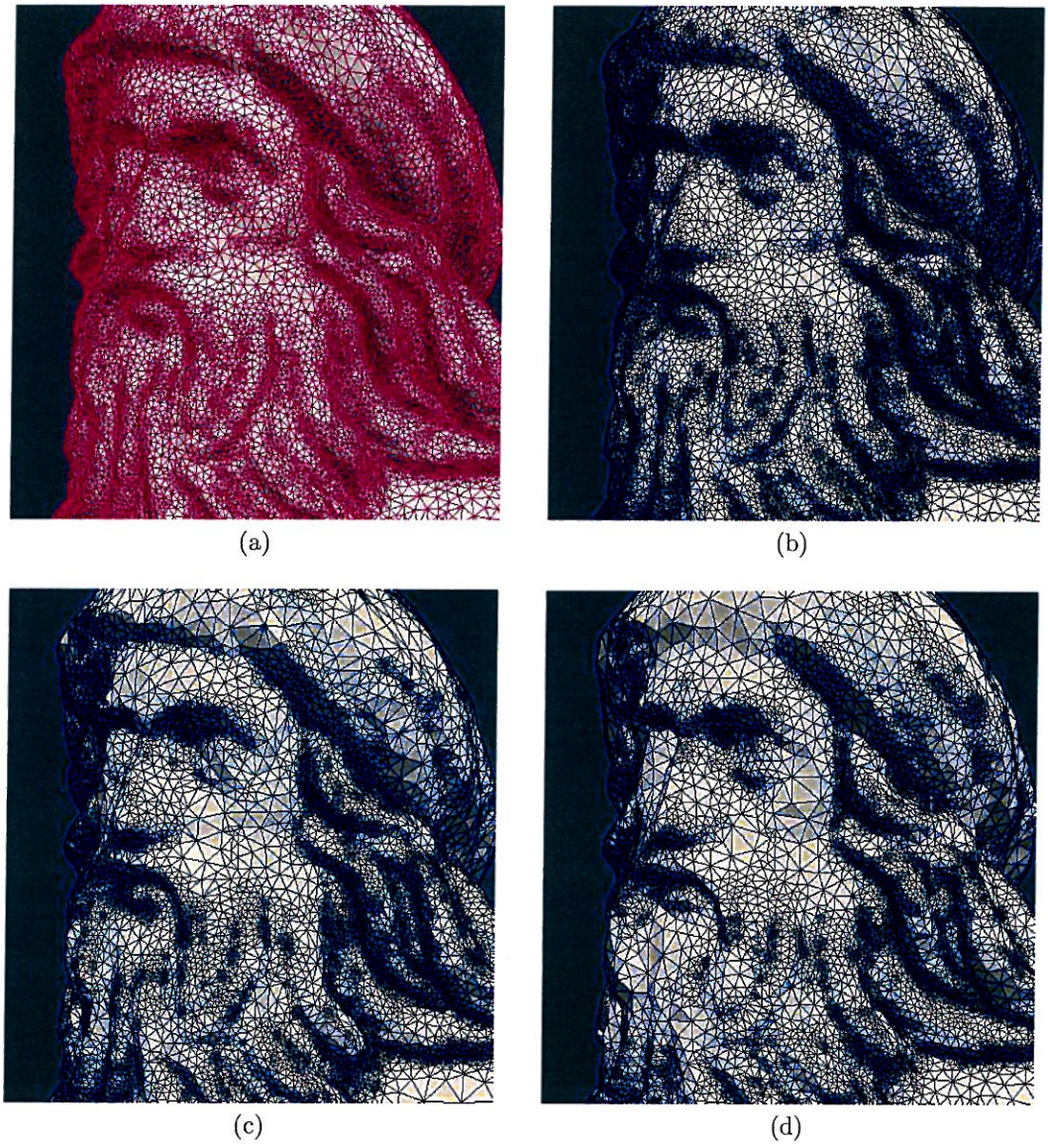


Figure 5.13: Zoom on the statue's face.

The figures 5.13 are a zoom in the area of the statue's face. It is interesting to watch the delimitation between the forehead and the hat. In this case the same precedent remark can be done: there is a fusion between the forehead and the hat after the simplification 3 (picture (d)) baecause of the sharp edges' treatment. Nevertheless some other regions are quite well preserved like the hair and the left eye. Another remark that can be done is the deterioration of the geometry of the cheeks an of the right eye. This deterioration starts with the second simplification (c) and continues with the third

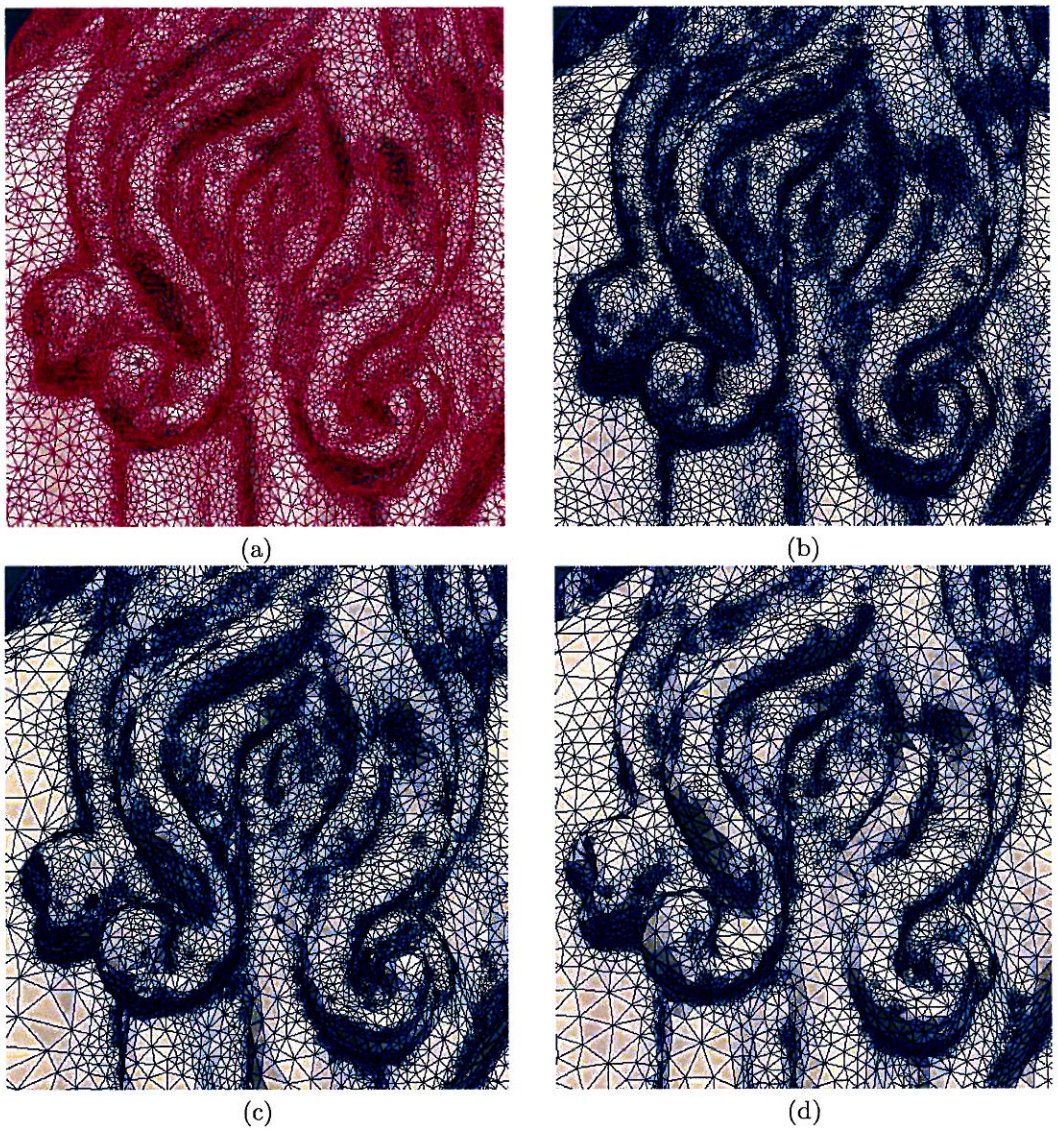


Figure 5.14: Zoom on the statue's bear.

one (d). So it is not due to the sharp line treatment but simply to the tolerance parameters which are to much permissive.

The figures 5.14 are a zoom in the area of the statue's beard. Unlike the precedent exemples, in this region the corsening respect very well the initial geometry of the bear, even for the third simplification. There is no flatening or fusion of the diferent parts of the beard. The reason is probably that the programm has well recognized the geometrical features of the beard during the

pre-treatment part.

Chapter 6

Concluding Remarks and Future Work

6.1 What needs to be done to go further

This repport has described what we obtained and the objectives we have reached after five months of work on the surface simplification problem. But our work is still in progress and one last month of internship with the CEA remains to be done. That is why we are going to explore two axes of improvements under the direction of the Profesors J. Sarrate and F. Ledoux.

The first improvement of our method could be the implementation of a volume control. Indeed, for some physical reasons, it appears that the preservation of the total inner volume of a surface mesh is a very important criterion for the CEA because of the mass conservation. However it seems that so far our simplification algorithm may not respect that criterion with the wished precision. That is why we are going to try to develop a tool that control the volume degradation. The main challenge is that the controled value (the global volume) is global whereas the implementation of all the criterions is local. So we have find the appropriated local control degradation of the volume, which may not be an easy task.

The second possible improvement could be a change in the way the nodes are relocated. Indeed the quadric used for the present relocation is quite rustic, and is not able to capture the surrounding geometry in the case the node is in a distorted area or if the node has a important number of neighbours. Then the programm should be able to detect this kind of nodes and relocate them using a more complex geometric surface whereas the others nodes would keep their rustic quadric.

6.2 Conclusion

It was a big challenge for me to do this master thesis in a very exciting environment (CEA and UPC). It was also a big challenge because I did not have deep knowledges in meshing science before this year. I actually learned a lot in term of programming in object-oriented language. I did not really know how to programm in C++ before and I was not enough sensible to data storage and computational efficiency problems.

Our programm has still some imperfections as you can have noticed reading the part 5 of this repport. But it works quite well for reasonnable tolerances and some significant improvements has been made from the original work of (Borouchaki and Frey, 2005). That is why I am globally proud to have done this thesis.

Chapter 7

Annex: algorithms

Algorithm 7.1 Global Algorithm of the method

1: **procedure** `Algorithm`(X, T) ▷ Coordinates and Connectivities
2: Discriminate nodes on the boundary nodes and on the special curves
3: Store these nodes in `pBC`
4: Create a data structure for the edges : `DatasEdges`
5: set $\delta_0 < \delta$ and $\theta_0 < \theta$ ▷ Tolerances Hausdorff et angulaire
6: **for** n iteration **do**
7: Sort the edges according to their qualities
8: (`X`, `T`) = `EdgeRemoval`(`X`, `T`, `DatasEdges`, `pBC`) ▷ Edge Collapse
9: (`T`) = `EdgeSwapp`(`X`, `T`, `DatasEdges`) ▷ Edge Swapp
10: (`X`) = `EdgeRelocalization`(`X`, `T`, `pBC`) ▷ Edge Relocalization
11: increment Δ and Θ
12: **end for**
13: **end procedure**

Algorithm 7.2 Algorithm of the EDGE COLLAPSE

procedure `EdgeRemoval`$(X, T, DatasEdges, pBC)$
Compute (`N`) = `Neighbours(T)` ▷ Matrix of neighbouring elem for each node
Initialize the pointers : `pX`, `pE` and `pT`
for j=1 **to** number of edges **do**
if the j^{th} Edge has not been already suppressed **then**
Compute the Balls of triangles : $\mathscr{B}(P)$ and $\mathscr{B}(Q)$
Remove the 2 common triangles
if P on boundary **or**(P and Q) on boundary **then**
P is collapsed virtually
Compute: old $q(\mathscr{B}(P))$, new $q(\mathscr{R}(Q))$
Check Orientations of the new triangles
if the degradation from $q(\mathscr{B}(P))$ to $q(\mathscr{R}(Q))$ is okay *and* the direction is okay **then**
Check Conditions on the Normals
if the conditions on the Normals are okay **then**
Check Conditions on H_d
end if
end if
end if
if Q on boundary **or**(P and Q) on boundary **then**
Q is collapsed virtually
Compute: old $q(\mathscr{B}(Q))$, new $q(\mathscr{R}(P))$
Check Orientations of the new triangles
if the degradation from $q(\mathscr{B}(R))$ to $q(\mathscr{R}(P))$ is okay *and* the direction is okay **then**
Check Conditions on the Normals
if the conditions on the Normals are okay **then**
Check Conditions on H_d
end if
end if
end if
If the two config are possible, we choose the one that gives the best quality
$q_{max} = \min q(\mathscr{R}(Q)), q(\mathscr{R}(P))$
if $q_{max} \geq \beta q_{old}$ **then**
if $q_{max} = q(\mathscr{R}(Q))$ **then**
P is collapsed, colnode=P and fixnode=P
else
Q is collapsed, colnode=Q and fixnode=P
end if
end if
if a node is collapsed **then**
if The Edge is not on the Boundary **then**
Put the 2 triangles in a standard config.
else if The Edge is part of the Boundary
Check the Condition on Boundary edges.
end if
if The Edge is not on the Boundary **then**
Find the 2 collapsed and the 2 receiving edges
Update the matrix `N`, the pointer `pT` and the distance H_d
Update the pointers : `pX` and `pE` using recursivity.
else if the Boundary Condition is okay
Find the unique collapsed and the unique receiving edge
Update the matrix `N`, the pointer `pT` and the distance H_d
Update the pointers : `pX` and `pE` using recursivity.
end if
end if
end if
end for
Update the Real datas `X`, `T`, `pBC` and `DatasEdges` using the pointers
end procedure

Algorithm 7.3 Edge Swap

```
1: procedure EdgeSwap(X, T, DatasEdges)
2:     Sort the Edges according to the quality
3:     Initialize n_loop and define the tolerance angle tol
4:     while n_loop < Nmax_loop or a significant nb of node is swapped do
5:         for i = 1 to NbEdges do
6:             the current edge Edge(i) has two neighbouring triangles: tri1 and tri2
7:             if Edge(i) is not on the boundary then
8:                 Check the current configuration of the triangles
9:                 if The configuration is not standard then
10:                    Re-number the triangles in a standard configuration
11:                end if
12:                Compute q_old the old quality of Edge(i)
13:                Compute q_new the new quality as if Edge(i) was swapped
14:                Compute n_1 and n_2 the normals of tri1 and tri2
15:                Compute the new orientation as if Edge(i) was swapped
16:                if angle(n_1, n_2) > tol and q_new > q_old and the orientation is conserved then
17:                    Do the Swapping, which contains the following operations:
18:                    Update the datas of the swapped edges in DatasEdges
19:                    Update the verteces and the connectivities of tri1 and tri2
20:                    Update the datas of all the other edges of tri1 and tri2
21:
22:                end if
23:            end if
24:        end for
25:        n_loop = n_loop + 1
26:    end while
27: end procedure
```

Algorithm 7.4 Relocalization of the nodes

procedure `Relocalization`(X, T, pBC)
  for $i = 1$ to nb of nodes **do**
    Current node : n_i
    if node(i) is neither on the boundary or on a critical curve **then**
      Get $\mathscr{B}(n_i)$ the ball of elements connected to node(i)
      Get the coordinates of the points of $\mathscr{B}(n_i)$
      Compute $\vec{n}_{pseudo}$ the pseudoNormal at n_i
      Compute the vectors of the local basis at n_i : $(\vec{n}_{pseudo}, \vec{e_1}, \vec{e_2})$
      Compute the Transformation matrix $\mathscr{M}$ from the global to the local basis
      $[a, b, c] =$ `getCoeffs`$(\mathscr{B}(\mathtt{n_i}))$ ▷ Get the equation of the Quadric
      Compute the coordinates of $P*$ ▷ Edge Swapp
      Projection of $P*$ on the quadric:
      for all triangles belonging to $\mathscr{B}(node_i)$ **do**
        set U_0 to P^*, $k = 0$ ▷ Initializations
        while $\|U_{k+1} - U_k\|$ is not sufficiently small **do**
          Apply the newton step to the function $F(U_k + t\nabla F(U_k))$:
          $U_{k+1} = U_k - \frac{F(U_k)}{\|\nabla F(U_k)\|^2}\nabla F(U_k)$
          $k = k + 1$
        end while
        return U_{k+1} ▷ Coordinates of the projected Point
      end for
    end if
  end for
end procedure

Bibliography

Borouchaki, H. and P. Frey (2005). Simplification of surface mesh using hausdorff envelope. *Computer Methods in Applied Mechanics and Engineering 194*(48-49), 4864 – 4884. Unstructured Mesh Generation.

Cohen, J., A. Varshnay, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright (1996). Simplification enveloppes. In *Proc. Siggraph'96*, pp. 119–128.

Frey, P. and H. Borouchaki (1997). Maillage géométrique de surfaces. partie ii: appauvrissement. In *RR-INRIA*, Volume 22.

Hamann, B. (1993). Curvature approximation for triangulated surfaces. In *Comput. Suppl.*, Volume 8, pp. 139–153.

Hamann, B. (1994). A data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design 11*(2), 197 – 214.

Hoppe, H., T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle (1993). Mesh optimization. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, pp. 19–26. ACM.

Ledoux, F., Y. Bertrand, and J. Weill (2009). Generic programming for designing a mesh data structure. In *11th ISGG Numerical Grid Conference.*

Roca, X. and J. Sarrate (2005). An automatic and general least-squares projection procedure for sweep meshing. In *Comput. Methods Appl. Mech. Engrg.*, Volume 1, pp. 10–30.