UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

João Vitor de Sá Hauck

Iterative edge length interval constraining in triangular meshes based on local parametrization

Juiz de Fora

2015

UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

João Vitor de Sá Hauck

Iterative edge length interval constraining in triangular meshes based on local parametrization

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Marcelo Bernardes Vieira

Juiz de Fora

2015

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

Hauck, João Vitor de Sá.
Iterative edge length interval constraining in triangular meshes based on local parametrization / João Vitor de Sá Hauck. -- 2015.
74 p.

Orientador: Marcelo Bernardes Vieira
Dissertação (mestrado acadêmico) - Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação, 2015.

1. Iterative remeshing. 2. Edge length equalization. 3. Edge length constraining. 4. Local parametrization. I. Bernardes Vieira, Marcelo, orient. II. Título.

João Vitor de Sá Hauck

Iterative edge length interval constraining in triangular meshes based on local parametrization

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Aprovada em 18 de Setembro de 2015.

BANCA EXAMINADORA

Prof. D.Sc. Marcelo Bernardes Vieira - Orientador
Universidade Federal de Juiz de Fora

Prof. D.Sc. Rodrigo Luis de Souza da Silva
Universidade Federal de Juiz de Fora

Prof. D.Sc. Sócrates de Oliveira Dantas
Universidade Federal de Juiz de Fora

Prof. D.Sc. Vitor Rafael Coluci
Universidade Estadual de Campinas

ACKNOWLEDGMENTS

I'd like to thank my family, specially my parents Kelle and Marcelo for their support. I also would like to thank my friends from the GCG, PGCC and the ones from the graduation. Additionally, I'd like to specially thank my girlfriend Alessandra. She was the only one with me since the graduation, and she helps with the design of some figures. Finally, I'd like to thank Marcelo Caniato for helping with my troubling text and Marcelo Bernardes for advising me during these two years of work.

"You will find men who want to be carried on the shoulders of others, who think that the world owes them a living. They don't seem to see that we must all lift together and pull together."

(Henry Ford)

RESUMO

Malhas com restrições no comprimento das arestas são úteis para diversas aplicações, especialmente para simulações de processos químicos e físicos. Este trabalho apresenta um método iterativo para remalhar uma malha triangular arbitraria de variedade 2 em uma malha com o comprimento de todas as arestas dentro de um intervalo de restrição definido pelo usuário. O método usa operações estelares para ajustar a quantidade de vértices e triângulos no modelo e para melhorar a valência dos vértices. Ele também aplica o operador Laplaciano em um espaço paramétrico local para melhorar a distribuição de vértices sobre a superfície. Propõe-se, uma otimização não linear, aplicada localmente, para os casos em que a malha é praticamente regular. Perdas geométricas são evitadas pela realização de uma projeção sobre a superfície original. O método proposto resulta em uma malha praticamente regular, com os vértices distribuídos uniformemente sobre a superfície. A dual da malha é usada em simulações de nano estruturas de carbono como uma aplicação do método. A principal contribuição deste trabalho é uma nova abordagem para restringir explicitamente o comprimento das arestas em um intervalo dado. Nosso método ainda garante baixa perda global de geometria e baixo custo de memória em comparação com métodos disponíveis na literatura.

Palavras-chave: Método iterativo. Equalização do comprimento das arestas. Restrição do comprimento das arestas. Parametrização local.

ABSTRACT

Meshes with constraints in the edge length are useful for several applications, specially for chemical and physical simulations. This work presents an iterative method for remeshing an arbitrary triangular 2-manifold mesh into a mesh with all edge lengths within an user-defined constraining interval. The method uses stellar operations to adjust the amount of vertices and triangles in the model and for improving the valence of the vertices. It also applies the Laplacian operator in a local parametric space to improve the distribution of the vertices over the surface. We propose a nonlinear optimization, locally applied, for cases in which the mesh is almost regular. Geometric losses are prevented by performing a projection over the original surface. Our method results in a nearly regular mesh, with vertices uniformly distributed over the surface. The dual of the mesh is used in simulations of carbon nanostructures as an application of the method. The main contribution of this work is a new approach for constraining the edge length within an explicitly given interval. Our method also ensures lower global geometry losses and lower memory cost in comparison to methods available in the literature.

Keywords: Iterative remeshing. Edge length equalization. Edge length constraining. Local parametrization.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Geometric models are essential for several computer applications, such as real time rendering, simulations of physical and chemical phenomena. However, as computers are discrete machines, these models must be discretized in order to be represented on them. These discrete representations are commonly named meshes, and are usually composed of vertices and polygons which represent the original surface. This growing need for meshes leads to the development of several techniques for mesh generation. For example, there are direct modeling softwares (DEMBOGURSKI et al., 2013; SCHöBERL, 1997) and techniques based on computer vision for 3D-scanning (ROCCHINI et al., 2001; VIEIRA et al., 2005).

A number of techniques like those mentioned before have been largely used for creating meshes. Despite this, in general, meshes built with those techniques are usually not well sampled. Besides, they may also contain polygons which are almost degenerated. A degenerate polygon contains an internal angle equal to zero, or a repeated vertex. A model with these characteristics rarely fits the requirements for being used in simulation methods. On that matter, remeshing methods can improve some quality aspects of those meshes. For this reason, development of remeshing methods becomes an interesting research field.

Defining general quality measures is not an easy task. Despite Bommes et al. (2009) points out some commonly required quality aspects, a globally valid quality measure may never be proposed, as it usually depends on the constraints required by each specific application. For instance, real-time applications may require a simplification of the original mesh, in order to improve its performance. On the other hand, physical and chemical applications (IIJIMA et al., 1991; RAPAPORT, 1996) demand several constraints capable of guaranteeing the fidelity of the results, such as valid vertices valence (number of incident edges), edge length in a precise interval, overall vertices distribution, angle restrictions, and so on. Even when there is no evident requirement, an improvement on connectivity, regularity and vertices distribution may attenuate numerical errors, improving the performance for a large array of applications.

1.1 PROBLEM DEFINITION

Let $\mathcal{M}$ be an arbitrary 2-manifold triangular surface immersed in $\mathbb{R}^3$. This mesh $\mathcal{M}$ can be divided into (V, E, P), where V are the vertices, E the edges and P the polygons of $\mathcal{M}$. Given such a mesh and an interval $[e_{min}, e_{max}]$, the problem is to generate a mesh where every edge $e \in E$ has its length constrained to the interval $[e_{min}, e_{max}]$ as in Equation 1.1. This constraint must be enforced while preserving the original topology and also fairly preserving the original geometry.

$$e_{min} \leq |e| \leq e_{max}, \forall e \in E \tag{1.1}$$

There are some additional features that can be improved. As for instance, the valence of an vertex should be 6 if it lies in the interior of a surface, and 4 if it lies on the border (SURAZHSKY; GOTSMAN, 2003). A vertex that attends this is called a regular vertex, by contrast if it does not attend it is named a irregular vertex. This is also an advantage for carbon nanostructures simulations, this will be became clearly as you read the next chapters. Also the average edge length should be as closer to $\frac{e_{min}+e_{max}}{2}$ as possible and the standard deviation should be very small.

1.2 OBJECTIVES

The main goal of this work is to propose a method for edge length interval constraining capable of preserving the topology of the input mesh, besides minimizing the geometric distance between the output mesh and the original one. It is also essential to reduce the amount of vertices with extremely irregular valence.

A secondary objective is to improve the distribution of vertices, polygons and angles in trivalent meshes. Trivalent meshes are meshes in which each vertex has its valence equal to three. This makes the processed mesh more suitable for usage as a carbon nanostructure, for example. This is possible because we can convert the trivalent mesh into a triangular by computing its dual, and after we process the triangular mesh we compute the dual again. Additionally, an analysis of the behavior of the resulting meshes in physical simulations shall be presented.

2 RELATED WORKS

As mentioned before, geometric models are needed for a huge number of applications. In general, however, the available meshes do not meet the requirements for specific applications. Thus, a variety of surface remeshing methods have been proposed. These methods can be divided mainly into two categories: global parametrization (2.1) and explicit remeshing (2.2). The first category is more formal, and usually has a finer control of the final mesh. However, computing a global parametrization is complex and consequently slow. On the other hand, explicit remeshing applies simpler operations directly over the mesh until some stopping criterion is reached. This is faster than the global parametrization, but does not present any guarantee that the final objective will be reached.

2.1 GLOBAL PARAMETRIZATION APPROACH

Global parametrization methods usually create a bijective map of the surface in $\mathbb{R}^3$ onto a $\mathbb{R}^2$ disk-like surface. However, creating a global parametrization presents a series of drawbacks. For instance, the result greatly depends on the parametrization strategy. Besides, in order to compute a global parametrization the model has to be homeomorphic to the disk. As an arbitrary model may not be homeomorphic to the disk, a graph cut algorithm must be used to open the model in a plane. As a result, these methods depend as well on the cut scheme.

Despite these drawbacks, there are several works using this strategy in an effort to create remeshing algorithms. For example, the work by Bommes et al. (2009) uses a scheme called N-Symmetric fields (RAY et al., 2008) to build a parametrization, in an effort to create a highly regular quadrangular mesh. This method presents good results, but it is quite complex and computationally expensive, as it requires solving a integer mixed system of equations. Even though this work is not directly interested in edge lengths or trivalent meshes, the work of Pampanelli (2011) uses it to produce a high quality trivalent mesh. This trivalent mesh may be used for physics simulations or for generating a regular triangular mesh by simply computing its dual.

Although the work of Huang et al. (2011) has another goal, it also uses the parametrization proposed by Ray et al. (2008). It focuses on obtaining a mesh in which the angle

between two arbitrary edges is approximately 60 degrees. Moreover, this method requires a set of feature lines in order to compute the N-Symmetric field. These feature lines can be either given by the user, or estimated by the algorithm. This estimation also has a high computational cost. This work is not directly concerned with edges lengths, but its restrictions lead to a mesh where almost all lengths are equal.

The work presented by Pietroni et al. (2010) is able to create an isometric mesh, but it also requires a set of feature lines. As stated before, estimating this set is computationally expensive. Additionally, this work generates a mesh where all edges lengths tend to be equal.

The drawbacks of global parametrization and its usual poor computational performance were already mentioned in the beginning of the section. As a direct effect of that, these methods usually cannot be used for geometrically or topologically complex, or even large, models. As our work deals with such models, we chose an explicit remeshing approach.

2.2 EXPLICIT REMESHING APPROACH

Explicit remeshing methods usually make use of local/global operations for adjusting the mesh until it reaches some kind of stopping criteria. These methods are commonly faster than methods based on global parametrization. As a consequence, there are plenty of methods based on this approach. For instance, Botsch and Kobbelt (2004) propose a method based on local operations followed by global relaxations. This strategy is somewhat like ours, but it cannot be used when the desired edge's length mean is far from the original length average.

The work of Surazhsky and Gotsman (2004) presents a method for creating a suitable mesh for morphing. In order to achieve such a mesh, it first tries to equalize all triangle's areas. This equalization leads to an uniform distribution of vertices, though the triangles may be almost degenerated. So it alternately performs angle smoothing and weighted angle-improving edge flips. Although this work is not interested in lengths of edges, its method leads to an almost uniform edge length distribution. This method presents good results, but it can only be used in planar meshes. It cannot be used for constraining the edges lengths either.

Vorsatz et al. (2003) presents another very interesting work, which uses a set of local

operations combined with relaxations. These relaxations occur in a local parametric space. This allows it to overcome the main drawbacks of the global parametrization, while preserving the ease of working in a 2D space. Even though this method is not directly concerned with edges lengths, it also requires an interval for them, but this interval is used for controlling the desired mesh resolution. The interval in their work also has to fulfill the condition $e_{max} \geq 2e_{min}$. Our work uses the key idea of local parametrization, but we overcome this interval limitation.

For the problem of edge length equalization, an explicit remeshing method was proposed by Peçanha et al. (2013). In an effort to accelerate its convergence, this algorithm creates a priority list for the edge split and edge collapse operations presented in Section 3.5. Another important contribution is the division of each iteration into four steps. Each step has a well defined goal. The first step adjusts the amount of vertices and polygons over the mesh, while the second step improves the connectivity of the mesh. Then, the third one uniformly distributes the vertices over the surface. Finally, the last one attenuates the geometric distance to the original surface. Our method uses this elegant division as well as the priority list proposed. Plus, differently from Peçanha et al. (2013), we also make use of the priority list during the second step. We made advances in the third step as well.

Our previous works (HAUCK et al., 2014, 2015) were the firsts in which edges lengths were explicitly constrained into a given interval. We also proposed a new measure of error used either as a post-processing step (HAUCK et al., 2014) or included in later iterations as a replacement for the well known Laplacian filter (HAUCK et al., 2015). The method developed by (HAUCK et al., 2015) was later used by Silva (2014) for modeling carbon nanostructures.

In this dissertation the method presented in (HAUCK et al., 2015) is improved in two different ways. Firstly, we propose a classification scheme for the edge flip operation. In possession of this classification, a new priority list is built and the edge flip operations occur in accordance with it. Finally, we make use of a local parametric space for the Laplacian filter whenever it is possible. This allows the Laplacian filter to become simpler while its effectiveness is increased, as will be discussed in Chapter 5.

3 FUNDAMENTALS

For a better understanding of this dissertation, some concepts and definitions must be elucidated. As this work uses polygon meshes for representing 2-manifold surfaces, a briefly description of N-manifolds, surfaces, surface representation and polygon will be presented. We will also discuss some stellar operations used by our method, namely the edge flip, edge collapse and edge split operations. Other topics to be discussed include parametrization schemes and the Laplacian operator. Even though this method is based on explicit remeshing, it locally uses a parametrization for improving the Laplacian operator. In order to clarify the notation used to represent vertices in this work, we use U when the vertex is on the parametric space and V otherwise.

3.1 N-MANIFOLD

An *N-manifold* is a space where the neighborhood of each point can be approximated by an euclidean space of dimension N (GUILLEMIN; POLLACK, 1974). This work focuses on 2-manifold spaces. More precisely, it focuses on 2-manifold surfaces immersed in the Euclidean space of dimension 3, commonly known as $\mathbb{R}^3$.

3.1.1 SURFACE

A *surface* is a 2-manifold space immersed in $\mathbb{R}^3$. However, this work is interested only in compact orientable surfaces. A surface is called *compact* if it has a finite area. In addition, a surface is *orientable* if it has two sides. Although it is not intuitive, there are surfaces which are not orientable. A classical example is the Möbius strip, depicted in Fig (3.1). This surface can be obtained by giving a paper strip a half-twist before connecting its two ends, thus forming a loop.

3.2 SURFACE REPRESENTATION

Among the ways in which a surface can be represented, we can point out the parametric and the implicit forms. *Parametric* surfaces are represented by a vectorial function $f : \mathbb{R}^2 \rightarrow S : \mathbb{R}^3$. The function f maps the two-dimensional parametric space $\Omega \subset \mathbb{R}^2$ onto the

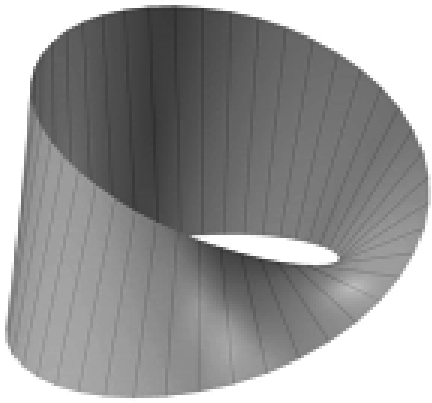


Figure 3.1: The Möbius strip, an example of a non-orientable surface (KUSHNER et al., 2007).

surface S immersed in $\mathbb{R}^3$. On the other hand, in the *implicit* representation, the surface is represented by the zeros of a function $F(x) = 0$, where $x \in \mathbb{R}^3$. Despite the elegance of these two forms of representation, a complex surface usually cannot be represented by an unique function. In order to represent these complex surfaces, the domain must be divided, defining a separate function for each piece of the domain. These functions only need to approximate the surface locally, so they can be simpler than another one which approximates the surface globally. In this work, surfaces are divided in triangles and a linear function is used for locally approximating the surface. To do so, it uses polygon meshes, which will be detailed in the next section.

3.2.1 POLYGON MESHES

As computers are discrete machines, continuous surfaces must go through some sort of discretization process in order to be represented in computers. Nowadays, the most common approach is to represent surfaces as polygon meshes. Figure 3.2 shows how polygon meshes may be used to represent real objects with high fidelity.

A *polygon mesh* is a simple representation for surfaces, where the surface is locally approximated by planar polygons. The mesh is structured into sets of faces, edges and vertices. Each vertex represents a point in the space. An edge is a connection between two vertices, and a polygon, also known as a face, is a planar closed set of edges. In addition, the polygons are usually (but not always) convex due to the simplicity of convex polygons.

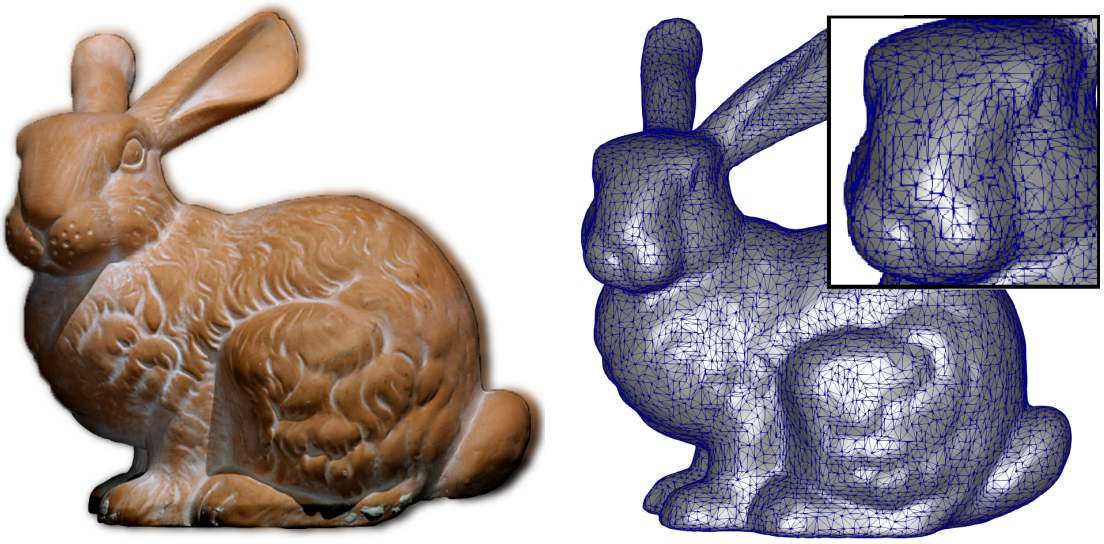


Figure 3.2: The real bunny object, and its polygon mesh representation using the Graphite renderer (ALICE, 2000-2012).

The vast majority of applications, including this one, focus on triangular meshes. This makes sense, since the triangle is the simplest polygon which defines a plane. Plus, since three points are needed for defining a plane, a triangle is the only polygon which defines it without ambiguity. In addition, convex polygon meshes may be converted into triangular meshes without great effort by a process called triangulation.

This work also benefits from another property of triangular meshes: the dual of a triangular mesh is a trivalent mesh, which has several applications in chemical and physical simulations. Examples of these applications will be presented in Chapter 6.

3.3 DUAL MESH

Given a 2-manifold mesh, its dual is another 2-manifold mesh with the same topology. Each vertex of the primal mesh is represented by a polygon in the dual and, likewise, the polygons of the primal mesh are represented by vertices in the dual mesh (TAUBIN, 2001).

The number of vertices of a polygon on the dual mesh corresponds to the valence of its correspondent primal vertex, e.g. a vertex with valence 3 forms a triangle in the dual, while one with valence 6 forms a hexagon. This allows us to deal with triangular meshes, more common and easy to handle, instead of directly processing trivalent meshes to model carbon nanostructures.

3.4 REMESHING

Given an input mesh $\mathcal{M}$, a remeshing method is one capable of generating a new mesh $\mathcal{M}'$, which fairly represents the original mesh $\mathcal{M}$ while at the same time improving some quality criteria of $\mathcal{M}$ (BOTSCH et al., 2010). There is not a precise criteria for defining when the mesh $\mathcal{M}'$ fairly represents $\mathcal{M}$. Even though each application may use its own measure of similarity between $\mathcal{M}'$ and $\mathcal{M}$, some criteria are used more often. These criteria may be geometric, topological, or even based on keeping some key vertices unchanged. For the quality aspects, in an analogous way to the similarity aspect, each application can also define its own quality measure for being improved. Although Bommes et al. (2009) enumerates some commonly used criteria, these quality aspects depend more on the application than the similarity ones. Thus, a quality aspect which is essential in an application may be useless in another one. This dissertation is mainly concerned with the edges' length and the valence of the vertices, as stated in Section 1.2.

3.5 STELLAR OPERATIONS

There are several kinds of operations for changing the mesh's structure. Stellar operations are a subset of these operations which preserve the mesh's topology. They can be divided into three groups in accordance with how they change the amount of faces. If an operation increases the amount of faces, it is called a refinement operation. However, if the amount of faces diminishes, the operation is classified as a simplification. Otherwise it is a neutral operation.

This work uses only three stellar operations. Those are *edge split* (refinement), *edge collapse* (simplification) and *edge flip* (neutral). Each one of them will be addressed in the following sections.

3.5.1 EDGE SPLIT

This operation splits an edge into two new edges. The process consists of inserting a new vertex somewhere over the edge. Then the new vertex is also connected to its opposed vertices. At the end of the operation, the amount of vertices increases by one, the number of edges by three, and the amount of triangles increases by two. The operation is depicted in Figure 3.3.

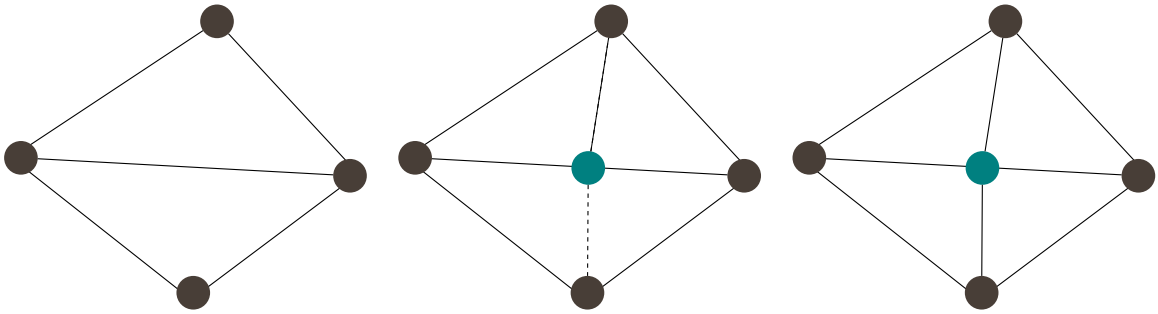


Figure 3.3: Example of an edge split operation.

3.5.2 EDGE COLLAPSE

The collapse operation basically removes the edge being collapsed and one of its vertices. Each edge previously connected to any vertex of the removed one is then connected to the preserved vertex. As a consequence of this operation, an edge, a vertex and two triangles of the mesh are removed. This operation is illustrated in Figure 3.4.

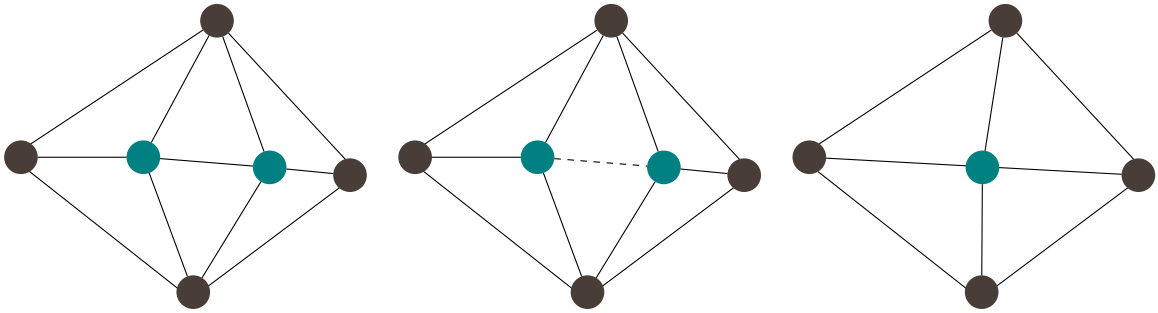


Figure 3.4: Example of an edge collapse operation.

3.5.3 EDGE FLIP

This operation has no effect in the total amount of vertices, edges or polygons, so it is used in an effort to improve the mesh's connectivity. As depicted in Figure 3.5, the flip operation consists in the replacement of an edge a_i, by a new one connecting the opposing vertices of a_i. In this work, we only consider a flip valid if it does not decrease too much the minimum angle of the affected triangles. More precisely, the flip is valid if the minimum angle after the flip is greater than half of the original minimum angle. This prevents the creation of degenerate triangles.

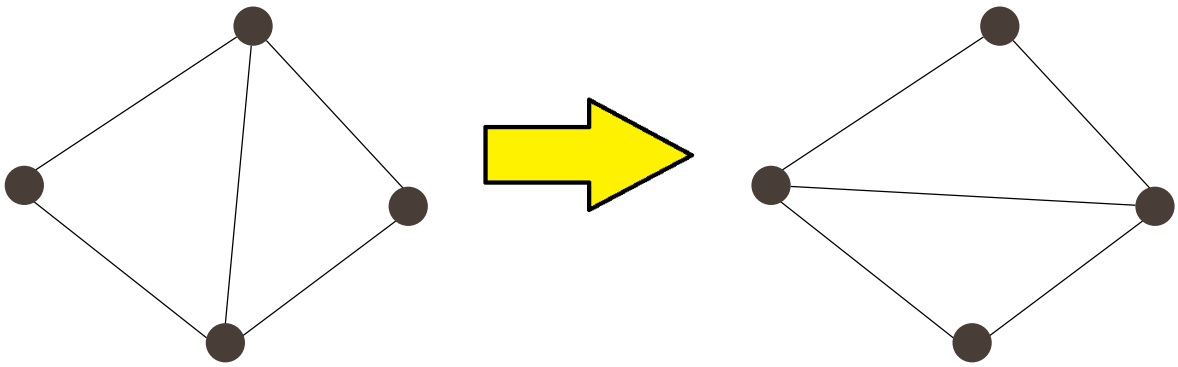


Figure 3.5: Example of an edge flip operation.

3.6 PARAMETRIC SPACES

Surfaces usually are embedded in $\mathbb{R}^3$, although several procedures become easily if the surface is embedded in $\mathbb{R}^2$. As a large variety of procedures benefit from the surface being embedded in a $\mathbb{R}^2$ space, several methods were developed for mapping a surface onto such a space, which is also called a parametric space. Figure 3.6 illustrates a surface and its parametric space.

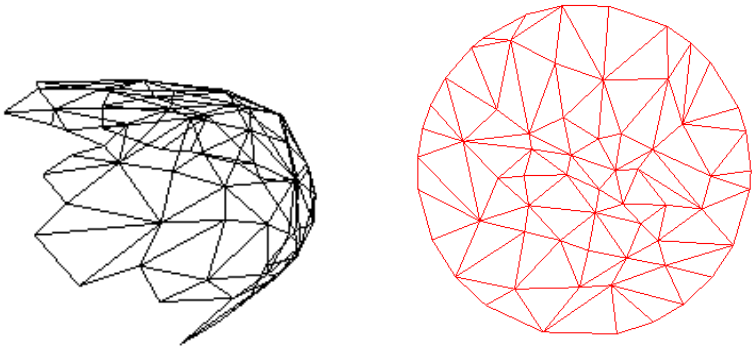


Figure 3.6: The first image shows the original mesh, the second one illustrates the parametric space.

There are several examples of parametrization in the literature. For example, each method in Section 2.1 uses a specific parametrization method. Even though a global parametrization is expensive and complex, there are simple and efficient methods for cases where the topology of the model is constrained. Those methods usually require the topology to be homeomorphic to a disk (FLOATER, 1997) or a sphere (GOTSMAN et al., 2003). Since locally the surface is homeomorphic to the disk, this dissertation uses the

parametrization scheme proposed by Floater (1997) for creating a local parametrization around a vertex.

This scheme by Floater (1997) extends the parametrization of curves, allowing it to also handle bounded triangular surfaces. The key idea of his work is to make each point U_i in the parametric space a convex combination of its neighbors. Thus, U_i can be written as in Equations (3.1, 3.2):

$$U_i = \sum_{U_j \in \mathcal{S}} \lambda_{ij} U_j, \tag{3.1}$$

$$\sum_{j} \lambda_{ij} = 1 \tag{3.2}$$

where U_i is a vertex not on the border, and U_j are the vertices of the surface $\mathcal{S}$, and λ_{ij} is a non-negative weight associated to the edge connecting U_i and U_j. If the edge does not exist, $\lambda_{ij} = 0$. If we map the boundary of the surface onto a K-sided convex polygon immersed in $\mathbb{R}^2$, then we can rewrite the Equation 3.1.

$$U_i - \sum_{U_k \notin \mathcal{B}} \lambda_{ik} U_k = \sum_{U_j \in \mathcal{B}} \lambda_{ij} U_j, \tag{3.3}$$

where U_i and U_k are the vertices of the surface that are not on the border, and U_j are the vertices on the border of the k-sided convex polygon. Since the position of U_j is known, Equation 3.3 may be rewritten as a set of two linear systems, one for each coordinate of U_i.

There are several weights for λ_{ij}, and each one of them leads to a different parametrization. We must carefully choose the weights which mostly preserve the geometric details of the original surface. Floater (1997) first proposes 3 weights, and, in a later work (FLOATER, 2003), he proposes another one. Among those weights, two can be pointed out: the shape-preserving one and the mean value. Floater (2003) argues that these two present better results.

Both weights are a generalization of the well known barycentric coordinates. The shape-preserving one is an average of the barycentric coordinates of the triangles containing V_i. Each triangle is composed by the vertices in the first star of V_i. These weights depend continually (but not smoothly) on the positions of V_i and on its first start vertices.

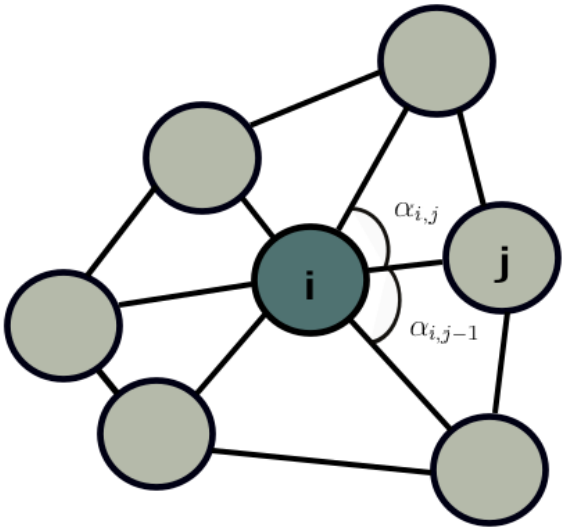


Figure 3.7: The angles $\alpha_{i,j-1}$ and $\alpha_{i,j}$ will be used for computing the weights λ_{ij}.

The mean value weights are defined by the Equations (3.4, 3.5):

$$\lambda_{ij} = \frac{wp_{ij}}{\sum_j wp_{ij}} \tag{3.4}$$

$$wp_{ij} = \frac{tan(\frac{\alpha_{i,j-1}}{2}) + tan(\frac{\alpha_{i,j}}{2})}{||v_j - v_i||}, \tag{3.5}$$

where $\alpha_{i,j-1}$ and $\alpha_{i,j}$ are the angles depicted in Figure 3.7. wp_{ij} is an intermediate weight related to the restriction defined in Equation 3.2. Each weight λ_{ij} depends smoothly on the positions of V_i and on its first start vertices. Additionally, computing those weights is easier and faster than computing the shape-preserving ones.

In Section 4.3.1, we will explain how this parametrization scheme is used together with the Laplacian filter.

3.7 LAPLACIAN OPERATOR

The dispersion in $\mathbb{R}^n$ of a function $f : \mathbb{R}^n \to \mathbb{R}$ may be measured by the Laplacian operator ∇^2, which is defined by Equation 3.6:

$$\nabla^2 f = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}. \tag{3.6}$$

A mesh can be seen as a three-dimensional graph signal(TAUBIN et al., 2000), and the Laplacian increases in conjunction with the frequency. Hence, if we find values for

$(x_1 \cdots x_n)$ which minimizes the Laplacian, we attenuate the higher frequencies of the mesh. In this work, we focus on finding values for $(x_1 \cdots x_n)$ which minimize the value of the Laplacian operator. So we are in fact applying a low-pass filter over the three-dimensional graph signal.

Taubin (1995) shows that the problem of smoothing a polygonal surface is equivalent to minimize its higher frequencies. In order to minimize those frequencies, we can minimize the value of the Laplacian operator, as pointed out before. To do so, a linear approximation for the Laplacian operator, also proposed by Taubin (1995), is used due to its simplicity. This linear approximation is given by Equation 3.7:

$$\mathcal{L}(V_i) = \sum_{V_j} w_{ij}(V_j - V_i), \tag{3.7}$$

where V_j is each vertex in the neighborhood of V_i, w_{ij} is a weight associated to each pair of vertices.

A simple search in the literature reveals that several formulations for the weights w_{ij} were proposed. For instance, there are weights based on the length of the edges (TAUBIN, 1995), on the cotangent (ALLIEZ et al., 2002) and one based on the star number (PEÇANHA et al., 2013). This work uses the approach based on the star number. However, in an effort to improve the Laplacian effectiveness and simplify its application, we apply it on a local parametric space.

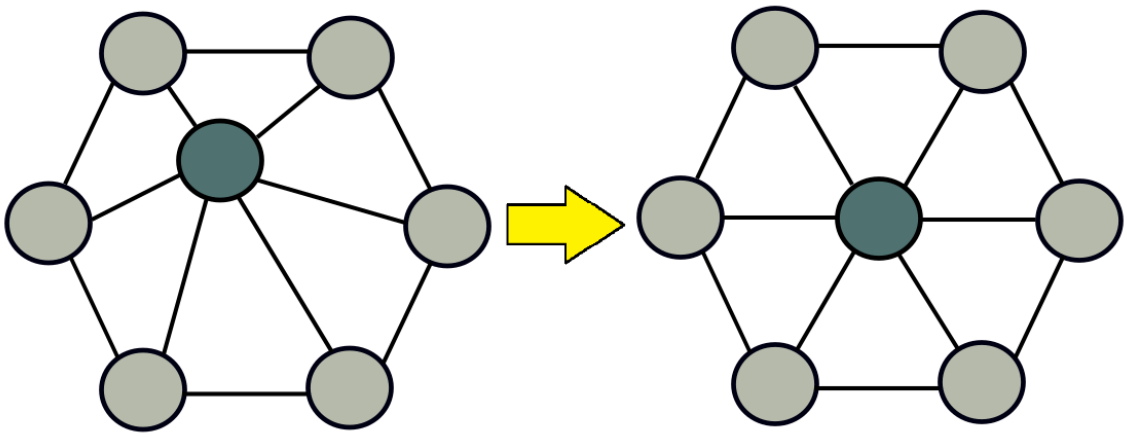


Figure 3.8: Application of the Laplacian filter

The Laplacian filter tends to put every vertex in the barycenter of its neighbors. This process is depicted in Fig 3.8. Plus, the Laplacian also decreases the standard deviation

of the edges' length, leading to a smoother and more equalized mesh.

4 PROPOSED METHOD

This dissertation proposes a method for generating a mesh without *long* or *short* edges. An edge a_i is classified as:

$$\begin{array}{ll} \text{long}, & \text{if } |a_i| > e_{max} \\ \text{short}, & \text{if } |a_i| < e_{min} \end{array} ,$$

where e_{min} and e_{max} are, respectively, the minimum and the maximum values allowed for the length of an edge. Although not the main goal, this method also seeks a mesh in which the valence of each vertex is as close as possible to its ideal valence.

The input for this algorithm is a tuple ($\mathcal{M}$, e_{min}, e_{max}, n, kl, kn, p, l, w), where $\mathcal{M}$ is a triangular mesh, e_{min} and e_{max} are the lower and upper bounds of the constraining interval, and n is the maximum number of iterations allowed. kl and kn are the star numbers used, respectively, in the Laplacian filter and in the nonlinear optimizer. The parameter p defines the number of iterations before the method updates the mesh in which it projects the vertices. Finally, l is a threshold used for deciding how the method optimizes the vertices positions (Laplacian filtering or nonlinear optimization), and w is the weight used in the parametrization, which may be the shape-preserving one or the mean value. After each iteration, if the mesh has fewer long and short edges than the currently best mesh, we save it. We also save the mesh if it has the same amount of long and short edges, but a higher percentage of regular vertices.

As an implementation detail, we use the half-edge data structure to represent meshes (WEILER, 1985). The proposed method is outlined in Algorithm 1 and in the flowchart (Fig. 4.1). Its steps will be explained in detail in the next sections.

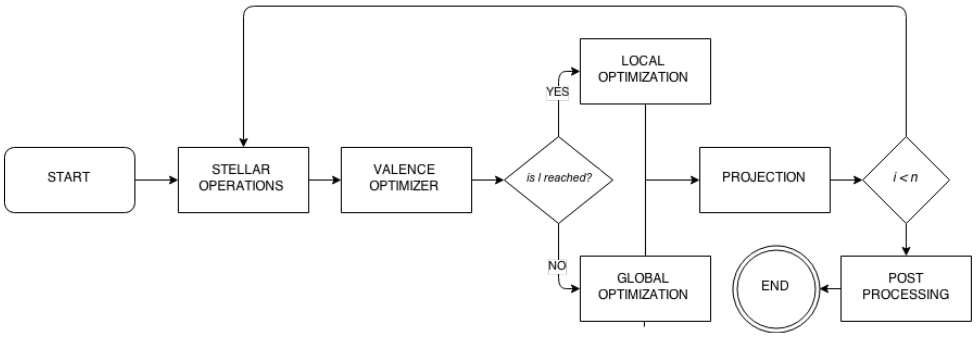


Figure 4.1: Flowchart showing the steps of the proposed algorithm. The algorithm stops anytime it achieves a mesh with all edges lengths within the given interval.

Algorithm 1: UniformRemeshing($\mathcal{M}$, e_{min}, e_{max}, n, kl, kn, p, l,w)

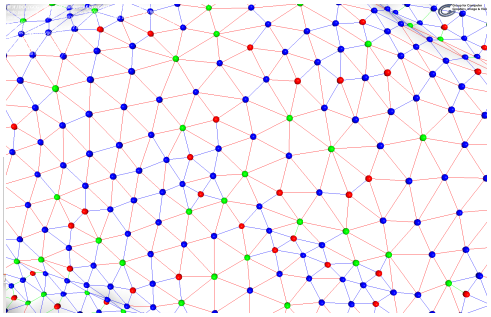
```
M' = Copy(M)
m = (e_min + e_max) / 2
while (short + long) > 0 and iter < n do
    if p > 0 and (iter mod p) = 0 then
        M = Copy(M')
    end if
    StellarOperations(M')
    CorrectValence(M')
    if CalcEdgesPercent(M') ≤ l then
        NonLinearOptimizer(M', kn)
    else
        LowPassFiltering(M', kl,w)
    end if
    Projection(M, M')
    SaveBestMesh(M')
end while
if (short + long) > 0 then
    M' = GetBestMesh(M')
    PostProcess(M')
end if
return M'
```

The resulting effects of each step are depicted in Fig 4.2. For each step, excluding the projection, one Figure representing its effects is presented. The first image shows the original mesh, the second one the mesh after the stellar operation. The third illustrates the application of the valence optimizer. Finally the last figure shows the effect of the Laplacian Operator.

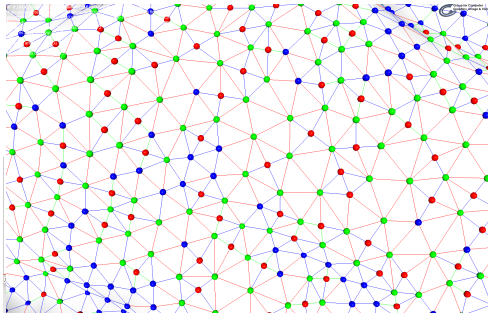
4.1 STELLAR OPERATIONS WITH PRIORITY LIST

The first step uses stellar operations in an effort to adjust the amount of vertices, polygons and edges of the mesh. For an arbitrary mesh, some regions may need to be refined while others may have to be simplified (PEÇANHA et al., 2013). Therefore, this procedure must be applied locally.

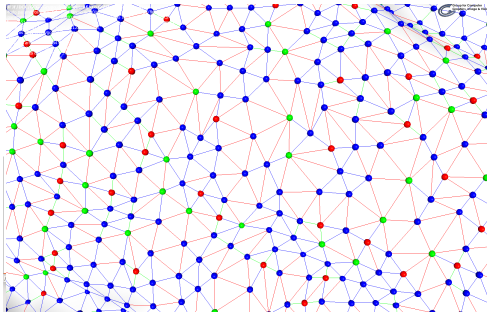
The average edge length directly depends on the amount of vertices, edges and polygons. As this average should converge to the mean of the constraining interval m, the method must adjust these amounts in order to achieve its objectives. The resulting mesh should have an average edge length equal to m. Thus, if the current average m_i is much smaller than m, a strong simplification is performed in a single iteration. This could lead



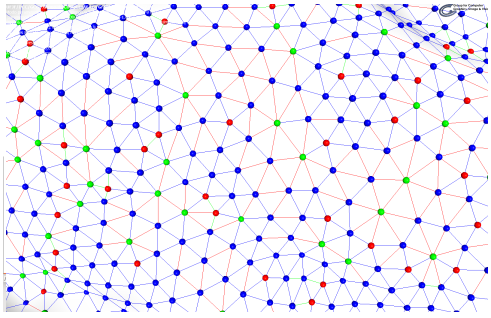
(a) The original mesh.



(b) The model after the application of stellar operations. *Long* edges are split while *short* edges are collapsed.



(c) Result of the valence optimizer step. The valence of most vertices become equal to its ideal valence, as we can see by the amount of blue vertices.



(d) Result of the optimization step, in which the edges are equalized. The triangles become more regular, and the length of most edges are constrained to the interval (edges in blue).

Figure 4.2: Overview of the Adaptive Remeshing steps.

to a degenerate mesh. In order to prevent this, the method uses intermediate values for e_{min} and e_{max}, which allows for smoother changes in the mesh. Those values are defined as:

$$\begin{aligned} e^i_{min}, &= MIN(2 \cdot m_i, m) - \tfrac{e_{max} - e_{min}}{2} \\ e^i_{max}, &= MIN(2 \cdot m_i, m) + \tfrac{e_{max} - e_{min}}{2} \end{aligned} \quad ,$$

Every edge with length smaller than e^i_{min} or greater than e^i_{max} is included in a list for later processing. As the processing order influences the result, a priority criterion must be defined. Peçanha et al. (2013) proposes a priority list in order to guarantee the stability of the method while improving its convergence. Finally, in order to improve the scheme proposed by (PEÇANHA et al., 2013), this method positions both the new vertex from the edge split operation and the remaining vertex of the edge collapse operation,

minimizing the Equation 4.1 proposed by (HAUCK et al., 2014).

$$\sum_{V_j} (|V_i - V_j| - m)^2, \tag{4.1}$$

where V_i is the vertex being repositioned and V_j is a vertex connected to V_i.

4.2 VALENCE OPTIMIZER WITH PRIORITY LIST

Vertex valence plays an important role on the quality of a mesh. It also has a huge impact on the dual mesh. For example, a vertex with valence 6 generates a hexagon on the dual mesh, while a vertex with valence 5 results in a pentagon, and so on. Some applications are interested in a highly regular trivalent mesh, which is the dual for the triangular mesh, composed mostly of hexagons. For those applications, the mesh connectivity is also decisive.

The ideal valence of a vertex may be estimated. The neighborhood of an internal vertex can be approximated by a disk, so the angle sum must be equal to 360 degrees. An equilateral triangle has all three internal angles equal to 60 degrees. Thus, in an ideal scenario, the valence of each vertex should be equal to $\frac{360^o}{60^o}$. However, if the vertex lies on the border, its neighborhood can be imagined as a half disk. In this case, two edges are cut off, hence the ideal valence becomes 4.

An error measure for mesh valence was proposed by (SURAZHSKY; GOTSMAN, 2003). This error measure $\mathcal{E}_{val}$ is defined as in Equation 4.2:

$$\mathcal{E}_{val} = \sum_{V_i \in \mathcal{M}} (IdealV(V_i) - Val(V_i))^2, \tag{4.2}$$

where V_i is a vertex that belongs to the mesh $\mathcal{M}$, $IdealV(V_i)$ is the ideal valence of the vertex V_i , and $Val(V_i)$ is the actual valence of V_i.

In order to improve the mesh's connectivity, a series of edge flips may be done. However, as already mentioned, the order of each flip significantly changes the final result. In an effort to ensure method stability, besides improving its convergence and effectiveness, this work proposes the creation of a priority list. First, for each edge we assign a number

$\mathcal{I}mp$ which is defined in Equation 4.3:

$$\mathcal{I}mp = \mathcal{E}_{val}^{bf} - \mathcal{E}_{val}^{af}, \tag{4.3}$$

where $\mathcal{E}_{val}^{bf}$ and $\mathcal{E}_{val}^{af}$ are, respectively, the error measures $\mathcal{E}_{val}$ before and after performing the edge flip. The priority list is built taking this $\mathcal{I}mp$ values as reference. Edges with higher $\mathcal{I}mp$ have higher priority. If two edges have the same $\mathcal{I}mp$, the most distant edge from the center of the interval has a higher priority. This ensures that edges which are most distant from the objective are flipped earlier.

After the priority list is built, each valid edge flip operation is executed in accordance with the list. An edge flip operation is considered valid if it simultaneously reduces $\mathcal{E}_{val}$ and does not decrease the associated minimum angle by more than a half. This guarantees that each flip operation always reduces the connectivity error. Plus, it also makes sure that it does not generate a degenerate triangle. Algorithm 2 is an overview of this step.

Algorithm 2: CorrectValence($\mathcal{M}'$)

```
do
    foreach A_i ∈ M' do
        if calcValImprovement (A_i)>0 then
            priorityList.Add(A_i)
        end if
    end foreach
    SortDescent(priorityList)
    foreach A_i ∈ priorityList do
        if calcValImprovement (A_i)>0 then
            preMin_α = getMinalpha(A_i)
            EdgeFlip(A_i)
            postMin_α = getMinalpha(A_i)
            if postMin_α < preMin_α / 2 then
                UndoEdgeFlip(A_i)
            else
                NFlips = NFlips + 1
            end if
        end if
    end foreach
while NFlips > 0;
```

4.3 LOW PASS FILTER

The vertices must be uniformly distributed over the surface in order to have equalized edge lengths. For achieving this uniform distribution, the method optimizes the positions of the vertices by applying a Laplacian filter. As already pointed out in Section 3.7, its application minimizes the higher frequencies of the mesh while smoothing it. However, we do not apply the Laplacian directly. Instead, we apply it in a parametric space as described in the following section.

4.3.1 LOCAL PARAMETRIZATION

In an effort to minimize the geometric error and maximize the effectiveness of the filter, we must constrain the vertices over the surface during its application. The simplest way to do this is by mapping the surface onto $\mathbb{R}^2$. However, this may be extremely inefficient. So we map the kl-star of each vertex onto a disk like surface. Then, for each vertex, the filter is applied in the parametric space.

The kl-star is homeomorphic to the disk in almost all cases, so we can apply the method proposed by Floater (1997). Figure 4.3 illustrates a 2-star around a vertex. The border of the vertex star is mapped onto a circle, in such a way that the length of each edge is the same both in the parametric space and in the original surface. We use two weights for λ_{ij}, as defined in Section 3.6: the Shape-Preserving and the Mean Value. The uniform weight and the length-based one were also implemented, but the results were even worse than applying the Laplacian directly. In the cases where the kl-star is not homeomorphic to the disk, the parametrization fails and the ordinary Laplacian operator is applied.

4.3.2 LAPLACIAN FILTER

This method uses an iterative approach for applying the Laplacian filter. If a local parametrization could be created around a vertex, it applies the Laplacian in the parametric space. Otherwise, it directly applies the Laplacian constrained to the tangent plane.

This work uses the kl-star of a vertex for computing its new position. However, the kl-star becomes extremely unbalanced when the vertex is close to the border, producing

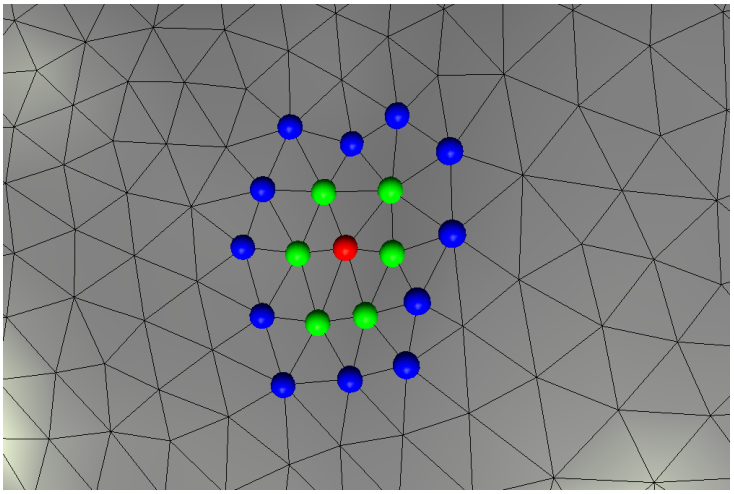


Figure 4.3: The 2-star around a vertex V_i. The vertex V_i is in red, the first star in green and the second star in blue.

not so good results. Therefore, if the kl-star of a vertex contains the border, we do not use the kl-star (S_{kl}) directly. In order to maximize the balance, we use a kl_i-star as defined in Equation 4.4:

$$kl_i = min\{Star(V_j)\}, \forall V_j \in \mathcal{B} \cap S^i_{kl}, \tag{4.4}$$

where kl_i is the maximum star number for the vertex V_i, $Star(V_j)$ is the star in which V_j lies, and V_j is a vertex located simultaneously on the border ($\mathcal{B}$) and in the kl-star of the vertex V_i (S^i_{kl}). Although this restriction constrains the compression effect close to the border, it cannot eliminate the effect completely.

A secondary effect of this restriction is the constrainment of the vertices located over the border. This is a result of the fact that the kl_i of a vertex on the border is 0. This may be a problem, since those vertices cannot be moved to a better position. Thus, we propose another strategy for repositioning these vertices. For each vertex V_i on the border, a quadratic curve interpolating the two edges of the border containing V_i is computed. The vertex V_i is then repositioned over this quadratic curve, minimizing Equation 4.1, the same one used for the stellar operations.

Algorithm 3 presents an overview of the aforementioned step.

4.4 NONLINEAR OPTIMIZER

After some iterations of the algorithm, the Laplacian operator has its effectiveness greatly reduced. This is specially true in regions with a high curvature. Those regions cannot

Algorithm 3: LowPassFiltering($\mathcal{M}'$, k,w)

foreach $V_i \in \mathcal{M}'$ **do**
  $kStar$=`getKStar` (V_i,k)
  if `ComputeParametrization`($kStar$,w) **then**
    $fat = 0$
    $V_i^p = \overline{0}$
    foreach $V_j \in kStar$ **do**
      $V_i^p = V_i^p + \frac{V_j^p}{2^{star-1}}$
      $fat = fat + \frac{1}{2^{star-1}}$
    end foreach
    $V_i^p = \frac{V_i^p}{fat}$
    V_i' =`Computed3dPosition`(V_i^p)
  else
    $V_i' = \overline{0}$
    foreach $V_j \in kStar$ **do**
      V_i'=$V_i' + \frac{V_j}{star}$
      fat=$fat + \frac{1}{star}$
    end foreach
    $V_i' = \frac{V_i'}{fat}$
    $D_i = V_i'$-V_i
    $D_i = D_i$ -`projection` (D_i,N_i)
    $V_i' = V_i + D_i$
  end if
end foreach
foreach $V_i \in \mathcal{M}'$ **do**
  if $V_i \in \mathcal{B}$ **then**
    `BorderOptimizer`(V_i)
  else
    $V_i = V_i'$
  end if
end foreach

be corrected by a simple operator like the Laplacian. Hence, a new strategy is required for obtaining further improvements in those regions. A specific optimization for edge length equalization is proposed in (HAUCK et al., 2014, 2015). This optimization is computationally expensive, so we must apply it following a local scheme. A new priority list of edges is created, and this new optimization is applied to the most troubling edges and to the region around them.

To support this nonlinear optimization procedure, Hauck et al. (2014) propose an error

measure for a region, as defined in Equation 4.5:

$$Error_a = \sum_{V_i} \sum_{V_j} (|V_i - V_j|^2 - m^2)^2, \tag{4.5}$$

where V_i are the vertices in the region, V_j are the vertices connected to V_i, and m is the center of the given interval.

If $Error_a$ is minimized, the lengths of the edges become closer to m. However, since the vertices displacements have three degrees of freedom, the local geometry may suffer a huge loss. Thus, we constrain those displacements to the tangent plane. In order to do so, we compute an orthonormal base for each vertex V_i using the normal vector in V_i. This local base is defined as $< T_i, T_i', N_i >$, where T_i and T_i' are the directions over the tangent plane, and N_i is the normal direction. Using this base we can write the displacement of each vertex V_i as defined in Equation 4.6

$$D_i = \alpha_i \cdot T_i + \beta_i \cdot T_i' + \gamma_i \cdot N_i \tag{4.6}$$

where α_i and β_i are the displacements over the tangent plane, and γ_i is the displacement in the normal direction. Since we want to constrain the displacement to be over the tangent plane, we must set $\gamma_i = 0$. So we can write a new error measure in which the only variables to be minimized are α_i and β_i.

$$Error_b = \sum_{V_i} \sum_{V_j} (|V_i + \alpha_i \cdot T_i + \beta_i \cdot T_i' - (V_j + \alpha_j \cdot T_j + \beta_j \cdot T_j')|^2 - m^2)^2, \tag{4.7}$$

where both α_j and β_j are set to zero when the index j does not exist. The error defined in Equation 4.7 can be minimized without significant geometric losses.

Similarly to the stellar operations step, the order in which the troubling edges are processed affects the final result. Considering that some geometry is lost each time we run this optimization, we want to remove all troubling edges by optimizing as few regions as possible. Troubling edges are classified in accordance with the amount of long and short edges present in their neighborhood. The neighborhood is the *knl*-star of the edge, and *knl* is a parameter of the method. Applying the optimization to regions containing more long and short edges may reduce their total amount quicker. Thence, we create a new priority list, with the priority p_j of each edge being defined as $p_j = n_j^l + n_j^s$. In this relation,

n_j^l and n_j^s are, respectively, the amount of long and short edges in the neighborhood of an edge A_j. Ultimately, each edge is processed in accordance with its priority.

This nonlinear optimization was first used as a post processing step (HAUCK et al., 2014), and finally as a replacement for the Laplacian operator in later iterations (HAUCK et al., 2015). Despite this step is used as it is in (HAUCK et al., 2015), we do not fix the star number in which the optimization takes place. Instead, this is a new parameter for the method and its effects will be further analyzed in Chapter 5. This nonlinear optimization also has a drawback for vertices over the border. If it optimizes a vertex on the border, the vertex will certainly be moved outside. So, to preserve the border, we do not optimize vertices over it. If a troubling edge contains a vertex on the border, we run the same optimization presented in Subsection 4.3.2.

4.5 PROJECTION

Following the previous steps, some vertices may be positioned outside the original surface $\mathcal{M}$. In order to prevent geometric losses, those vertices must be projected onto $\mathcal{M}$. Although the projection of a surface onto another surface seems easy, it is in fact a problem of high complexity (BOTSCH et al., 2010).

Peçanha et al. (2013) proposes a low-cost method, which achieves good results when the two surfaces are close enough. This method to project a vertex V_i onto $\mathcal{M}$ finds the closest vertex to V_i. The vertex V_i is then projected onto every triangle which contains the closest vertex. This generates a list of projections $\{P_1 \cdots P_n\}$, and we select the projection P_j which presents the smallest distance to V_i to be the final projection.

This dissertation extends the above method, as the k closest vertices are selected, instead of only the closest one. The method projects the vertex V_i onto every triangle which contains at least one of the k closest vertices. It also creates a list of projections, projecting the vertex V_i onto the projection P_j which presents the smallest value for $|V_i - P_j|$. This cannot find the optimal solution for every case, but it improves the approximation. In our tests, the projection improves as k increases until it reaches the value 5. Thus, in order to guarantee that the best projection is selected, we overestimate k setting it to 10. We also now consider the sum $|V_i - P_j| + \sum_{V_k} |V_i - V_k| - |P_j - V_k|$, where V_k are the vertices connected to V_i. This adds some information about the neighborhood of the vertex in the projection, which also improves the process.

The mesh $\mathcal{M}_p$ in which the projections are performed may be updated over the iterations. In order to control how often $\mathcal{M}_p$ is changed, the algorithm uses the parameter p. At every p iterations, the projection mesh is replaced by the current mesh $\mathcal{M}'$. This relaxes even more the mesh, improving the convergence. However, the geometric error also increases, as the mesh is being projected onto a mesh $\mathcal{M}_p$ which is progressively more distant from the original mesh $\mathcal{M}$. If minimizing the geometric distortions is desired, p must be set to 0 so the mesh $\mathcal{M}_p$ is never replaced.

The projection procedure is detailed in Algorithm 4.

Algorithm 4: Projection($\mathcal{M}$, $\mathcal{M}'$)

```
foreach V_i ∈ M' do
    Nearest = getNearest (V_i,M,10)
    Projections = getProjections(V_i,Nearest)
    foreach P_j ∈ Projections do
        dist_j = |V_i − P_j|
        dist_f = |V_i − P_f|
        foreach V_k do
            dist_j = dist_j + |V_i − V_k| − |P_j − V_k|
            dist_f = dist_f + |V_i − V_k| − |P_f − V_k|
        end foreach
        if dist_j ≤ dist_f then
            P_f = P_j
        end if
    end foreach
    V_i = P_j
end foreach
```

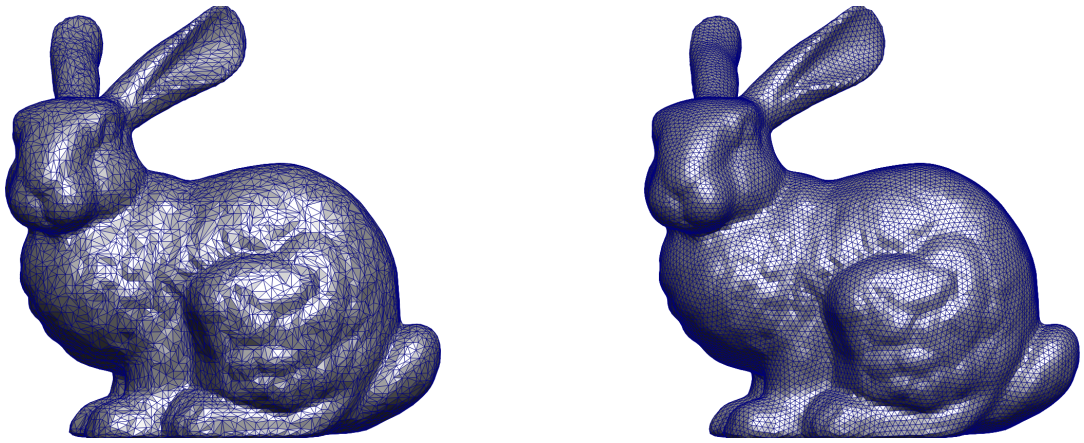
4.6 POST PROCESSING

In some complex models, the defined n iterations may not be enough to remove all long and short edges. In those cases, the method applies the nonlinear optimizer without projecting the result while the amount of long and short edges are decreasing. This adds some geometric error for each time it runs the nonlinear optimizer, but can put a large number of edges within the interval. As a consequence, better results are found when the method does not need to enter this step. Even though better results are presented when the method do not use this step, in some complex models this may be the only way for achieving the desired objective.

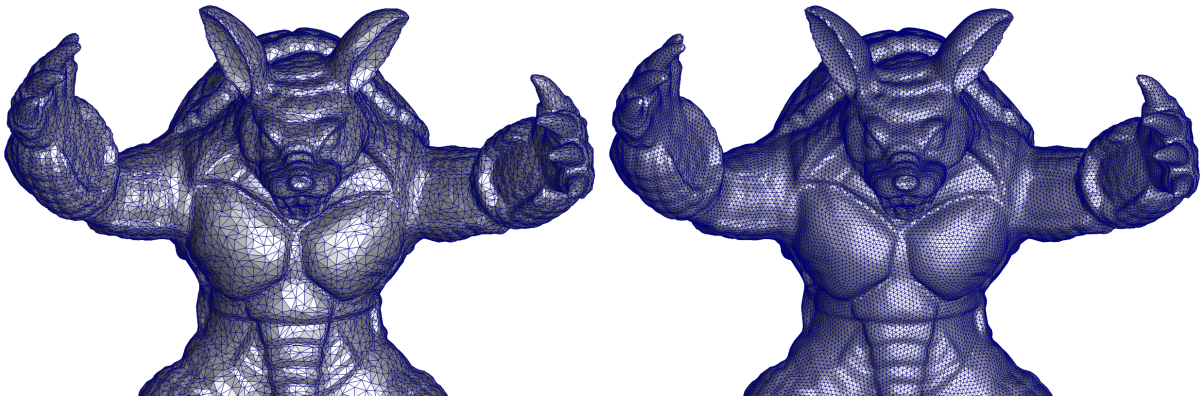
5 RESULTS

This chapter presents the experimental results of the proposed method. It also analyzes the impact of each parameter of the method on the results. Furthermore, observed drawbacks of the method will also be discussed here. An application of this method for modeling carbon nanostructures will be presented in details in Chapter 6. All tests presented here were run on an Intel Xeon(R) CPU 31220 @ 3.10GHz x 4 computer with 8GB of RAM. The algorithm is implemented in C++, without any kind of parallelism.

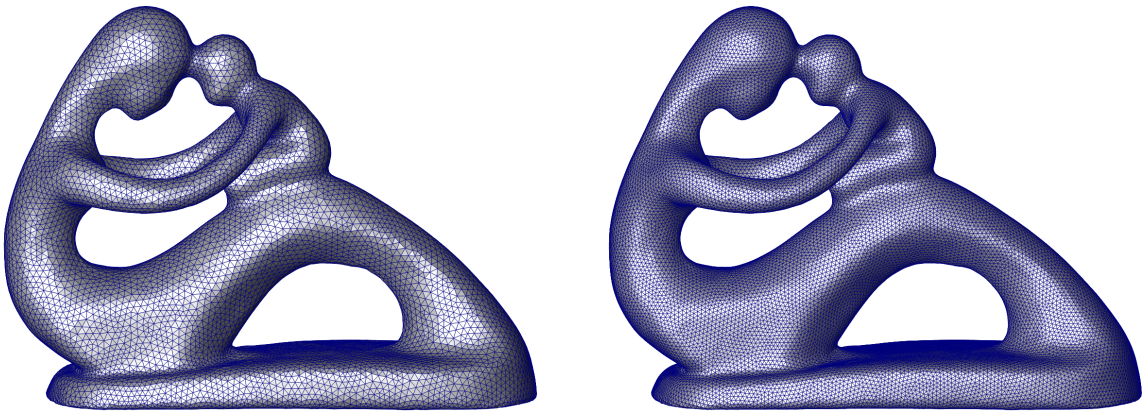
In order to generate the results, we ran tests over five models. The first one is the Bunny, which is a commonly used model, so we use it for comparison. The second is the Fertility model. This one contains a genus greater than 0, so we use it to show how the method behaves in such models. The third model is the Armadillo, which was chosen for being a large model. It allows us to show how the method performs for large inputs. The fourth one is the Hand model, used here because it contains a border. Finally, the last model is the Octopus. This model has a variable density of vertices, along with extremely high curvature points in its tentacles. It will be used only in Section 5.6 in order to demonstrate a flaw case of the method. Figure 5.1 shows each model used in the tests along with their processed version.



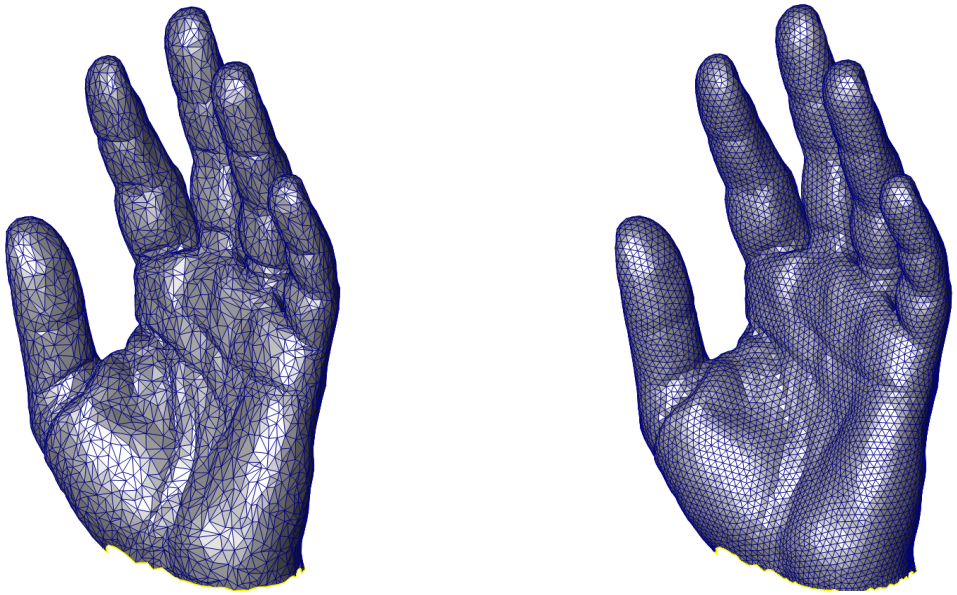
(a) Bunny



(b) Armadillo



(c) Fertility



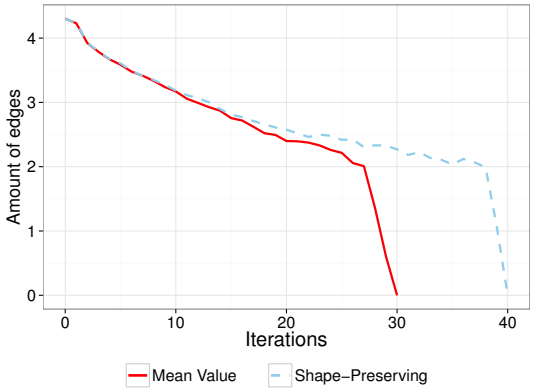
(d) Hand

Figure 5.1: These images show the original models on the left and the processed versions on the right. The weight used in the local parametrization was the Mean Value and the other parameters were:(1.2,1.8,150,2,2,0,0.5).

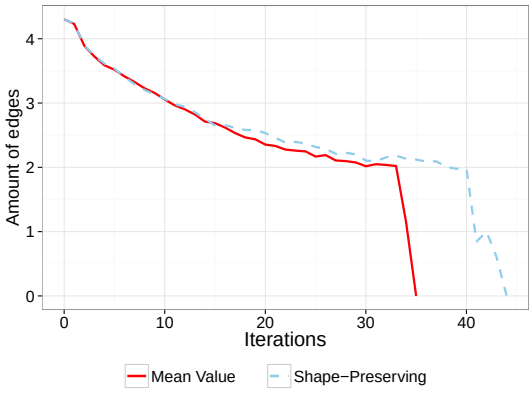
5.1 PARAMETRIZATION AND LAPLACIAN OPERATOR

In this section, the Shape Preserving and Mean Value weights for the parametrization are compared. The parametrization is extensively affected by the star number kl, so this parameter was also varied for these tests. In order to analyze the impact of these parameters, the evolution of long and short edges for each model is plotted. For visualization purposes, we use a log_{10} scale for the amount of long and short edges.

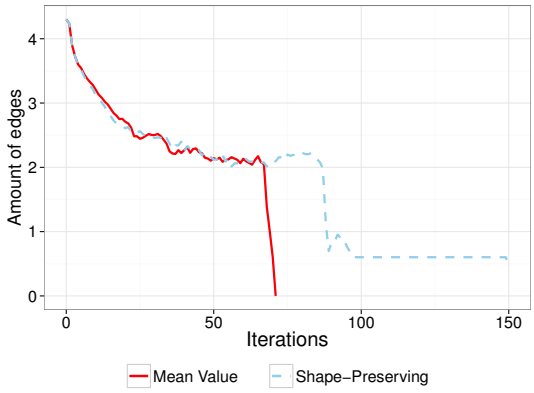
Figures 5.2, 5.3, 5.4 and 5.5 show the amount of long and short edges per iteration for the models. Each figure presents one graph for each kl value, and each graph contains both weights for the parametrization. Since the parametrization being analyzed only affects the Laplacian operator, the parameter l was fixed and equals to 0.5, while the parameter knl is equal to 2. The parameter p was also fixed and set to 0, as this represents the most troubling scenario.



(a) Evolution for $kl = 2$



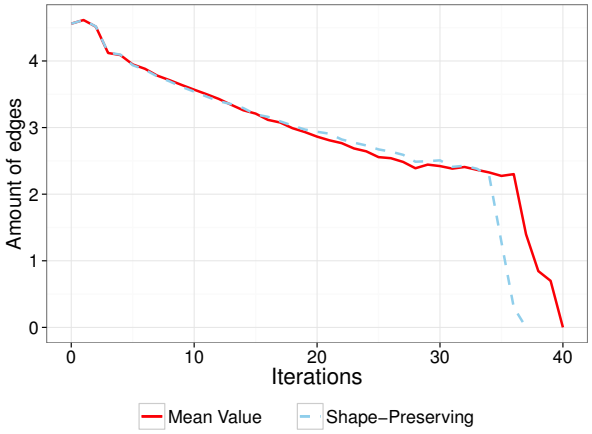
(b) Evolution for $kl = 3$



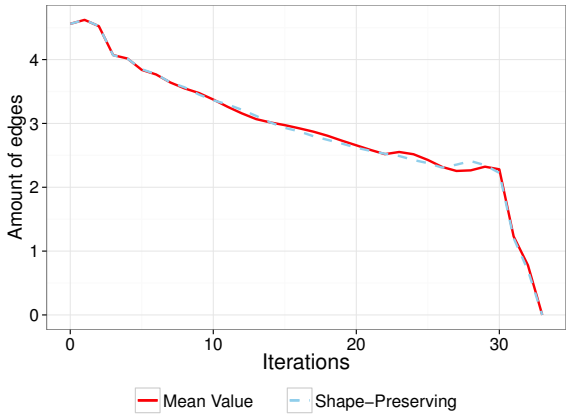
(c) Evolution for $kl = 4$

Figure 5.2: Evolution of long and short edges per iteration for the Bunny model. The amount of long and short edges are in log_{10} scale.

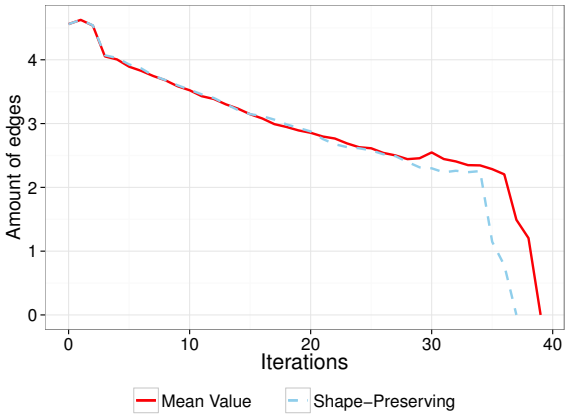
Results shown that, except for the Fertility model with $kl = 3$ (Graph in Figure 5.3b),



(a) Evolution for $kl = 2$



(b) Evolution for $kl = 3$



(c) Evolution for $kl = 4$

Figure 5.3: Evolution of long and short edges per iteration for the Fertility model. The amount of long and short edges are in log_{10} scale.

$kl = 2$ yields the best results. The Fertility is an easy model for refinement, without extremely high curvature regions or borders. Therefore we can conclude that raising kl accelerates the convergence rate for simple regions.

The Hand (Graph in Figure 5.4) model is an special case, the only model with border. The method is not capable of achieving a mesh with all edges within the given interval. This is explained by the additional restrictions added for preserving the border. They are effective in preserving the geometry of the border, but constrains the method preventing it from reach the final objective.

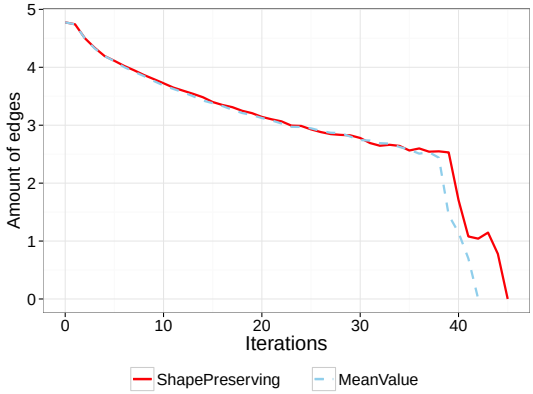
The comparison between Mean Value weight and the Shape-Preserving one is not conclusive, as it depends on the model. For example, the Bunny model presents the best results when using the Mean Value weight. On the other hand, the Fertility model presents better results using the Shape-Preserving one. This may be explained as the Shape-

(a) Evolution for $kl = 2$

(b) Evolution for $kl = 3$

(c) Evolution for $kl = 4$

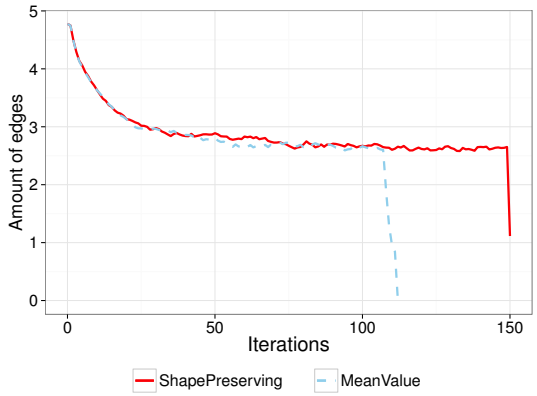
Figure 5.4: Evolution of long and short edges per iteration for the Hand model. The amount of long and short edges are in log_{10} scale.



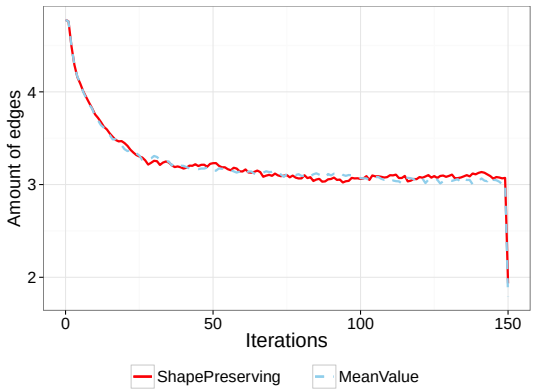
(a) Evolution for $kl = 2$

Figure 5.5: Evolution of long and short edges per iteration for the Armadillo model. The amount of long and short edges are in log_{10} scale.

Preserving one maps the neighborhood of each vertex onto a plane before computing the weights, while the Mean Value computes the weights directly. So meshes in which



(b) Evolution for $kl = 3$



(c) Evolution for $kl = 4$

Figure 5.5: Evolution of long and short edges per iteration for the Armadillo model. The amount of long and short edges are in log_{10} scale.

Table 5.1: Total time and the average time per iteration.

Model	**Parametrization**	***kl***	**Iterations (It)**	**Total time (s)**	**Avg. time per It (s)**
Bunny	Shape Preserving	2	40	279.793	6.995
Bunny	Mean Value	2	30	153.630	5.121
Bunny	Shape Preserving	3	44	512.660	11.651
Bunny	Mean Value	3	35	204.808	5.852
Bunny	Shape Preserving	4	150	2125.789	14.172
Bunny	Mean Value	4	71	513.794	7.237
Armadillo	Shape Preserving	2	45	1536.370	34.142
Armadillo	Mean Value	2	42	1178.062	28.049
Armadillo	Shape Preserving	3	150	7112.029	47.414
Armadillo	Mean Value	3	112	3343.206	29.850
Armadillo	Shape Preserving	4	150	10486.290	69.909
Armadillo	Mean Value	4	150	4924.962	32.833
Fertility	Shape Preserving	2	37	596.764	16.129
Fertility	Mean Value	2	40	487.634	12.191
Fertility	Shape Preserving	3	33	804.650	24.383
Fertility	Mean Value	3	33	468.438	14.195
Fertility	Shape Preserving	4	37	1379.940	37.296
Fertility	Mean Value	4	39	697.291	17.880
Hand	Shape Preserving	2	150	533.302	3.553
Hand	Mean Value	2	150	348.160	2.321
Hand	Shape Preserving	3	150	934.763	6.232
Hand	Mean Value	3	150	444.773	2.965
Hand	Shape Preserving	4	150	1605.294	10.702
Hand	Mean Value	4	150	639.963	4.266

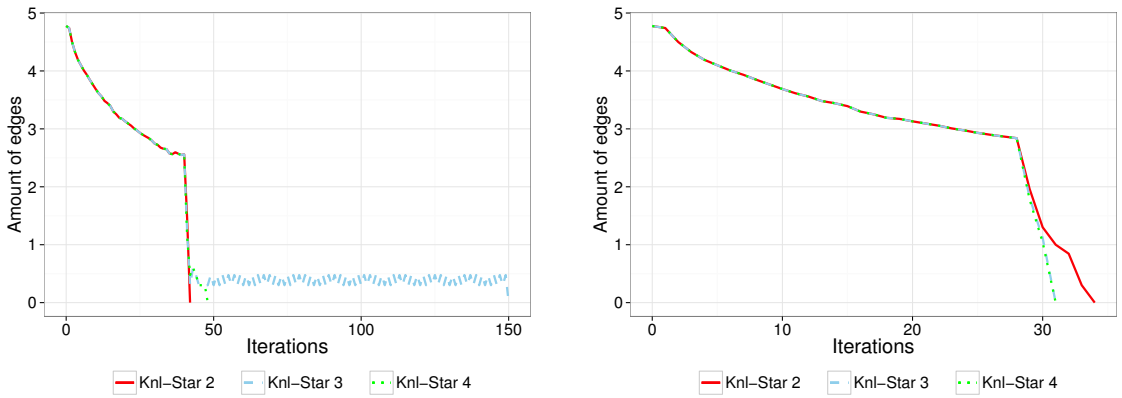
the the neighborhood of each vertex is not closer to a plane present better results with the Mean Value weight. Additionally, as kl increases, the local parametrization area becomes larger. This also raises the relative time spent computing the parametrization, increasing the performance impact of changing the weights from the Mean Value to the Shape-Preserving one as seen in Table 5.1. Even when considering the best case for each

parametrization weight, the results are not so different, but the Mean Value is faster. For this reason, it was used in the majority of the following tests.

5.2 NONLINEAR OPTIMIZER

This section shows the influence of parameters knl and l in the final result. To do so we ran several test varying these parameters.

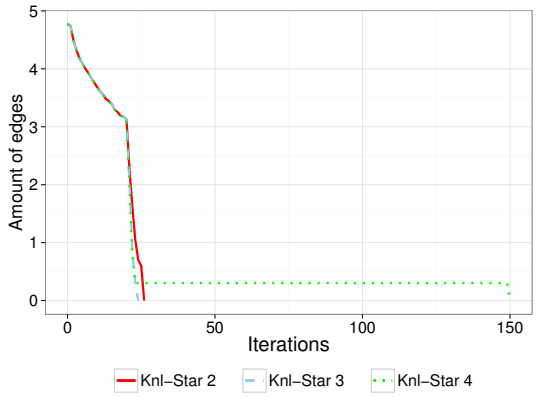
Figures 5.6, 5.7, 5.8 and 5.9 show the results for each test model. Each sub-figure illustrates a different l, with each one of the three lines representing a different knl value. Even though a higher l accelerates convergence, it can easily lead to a local minimum. However, if l is too small, the method may never reach the nonlinear optimization phase (Fig. 5.8a), preventing its convergence. The best value for l, when only the mesh's quality is a concern, is the lowest value in which the method can enter the nonlinear phase. Increasing knl may slow down a little the convergence rate. Nevertheless, for some models, a knl higher than 2 results in a local minimum. Thus, it is recommended to adopt a knl value equal to 2.



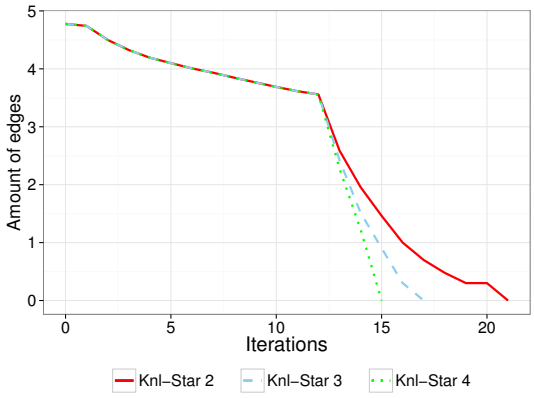
(a) Evolution for $l = 0.5$ (b) Evolution for $l = 1$

Figure 5.6: Evolution of long and short edges per iteration for the Armadillo model. The amount of long and short edges are in log_{10} scale.

Sometimes the method presents an oscillatory behavior (for example, Figure 5.6). This could be explained as the interaction between the nonlinear optimizer and the stellar operations. For a better understanding, an example of the problem will be analyzed. First, the nonlinear optimizer finds the best position for the vertices. However, some edge still remains bigger than e_{max}. In the next iteration, the stellar operation step split the long edge. The nonlinear optimizer computes the best positions for the vertices again, but

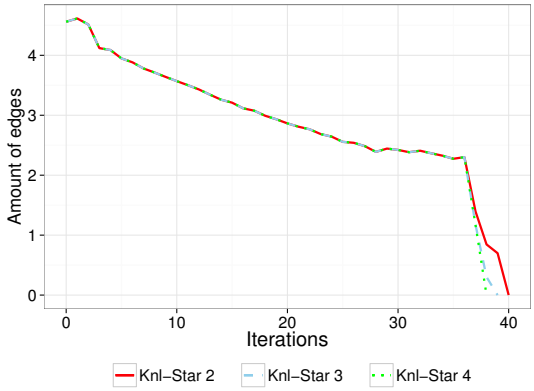


(c) Evolution for $l = 2$

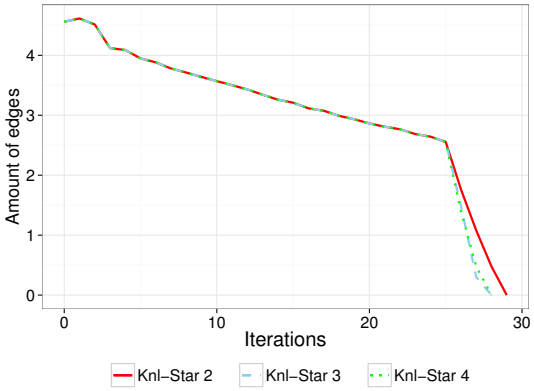


(d) Evolution for $l = 5$

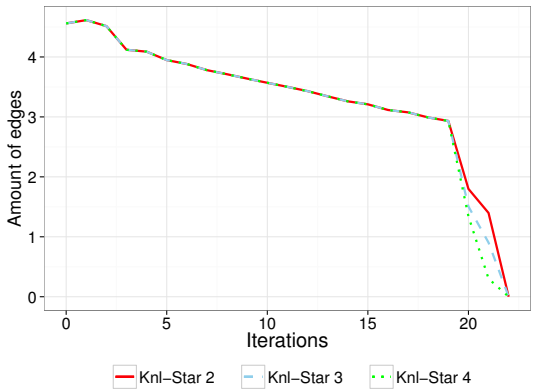
Figure 5.6: Evolution of long and short edges per iteration for the Armadillo model. The amount of long and short edges are in log_{10} scale.



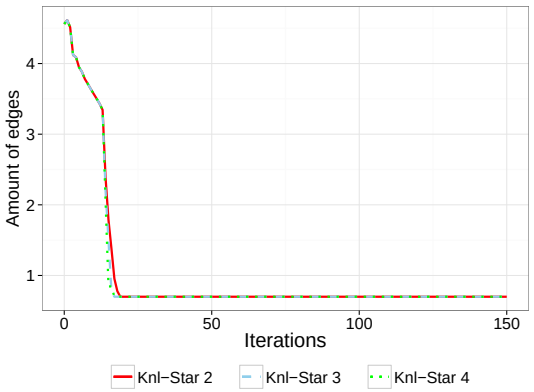
(a) Evolution for $l = 0.5$



(b) Evolution for $l = 1$



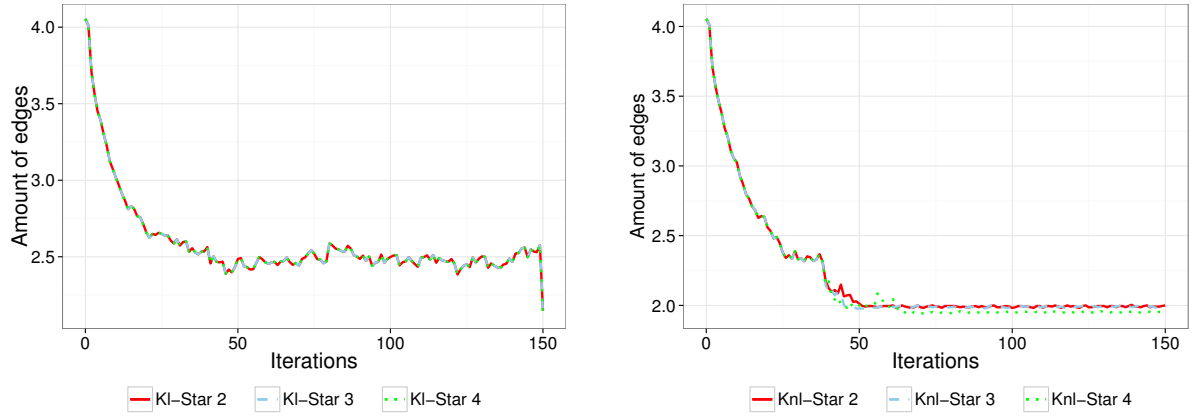
(c) Evolution for $l = 2$



(d) Evolution for $l = 5$

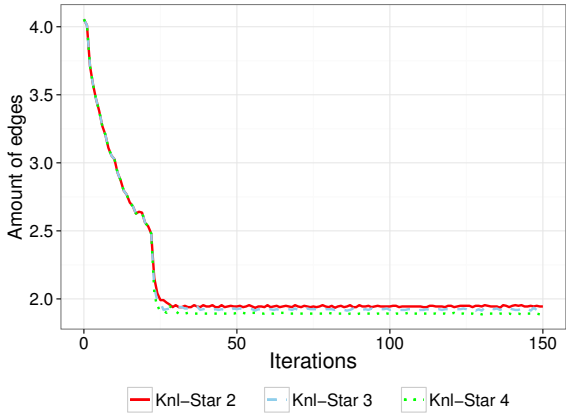
Figure 5.7: Evolution of long and short edges per iteration for the Fertility model. The amount of long and short edges are in log_{10} scale.

this time with the extra vertex added by the edge split. This results in an edge becoming smaller than e_{min}. Finally, in the next iteration the stellar operation step collapses the short edge. Now the method has returned to the state where we started the example, as

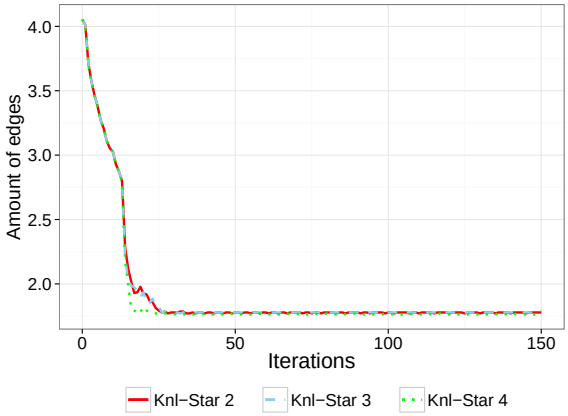


(a) Evolution for $l = 0.5$

(b) Evolution for $l = 1$



(c) Evolution for $l = 2$

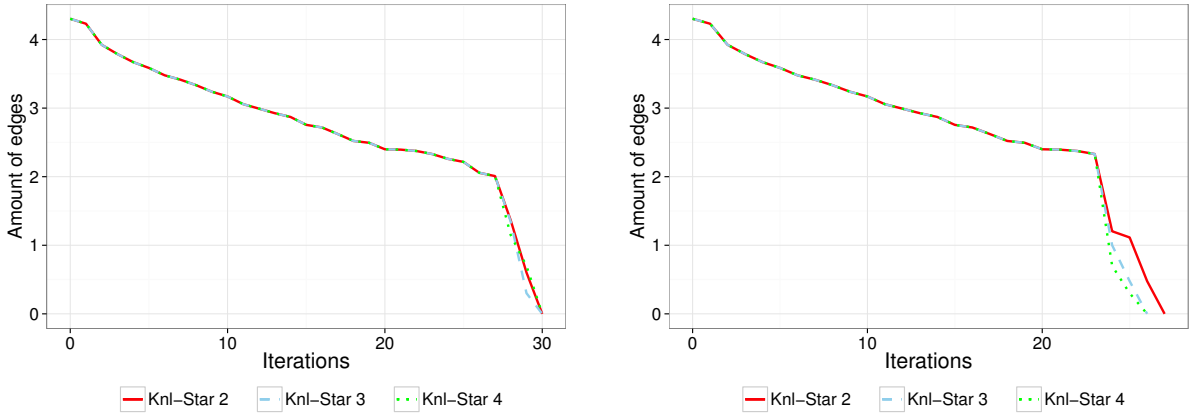


(d) Evolution for $l = 5$

Figure 5.8: Evolution of long and short edges per iteration for the Hand model. The amount of long and short edges are in log_{10} scale.

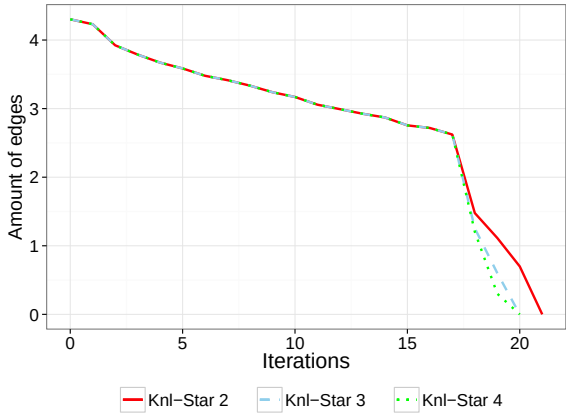
the extra vertex was removed with the edge collapse.

In order to have a better understanding of the impact of l over the quality of the meshes, the histograms of the edges lengths are depicted for each model in Figure 5.10. Those histograms evince that lower values for l present better results than higher values, except for the Hand model. This is expected because the method spends more iterations improving regions in which the edges are already within the interval. For the Hand model, in addition, the method changes the border during those iterations. As the border has several constraints for preserving the geometry, when a long or short edge appears, those changes cannot always be undone.

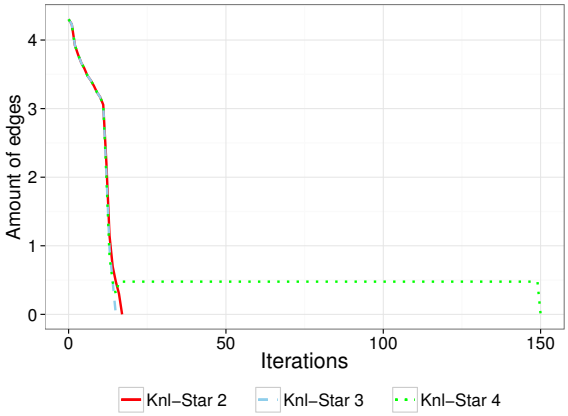


(a) Evolution for $l = 0.5$

(b) Evolution for $l = 1$



(c) Evolution for $l = 2$



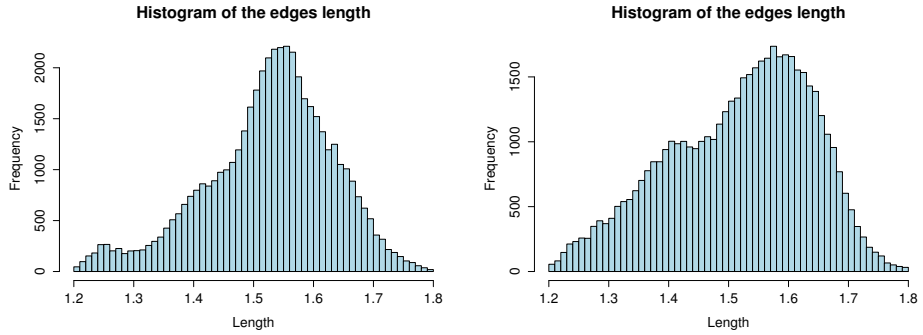
(d) Evolution for $l = 5$

Figure 5.9: Evolution of long and short edges per iteration for the Bunny model. The amount of long and short edges are in log_{10} scale.

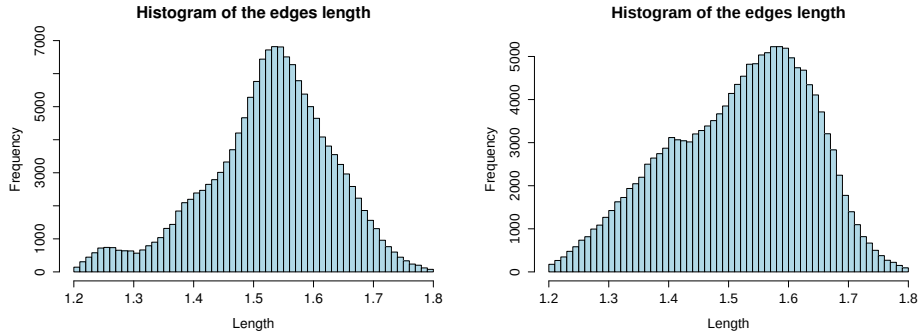
5.3 PROJECTION

This section presents the results for different values of p. The results are analyzed taking into consideration the convergence and the fidelity of the mesh. In order to compare the differences between two surfaces, Cignoni et al. (1998) proposes an error measure called Hausdorff distance. As the projection is used to minimize the distance between the original and the processed one, the Hausdorff distance is used to measure how the parameter p affects the distance between them. For computing the Hausdorff distance, the software MeshLab (CIGNONI et al., 2008) was used.

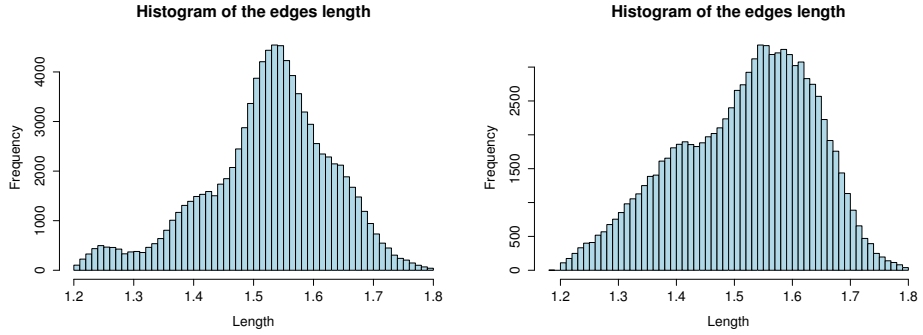
Table 5.2 shows the Hausdorff distance, the number of iterations performed and the percentage of regular vertices for each model, only varying the parameter p. As the method for computing the Hausdorff distance uses a Monte Carlo approach, the Hausdorff max value may decrease even when p increases, however the Hausdorff root mean square (RMS)



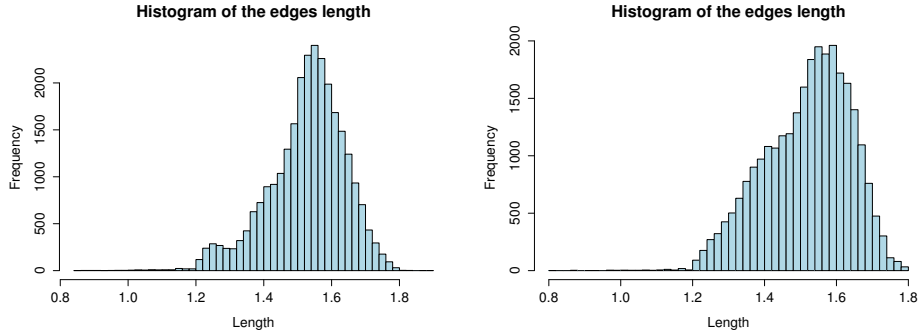
(a) Histogram of the Bunny model with $l = 0.5$ (left) and $l = 5$ (right)



(b) Histogram of the Armadillo model with $l = 0.5$ (left) and $l = 5$ (right)



(c) Histogram of the Fertility model with $l = 0.5$ (left) and $l = 5$ (right)



(d) Histogram of the Hand model with $l = 0.5$ (left) and $l = 5$ (right)

Figure 5.10: Distribution of edge lengths in the models after processing with different l values.

always increases along with p.

Table 5.2: How the method reacts to changes in the parameter p.

p	Model	Iteration	Hausdorff max	Hausdorff RMS	Regular Vertices
0	Bunny	30	0.482711	0.037560	86.788922
10	Bunny	31	0.771324	0.056754	86.879900
25	Bunny	30	0.738167	0.040200	86.829786
50	Bunny	30	0.482711	0.037560	86.788922
0	Armadillo	42	0.596794	0.047534	87.134187
10	Armadillo	37	1.063591	0.075737	87.277898
25	Armadillo	40	0.689957	0.055863	87.032985
50	Armadillo	42	0.596794	0.047534	87.134187
0	Fertility	40	0.400242	0.023533	87.331223
10	Fertility	35	0.465996	0.038418	87.261435
25	Fertility	40	0.374687	0.026338	87.290300
50	Fertility	40	0.400242	0.023533	87.331223
0	Hand	150	0.528416	0.040157	86.049544
10	Hand	150	0.797670	0.077079	86.805937
25	Hand	150	0.627367	0.054135	86.338857
50	Hand	150	0.523630	0.047686	86.377507

Additionally, MeshLab was also used for generating a color map of the Hausdorff distance over the mesh. Figures 5.11, 5.12, 5.13 and 5.14 illustrates the Hausdorff distance over the four meshes for $p = 10$, giving an idea of which parts of the mesh have bigger geometric losses.

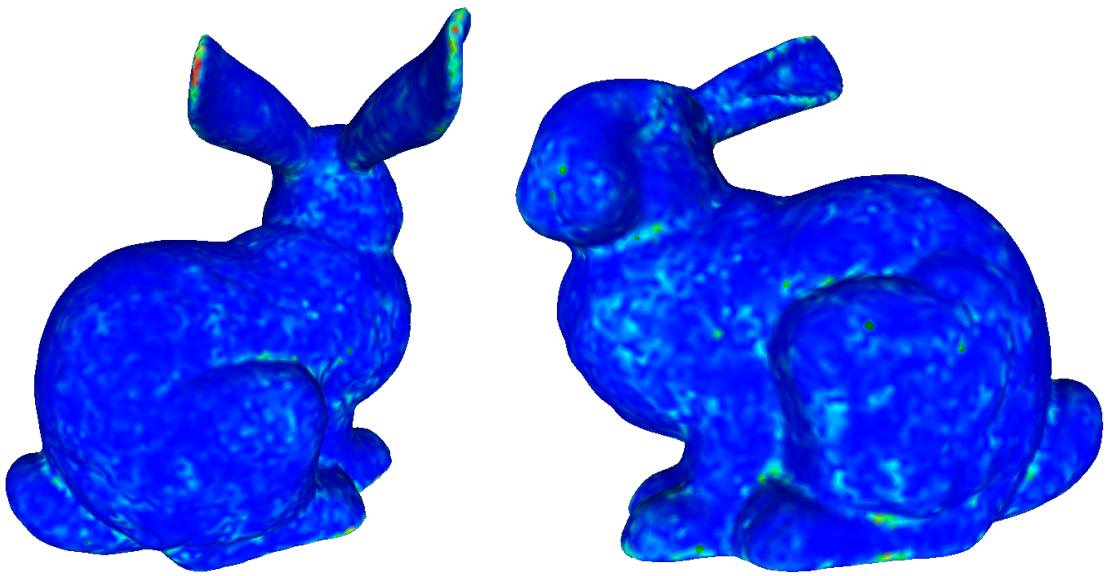


Figure 5.11: Hausdorff distance of the processed Bunny model to the original one onto a blue-green-red scale.

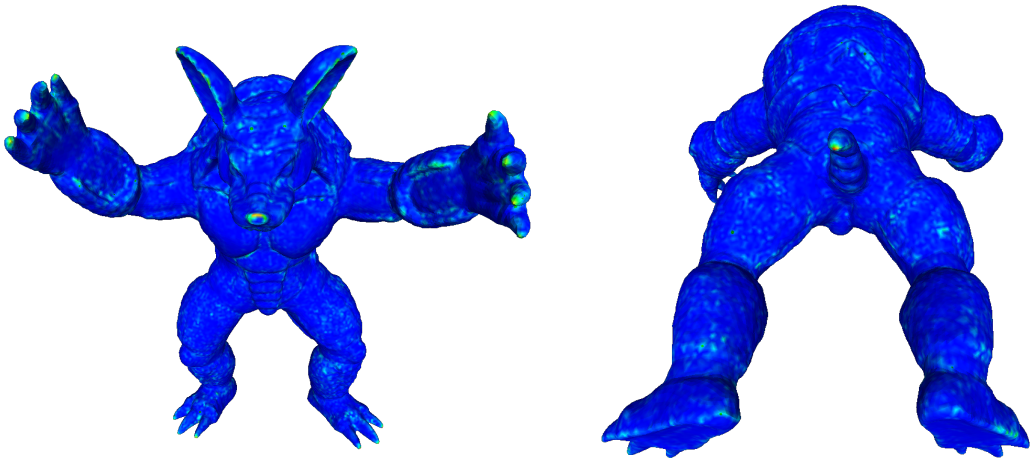


Figure 5.12: Hausdorff distance of the processed Armadillo model to the original one onto a blue-green-red scale.

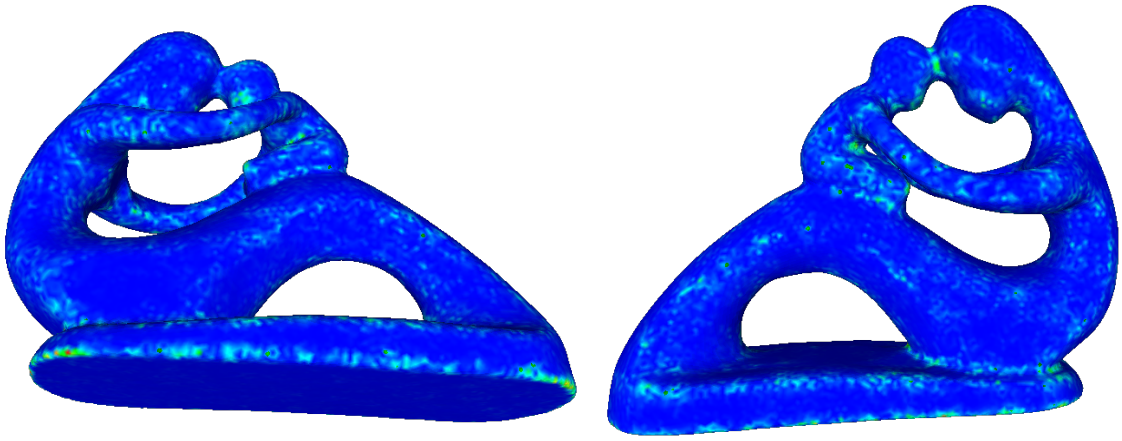


Figure 5.13: Hausdorff distance of the processed Fertility model to the original one onto a blue-green-red scale.

Figure 5.14 clearly shows that our efforts to preserve the boundary were effective, as the border does not present the highest Hausdorff distance. It is also possible to notice from Figures 5.11, 5.12, 5.13 and 5.14 that regions with higher curvature are also the ones with the greatest Hausdorff distance. This is expected, since the Laplacian operator smooths those regions, while the constant change in the projection mesh make this smooth permanent.

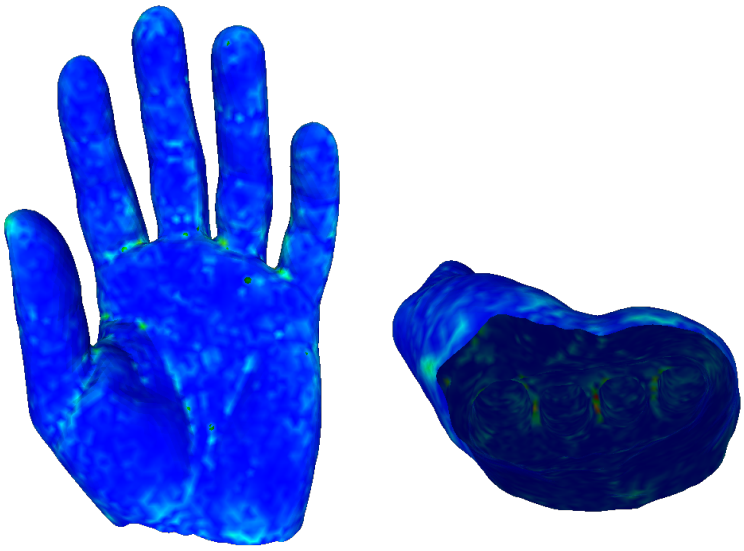
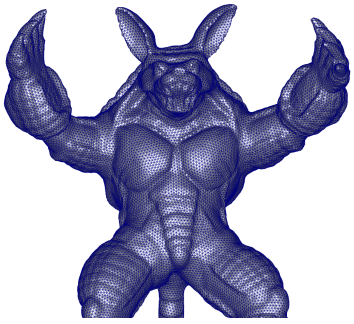


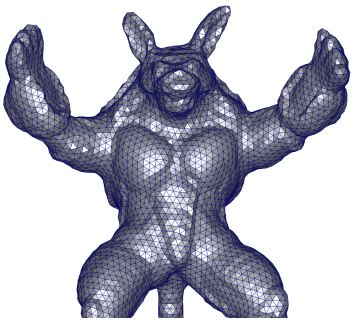
Figure 5.14: Hausdorff distance of the processed Hand model to the original one onto a blue-green-red scale.

5.4 INTERVAL

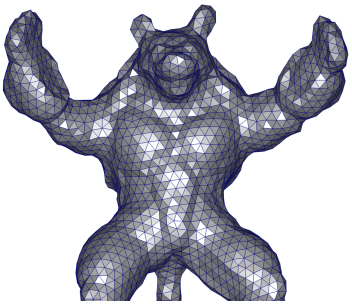
This section shows how changes in the desired interval affects the method. In order to do so, we performed tests varying the constraining interval for each one of the four models.



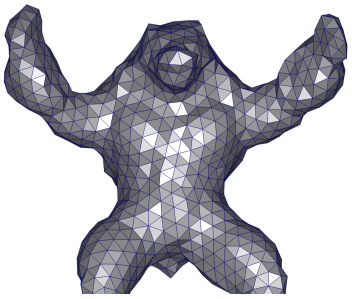
(a) Processed Armadillo with $e_{min} = 1.44$ and $e_{max} = 2.16$.



(b) Processed Armadillo with $e_{min} = 3.2$ and $e_{max} = 4.8$.

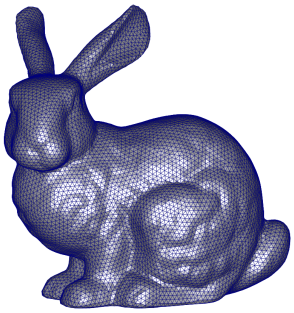


(c) Processed Armadillo with $e_{min} = 4.8$ and $e_{max} = 7.2$.

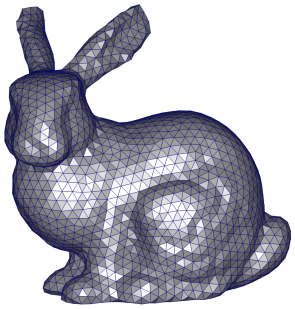


(d) Processed Armadillo with $e_{min} = 7.2$ and $e_{max} = 10.8$.

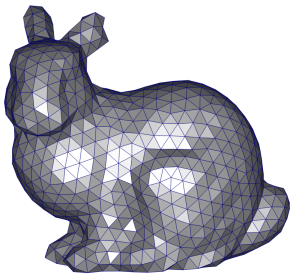
Figure 5.15: Resulting Armadillo model for different constraining intervals.



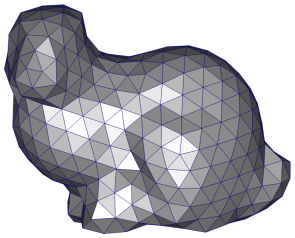
(a) Processed Bunny with $e_{min} = 1.44$ and $e_{max} = 2.16$.



(b) Processed Bunny with $e_{min} = 3.2$ and $e_{max} = 4.8$.

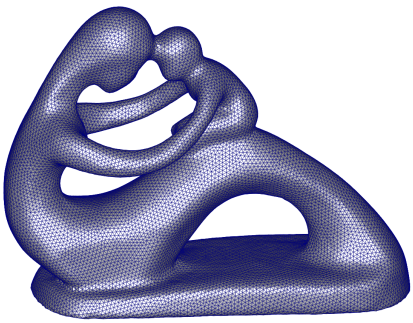


(c) Processed Bunny with $e_{min} = 4.8$ and $e_{max} = 7.2$.

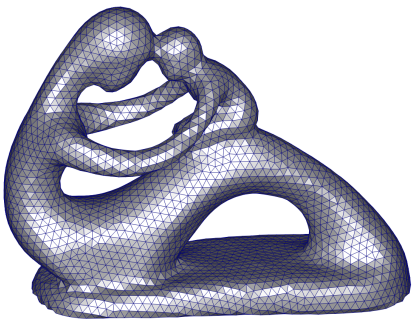


(d) Processed Bunny with $e_{min} = 7.2$ and $e_{max} = 10.8$.

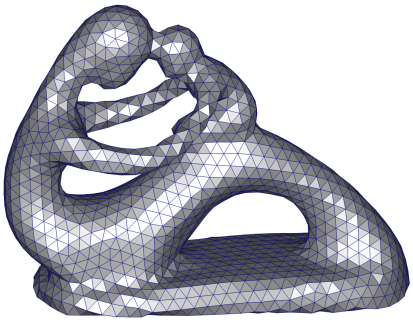
Figure 5.16: Resulting Bunny model for different constraining intervals.



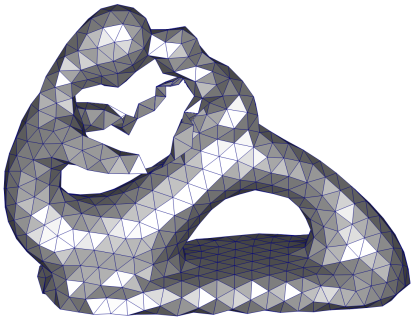
(a) Processed Fertility with $e_{min} = 1.44$ and $e_{max} = 2.16$.



(b) Processed Fertility with $e_{min} = 3.2$ and $e_{max} = 4.8$.

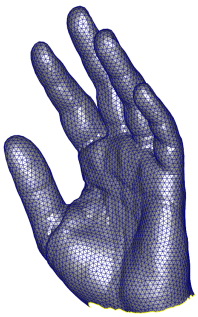


(c) Processed Fertility with $e_{min} = 4.8$ and $e_{max} = 7.2$.

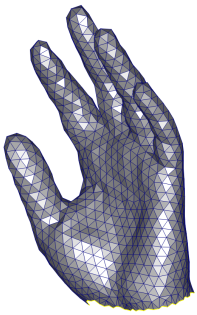


(d) Processed Fertility with $e_{min} = 7.2$ and $e_{max} = 10.8$.

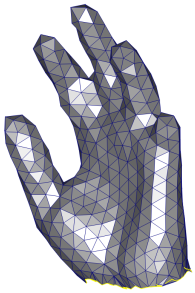
Figure 5.17: Resulting Fertility model for different constraining intervals.



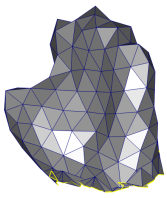
(a) Processed Hand with $e_{min} = 1.44$ and $e_{max} = 2.16$.



(b) Processed Hand with $e_{min} = 3.2$ and $e_{max} = 4.8$.



(c) Processed Hand with $e_{min} = 4.8$ and $e_{max} = 7.2$.



(d) Processed Hand with $e_{min} = 7.2$ and $e_{max} = 10.8$.

Figure 5.18: Resulting Hand model for different constraining intervals.

Figures 5.15, 5.16, 5.17 and 5.18 show the impact of different intervals over the geometry of the models. Each original model has an initial edge length average of 2.25. It is possible to notice that, for every model, when the mesh is refined the geometry is largely preserved. Slightly simplifications also fairly preserve the geometry. However, when a huge simplification is performed, the method cannot preserve the geometry. This makes sense, since information has to be lost due to the reduction in the numbers of points and triangles representing the same model.

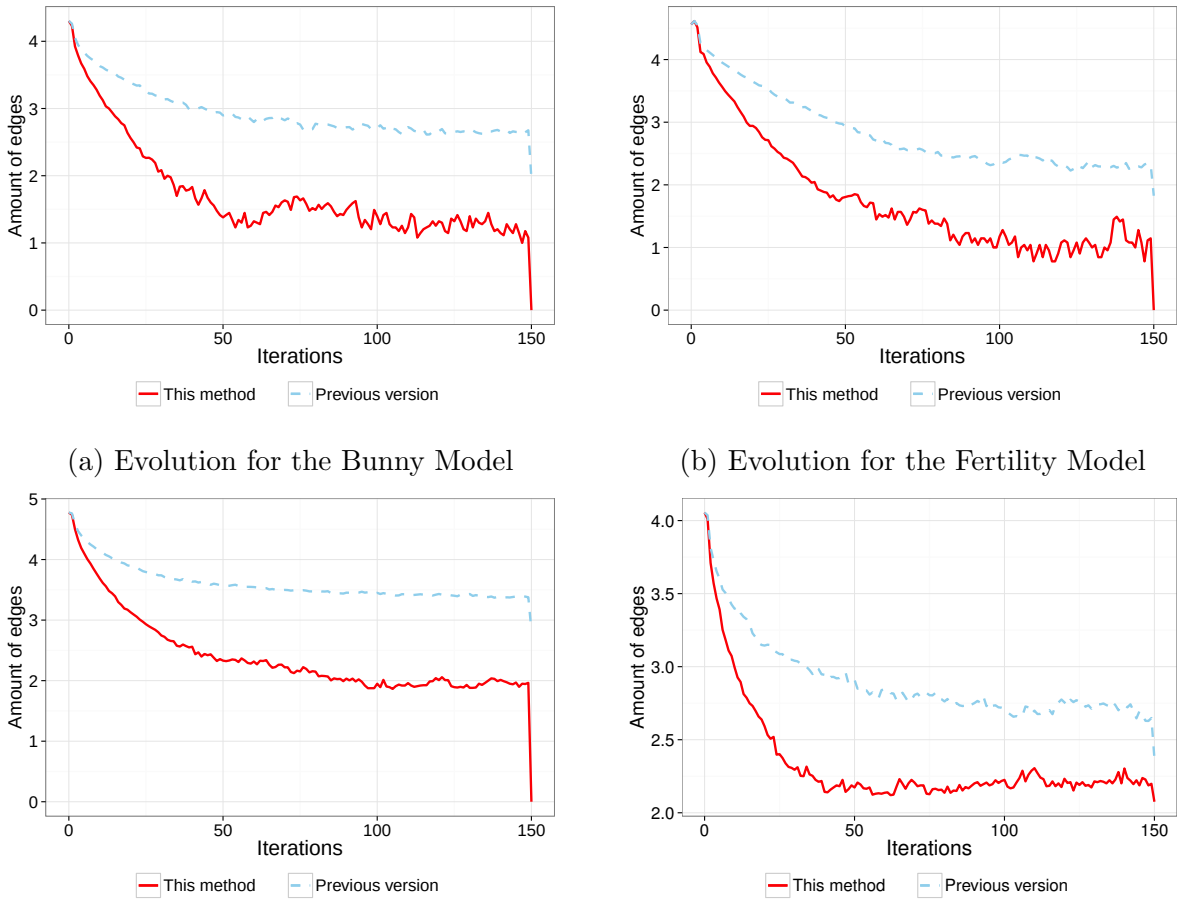
5.5 COMPARISON WITH PREVIOUS WORKS

This work is an improvement of the work by (HAUCK et al., 2015). This work enhances the Laplacian operator and the valence optimizer step. The Laplacian operator now is applied in a local parametric space. As for the valence optimizer step, a new priority list is included to minimize the maximum valence error per vertex $\mathcal{E}val_i$.

The first comparison is related to the convergence rate, excluding the nonlinear optimizer as it remains unchanged. To do so, we run the method with the parameter $l = 0$ for 150 iterations. Table 5.3 presents the results for each test. Every test presented in the

table uses the same parameters (1.2,1.8,150,2,2,0,0.5,MEANVALUE), with the except of kn and w which are not used in the previous version of the method. It is possible to notice that even with the parameter p set to 0, the current method constrains all edges lengths within the given interval, except for the Hand model, while the previous one cannot do the same.

Figure 5.19 shows a graph for each model containing the evolution of long and short edges for both versions of the method. The current version is clearly superior, as the amount of long and short edges is always smaller than in the previous version.



(a) Evolution for the Bunny Model

(b) Evolution for the Fertility Model

(c) Evolution for the Armadillo Model

(d) Evolution for the Hand Model

Figure 5.19: The evolution of long and short edges for this method and its previous version (HAUCK et al., 2015). The amount of long and short edges are in log_{10} scale.

The method of Hauck et al. (2015) uses a non quadratic error for the valence optimizer (Equation 4.2). That approach can achieve a higher percentage of vertices with the ideal valence, but results in several vertices with $\mathcal{E}val_i$ greater than 1. In the current version of the method, there is not a single vertex with $\mathcal{E}val_i$ greater than 1. This is especially useful for the application that will be presented in Chapter 6, as carbon nanostructures usually

Table 5.3: Comparison of the current method with its previous version (HAUCK et al., 2015).

Method	**Model**	**Long/Short**	$\overline{x}$	S	$\mathcal{E}val_i = 0$	$\mathcal{E}val_i = 1$	$\mathcal{E}val_i > 1$
Current	Bunny	0	1.52789	0.104681	87.35%	12.65%	0.00%
Previous	Bunny	90	1.54694	0.103674	89.47%	10.29%	0.24%
Current	Fertility	0	1.52923	0.102217	87.84%	12.16%	0.00%
Previous	Fertility	66	1.54877	0.099626	90.09%	9.78%	0.13%
Current	Armadillo	0	1.52765	0.104133	88.12%	11.88%	0.00%
Previous	Armadillo	676	1.54587	0.109272	88.45%	10.88%	0.67%
Current	Hand	118	1.52668	0.110736	86.37%	13.63%	0.00%
Previous	Hand	235	1.54419	0.112366	88.45%	9.81%	1.74%

do not form quads and/or octagons. These polygons only appear in meshes containing at least one vertex with $\mathcal{E}val_i > 1$.

5.6 FLAW CASES

Even though the method has improved a lot over the previous versions, it still has some drawbacks. For instance, if a strong simplification is required, it almost destroys the original geometry. This is expected, as a lot of information will be lost. The method also has problems when the model presents a huge variation over vertices distribution. The Octopus model is an example of this. It is oversampled in some areas of high curvature, so in those areas a strong simplification is performed, locally destroying the geometry.

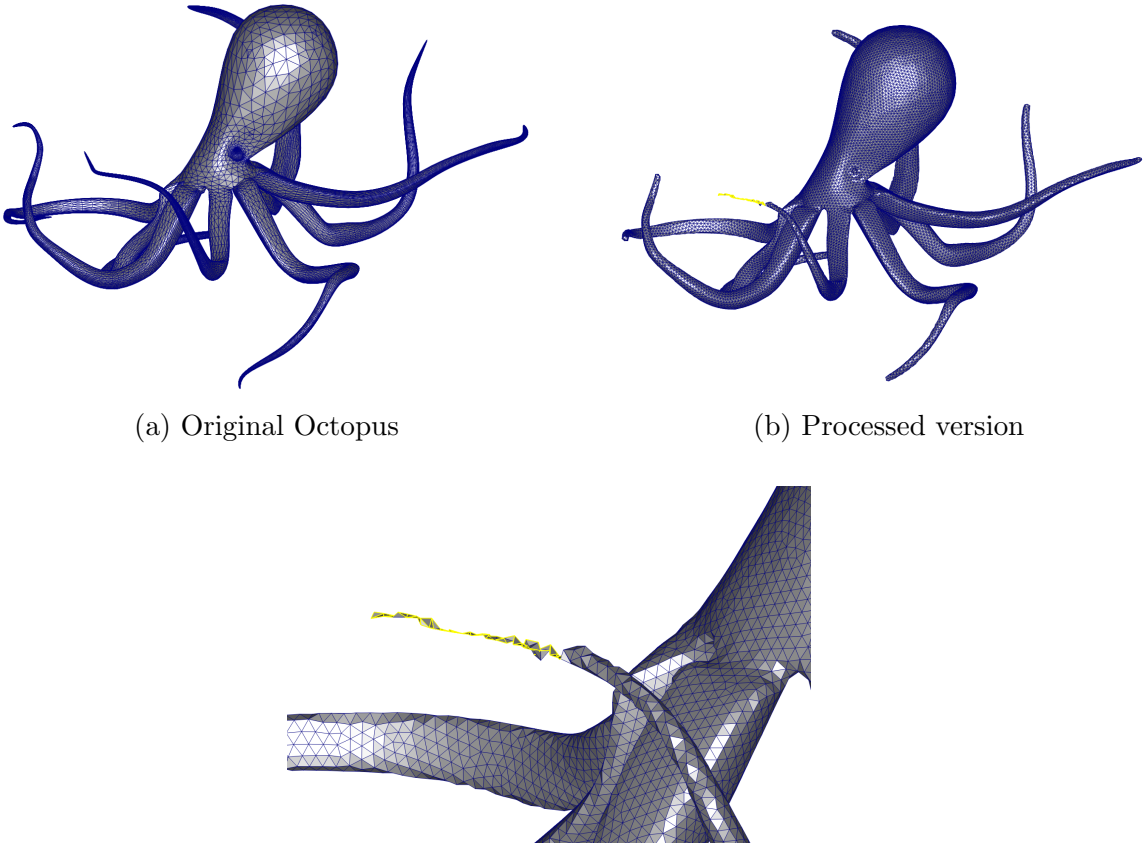
A clear example of failure when a strong simplification is performed can be seen in the Fertility model depicted in Figure 5.17d on page 45. As evinced by the arm of fertility, not only the geometry is lost, but the topology may also be destroyed.

Figure 5.20 shows the original Octopus model and its processed version. It can be noticed that the model presents a huge geometric loss. The processed version also has problems in the topology, as a border is added in the tentacles.

The method also fails to put every edge within a too tight constraining interval. This problem usually appear when $e_{min} > \frac{2}{3}e_{max}$. In those cases the Laplacian operator cannot diminish the amount of long and short edges because a large amount of stellar operations are done per iteration. Thus, it is recommended that Inequality 5.1 be respected.

$$e_{min} \leq \frac{2}{3}e_{max} \tag{5.1}$$

If this restriction is not respected, the method may achieve an almost uniform mesh, but



(a) Original Octopus

(b) Processed version

(c) Zoom on the problematic tentacle

Figure 5.20: The result of processing the Octopus model.

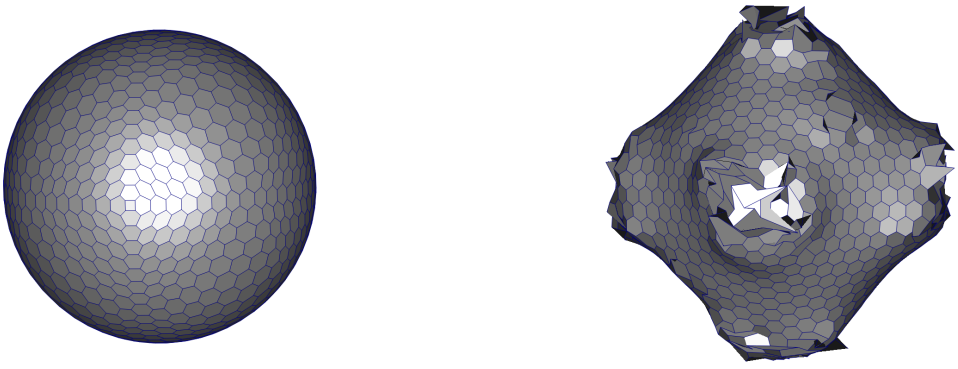
it usually cannot constrain every edge into the given interval.

6 AN APPLICATION IN PHYSICS

The method proposed in this work may be used for modeling carbon nanostructures (SILVA, 2014). In order to do so, an arbitrary mesh is processed, constraining all its edges to the interval in which Carbon-Carbon bounds exist. In an effort to validate this claim, the resulting mesh is simulated using molecular dynamics. This usually needs an expression for the potential energy, and this energy is combined with Newton's laws to determine the movement of each atom in the simulation. The simulations performed in this work use REBO2 (BRENNER et al., 2002) for computing the potential of the Carbon-Carbon interactions, a referential temperature of 300 kelvin and a time step of 1 femtoseconds. The processed models presented in this chapter were processed using the parameters (1.2,1.8,150,2,2,0,0.5,MEANVALUE).

6.1 ORDINARY MODELS

The first results here presented are simple structures. Despite their simplicity, these structures still gather the interest of the scientific community, due to the fact they are useful for experimentation. In this work, we used a sphere and a helix.

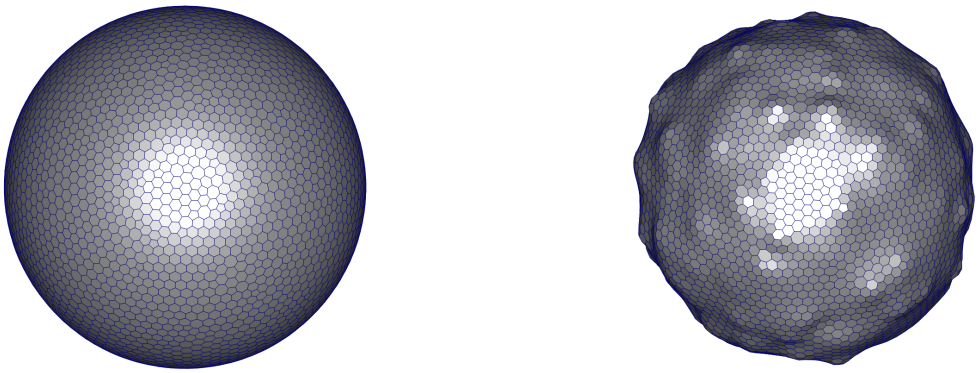


(a) Original Sphere model.

(b) After 10 picoseconds of simulation, keeping the original connectivity.

Figure 6.1: The result of simulating the Sphere model.

Figure 6.1 shows the original sphere model and after being simulated as a carbon nanostructure for 10 picoseconds. Even though models seem uniform, they lose stability when simulated as a carbon nanostructure.



(a) Processed Sphere.

(b) After 10 picoseconds of simulation, keeping the original connectivity.

Figure 6.2: The result of simulating a sphere processed by this method.

Figure 6.2 shows a sphere processed by our method after being simulated as a carbon nanostructure for 10 picoseconds. Even though it is possible to notice some deformations, a huge improvement was obtained when compared to Figure 6.1.

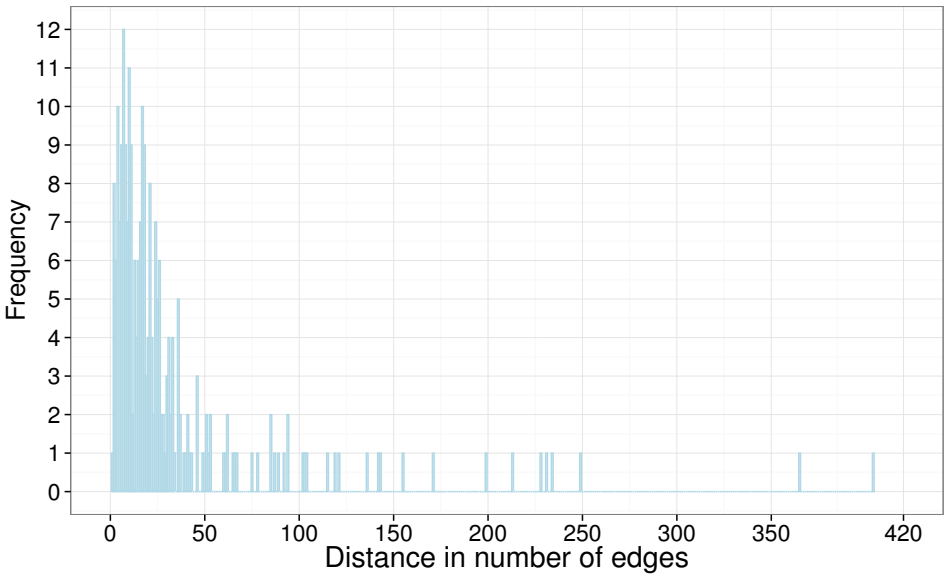


Figure 6.3: Histogram of the distance, in number of edges, from a pentagon to its closest pentagon on the sphere.

The sphere is a special structure, which has symmetry in all directions, hence it can be used to measure how well the method distributes irregular polygons. We use the number of edges as a measure of distance for computing two histograms. The first one (Fig. 6.3) measures the distance between a pentagon and its closest pentagon, the second one (Fig. 6.4) likewise measures the distance between a heptagon and its closest heptagon. It is possible to notice that pentagons and heptagons have on most times an irregular neighbor

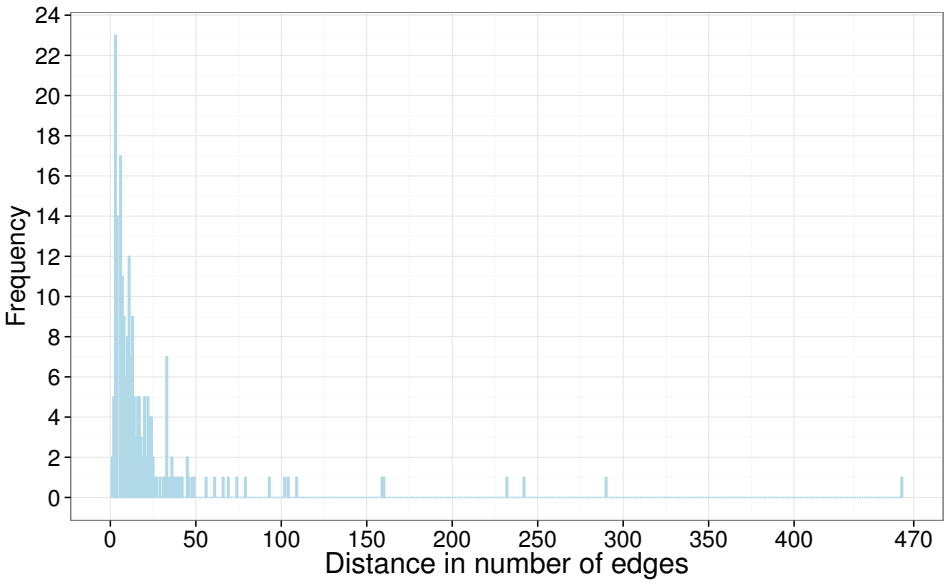


Figure 6.4: Histogram of the distance, in number of edges, from a heptagon to its closest heptagon on the sphere

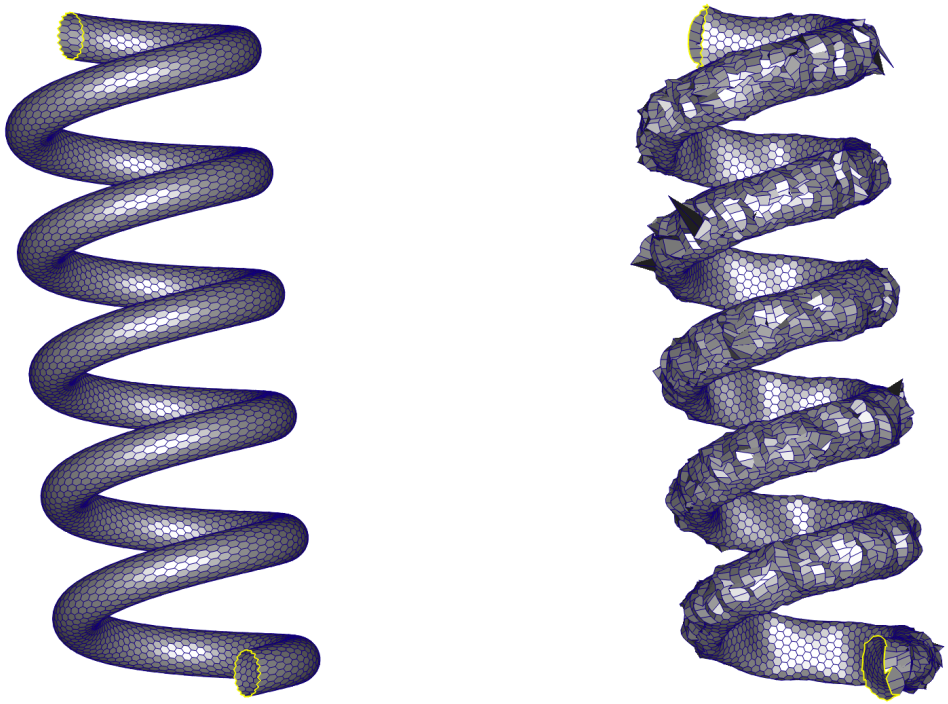
with a distance smaller than 50 edges. This suggests that pentagons and heptagons are well distributed over the surface, with few exceptions.

Figure 6.5 shows the original helix model and the result of its simulation as a carbon nanostructure for 10 picoseconds. Although the model is visually well sampled, it is not stable when simulated. Figure 6.6 also shows a helix being simulated, but this time it was previously processed by our method. The structure now suffers some deformations, but clearly maintains its shape. Even though a helix is a simple structure, there are several applications for them as carbon nanosprings.

6.2 GEOMETRICALLY AND TOPOLOGICALLY MORE COMPLEX MODELS

The method proposed in this work is really useful for modeling complex structures of carbon, which cannot be modeled directly by any mathematical technique. For these structures, a method like this may be the only way for modeling them. In this section we will simulate the Fertility, Horse, Armadillo, Elk and Woman models as carbon nanostructures.

Figure 6.7 shows the original Fertility model and the resulting mesh after simulating it as a carbon nanostructure for 10 picoseconds. The model is not stable, and it is



(a) Original helix model.

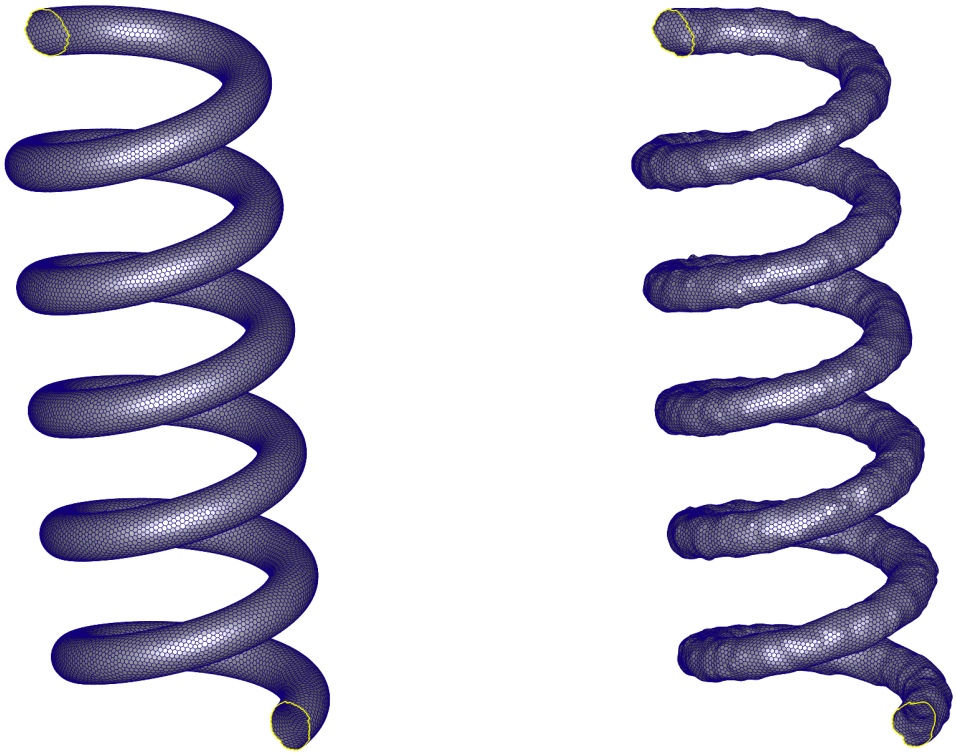
(b) After 10 picoseconds of simulation, keeping the original connectivity.

Figure 6.5: The result of simulating original helix as a carbon nanostructure.

completely destroyed after the simulation. On the other hand, Figure 6.8 shows the processed version of the Fertility model and the result of its simulation. Although the simulated model has some geometric distortions, the model is pretty stable and the overall geometry is preserved. One may observe that the geometric distortions are higher in the regions around a non-hexagon polygon. This could be an important clue for future works.

The Fertility model has a symmetry in the plane parallel to the camera plane in Figure 6.8. The amount of irregular polygons of each region was computed. The first region has 1987 pentagons and/or heptagons while the second one has 2021. As one may see the amount is almost the same, revealing that even though our method does not explicitly constrain the distribution of irregular polygons over the surface, they were equally distributed in this example.

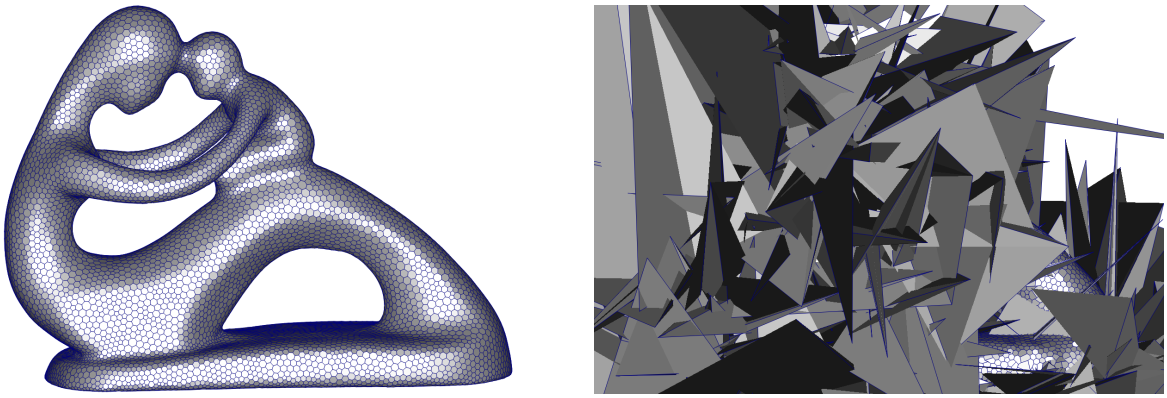
The figure 6.9 shows the simulation of a horse model. This model is smaller than the Fertility model, allowing the analysis of the method in a model with fewer vertices and polygons. The simulation in this case greatly preserve the geometry. For this model and



(a) Processed helix.

(b) After 10 picoseconds of simulation, keeping the original connectivity.

Figure 6.6: The result of simulating a helix processed by this method.



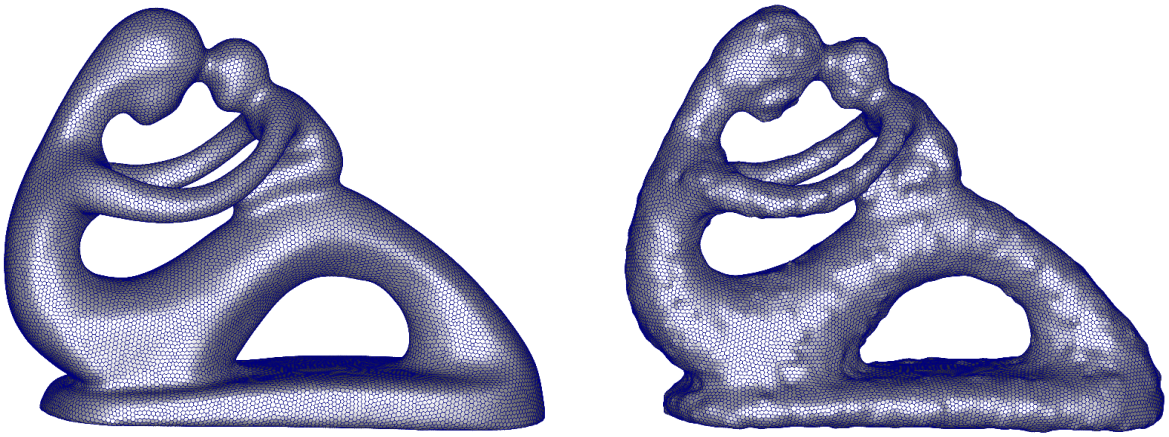
(a) Original Fertility model.

(b) After 10 picoseconds of simulation, keeping the original connectivity.

Figure 6.7: The result of simulating the original Fertility model.

the next ones, we present only the simulation of the processed model, since the previous results already shown that the original model is not stable.

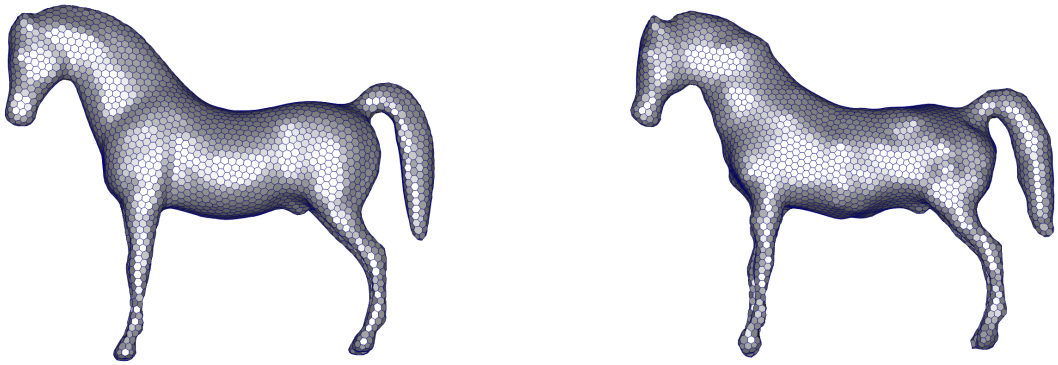
Figure 6.10 illustrates the behavior in a bigger and more geometrically complex model, the Armadillo. This model lost much of its geometric details, but maintains its general form. This is the last model presented in which the method was capable of generating a stable carbon nanostructure.



(a) Processed Fertility model

(b) After 10 picoseconds of simulation, keeping the original connectivity.

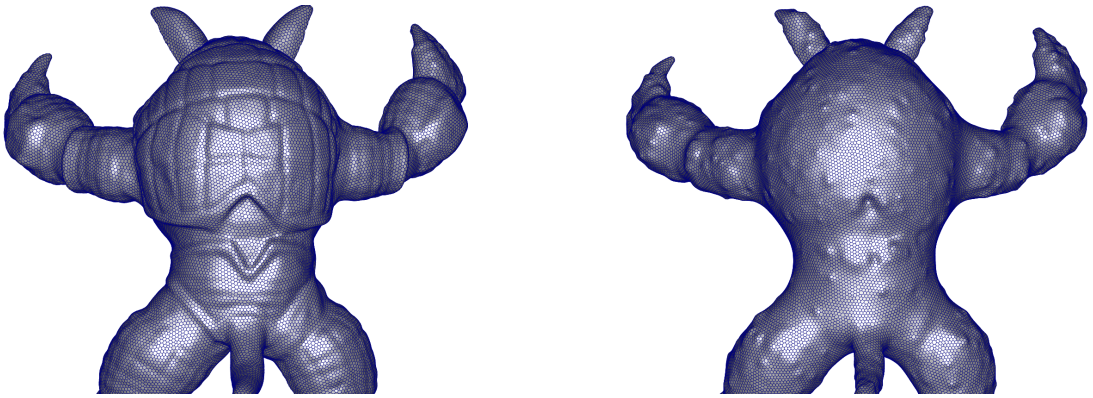
Figure 6.8: The result of simulating the processed Fertility model.



(a) Processed Horse model

(b) After 10 picoseconds of simulation, keeping the original connectivity.

Figure 6.9: The result of simulating the processed Horse model.



(a) Processed Armadillo model

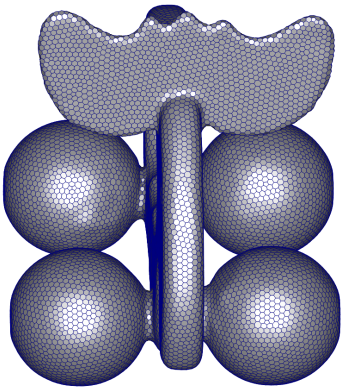
(b) After 10 picoseconds of simulation, keeping the original connectivity.

Figure 6.10: The result of simulating the processed Armadillo model.

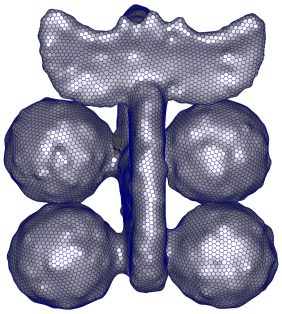
This model also has an obvious symmetry, dividing the model on two sides (right and left) in Figure 6.10. The amount of irregular polygons was computed for each side. The right side has 3161 irregular polygons while the left side has 3147. The results in this model are similar to the ones found in the Fertility model, with equally distributed pentagons and heptagons in the model.

6.2.1 FLAW CASES

Even though the method can transform a variety of models in carbon nanostructures, some models may have geometric and/or topological features which cannot be modeled with carbon-carbon bonds. This subsection presents two models with those characteristics.

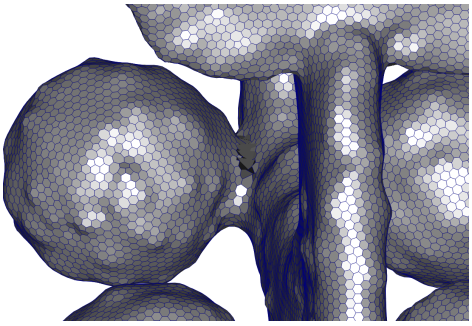


(a) Processed Elk model



(b) After 10 picoseconds of simulation, keeping the original connectivity.

Figure 6.11: The result of simulating the processed Elk model.

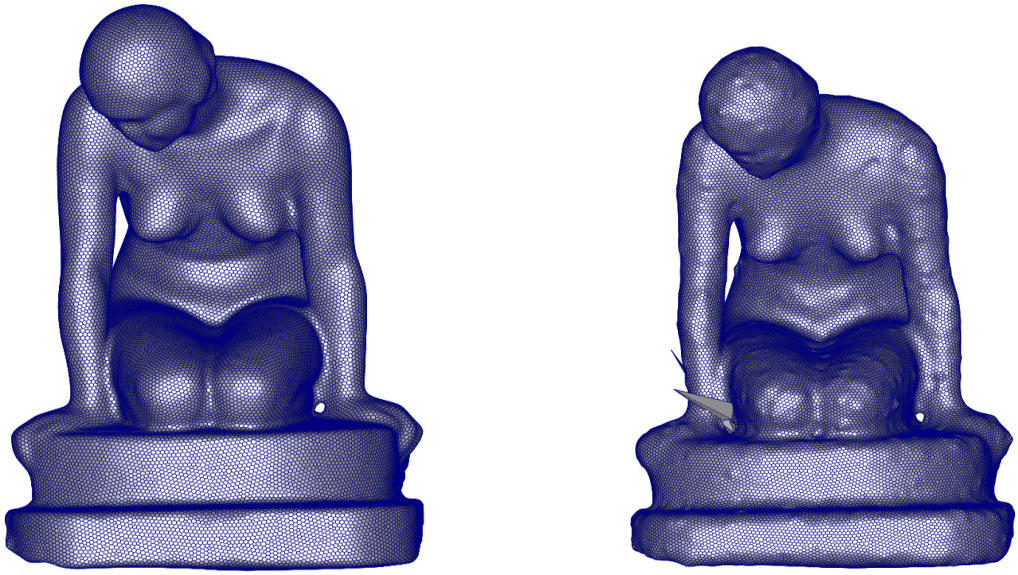


(c) Zoom in the connection between the sphere and the model.

Figure 6.11: The result of simulating the processed Elk model.

Figure 6.11 shows a model with challenging geometry, as seen on the connection between the spheres and the main body of the model. These regions present a huge curvature,

which is also changing fast. For these reasons even the processed version of model is not stable. The connection between the sphere and the model presents invalid polygons, as illustrated in Figure 6.11c.



(a) Processed Woman model

(b) After 10 picoseconds of simulation, keeping the original connectivity.

Figure 6.12: The result of simulating the processed Woman model.

Finally the most challenging model is the Woman model. This model has a genus greater than it appears, as the region closer to the right hand has a tunnel. As for the left hand, there is an intrusion, which almost forms a tunnel (Fig. 6.12). Figure 6.12 illustrates the resulting of simulating this model as carbon nanostructure. The simulation clearly fails next to the left hand, probably because in the tiny intrusion the curvature is higher than a carbon bond can support.

7 CONCLUSION

This work presented a method for remeshing an arbitrary model into another one with all edges within an user-defined interval. As demonstrated in Chapter 6, the method is also capable of transforming several arbitrary models into carbon nanostructures. This work also analyzed the effects of each parameter over the results.

The inclusion of a local parametric space for Laplacian operation proves its value, as the method convergence improves, in comparison to our previous work (HAUCK et al., 2015). It also diminishes the geometric error introduced by the Laplacian operator, as the new vertex position computed by the operator is already on the surface. In addition, the inclusion of a priority list in the valence optimizer improves method stability, since a fixed order is imposed for applying the flips. Another effect of this priority list is the reduction of the vertices with the valence error $\mathcal{E}val$ greater than 1. All these improvements allow the usage of lower l values, which presents better results.

Even though the method usually presents satisfactory results, there are some cases in which the method fails, as discussed in Section 5.6. For instance, meshes with points of extremely high curvature and/or variable density of vertices are especially challenging. The method also does not converge when the constraining interval is too tight.

There are plenty of improvements that could be done to this method. The most obvious improvement is to extend the local parametrization for the nonlinear optimization. Another possibility is to use a mass spring system to smooth the mesh while maintaining the edges lengths inside the desired interval. The method could also automatically adjust its parameters, improving its usability. This automatic parametrization could lead to better results, as the parameters may change during the iterations in a similar way to the e^i_{min} and e^i_{max}.

Additionally, our method could be extended in order to have a variable constraining interval over the model. This is useful when some regions of the mesh must be extremely detailed while others have not. In our method the entire model should be oversampled to increase the odds of achieving the global constraint requirement.

In order to improve the performance of the method in the generation of carbon nanostructures, an optimization step for the dual mesh could be added. This is useful as the

dual mesh may still present some long or short edges even when the primal does not.

Another problem that requires attention for generating carbon nanostructures is the number of interior vertices with valence not equal to 6 and their positions. Those vertices generate polygons which are not hexagons. These polygons, during the simulation, create distortions over the geometry. Thus, a new algorithm for the valence optimizer step which takes this into account could be proposed.

REFERENCES

ALICE. **Graphite: an experimental 3d modeler**. 2000–2012. `http://alice.loria.fr/software/graphite`.

ALLIEZ, P.; MEYER, M.; DESBRUN, M. Interactive geometry remeshing. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 21, n. 3, p. 347–354, jul. 2002. ISSN 0730-0301.

BOMMES, D.; ZIMMER, H.; KOBBELT, L. Mixed-integer quadrangulation. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 28, n. 3, p. 77:1–77:10, jul. 2009. ISSN 0730-0301.

BOTSCH, M.; KOBBELT, L. A remeshing approach to multiresolution modeling. In: **Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing**, 2004. (SGP '04), p. 185–192. ISBN 3-905673-13-4.

BOTSCH, M.; KOBBELT, L.; PAULY, M.; ALLIEZ, P.; LÉVY, B. **Polygon mesh processing**, 2010.

BRENNER, D. W.; SHENDEROVA, O. A.; HARRISON, J. A.; STUART, S. J.; NI, B.; SINNOTT, S. B. A second-generation reactive empirical bond order (rebo) potential energy expression for hydrocarbons. **Journal of Physics: Condensed Matter**, v. 14, n. 4, p. 783, jan. 2002.

CIGNONI, P.; CORSINI, M.; RANZUGLIA, G. Meshlab: an open-source 3d mesh processing system. **Ercim news**, v. 73, p. 45–46, 2008.

CIGNONI, P.; ROCCHINI, C.; SCOPIGNO, R. Metro: measuring error on simplified surfaces. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**, 1998. v. 17, n. 2, p. 167–174.

DEMBOGURSKI, R.; DEMBOGURSKI, B.; SILVA, R. L. Souza da; VIEIRA, M. B. Interactive mesh generation with local deformations in multiresolution. In: MURGANTE, B.; MISRA, S.; CARLINI, M.; TORRE, C.; NGUYEN, H.-Q.; TANIAR, D.; APDUHAN, B.; GERVASI, O. (Ed.). **Computational Science and Its Applications**

– **ICCSA 2013**, 2013, (Lecture Notes in Computer Science, v. 7971). p. 646–661. ISBN 978-3-642-39636-6.

FLOATER, M. S. Parametrization and smooth approximation of surface triangulations. **Computer aided geometric design**, Elsevier, v. 14, n. 3, p. 231–250, 1997.

FLOATER, M. S. Mean value coordinates. **Computer aided geometric design**, Elsevier, v. 20, n. 1, p. 19–27, 2003.

GOTSMAN, C.; GU, X.; SHEFFER, A. Fundamentals of spherical parameterization for 3d meshes. In: ACM. **ACM Transactions on Graphics (TOG)**, 2003. v. 22, n. 3, p. 358–363.

GUILLEMIN, V.; POLLACK, A. Differential topology. In: **Differential Topology**, 1974.

HAUCK, J. D. S.; SILVA, R. D.; VIEIRA, M.; SILVA, R. D. Adaptive remeshing for edge length interval constraining. **Journal of Mobile Multimedia**, v. 11, n. 1-2, p. 75–89, 2015. Cited By 0.

HAUCK, J. V.; SILVA, R. N. da; VIEIRA, M. B.; SILVA, R. L. d. S. da. Iterative remeshing for edge length interval constraining. In: **Computational Science and Its Applications–ICCSA 2014**, 2014. p. 300–312.

HUANG, J.; ZHANG, M.; PEI, W.; HUA, W.; BAO, H. Controllable highly regular triangulation. **Science China Information Sciences**, SP Science China Press, v. 54, n. 6, p. 1172–1183, 2011. ISSN 1674-733X.

IIJIMA, S. et al. Helical microtubules of graphitic carbon. **Nature**, London, v. 354, n. 6348, p. 56–58, 1991.

KUSHNER, A.; LYCHAGIN, V.; RUBTSOV, V. **Contact geometry and nonlinear differential equations**, 2007.

PAMPANELLI, P. C. P. **Mesh Generation Through the Mapping of Triangular Models into Rhomboid Space**. Dissertação (M.Sc. Dissertation) — Universidade Federal de Juiz de Fora, Advisor: Marcelo Bernardes Vieira, Co-advisors: Marcelo Lobosco and Sócrates de Oliveira Dantas, 2011.

PEÇANHA, J. P.; FILHO, J. S.; VIEIRA, M. B.; LOBOSCO, M.; DANTAS, S. Iterative method for edge length equalization. **Procedia Computer Science**, Elsevier, v. 18, p. 481–490, 2013.

PIETRONI, N.; TARINI, M.; CIGNONI, P. Almost isometric mesh parameterization through abstract domains. **IEEE Transaction on Visualization and Computer Graphics**, v. 16, n. 4, p. 621–635, July/August 2010.

RAPAPORT, D. C. **The Art of Molecular Dynamics Simulation**, 1996. ISBN 0521445612.

RAY, N.; VALLET, B.; LI, W.-C.; LéVY, B. N-symmetry direction field design. In: **ACM Transactions on Graphics**, 2008. Presented at SIGGRAPH.

ROCCHINI, C.; CIGNONI, P.; MONTANI, C.; PINGI, P.; SCOPIGNO, R. A low cost 3d scanner based on structured light. **Computer Graphics Forum**, Blackwell Publishers Ltd, v. 20, n. 3, p. 299–308, 2001. ISSN 1467-8659.

SCHöBERL, J. Netgen an advancing front 2d/3d-mesh generator based on abstract rules. **Computing and Visualization in Science**, Springer-Verlag, v. 1, n. 1, p. 41–52, 1997. ISSN 1432-9360.

SILVA, R. N. da. **Modeling Nanocarbon Structures using Adaptive Remeshing**. Monografia (Monography) — Universidade Federal de Juiz de Fora Advisor: Marcelo Bernardes Vieira, 2014.

SURAZHSKY, V.; GOTSMAN, C. Explicit surface remeshing. In: EUROGRAPHICS ASSOCIATION. **Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing**, 2003. p. 20–30.

SURAZHSKY, V.; GOTSMAN, C. High quality compatible triangulations. **Engineering with Computers**, Springer, v. 20, n. 2, p. 147–156, 2004.

TAUBIN, G. A signal processing approach to fair surface design. In: **Proceedings of the 22nd annual conference on Computer graphics and interactive techniques**, 1995. (SIGGRAPH '95), p. 351–358. ISBN 0-89791-701-4.

TAUBIN, G. Dual mesh resampling. In: IEEE. **Computer Graphics and Applications, 2001. Proceedings. Ninth Pacific Conference on**, 2001. p. 180–188.

TAUBIN, G. et al. Geometric signal processing on polygonal meshes. **Eurographics State of the Art Reports**, v. 4, n. 3, p. 81–96, 2000.

VIEIRA, M. B.; VELHO, L.; SA, A.; CARVALHO, P. C. A camera-projector system for real-time 3d video. In: IEEE. **Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on**, 2005. p. 96–96.

VORSATZ, J.; RöSSL, C.; SEIDEL, H.-P. Dynamic remeshing and applications. In: **Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications**, 2003. (SM '03), p. 167–175. ISBN 1-58113-706-0.

WEILER, K. Edge-based data structures for solid modeling in curved-surface environments. **IEEE Computer graphics and applications**, n. 5, p. 21–40, 1985.