Universidade Federal de Juiz de Fora
Instituto de ciências Exatas
Bacharelado em Ciência da Computação

Iterative method for edge equalization of triangular meshes

João Vitor de Sá Hauck

JUIZ DE FORA
AGOSTO, 2013

Iterative method for edge equalization of triangular meshes

João Vitor de Sá Hauck

Universidade Federal de Juiz de Fora

Instituto de ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Marcelo Bernardes Vieira

JUIZ DE FORA

AGOSTO, 2013

Iterative method for edge equalization of triangular meshes

João Vitor de Sá Hauck

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Marcelo Bernardes Vieira
Doctor in Computer Science

Socrates de Oliveira Dantas
Doctor in Physics

Rodrigo Luis De Souza Da Silva
Doctor in Civil Engineering

JUIZ DE FORA

29 DE AGOSTO, 2013

To my friends and my sister.

To my Parents for their support and sustenance.

Resumo

Este trabalho apresenta um método para remalhamento de superfícies triangulares. O método obtem uma nova superfície de maneira que qualquer aresta está dentro de um intervalo predefinido $[e_{min}, e_{max}]$. A entrada do processo é uma malha de 2-variedade com geometria e topologia arbitrarias. O algoritmo proposto é iterativo e consegue ajustar automaticamente a quantidade de vértices e triângulos necessários através das operações estelares. Um filtro passa-baixa também é aplicado para retirar as altas frequências. O algoritmo gera uma malha triangular de 2-variedade, com os vértices distribuídos de maneira quase uniforme sobre a superfície de entrada. No fim do processo, praticamente todas as arestas estão dentro do intervalo. A malha dual desta malha triangular é uma malha trivalente. Este tipo de malha tem muitas aplicações em simulações de nano estruturas de carbono.

Palavras-chave: equalização do comprimento de aresta, operações estelares, remalhamento.

Abstract

This work presents a method for remeshing triangular surfaces. This method obtains a new mesh in such a way that any edge is within a predefined interval $[e_{min}, e_{max}]$. The input of the process is a 2-manifold mesh with arbitrary geometry and topology. The proposed algorithm is iterative and is able to automatically adjust the number of vertices and polygons through stellar operations. A low pass filter is also applied to remove higher frequencies. This algorithm generates a triangular 2-manifold mesh, with the vertices spreaded almost uniform over the input surface. At the end of the process, almost all edges are within the interval. The dual mesh of this triangular mesh is a trivalent mesh also very uniform. This kind of mesh has many applications in simulations of carbon nano structures.

Keywords: edge length equalization, stellar operations, remesh.

Acknowledgments

This work would not be possible without the support of my parents Marcelo Moreira Hauck and Kelle Pereira de Fátima de Sá Hauck. That is why I thank them. I also thank my advisor Marcelo Bernardes Vieira for helping me with my confusing text, my friends for the emotional support in the entire graduation and the Universidade Federal de Juiz de Fora for offering this course completely for free.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introduction

This work aims the production of a method capable of, from a given mesh, generate a new mesh with edge lengths that are within a minimum and maximum value.

To understand this work it is necessary that some basic concepts are clear. These concepts are: polygon meshes to represent surfaces, triangular meshes, stellar operations, the laplacian operator and remeshing. To represent a surface in the computer memory a discretization is necessary. To achieve this there are several manners. One of the most used is to represent objects through polygon meshes. Polygon meshes are surfaces represented by discrete polygons, each polygon representing the surface locally and the entire surface is represented by all polygons together.

The polygon meshes can be made with different types of polygons, for example triangles, hexagons, quads, pentagons. In this work we are going to focus only in triangular meshes due to their great utility and practical use. Other very important mesh in this work are the trivalent meshes, in which each vertex has exactly three edges. They tend to form hexagons in low curvature surfaces and have recently attracted attention from the computer geometry community (M. Nieser et al, 2010). They are important for several applications, such as the physics simulation of carbon nano structures (Quinelato et al, 2010).

Stellar operations are a classic set of operations in polygon meshes known for keeping the Euler characteristic unchanged as in equation 1.1

$$v_b - e_b + f_b = v_a - e_a + f_a, \tag{1.1}$$

where v_b is the original number of vertices, e_b is the original number of edges, f_b is the original number of faces, v_a is the new number of vertices, e_a is the new number of edges, and f_a is the new number of faces. They are formed basically by the operations of edge split, vertex split, edge collapse, vertex collapse and edge flip (Oliveira et al, 2012).

The Laplacian operator is a common operator often used as a filter, as it tends to

remove the higher frequencies. Basically it is an operator that try to reposition the vertex in the centroid of its neighbors. In this work, we use the laplacian to act not only in the first neighbors but in the k-neighborhood with a differential weight to each neighborhood (Oliveira et al, 2012).

According to Mario et al (2010a), remeshing is an operation that generates a new mesh with greater quality than its original version. The term quality is dependent on the objective. For example, to minimize memory, the objective can be to reduce the number of polygons and vertices without losing accuracy. For this work, a quality mesh is the one that contains all edge lengths in the target interval. This is extremely useful in atomic simulations where the physically plausible distances between the atoms are known and limited.

1.1 Problem definition

Given an arbitrary triangular mesh $\mathcal{M}$, composed of edges $E_i \in \mathcal{M}$, generate a new trivalent mesh with the same geometry and topology, but having all edge lengths in the interval $[e_{min}, e_{max}]$. In other words, $e_{min} \leq |E_i| \leq e_{max} \ \ \forall\, E_i \subset \mathcal{M}$.

1.2 Motivation

In this work, we propose a new iterative process to transform a triangular mesh in a trivalent mesh whose edge lengths are within a predefined interval. The method was conceived to be used with most 2-manifold models found publicly.

In a previous work (Oliveira et al, 2012),(Oliveira et al, 2013), an iterative method was developed to force the median and standard-deviation of the mesh to be below expected thresholds. This works has the drawback of allowing some edges to be much greater than the overall average. The imposition of a minimum and maximum value for edge lengths is much more restrictive since the majority of the edges cannot compensate the extra size of a few outliers. This problem statement constrains even further the space of possible solutions for the mesh, and is a motivator for this work.

1.3 Objectives

The main objective of this work is to develop a method to transform a triangular mesh in a trivalent mesh with all edges within an interval. This method must preserve the geometry and topology of the surface.

As a secondary objective, the method also try to obtain a hexagonal mesh, in order to be practical for nano structure simulations. Another objective is to have low memory cost since some input meshes might have hundreds of thousands vertices. Methods based on linear system solving require large amounts of installed memory to process big meshes. We propose a iterative method capable of processing the large meshes found publicly.

2 Basics Concepts

2.1 Manifolds

A *manifold* represents a n dimension surface immersed in $k, k \geq n$ dimension space. In this kind of surfaces the neighborhood of each point can be reduced to an n dimension euclidean space (Guillemin and Pollack, 1974).

A surface is *orientable* if it has two sides. As strange as it can be, there are some surfaces that do not have two sides. The Möbius strip is a classic example and is depicted in Figure 2.1.

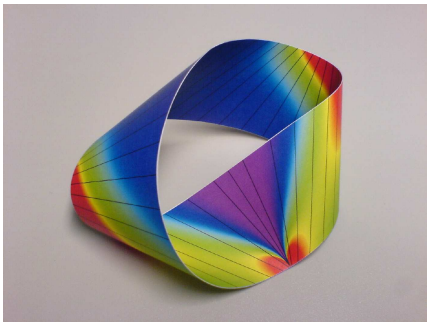

Figura 2.1: This is an example of non orientable surface (Starostin et al, 2010).

This work will be focused in surfaces that are orientable.

2.2 Euler characteristic

The Euler characteristic is a topological invariant of surfaces. In other words, any two surfaces that are homeomorphic will have the same Euler characteristic, it does not matter their size, shape or any other characteristic of the surface.

The geometric methods of this work must preserve the Euler characteristic in every operation to consequently preserve the topology of the surface.

2.3 Triangular meshes

A surface could be represented in many manners, a possible discretization with great use in computer graphics, due this simplicity and versatility are the triangular meshes. This is a especial kind of mesh where the surface is approximated through triangles, represented by their distinct and non-collinear three vertices $[A, B, C]$. Due to the barycentric parameterization, each point P in a triangle $[A, B, C]$ is represented as an unique linear combination:

$$P = \alpha A + \beta B + \gamma C, \tag{2.1}$$

$$\alpha + \beta + \gamma = 1. \tag{2.2}$$

Triangulation is an efficient way for approaching every point over the surface with a very simple data structure. Besides the simplicity, this approach has a great power of representation. Any three distinct and non-collinear points always generate a plan, which is simpler to deal geometrically.

The vertex *valency* is the number of incident edges. An equilateral triangle has internal angles equals to sixty degrees. In a scene of ideal valency for the purposes of this work, each vertex should have a valency of $\frac{360^o}{60^o} = 6$. In this work, a vertex with valency six is called a regular vertex and all other vertex valencies are called irregular vertex (Oliveira et al, 2012).

In this work, all input meshes will be implicitly orientable and represented by a triangular mesh.

2.4 Stellar Operations

Stellar Operations are operations that preserve the Euler characteristic of the mesh (Mario et al, 2010a), they are based on division, remotion or changing edges from the model however maintaining the sum vertices - edges + faces of the surface as showed in Equation 1.1. Consider the vertices V_1, V_2, V_3 and V_4 forming two triangles $f_1 = (V_1, V_2, V_3)$ and $f_2 = (V_1, V_3, V_4)$. The vertices adjacent to the edge $A_i = \overline{V_1 V_3}$ are V_1 and V_3, the opposite

vertices to A_i are V_2 and V_4. The set of vertices from the triangles adjacent to A_i is $\{V_1, V_2, V_3, V_4\}$

2.4.1 Edge flip

This operation flips an edge as depicted in Figure 2.2.

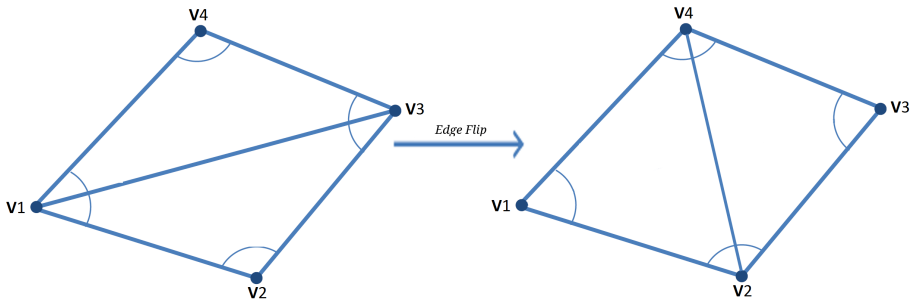

Figura 2.2: Edge Flip (Oliveira et al, 2012).

In this operation the edge A_i is changed by a new edge linking the opposite vertices of A_i, the Euler characteristic is preserved.

2.4.2 Edge split

There are four vertices involved in a edge split. After the split, a new vertex is created over the split edge, two new faces will be added in the mesh. The valency of the opposite vertices will be increased by one as depicted in Figure 2.3.

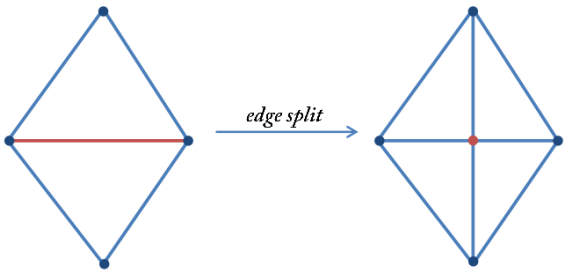

Figura 2.3: Edge Split (Oliveira et al, 2012).

2.4.3 Edge collapse

This operation removes an edge from the mesh as depicted in Figure 2.4. There are four

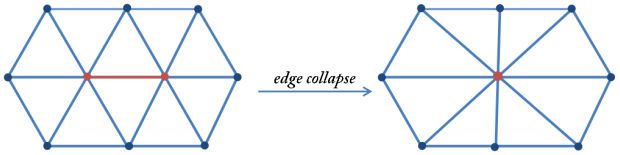

Figura 2.4: Edge collapse (Oliveira et al, 2012).

vertices involved in a collapse. An edge A_i is selected and removed, the faces containing the edge A_i are removed too. Given the vertices that form the edge $\overline{V_iV_j}$, only one is maintained and its valency va_i becomes $va_i + va_j - 4$.

2.5 Remesh

According to Mario et al (2010a), a remesh is defined as: Given a surface represented by a mesh $\mathcal{M}$, create a new mesh $\mathcal{M}'$, such that the new mesh satisfy a certain quality standard and $\mathcal{M}'$ is enough next of $\mathcal{M}$. So remesh is a very important technique when you a have a certain standard to be achieved by a surface. In this case, the goal is to have the edges length within a predefined interval. The definition of "enough next" is vague. It could be topological, geometric or defined by any constraint. In general, the topology must be preserved while other characteristics might be approximately the same as the original (Surazhsky et al, 2003).

2.6 Lowpass filtering

A lowpass filter removes high frequencies from the mesh, making it more smooth. There are several lowpass filters and, in this work, we use a Laplacian filter. More precisely, we use a modified Laplacian filter which uses more than one ring around the vertices (k-rings) and is constrained to be in a previously defined tangent plan.

The classic laplacian filter is defined as:

$$\nabla^2 f = \frac{\partial^2 f}{\partial^2 x_1} + ... + \frac{\partial^2 f}{\partial^2 x_n}.$$

It is a measure of the dispersion in $\mathbb{R}^n$ of a function f. Taubin et al (1995) propose a discrete approach to the Laplacian operator. The approach is:

$$L(V_i) = \sum_{V_j} w_{ij}(V_i - V_j), \tag{2.3}$$

with V_j in the neighborhood of V_i. In the literature, many weights were proposed for w_{ij}. There are schemes based on cotangent (Alliez et al, 2002) and neighborhood (Oliveira et al, 2012). The discrete Laplacian is largely used due to its simplicity. Basically, each vertex is moved to the median of its neighbors. This procedure tends to equalize edges lengths, minimizing the standard deviation. The Laplacian must be zero to achieve these properties and the system to solved is given by:

$$\sum_{v_j} w_{ij}(v_i - v_j) = 0.$$

In this work, we use an iterative approach that approximately gives the same results of the Laplacian filter.

3 Proposed method

The input for the algorithm that uniforms the edges is a tuple $(\mathcal{M}, e_{min}, e_{max}, k, n, pr, f)$, where $\mathcal{M}$ is the triangular mesh,e_{min} is the smallest edge length allowed , e_max is the biggest edge length allowed, k the number of rings used at the Laplacian optimization step, n the number of iterations, pr is the number of iterations before the original mesh is replaced by the current mesh in order to relax next projections, f is the maximum degradation for the triangles smallest angle involved in the flip. The loss of the original geometry is smaller as f is closer to zero.

Algoritmo 1: UniformRemeshing$(\mathcal{M}, e_{min}, e_{max}, k, n, pr, f)$

```
M' = Copy(M)
m = (e_min + e_max)/2
for i = 1 to n do
    if pr > 0 and (i mod pr) = 0 then
        M = Copy(M')
    end if
    m_i = MIN(2 · CalculateEdgesAverage(M'), m)
    L_p = CreatePriorityList(M', m_i, e_min, e_max)
    StellarOperations(M', L_p)
    CorrectValency(M')
    LowPassFiltering(M', k)
    Projection(M, M')
end for
return M'
```

The method works as shown in Algorithm 1. Each step is separately explained ahead.

3.1 Stellar Transformations with Priority List

The mesh must be transformed according to the target median $m = \frac{e_{min}+e_{max}}{2}$. Less faces and vertices are needed to represent the model as m is higher. For the opposite, a lower m results in more faces and vertices.

The current edge's length is the criterion to decide if vertices are added or re-

moved. An arbitrary input mesh, however, can have regions which should be refined and others that should be simplified.

This step is based on the modification of *long* or *short* edges A_j which are classified as:

$$\begin{aligned} \text{long}, &\quad \text{if } |A_j| > m_i + \sigma, \\ \text{short}, &\quad \text{if } |A_j| < m_i - \sigma, \end{aligned}$$

where $\sigma = \frac{e_{max} - e_{min}}{2}$ and m_i is an intermediate target value for the i-th iteration, defined for the transition between the iteration i and $i+1$. This intermediate target value m_i avoids sudden changes on the mesh geometry if m is too different than the original mesh edge length average.

Edges with lengths within the closed interval $[m_i - \sigma,\ m_i + \sigma]$ remain unchanged during this step. Long and short edges are then candidates to be collapsed or split, respectively. For every edge $A_i \in \mathcal{M}'$ considered long, a new vertex is inserted over the edge, dividing it into two smaller edges by the stellar operation of *edge split*. Edges considered too small in respect to m_i are removed from the model using *edge collapse*.

When m is much greater than the current edges average, a strong mesh simplification is required. In a extreme example with high m, all edges will be candidates to be collapsed. Thus, the iteration's target m_i is set to be the minimum between the global target average m and two times the current edges average of $\mathcal{M}'_i$ (Algorithm 1). In that way, the difference between the averages of $\mathcal{M}'_i$ and $\mathcal{M}'_{i+1}$ changes smoothly and is unlikely to generate invalid triangles or change the topology of the mesh, due to the collapse of near long edges. The upper bound of two times the current average per iteration for m_i comes from the fact that the edges progressively tend to be equalized, and doubling the edges length of a star does not promote a severe local modification.

The order of application of stellar operations is important. We propose to process the longest and smallest edges first. A priority list L_p is created where the edges A_j, with higher deviation $d_j = ||A_j| - m_i|$, are the most important. The list L_p is the set of edges $\{A_1, ..., A_t\} \subset \mathcal{M}$, with $d_1 \geq d_2, ..., d_{t-1} \geq d_t$, where $A_j \in L_p$ if $|A_j| \notin [m_i - \sigma, m_i + \sigma] \in \mathbb{R}$ (Algorithm 2).

The stellar operations are performed after the set up of the list. The algorithm

Algoritmo 2: CreatePriorityList($\mathcal{M}', m_i, e_{min}, e_{max}$)

```
priorityList
σ = (e_max − e_min)/2
foreach A_i ∈ M' do
    if |A_i| > m_i + σ then
        priorityList.Add(A_i)
    else if |A_i| < m_i − σ then
        priorityList.Add(A_i)
    end if
end foreach
SortDescent(priorityList)
return priorityList
```

traverses the list verifying the type of each edge and applies the appropriate operation (*edge split* or *edge collapse*). All the vertices adjacent to an edge that was changed are marked as visited, if both vertices of the edge were visited than the edge is removed from the list L_p (Algorithm 3). This imposes a even more smoothly change between $\mathcal{M}'_i$ and $\mathcal{M}'_{i+1}$. Note that an edge can be in the list L_p and after an operation that affect one of its vertices not be *long* or *short* .

Algoritmo 3: StellarOperations($\mathcal{M}', L_p$)

```
foreach A_i ∈ L_p do
    if BothVertexVisited(A_i) then
        continue
    else if |A_i| > m_i + σ then
        EdgeSplit(A_i)
        VisitEdgeNeighbors(A_i)
    end if
    else if |A_i| < m_i − σ then
        EdgeCollapse(A_i)
        VisitEdgeNeighbors(A_i)
    end if
end foreach
```

In this work we developed a method to reposition the output vertices of edge split and edge collapse operations. It position the vertex over the original edge in a way that

minimize the equalization:

$$\sum_{V_j} ABS(|V_i - V_j| - m_i)^2, \tag{3.1}$$

where V_i is the vertex we want to position and V_j are the vertices connected to V_i.

3.2 Valency optimizer

After the edge adjustments by stellar operations, the valency of the vertices naturally tends to 6, which is mandatory to obtain uniform distribution in smooth regions. However, some vertices might have arbitrary valencies. A simple scheme to correct the valencies of the vertices based on *edge flip* operations is then applied (Mario et al, 2010a).

Consider the two triangles of Figure 2.2. For each edge of the mesh, the algorithm performs an *edge flip* operation and, if the valencies of the involved vertices become near to 6 and the angle $\min(\alpha_i)$ is greater than $f \cdot \min(\alpha_i)$ before the *edge flip* , the rotated edge is accepted, otherwise the operation is undone. An overview of the process is shown on Algorithm 4.

Algoritmo 4: CorrectValency($\mathcal{M}'$)

foreach $A_i \in \mathcal{M}'$ **do**
- $preMin_\alpha$ = `getMinalpha(`A_i`)`
- v_n = `SearchAdjacentVertices(`a_i`)`
- preDeviation = $|valency(V_1) - 6| + |valency(V_2) - 6| + |valency(V_3) - 6| + |valency(V_4) - 6|$
- `EdgeFlip(`A_i`)`
- $postMin_\alpha$ = `getMinalpha(`A_i`)`
- postDeviation = $|valency(V_1) - 6| + |valency(V_2) - 6| + |valency(V_3) - 6| + |valency(V_4) - 6|$
- **if** *(*postDeviation $\geq$ preDeviation*)* ***or*** *(*$postMin_\alpha \leq f \cdot preMin_\alpha$*)* **then**
  - `UndoEdgeFlip(`A_i`)`
- **end if**

end foreach

3.3 Lowpass Filtering

After the valency optimizer step, we proceed to the lowpass filtering. In this step, we use an iterative method to approach the constrained Laplacian. In the classical Laplacian filter, we add some additional constraints to reduce the geometry loss:

$$N_i \cdot T_i = 0, \ \forall T_i \in \mathcal{M}',$$

$$|T_i| = 0 \ \forall T_i \in \mathcal{B},$$

where N_i is the normal of the current mesh in the vertex V_i and T_i is the unknow displacement of the vertex V_i.

Algoritmo 5: LowPassFiltering($\mathcal{M}'$, k)

foreach $V_i \in \mathcal{M}'$ **do**
  $kStar$=`getKStar` (V_i,k)
  fat=0
  foreach $V_j \in kStar$ **do**
    V_i'+=$\frac{V_j}{star}$
    fat+=$\frac{1}{star}$
  end foreach
  V_i'=$\frac{V_i'}{fat}$
  T_i=V_i'-V_i
  T_i=T_i -`projection` (T_i,N_i)
end foreach
foreach $V_i \in \mathcal{M}'$ **do**
  if $V_i \notin \mathcal{B}$ **then**
    V_i+=T_i
  end if
end foreach

The iterative Algorithm 5 approximate the constrained Laplacian filtering described above. It calculates the new vertex position based on the k-neighborhood as proposed in Oliveira et al (2012). The first step is to compute for each vertex the new position without the application of the new constraints. This position is defined by the center of mass of all neighbors vertices weighted by their ring number in such a way that distant vertices contribute less than near vertices.

The second step is to impose the constraint $N_i \cdot T_i = 0$ by removing the vector

corresponding to the projection of T_i in N_i. When all displacements are computed, the vertices V_i are updated except those on the borders.

3.4 Projection

After the previous steps the new vertices tend to move away from the original mesh. As proposed in (Oliveira et al, 2013), this problem can be reduced by projecting the mesh computed in each iteration over the original mesh. However, the projection of a mesh over another is a complicate problem since the object can have arbitrary topology and geometry. Even if the mesh to be projected is close to the original, artifacts may appear due to wrong vertex correspondences. The algorithm works as follows: for each vertex we compute its nearest projection over some original triangles. These triangles are the incident over the 30 nearest vertices in the original mesh (Algorithm 6). To justify the 30 nearest vertices, we run several tests where using 30 nearest neighbors the resulting projection was exactly the same as if we project in all triangles. But this behaviour is not guaranteed for all cases because it is dependent on the geometry and topology of the mesh.

Algoritmo 6: Projection($\mathcal{M}$, $\mathcal{M}'$)

```
foreach V_i ∈ M' do
    Nearest=getNearest (V_i,M,30)
    foreach Triangle_j ∈ Nearest do
        P=projectionVT(V_i,Triangle_j)
        if |V_i − P| ≤ |V_i − MIN| then
            MIN=P
        end if
    end foreach
    V_i=MIN
end foreach
```

In our implementation of the Algorithm 6, we use a Kd-tree to increase the performance.

4 Results

In this section, the results of this work are shown. The method was implemented using C++ programming language and compiled using GCC 4.6.3. All tests were run in a Intel Xeon(R) CPU E31220 @ 3.10GHz x 4 computer that has 8 GBs of RAM. The graphic card was an AMD Radeon HD 5700 series.

4.1 Edge equalization by simplification

In this first scenario, we were testing the iterative method evolution in time (Table 4.1). The following tests were run over the fertility model (Figure 4.1). The parameters used in the tests were $n = 100, k = 2, e_{min} = 1.2, e_{min} = 1.8, pr = 0, f = \frac{1}{2}$ and the input mesh has an original edge length average of 0.75.

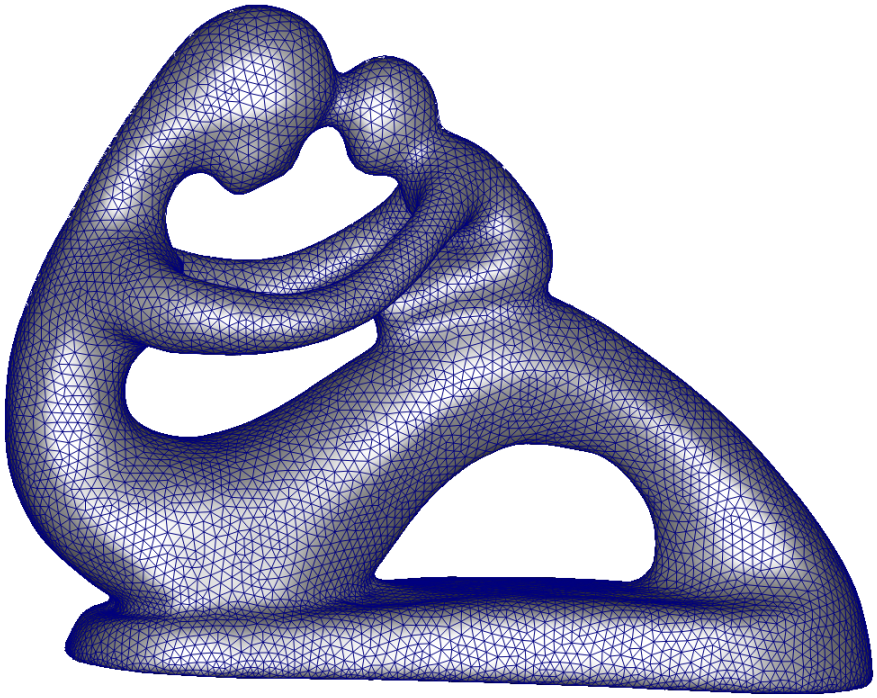

Figura 4.1: This is the input fertility model.

The table 4.1 columns are the iteration $Iteration$, the number of vertices $Vertices$,

Iteration	Vertices	Edges	Average	Deviation	Regular	Small	Big
1	9354	28080	0.934116	0.150386	72.011973	26702	3
6	3629	10905	1.488004	0.199443	73.215762	889	505
11	3492	10494	1.513421	0.187043	77.348225	623	477
16	3464	10410	1.518211	0.171593	79.907621	458	339
21	3425	10293	1.525191	0.163202	80.350365	369	304
26	3401	10221	1.529626	0.161322	81.770068	360	284
31	3400	10218	1.530237	0.165183	80.970588	377	291
36	3384	10170	1.532931	0.163540	82.092199	381	321
41	3405	10233	1.528674	0.159297	82.085169	362	274
46	3394	10200	1.530222	0.161740	81.644078	382	299
51	3382	10164	1.532950	0.157780	82.820816	340	277
56	3361	10101	1.536691	0.150204	84.141625	272	238
61	3380	10158	1.533097	0.151148	84.171598	300	228
66	3381	10161	1.532640	0.150045	85.241053	267	267
71	3382	10164	1.532510	0.151162	83.973980	300	217
76	3376	10146	1.533919	0.152444	84.360190	299	241
81	3371	10131	1.535301	0.148051	84.307327	270	240
86	3391	10191	1.530824	0.151205	84.370392	304	240
91	3416	10266	1.525567	0.159351	83.313817	393	253
96	3375	10143	1.534117	0.148669	85.214815	276	237

Tabela 4.1: This table shows the iterative method evolution in time. It clearly shows the refinement in the mesh, with the decreasing number of vertices.

number of *Edges*, the average edges length *Average*, the standard deviation of the average *Average*, the percent of regular vertices *Regular*, the number of edges that are smaller than the interval *Small* and the number of edges that are bigger than the interval *Big*.

As depicted in Figure 4.2 the method generates a almost uniform mesh without losing geometry accuracy. Even the trivalent mesh is close to the original geometry and almost uniform as depicted in Figure 4.3.

4.2 Edge equalization by refinement

As edge equalization by simplification we run tests for measuring the iterative method evolution in time (Table 4.2). To be more accuracy, the following results were running in the same fertility model used in edge equalization by simplification (Figure 4.1).The parameters used in the tests were $n = 100, k = 2, e_{min} = 1.2, e_{min} = 1.8, pr = 0, f = \frac{1}{2}$ and the input mesh has an original edge length average of 2.25.

As depicted in Figure 4.4 the method generates a almost uniform mesh without

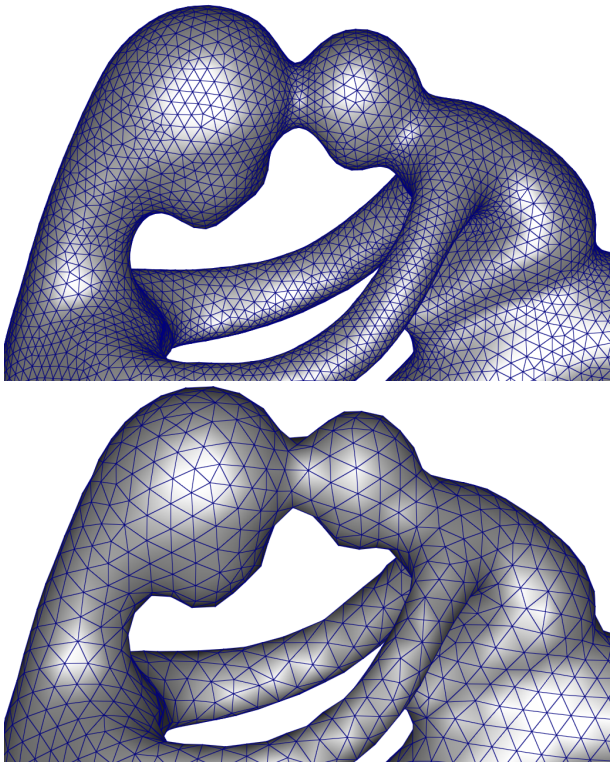

Figura 4.2: The first mesh is the input mesh, the second one is the mesh in the 100th iteration.

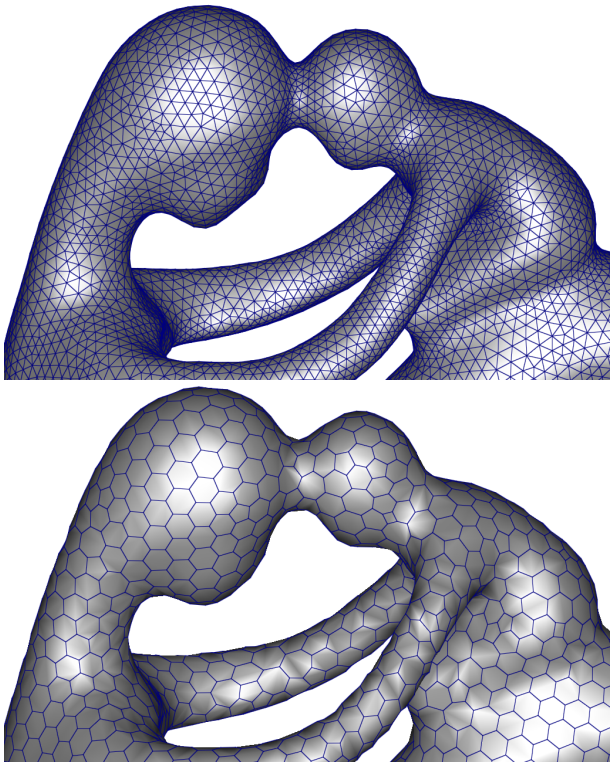

Figura 4.3: The first mesh is the input mesh, the second one is the hexagonal mesh in the 100th iteration.

losing geometry accuracy. As depicted in Figure 4.5 even the trivalent mesh is close to the original geometry and almost uniform.

Iteration	Vertices	Edges	Average	Deviation	Regular	Small	Big
1	17539	52635	2.045783	0.327601	67.478191	394	40852
6	32826	98496	1.492026	0.195853	72.156218	7353	4301
11	31479	94455	1.518870	0.164277	78.843038	3833	2717
16	31058	93192	1.526876	0.147635	82.326615	2558	1725
21	30690	92088	1.534388	0.133350	84.874552	1731	1154
26	30491	91491	1.538608	0.124447	86.199206	1225	729
31	30374	91140	1.541098	0.119997	87.041549	991	584
36	30289	90885	1.542784	0.115151	87.817359	740	420
41	30236	90726	1.543908	0.113076	88.176346	634	357
46	30195	90603	1.544772	0.110984	88.491472	522	363
51	30192	90594	1.544682	0.109332	88.665872	491	232
56	30124	90390	1.546264	0.107284	88.965609	351	224
61	30129	90405	1.546056	0.106334	89.146669	326	183
66	30114	90360	1.546384	0.106494	89.144584	328	188
71	30080	90258	1.547069	0.104416	89.507979	238	125
76	30076	90246	1.547142	0.104279	89.559782	214	129
81	30085	90273	1.546890	0.103804	89.566229	208	103
86	30083	90267	1.546922	0.103874	89.572184	209	97
91	30068	90222	1.547238	0.103024	89.696688	159	102
96	30084	90270	1.546902	0.103884	89.612419	204	109

Tabela 4.2: This table shows the iterative method evolution in time. It clearly shows the refinement in the mesh, with the increasing number of vertices.

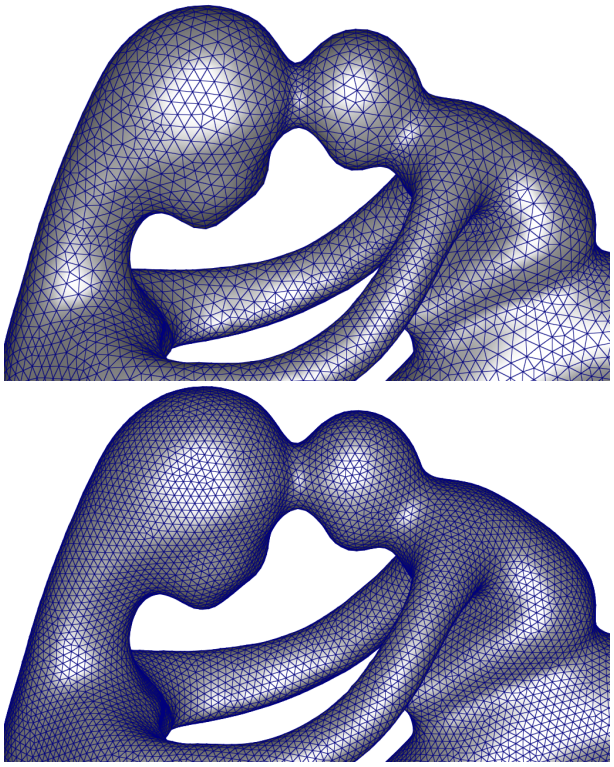

Figura 4.4: The first mesh is the input mesh, the second one is the mesh in the 100th iteration.

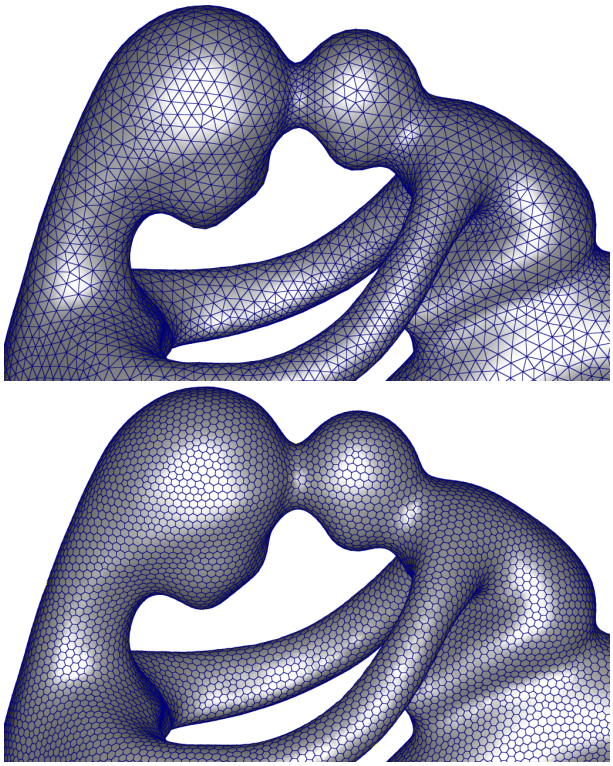

Figura 4.5: The first mesh is the input mesh, the second one is the hexagonal mesh in the 100th iteration.

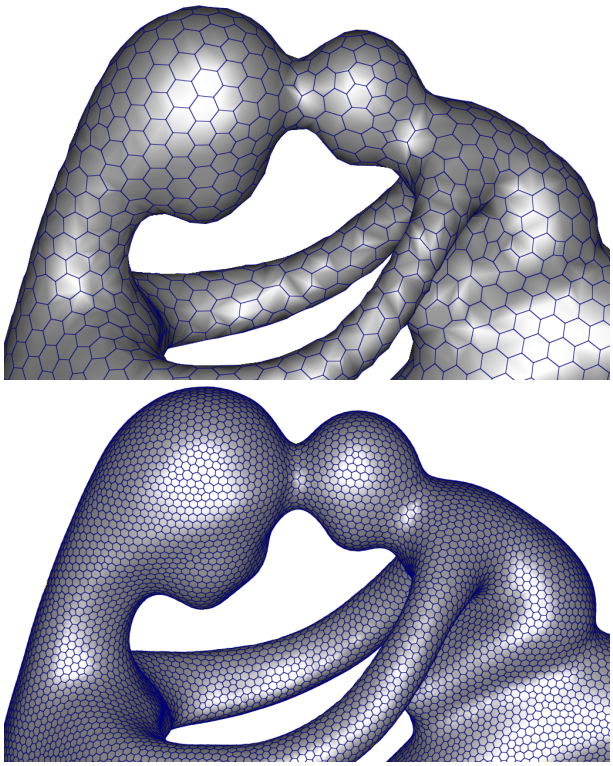

Figura 4.6: The first mesh is a simplified output mesh, the second one refined output mesh.

The edge equalization by refinement is better than by simplification because it preserve more the original geometry as depicted in Figure(4.6). Due this, the graphics

were based on edge equalization by refinement.

4.3 Per iteration graphs

The graphics in this section were generated using the following parameters $n = 100, k = 2, e_{min} = 1.2, e_{min} = 1.8, pr = 0, f = \frac{1}{2}$ and the input mesh has an original edge length average of 2.25.

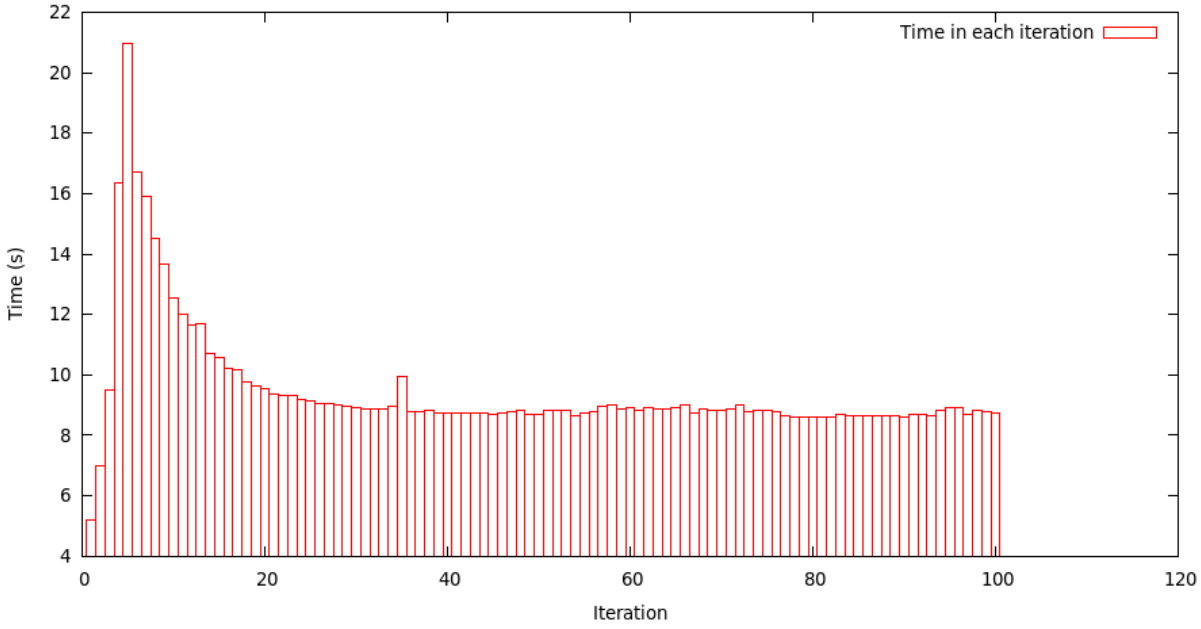

Figura 4.7: The boxes represent the time spent in each iteration.

As this is a refinement the number of edges and vertices increases over the early iterations, so the time spent increases as well. After a peak the time spent per iteration stabilizes around nine seconds as depicted in Figure 4.7.

The edge length average quickly converge to a value next to $m = \frac{e_{min}+e_{max}}{2}$ as depicted in Figure 4.8. The method changes the mesh so fast that the average oscillates around the expected average m before converging. Due to these drastic changes, the standard deviation decreases slowly to its minimum.

As depicted in Figure 4.9, the number o edges lengths out of interval decreases even more slowly than the standard deviation. this was expected as the imposition of a minimum and maximum value for edge lengths is much more restrictive since the majority of the edges cannot compensate the extra size of a few outliers.

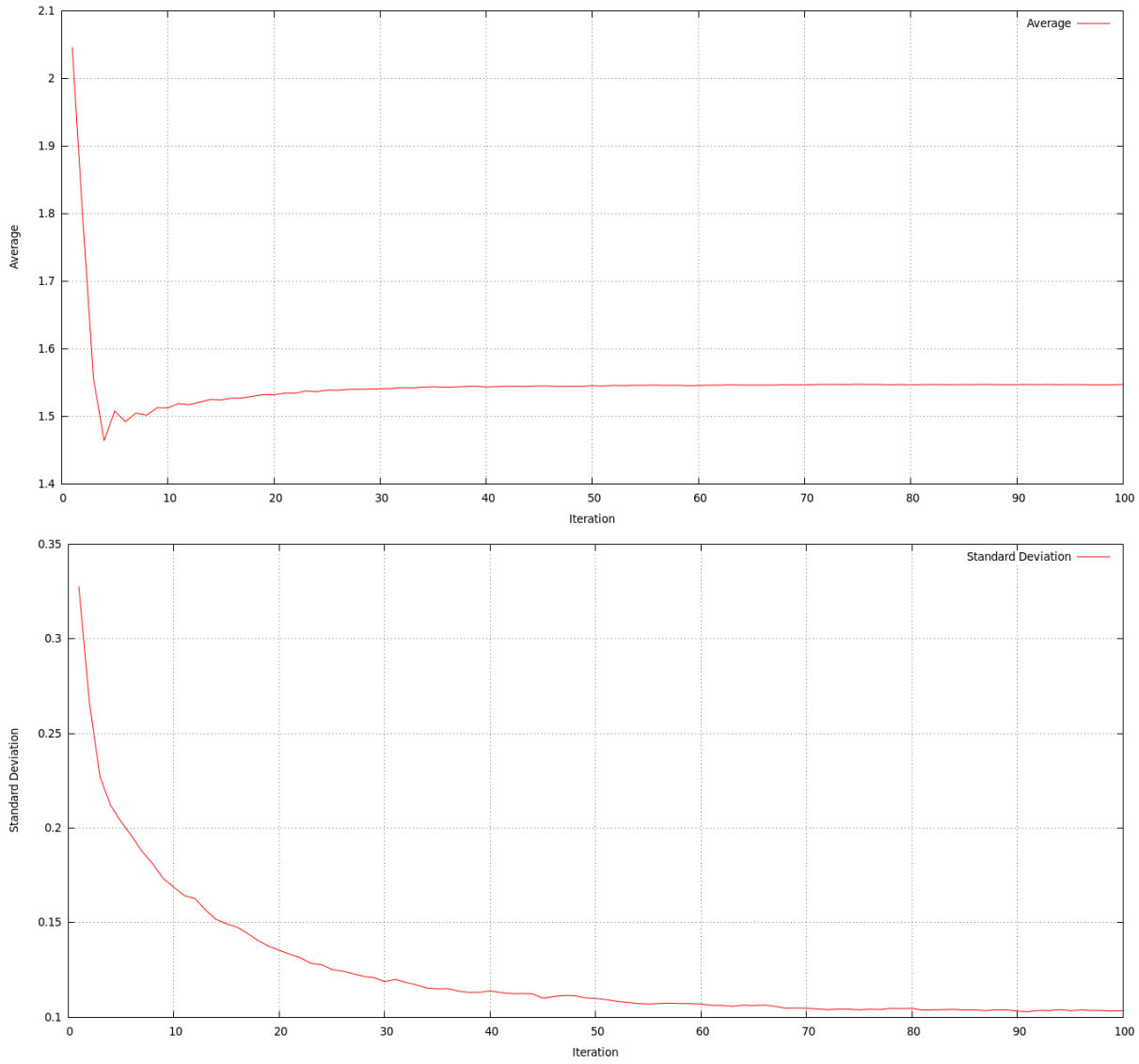

Figura 4.8: The first curve shows the edge length average in per iteration. The second curve is the standard deviation.

In Figure 4.10, we can see the method adapting the number of vertices in the model according to the constraints.

In Figure 4.11, we can see that the valency optimizer really makes a good work as the number of regular vertices slowly rises until it is more than 90%.

As depicted in Figure 4.12, the dual mesh is almost an uniform hexagonal mesh. There are almost no edges lengths out of the predefined interval. This is very important due the applications of this kind of mesh in physics.

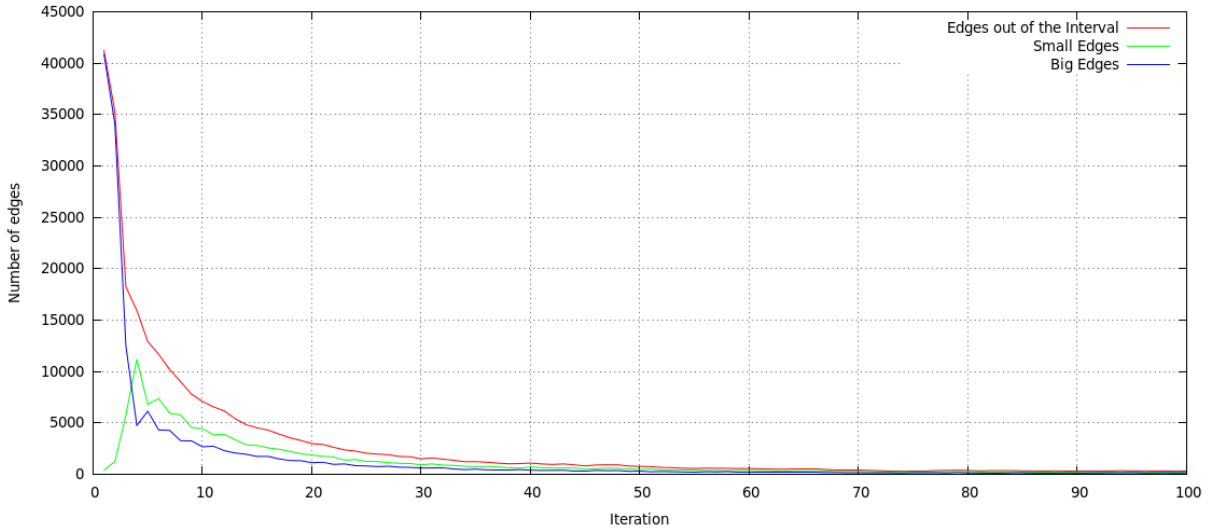

Figura 4.9: The red curve shows the number of edges lengths out of the interval per iteration. The green curve shows the number of small edges. The blue curve shows the number of big edges.

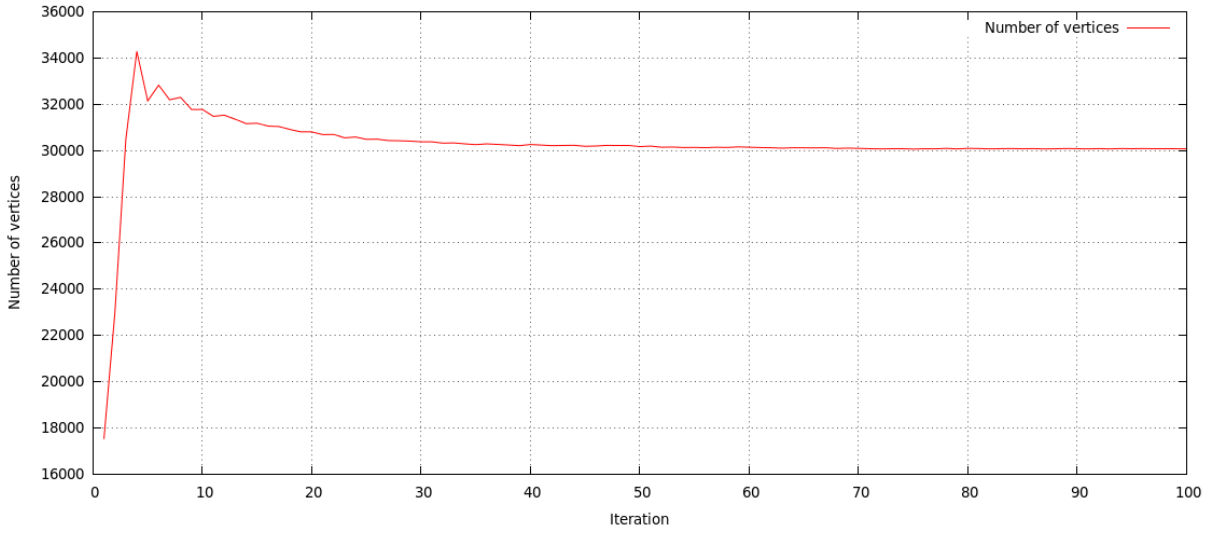

Figura 4.10: This shows the number of vertices per iteration.

4.4 Flaw case

This method can not do a strong simplification, as this will remove so much vertices, in such a way that it can not sample the entire surface. In the example (Figure 4.13) the average edges lengths were five times smaller than the average of the predefined interval.

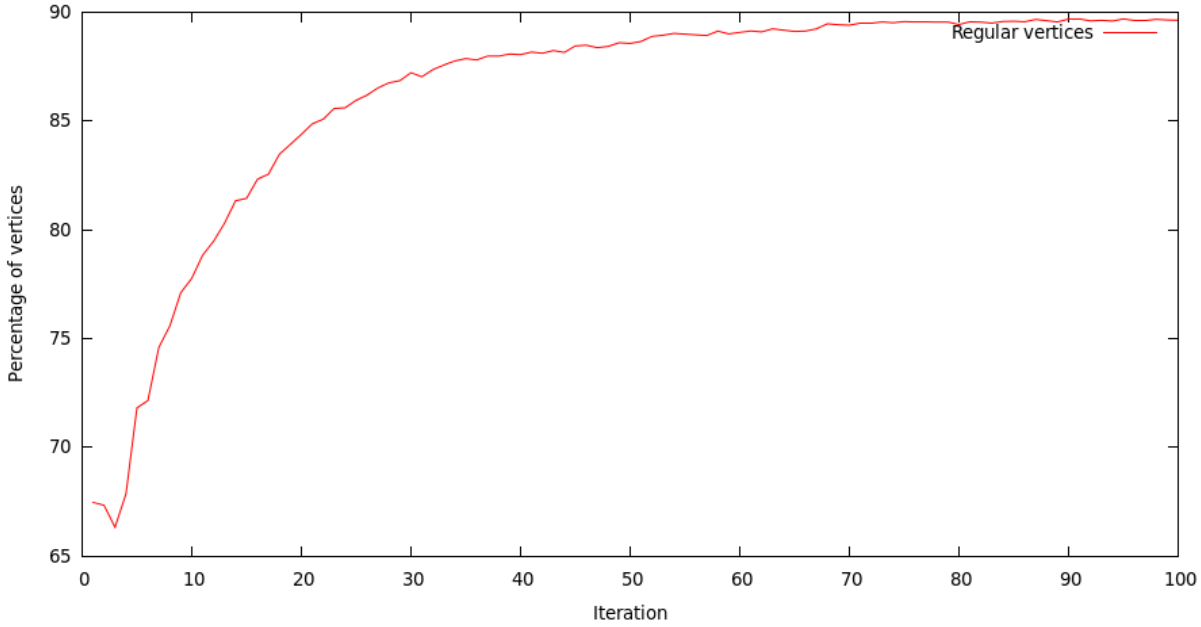

Figura 4.11: This shows the percent of regular vertices per iteration.

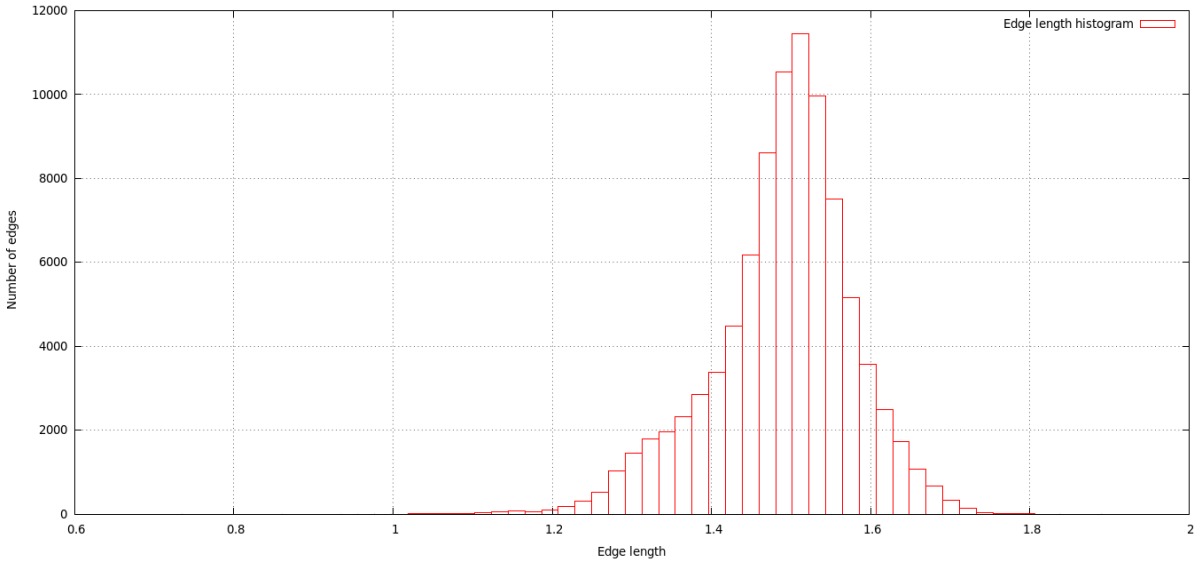

Figura 4.12: This is the histogram of the hexagonal mesh edges length at 100th iteration.

4.5 Examples

These are other examples of meshes processed using this method. The tests were run over the Rockarm (Figure 4.14) and Bunny (Figures 4.17,4.15,4.18) models with the same parameters $n = 50, k = 2, e_{min} = 1.2, e_{min} = 1.8, pr = 0, f = \frac{1}{2}$ and the input mesh has an original edge length average of 2.25.

The method is robust and can be applied even on larges meshes with thousands of vertices. The Rockarm model (Figure 4.14) has 20776 vertices and after the process

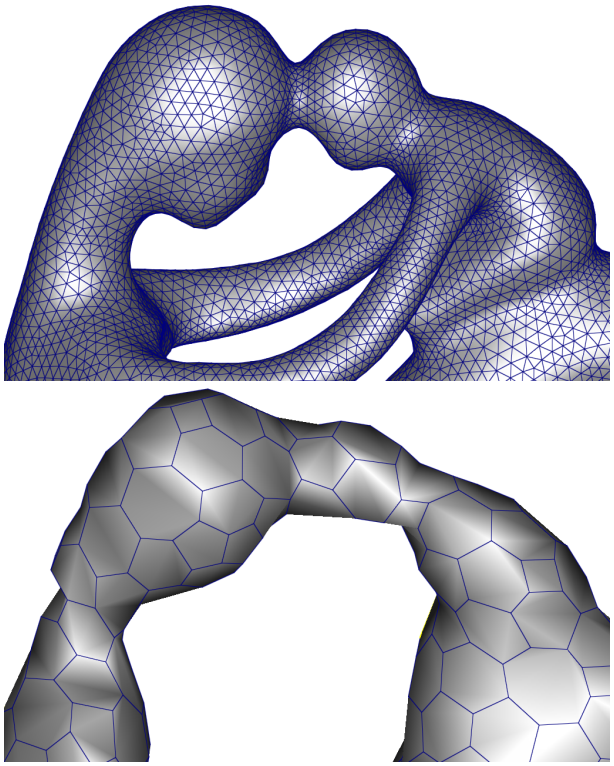

Figura 4.13: This is a flaw case of the method. The output mesh on the bottom loses almost the entire original mesh geometry.

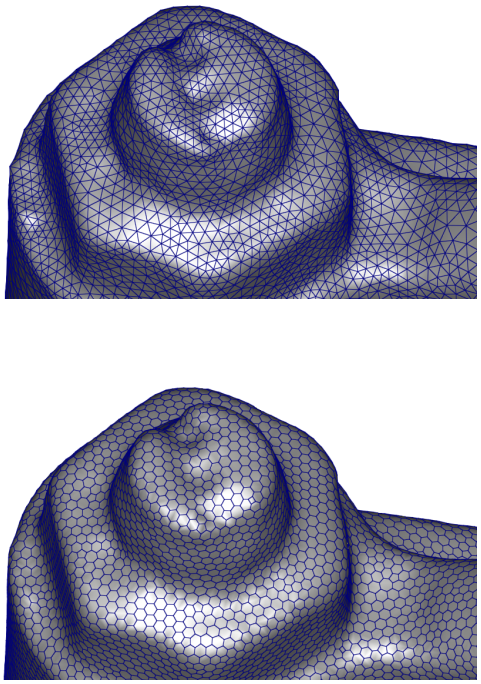

Figura 4.14: On the top is the input mesh, on the bottom the mesh after 50 iterations.

the output hexagonal model has 95020 vertices.

The method is powerful enough to handle extremely irregular meshes, as in the

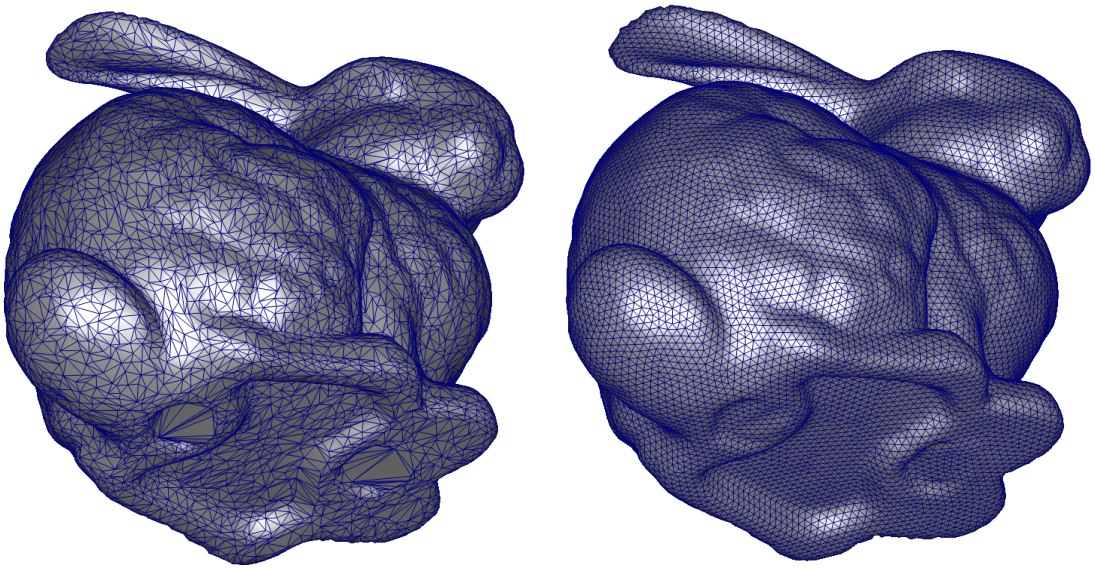

Figura 4.15: On the top is the input mesh, on the bottom the mesh after 50 iterations.

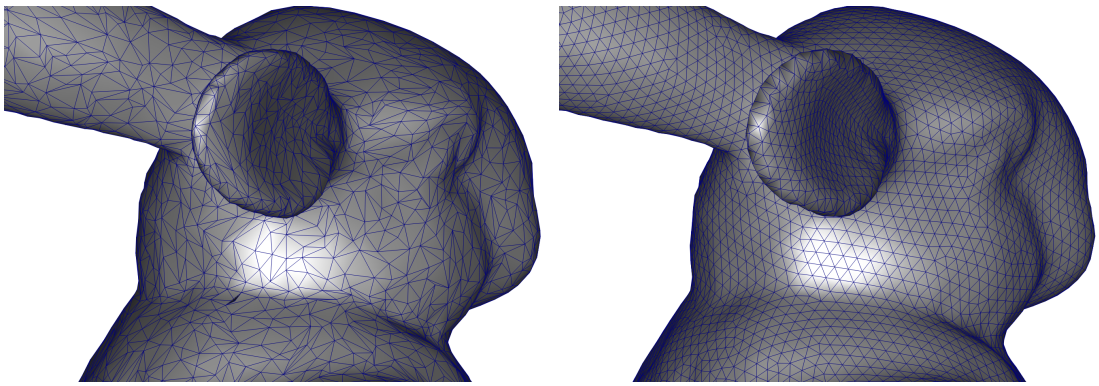

Figura 4.16: The same model depicted in Figure 4.15, with the ear is in focus.

bunny model depicted in Figures 4.15, 4.16, 4.17 and 4.18, this model has higher frequency regions as depicted in Figure 4.16 and vertices with extremely high valencies and edges with lengths much bigger or much smaller than the average as depicted in Figure 4.17.

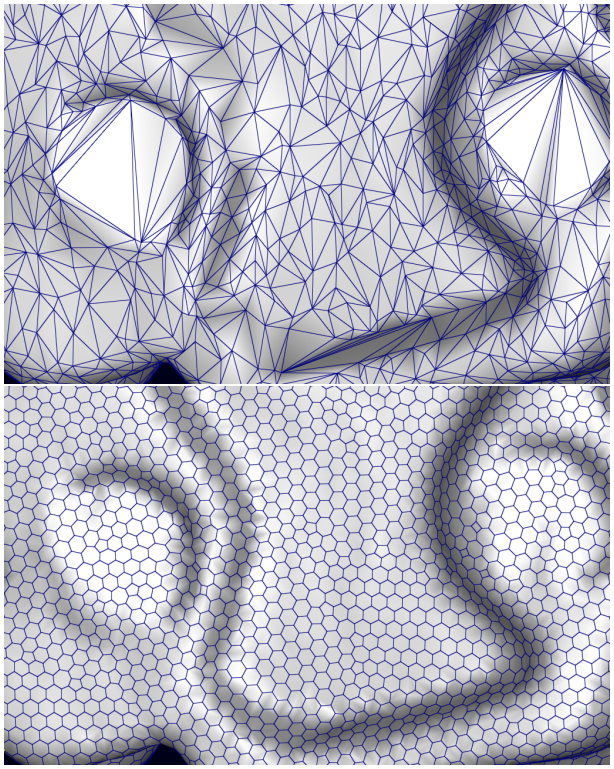

Figura 4.17: On the top is the input mesh, on the bottom the mesh after 50 iterations.

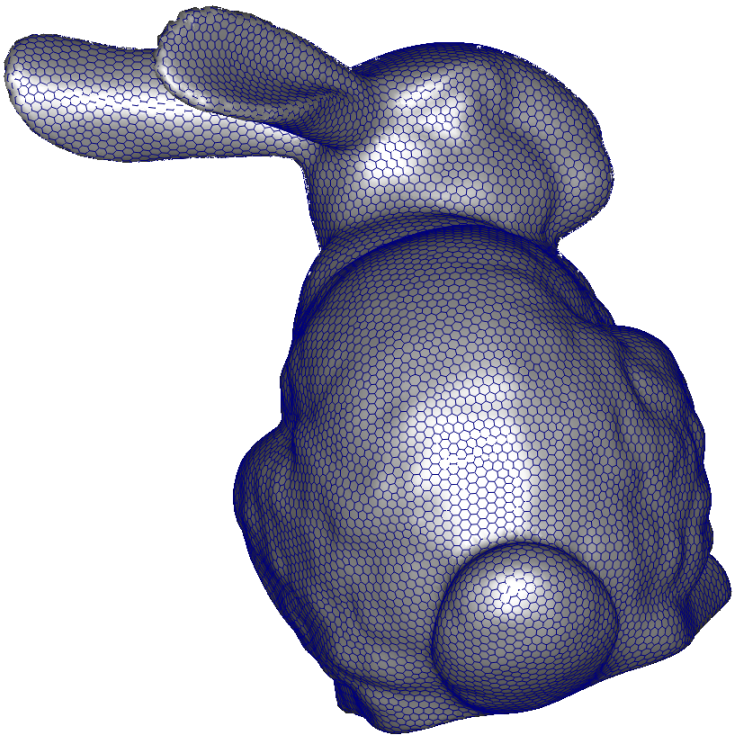

Figura 4.18: This is the trivalent bunny after 50 iterations.

5 Conclusion

This work proposes a low memory cost edge length equalization method. The edge length is an important mesh quality for physics simulations. Due to this, a method capable of edge equalization may be very useful as there are many publicly models that could be used in simulations after processed.

The method will not converge if the predefined interval is too tide. In our tests a good interval have to be enough spaced in such a way that the stellar operations will be applied rarely in the later iterations. This is often an interval obtained by fixing an average m , setting $e_{min} = m - 0.2m$ and setting $e_{min} = m + 0.2m$.

When the algorithm diverges, it is possible to see that the number of vertices in the priority list will continue high over the iterations and the projection will move vertices too far from their original positions. In that case setting a larger interval might reduce or solve this problem. Another problem is when the target interval forces the model to be extremely simplified.

Besides the e_{min} and e_{max}, the method is very sensitive to some parameters. The k rings can be fixed at $k = 2$, as our tests suggests that higher k does not improve very much the algorithm but increases the computational effort.

The frequency of projection per iteration pr is very dependent of the desired objective because it allows a greater geometry loss, but helps the method to converge in a more tide interval.

The maximum angle for flipping operations f does not affect the convergence very much, and a value of $f = 0.5$ gives good results in most cases. This prevents a great geometry loss but allows the algorithm to flip edges for improving the vertex valency in a balanced way.

5.1 Future works

This method needs a post process to remove few edges that are not within the predefined interval. Another thing to do is the implementation of another lowpass filter, for comparison with the actual Laplacian filter or even design a combined filter.

Referências Bibliográficas

Alliez, P., M. M. D. M. **Interactive geometry remeshing**. In: ACM Trans. Graph, volume 21, p. 347–354, 2002.

Guillemin, V.; Pollack, A. **Differential topology**. In: Differential Topology. Englewood Cliffs, New York, NY, 1974.

M. Nieser, J. Palacios, K. P. E. Z. **Hexagonal global parameterization of arbitrary surfaces**. In: ACM SIGGRAPH ASIA 2010 Sketches, p. 5:1–5:2, New York, NY, USA, 2010.

Mario Botsch, Leif Kobbelt, M. P. P. A. B. L. **Polygon mesh processing**. In: Polygon Mesh Processing. A K Peters, Ltd, Natick, Massachusetts, 2010.

de Oliveira, J. P. P. N. **Iterative method for generation of triangular meshes with uniform distribution**. 2012. M.sc. dissertation - Universidade Federal de Juiz de Fora, Advisor: Marcelo Bernardes Vieira, Co-advisors: Marcelo Lobosco and Sócrates de Oliveira Dantas.

de Oliveira, J. P. P. N. **Iterative method for edge length equalization**. In: International Conference on Computational Science, p. 481–490, Barcelona,Spain, 2013.

Thiago de Oliveira Quinelato, Patrícia Cordeiro Pereira Pampanelli, P. H. F. T. L. R. S. M. B. V. M. L. S. d. O. D. V. R. C. **Generation of nanotubes with minimal energy using rebo2**. In: ACM SIGGRAPH ASIA 2010, p. 5:1–5:2, New York, NY, USA, 2010.

Spivak, M. **A comprehensive introduction to differential geometry**. In: A Comprehensive Introduction to Differential Geometry. Publish or Perish, Inc., Houston, Texas, 1999.

Starostin, V. D. H. **The shape of a möbius strip**. In: Nature Materials, volume 6, p. 563 – 567, New York, NY, USA, 2007.

V. Surazhsky, C. G. **Explicit surface remeshing**. In: Proceedings of Eurographics Symposium on Geometry Processing, p. 17–28, Aachen, Germany, 2003.

Taubin, G. **A signal processing approach to fair surface design**. In: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques,SIGGRAPH '95, p. 351–358, New York, NY, USA, 1995.