Journal of Mobile Multimedia, Vol. 11, No. 1&2 (2015) 075–089

ADAPTIVE REMESHING FOR EDGE LENGTH INTERVAL CONSTRAINING

JOÃO VITOR DE SÁ HAUCK
Departamento de Ciência da Computação, Universidade Federal de Juiz de Fora
Juiz de Fora, Minas Gerais 36036-900, Brazil
jhauck@ice.ufjf.br

RAMON NOGUEIRA DA SILVA
Departamento de Ciência da Computação, Universidade Federal de Juiz de Fora
Juiz de Fora, Minas Gerais 36036-900, Brazil
ramon.nogueira@ice.ufjf.br

MARCELO BERNARDES VIEIRA
Departamento de Ciência da Computação, Universidade Federal de Juiz de Fora
Juiz de Fora, Minas Gerais 36036-900, Brazil
marcelo.bernardes@ice.ufjf.br

RODRIGO LUIS DE SOUZA DA SILVA
Departamento de Ciência da Computação, Universidade Federal de Juiz de Fora
Juiz de Fora, Minas Gerais 36036-900, Brazil
rodrigoluis@ice.ufjf.br

This paper presents a method for explicitly remesh an arbitrary input surface into a mesh with all edge lengths within a fixed interval. The process starts with an arbitrary triangular 2-manifold mesh. The proposed method is iterative and uses stellar operations to achieve the necessary amount of vertices and triangles. It also applies a technique to uniformly distribute the vertices of the model over the surface. At earlier stages of the algorithm, this technique is an approximation of the Laplacian filter. In order to preserve the geometry of the model, some constraints are added to the filter. At later stages, we replace the global uniformization strategy with a nonlinear optimizer, that performs only locally. A projection step is also applied at each iteration, to prevent the geometric distortions caused by the method. We also apply a post processing step to correct the final edges, if the standard iterations do not converge. Our method results in a very regular mesh, with uniform distribution of vertices. The dual trivalent mesh obtained by this mesh can be useful for several applications. The main contribution of this work is a new approach for edge length equalization, with explicit constraints definition, higher computational performance and lower global geometry losses if compared to previous works.

Keywords: iterative remeshing, edge length equalization, interval constraining

1 Introduction

In the last few years, the support of computational tools has become mandatory for many applications. As the cost of generating real objects for experimenting elevates, it is essential to proper simulate this objects in virtual space. On that account, the growing need of geometric models conducted to the development of many technologies for mesh generation.

There are, for instance, computer vision algorithms with 3D scanners [1, 2] or direct modeling softwares [3, 4]. The ways the data are presented by these technologies, however, rarely satisfy the requirements of any specific application. Therefore, the improvement of the quality of geometric models has become a primal research area in computer graphics.

There are some literary efforts to define the precise criteria for determining the quality of a mesh. Bommes *et al.* [5] enumerate some quality aspects most commonly required. However, the precise quality criteria usually depends on the requirements of the applications in which the models will be used. For real time applications, for example, a simplification of the model is preferable, in order to achieve high performance. In physics and chemistry simulations [6, 7], some constraints may be necessary to guarantee the fidelity of the results, e.g. constraints on edge lengths, valid vertices valency, polygon area restrictions, overall distributions of vertices, etc.

This work is interested in the edge lengths of triangular 2-manifold meshes. Specifically, our goal is to impose a constraining interval for the lengths. To do so, we iteratively remesh the model until all the edge lengths satisfy the defined constraints. Since the average of the edge lengths in a region impact on the amount of the edges and faces found there, the method applies stellar operations to adjust the amount of edges locally. It also applies, on the earlier iterations, an approximation of the Laplacian filter to relax the mesh. This relaxation becomes more local in the later iterations, when the Laplacian is replaced by a nonlinear optimization. In order to prevent the natural shrink caused by these applications, among with other geometric losses, we perform a projection over the original surface at each iteration. Finally, after the execution of the iterations, if there are any *long* of *short* edges, it performs a post processing step that usually eliminates the remaining problems. Although the process was improved to maintain the original geometry of the model, if compared to previous works, some local geometric losses may still occur, specially in regions with high curvature.

Our results indicate that the method generates models which satisfy the input constraining for most cases. Furthermore, in the final mesh the standard deviation of edge lengths tends to be low. The resulting mesh can also be used to generate a very regular trivalent mesh, by computing its dual. This kind of mesh may be greatly useful for physics applications, such as nano carbon simulations, which firstly motivated this work.

2 Related works

Surfacing remeshing is a process often applied in order to make certain model to meet some requirement for a specific application. For instance, the work of Liu *et al.* [8] aims to achieve a self-supporting surface, i.e., a surface that stands in static equilibrium without external support. Although a regular triangulation is not the final objective of that work, it is a necessary step for reaching it. Consequently, the method uses a power diagram [9] for obtaining a regular triangulation.

Since physics and chemistry simulations usually require more regular models, there are many works focused on obtaining more regular meshes. N-Symetric fields [10] can be used for generating highly regular polygonal meshes, although its method is computationally expensive. The work of Bommes *et al.* [5] applies the method for remeshing an arbitrary triangular mesh into a high quality quadrangular mesh.

This approach is also used by the work of Huang *et al.* [11], in which the goal is to obtain a mesh where the angle between two arbitrary edges of a triangle is 60^o. Through a set of feature lines of the model, this method can compute the N-Symmetric fields. This feature lines can either be defined by the user or automatically estimated. However, that estimation has a high computational cost.

The work presented in Pampanelli *et al.* [12] propose a method to obtain a regular trivalent mesh. This work estimates a regular quadrangular mesh, as proposed by Bommes *et al.* [5], then computes the rhomboid mapping of that quadrangular mesh. The resulting model is a regular trivalent mesh. Although it is not the main goal of that work, a very regular triangular mesh could be obtained by computing the dual from the resulting trivalent mesh.

Following a global parametrization strategy, Pietroni *et al.* [13] achieves an almost isometric mesh. They manage to obtain highly regular triangular meshes, but have the disadvantage of being computationally expensive, since it requires a set of feature lines of the model to be computed. Also, they are not explicitly concerned with edge lengths.

The method proposed by Botsch and Kobbelt [14] presents a method based on local adjustments followed by global relaxations. However, the functionality of this work is only guaranteed if the desired edge length is close to the average edge length of the original model. The approach of our work overcomes that limitation, and also demonstrates that the global relaxations can be replaced by local relaxations at later iterations to achieve better results.

Surazhsky and Gotsman [15] proposed a remeshing method based on area equalization and angle smoothing. Their method aims a mesh that maximizes the minimum angle of the triangles. Consequently, their approach build a high regular surface with triangle areas almost uniform. They also propose a new method based on angles to obtain a smooth surface. However, this method does not remove elements and thence is not suitable for simplifications.

The first work concerning to the edge lengths is presented in [16], where the goal is to obtain an average edge length that is close to a user defined value. Although that work achieve low values of standard deviation, it cannot guarantee that a specific edge length does not surpass the bounds of a constrained interval.

Following that, the work presented in Hauck *et al.* [17] succeeds to achieve models with all the edge lengths constrained to an input interval. Nonetheless, that work presents some geometric losses because the final step forces all edge lengths into the constrained interval, sacrificing the geometry. In an effort to preserve more geometric details, our new method inserts the local relaxation proposed in [17] into the iterations cycle. This allows the other steps of the algorithm to correct the geometric distortions caused by the local relaxation, and leads to a better resulting mesh projected over the original surface.

3 Proposed method

This method is an extension of the work presented in [17]. It also aims a mesh without any *long* or *short* edges a_j, classified as:

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

where e_{min} is the shortest edge length allowed and e_{max} is the longest edge length allowed.

The input for this algorithm is a tuple $(\mathcal{M}, e_{min}, e_{max}, n, k, p, l)$, where $\mathcal{M}$ is the triangular mesh, n the maximum number of iterations allowed, k the number of rings used at the

Laplacian optimization step, p the number of iterations before the original mesh is replaced by the current mesh in order to relax next projections and l is a threshold percentage of long and short edges used for deciding how the method optimizes the vertices distribution (Laplacian filtering or nonlinear optimization). At the end of each iteration, we save the resulting mesh if it has a lower amount of *long* and *short* edges than the current saved mesh. The Algorithm 1 is an overview of the proposed method.

$\mathcal{M}' =$ `Copy(`$\mathcal{M}$`)`
$m = \frac{e_{min}+e_{max}}{2}$
while $(short + long) > 0$ *and* $iter < n$ **do**
  if $p > 0$ *and* $(i \bmod p) = 0$ **then**
    $\mathcal{M} =$ `Copy(`$\mathcal{M}'$`)`
  end
  `StellarOperations(`$\mathcal{M}'$`)`
  `CorrectValency(`$\mathcal{M}'$`)`
  if `CalcEdgesPercent()` $\leq l$ **then**
    `NonLinearOptimizer(`$\mathcal{M}'$`,` k`)`
  end
  else
    `LowPassFiltering(`$\mathcal{M}'$`,` k`)`
  end
  `Projection(`$\mathcal{M}$`,` $\mathcal{M}'$`)`
end
`PostProcess(`$\mathcal{M}'$`)`
return $\mathcal{M}'$

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

Detailed information about the *CorrectValency* and *Projection* procedures can be found in [16]. The other steps will be explained ahead.

3.1 Stellar operations

The average value of edge lengths of the model is directly related to the amount of edges. Therefore, the purpose of the first step of the algorithm is to adjust the amount of vertices, so that the average edge length becomes near to the average value m of the interval. Thus, we apply stellar operations on the edges whose lengths are much greater or smaller than m. However, when the current average edge length of the model is much smaller than m, the mesh may be radically simplified. This situation could lead to a degeneration of the mesh on the earlier iterations. In order to prevent it, we calculate intermediate values ei_{min} and ei_{max} that only allow smooth transformations. These values are defined as:

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

where m_i is the current average edge length of the model.

The order of appliance of the stellar operations impacts on the results. For the best convergence, and also to ensure the method stability, we create a priority list of edges, as presented in [16]. The edges are then processed in the order established by the list.

If the edge length is shorter than ei_{min}, it is collapsed. Otherwise, if the edge length is longer than ei_{max}, then it is split. This modifies the amount of edges locally, since in an

arbitrary mesh some regions need to be refined while others need to be simplified. Each time an edge is operated, its original vertices and the new vertice created are marked as processed. If an edge already has both of its vertices processed, it is not operated. It is also not operated when the edge is neither shorter or longer anymore. This may occur when the position of one of its vertices was modified by a previous stellar operation.

Originally, both the remaining vertex of the edge collapse operation and the new vertex of the edge split operation can be placed in an arbitrary position over the operated edge. Hence, in order to optimize the convergence of our algorithm we compute the position that minimizes the equation:

$$\sum_{V_j} (|V_i - V_j| - m)^2, \tag{1}$$

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

3.2 Low-pass filter

To achieve equalized edge lengths, we have to distribute the vertices uniformly over the surface. To do so, after the valency correction step, as described in [16], the method proceed to a technique for optimizing the vertices' distribution. However, the strategy used for this optimization varies for distinct stages of the algorithm. For the earlier stages, we make use of a global optimization strategy, adjusting the distribution all over the mesh at once. For the later stages, we make use of a local optimization strategy, in witch we optimize only a small area around an edge at each time. The decision criterion for which strategy will be applied is the current percentage of *long* and *short* edges of the model. While that percentage is greater than the input value l, the algorithm opts for the global strategy. When this condition is no longer true, the method can proceed to the local strategy.

The global optimization method is a low-pass filtering over the surface. In this work, we use a modified version of the Laplacian filter.

The classic Laplacian filter is defined as:

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

It is a measurement of the dispersion in R^n of a function f. Taubin *et al.* [18] propose a discrete approach to the Laplacian operator. The approach is:

$$L(V_i) = \sum_{V_j} w_{ij}(V_i - V_j), \tag{3}$$

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 [19] and neighborhood [16].

The discrete Laplacian is largely used due to its simplicity. Basically, its appliance moves each vertex to the average of its neighbors. This procedure tends to equalize edge lengths, minimizing the standard deviation. The Laplacian must be zero to achieve these properties and the system to be solved is given by:

$$\sum_{v_j} w_{ij}(v_i - v_j) = 0. \tag{4}$$

The technique employed in this work does not solve the system. Instead, we run an iterative approximation that gives us almost the same results, significantly reducing the memory cost.

In the classical Laplacian filter, we add some additional constraints to reduce the geometry loss:

$$N_i \cdot D_i = 0,\ \forall D_i \in \mathcal{M}',$$
$$|D_i| = 0\ \forall D_i \in \mathcal{B},$$

where N_i is the normal of the current mesh in the vertex V_i and D_i is the unknown displacement of the vertex V_i. These constraints were imposed in the approximation as mandatory.

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

Algorithm 2: LowPassFiltering($\mathcal{M}'$, k)

The iterative Algorithm 2 approximates the constrained Laplacian filtering described above. It calculates the new vertex position based on the k-neighborhood as proposed in [16]. 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 $N_i \cdot D_i = 0$ by removing the vector projection of D_i in N_i. When all displacements are computed, the vertices V_i are updated, except those on the borders.

3.3 Nonlinear Optimizer

At a given point of the execution of the algorithm, the improving of the quality of the mesh through the Laplacian operator is greatly reduced. At some regions, specially the ones with higher curvatures, the excess of refinement cannot be compensated by the global optimization. In those circumstances, we need to apply a more local optimization strategy, which firstly

corrects the most troubling edges, and then cascades the optimization to its neighbors. To do so, Hauck *et al.* [17] proposes an error measurement for a region around an edge:

$$\sum_{V_i} \sum_{V_j} (|V_i - V_j|^2 - m^2)^2, \tag{5}$$

where V_i are the vertices in the forth star of the edge and V_j the vertices in the first star of V_i.

The minimization of Equation 5 approximates the edge lengths to m. Nonetheless, since the movement of the vertices assume three degrees of freedom, it may greatly distort the local geometry. Thence, we restrict those movements to the tangent plane. For each vertex V_i, we obtain an orthonormal base with its normal vector. This local base is defined as $< N_i, T_{i1}, T_{i2} >$, where N_i is the normal direction, and T_{i1}, T_{i2} are the directions over the tangent plane. That base is used to impose a restriction to the Equation 5, making the final error measurement to minimize as:

$$\sum_{V_i} \sum_{V_j} (|V_i + \alpha_i \cdot T_{i1} + \beta_i \cdot T_{i2} - (V_j + \alpha_j \cdot T_{j1} + \beta_j \cdot T_{j2})|^2 - m^2)^2, \tag{6}$$

where α_i and β_i are the variables in the function and represent the displacement over the tangent plane. Both α_i and β_i are set zero when the index i does not exist.

In this work, we use a conjugate gradient method, as seen in [20], for minimizing Equation 6. Due to performance and numerical issues, we do not perform the minimization for all the vertices in the model at once. Instead, we process the model per edge.

Similarly to the appliance of the stellar operations, the results of this step can vary in accord to the order in which the regions are operated. Considering the computational cost, we wish to minimize the geometric losses with the fewest amount of operations. Therefore, we prioritize to apply the transformations firstly in the regions where it will impact in the greatest amount of *long* or *short* edges. To do so, we propose the creation of a new priority list of edges. For this list, the priority P_j assigned to each edge a_j is defined as $P_j = N_j^l + N_j^s$, where N_j^l is the amount of *long* edges in the neighborhood of a_j, and N_j^s is the amount of *short* edges in the neighborhood of a_j. The neighborhood is considered to be all the edges until the forth star of a_j. The list L_p^o is the set $\{a_1, ..., a_t\} \subset \mathcal{A} \subset \mathcal{M}$, with $P_1 \geq P_2 \geq ... \geq P_t$. Following that, we perform the minimization of Equation 6 for each edge on the list, in order.

For the work [17], this nonlinear optimizer was presented as a post-processing step. Further experiments evinced that this operation would give better results as a replacement to the Laplacian filter at the later iterations. It is possible to realize that this occurs because the Laplacian loses its effectiveness as the troubling areas become more local. However, the local optimization alone could not solve this concentrated problem at once. Allowing it to be part of the iterations cycle increases its efficiency by complementing the local uniformization with the other operations. This has allowed us to reduce the amount of *long* and *short* edges even further without abdicating the projection, which generates better results, with yet lower geometric deformations. In most cases, the resulting mesh is now fully projected over the original surface.

3.4 Post processing

If the limit of n iterations is reached, for difficult models it is possible that the best resulting mesh still contains some *long* or *short* edges. In the occurrence of such cases, we perform the minimization of Equation 5 without constraints, allowing three degrees of freedom to each vertex. Although this resort solved the problem for all experiments performed, it can present significant geometry losses.

4 Experimental results

In this section we discuss the generated results of the proposed method. The algorithm was implemented using C++ programming language and compiled using GCC 4.6.3. All tests were performed in a Intel Xeon(R) CPU E31220 @ 3.10GHz x 4 computer with 8 GBs of RAM. The graphics card was an AMD Radeon HD 5700 series.

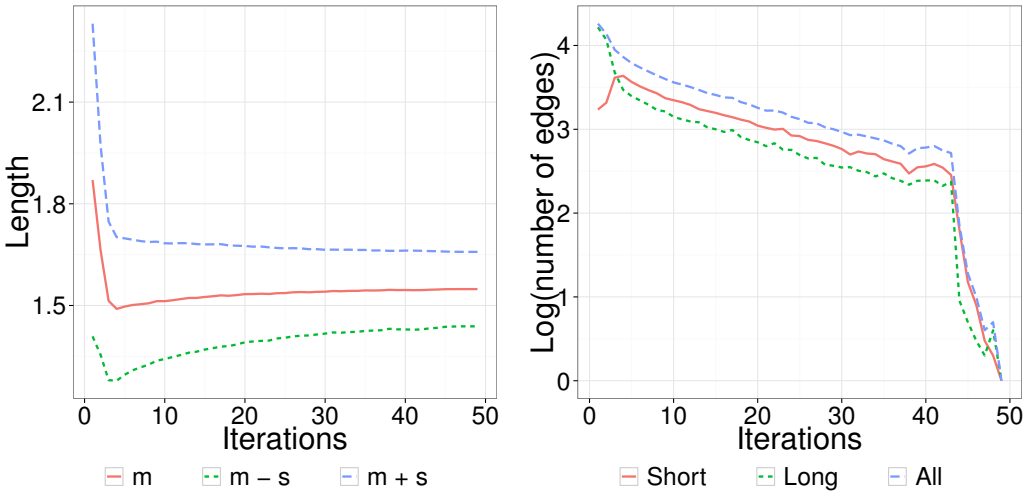

Fig. 1. Progression of Egea model through time

First we analyze the progression of the method over the time. Figure 1 exhibits the graphics for the progression of the *Egea* model, and Figure 2 shows the graphics for *Bunny* model. The graphics on the left illustrate the average edge length of the model through the iterations, and the deviation around it, where m is the average length and s is the standard deviation. The graphics on the right show the amount of *long* and *short* edges over the time. For visualization purposes, the y-axis is presented in a logarithmic scale with base ten. In both examples, we ran the method until there was no *long* or *short* edges.

The first thing noticeable is that the average edge length quickly converges to a value very close to the ideal average edge length. Furthermore, the amount of *long* and *short* edges for both models is reduced as the number of iterations increases. We can easily perceive an fast reduction after the point where the nonlinear optimizer starts. The final result for *Egea* mesh is depicted on Figure 4, and the Figure 3 illustrates the distribution of edge lengths for the final result.

We also depict the final results for the *Bunny* model in the Figure 5. This picture illustrates

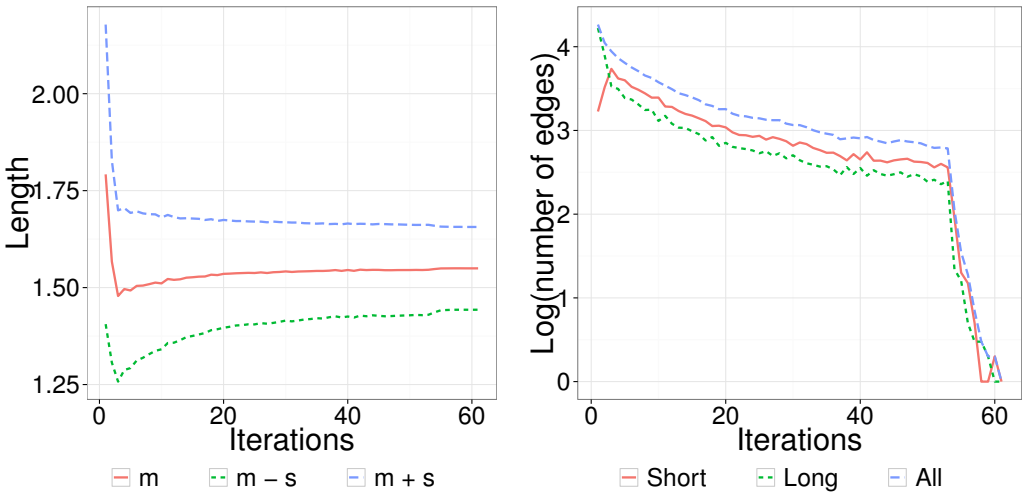

Fig. 2. Progression of Bunny model through time

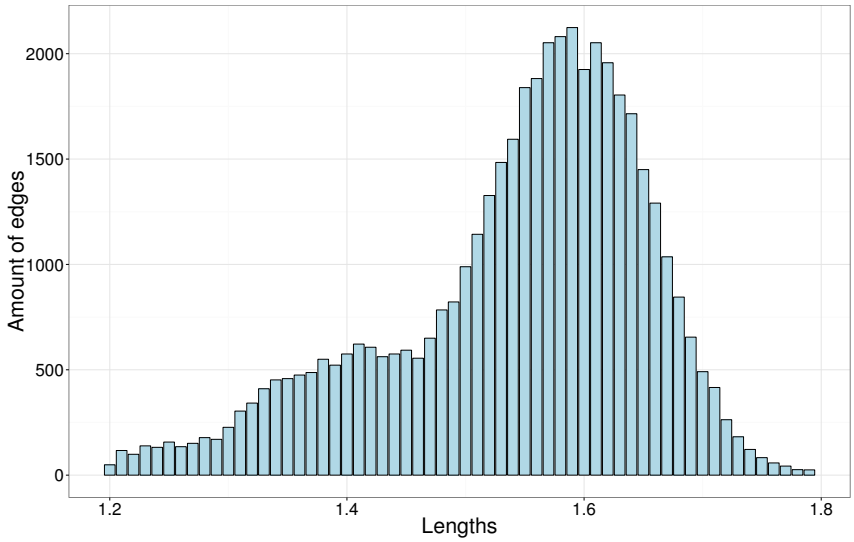

Fig. 3. Edge lengths for the processed *Egea* model, with $e_{min} = 1.2$ and $e_{max} = 1.8$.

the improvement of the method if compared to the one presented in [17]. The geometric distortions at the *Bunny* ear are greatly reduced in this new version. In fact, we achieve a resulting mesh with all the vertices projected over the original surface.

The experimental data from Egea model can be found in Table 1, where $\overline{x}$ is the average edge length, S the standard deviation, *It.* is the number of iteration needed to put all edges lengths within the given interval, and *Reg. Vert.* is the percentage of regular vertices with valency 6. The method performs better with $k = 2$, as the version presented in [17], so we ran all tests with this value. These experiments reveals the influence of the new parameter l over the method's convergence and the quality of the resulting model. For lower values of

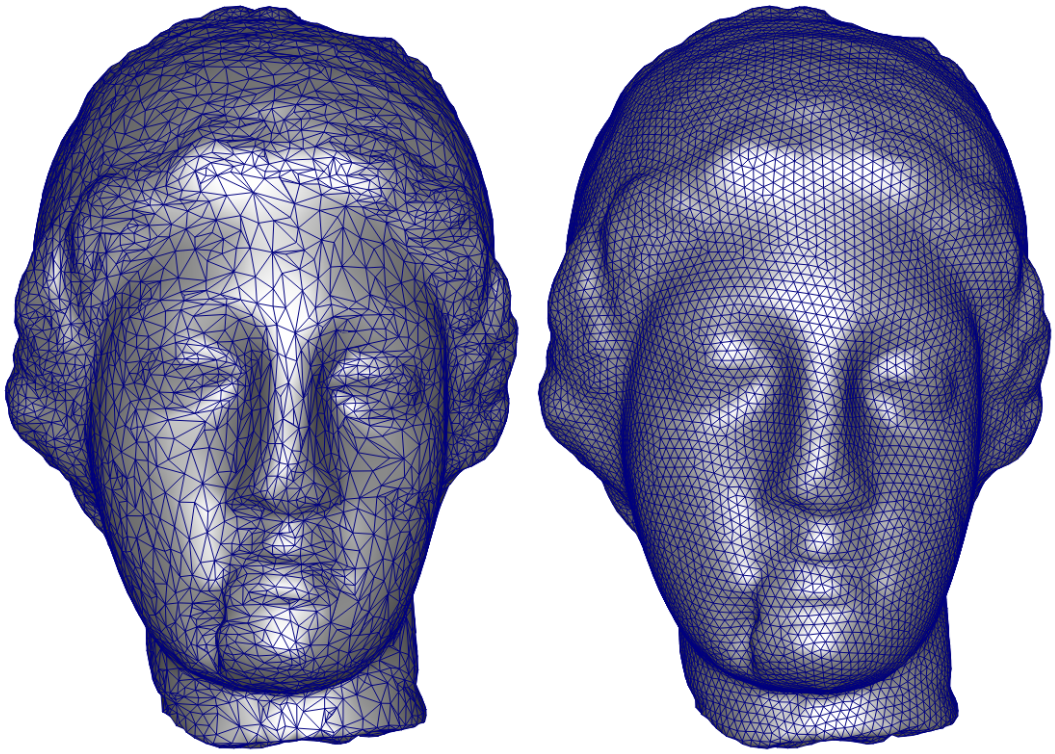

Fig. 4. *Egea* comparison between original mesh and processed mesh with parameters (1.2,1.8,100,2,0,2)

l the method takes more time before executing the nonlinear optimization. This leads to a slower convergence rate, even though the final mesh quality is usually better, presenting lower standard deviation and higher percentage of regular vertices. As we increase the value of l, the convergence tends to accelerate. However, there is a limit to this effect. If we increase l too much, the method's convergence starts to decrease. This is an expected result, since the local optimization is only effective for small regions. If there are still huge regions of *long* or *short* edges, the effectiveness of the local optimizer is greatly impaired.

The effect of l over the quality is much more direct: as l increases, the quality decreases. Again, this is an expected result. The nonlinear optimizer corrects small regions by ignoring the edges around it. This means that, unlike the global optimization, it does not aims a result that improves the quality of the mesh as a whole. As consequence, the quality of the mesh decreases. Yet, if we decreases l so much that the nonlinear optimization is never executed, the method will not converge and the post processing step will be applied, causing several geometric losses.

We can also notice that the parameter p is less effective than it is in [17]. However, it can present a greater role for the iterations performed with the nonlinear optimizer. As it smooths the original surface, it may reduce the probability of the method reaching a local minimum, like the one we see for the parameters (1.2,1.8,100,2,0,5). However, it is important

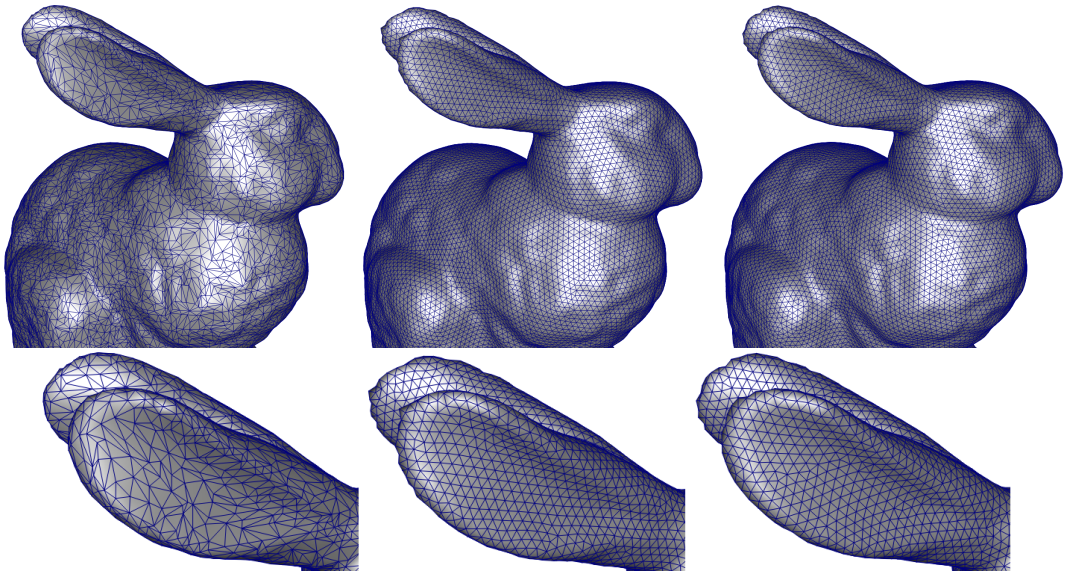

Fig. 5. *Bunny* Models. The first figure is the original mesh, the second one is the resulting mesh found in [17] and the last picture is our new result.

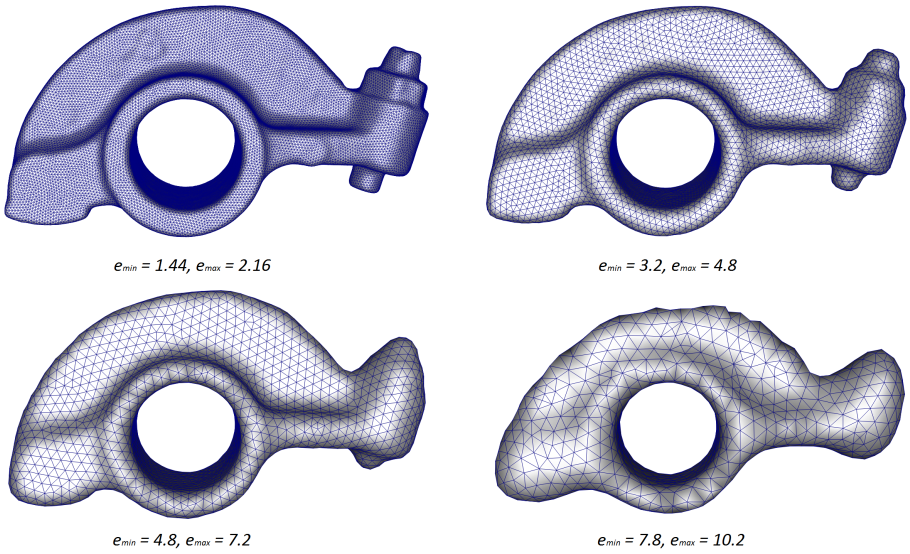

Fig. 6. Final result for *rockerarm* model with variant intervals for edge lengths. Each picture illustrates a specific length constraining.

to note that it can also slightly modify the geometry, as the projections are performed in a mesh gradually more distinct from the original, so the geometry distortions caused by each iteration becomes permanent.

Figure 6 presents the surface *rockerarm* with variant values for e_{min} and e_{max}. The results are shown from the state before the post processing step. The original surface is fairly

Table 1. *Egea* experiments with different k and p values

$(e_{min},e_{max},n,k,p,l)$	$\overline{x}$	S	It.	Reg. Vert. (%)	Time(s)
Model	2.25	0.702222	-	76.87	-
(1.2,1.8,100,2,0,2)	1.548329	0.109813	49	88.343476	165.046543
(1.2,1.8,100,2,0,5)	1.546942	0.115387	100	86.152724	399.453587
(1.2,1.8,100,2,0,8)	1.541189	0.131334	28	85.020750	204.223015
(1.2,1.8,100,2,0,10)	**1.534562**	0.137918	**21**	83.023665	193.354571
(1.2,1.8,100,2,10,2)	1.546141	0.110332	45	**88.394597**	184.454901
(1.2,1.8,100,2,10,5)	1.545709	0.122208	29	86.442711	174.859291
(1.2,1.8,100,2,10,8)	1.539932	0.134256	24	84.249951	194.527297
(1.2,1.8,100,2,10,10)	1.540058	0.140096	23	83.076721	220.757845
(1.2,1.8,100,2,25,2)	1.548902	**0.109361**	41	88.267669	157.827723
(1.2,1.8,100,2,25,5)	1.543660	0.120304	30	86.430659	**154.612424**
(1.2,1.8,100,2,25,8)	1.541183	0.131332	28	85.020750	207.849420
(1.2,1.8,100,2,25,10)	**1.534562**	0.137918	**21**	83.023665	192.297335
(1.2,1.8,100,2,50,2)	1.548329	0.109813	49	88.343476	161.671505
(1.2,1.8,100,2,50,5)	1.543919	0.121043	51	86.152724	201.896734
(1.2,1.8,100,2,50,8)	1.541189	0.131334	28	85.020750	204.984423
(1.2,1.8,100,2,50,10)	**1.534562**	0.137918	**21**	83.023665	193.273994

preserved when the method aims for a refined version of the model. Nonetheless, when the process seeks to perform a strong simplification, details of the original surface are naturally lost.

4.1 Applications

Regular meshes are very useful for computational simulations. For instance, in the simulations of nano-carbon structures the atoms and the bounds between them may be represented as a trivalent mesh. This kind of mesh can be obtained by triangular meshes. The dual mesh of a triangular mesh is a trivalent mesh. The ideal mesh for this kind of application is a hexagonal mesh. One may observe that if the primal mesh is not regular, there will be non hexagonal polygons on the dual. Thence, the most regular the primal mesh is, the best is its quality for computational simulations.

For this work, we present results for the dual mesh of the *rockerarm* model resulted from the appliance of our method. To analyze the quality of this model, we calculate the Lennard-Jones potential with $eps = 10.1$ and $\sigma = 0.9$. As depicted in Figure 7 the processed model does not have great energy variations. We can conclude that the resulting structure is much more stable than the original one for simulations. This result is very similar to the one obtained in the work presented in [17]. Nonetheless, the new result has a better preservation of the original *rockerarm* geometry, as the new resulting mesh is fully projected over the original surface.

Another application is the processing of a model so it becomes more stable for other numerical methods, e.g. finite elements. Due to the great regularity of the output mesh, several numerical issues are avoided.

5 Conclusion

This paper presents a method to remesh an arbitrary triangular 2-manifold mesh with all the edge lengths within an user defined interval. The main contribution of this work is a

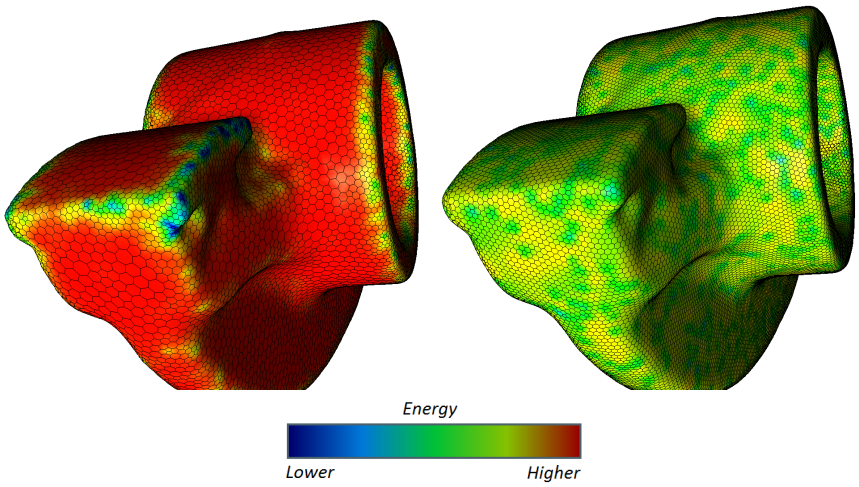

Fig. 7. The first picture is the original *rockerarm* model with potential from −33.4 to 1.21 and the second one is the processed model with potential from −17.59 to −3.08.

method that preserves more of the original geometry than the previous works. In addition, we achieve a method that has a well defined stop criterion. Consequently, we only have to limit the number of iterations n in order to prevent the method looping for the cases it does not converge. Moreover, the computational performance was also improved. Our tests showed that the new approach is faster than the previous work mainly because it converges in less iterations.

Our results also indicate that the method achieves the goal for a wide range of lengths, even without the need of proceeding to the post processing step. Furthermore, the resulting mesh fairly represent the original surface in most of cases, and can be useful for several applications.

The new parameter l greatly impacts on the results. For a lower l, the algorithm runs the global Laplacian filter over the mesh for a longer time, which leads to higher quality results, at the cost of more iterations. On the other hand, if its value is higher the method requires less iterations to converge, but the quality of the resulting mesh is reduced. If l is too low, the percentage is never reached by the global optimization and the algorithm does not converge. If l is too high, the method may not converge too because the nonlinear optimizer requires a good overall mesh quality to perform.

With regard to the parameter p, when its value is low, but not zero, the geometry is more likely to be softened, specially if many iterations are performed. Low values of p are useful for assuring the convergence, if the application allows the loss of some details of the original geometry.

For the minimum and maximum values allowed for edge lengths, as the values e_{min} and e_{max} are greater and the difference e_{min} - e_{max} is smaller, the final geometric losses are greater and the convergence of the method is slower.

One major problem faced by the method is to maintain the geometry when the model is being simplified. As a future work, a new approach that treats in a different way models that

are being simplified may be proposed.

Acknowledgements

Authors thank to FAPEMIG, CAPES and UFJF for financial support.

References

1. C. Rocchini, P. Cignoni, C. Montani, P. Pingi, and R. Scopigno. A low cost 3d scanner based on structured light. *Computer Graphics Forum*, 20(3):299–308, 2001.
2. Marcelo Bernardes Vieira, Luiz Velho, Asla Sa, and Paulo Cezar Carvalho. A camera-projector system for real-time 3d video. In *Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, pages 96–96. IEEE, 2005.
3. Renan Dembogurski, Bruno Dembogurski, RodrigoLuis Souza da Silva, and MarceloBernardes Vieira. Interactive mesh generation with local deformations in multiresolution. In Beniamino Murgante, Sanjay Misra, Maurizio Carlini, CarmeloM. Torre, Hong-Quang Nguyen, David Taniar, BernadyO. Apduhan, and Osvaldo Gervasi, editors, *Computational Science and Its Applications – ICCSA 2013*, volume 7971 of *Lecture Notes in Computer Science*, pages 646–661. Springer Berlin Heidelberg, 2013.
4. Joachim Schöberl. Netgen an advancing front 2d/3d-mesh generator based on abstract rules. *Computing and Visualization in Science*, 1(1):41–52, 1997.
5. David Bommes, Henrik Zimmer, and Leif Kobbelt. Mixed-integer quadrangulation. *ACM Trans. Graph.*, 28(3):77:1–77:10, July 2009.
6. Sumio Iijima et al. Helical microtubules of graphitic carbon. *nature*, 354(6348):56–58, 1991.
7. D. C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, New York, NY, USA, 1996.
8. Yang Liu, Hao Pan, John Snyder, Wenping Wang, and Baining Guo. Computing self-supporting surfaces by regular triangulation. *ACM Trans. Graph.*, 32(4):92:1–92:10, July 2013.
9. F Aurenhammer. Power diagrams: properties, algorithms and applications. *SIAM J. Comput.*, 16(1):78–96, February 1987.
10. Nicolas Ray, Bruno Vallet, Wan-Chiu Li, and Bruno Lévy. N-symmetry direction field design. In *ACM Transactions on Graphics*, 2008. Presented at SIGGRAPH.
11. Jin Huang, MuYang Zhang, WenJie Pei, Wei Hua, and HuJun Bao. Controllable highly regular triangulation. *Science China Information Sciences*, 54(6):1172–1183, 2011.
12. Patrícia Pereira Pampanelli, JP Peanha, Alessandra Matos Campos, Marcelo Bernardes Vieira, Marcelo Lobosco, and Sócrates de Oliveira Dantas. Rectangular hexagonal mesh generation for parametric modeling. In *Computer Graphics and Image Processing (SIBGRAPI), 2009 XXII Brazilian Symposium on*, pages 120–125. IEEE, 2009.
13. Nico Pietroni, Marco Tarini, and Paolo Cignoni. Almost isometric mesh parameterization through abstract domains. *Visualization and Computer Graphics, IEEE Transactions on*, 16(4):621–635, 2010.
14. Mario Botsch and Leif Kobbelt. A remeshing approach to multiresolution modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 185–192. ACM, 2004.
15. Vitaly Surazhsky and Craig Gotsman. High quality compatible triangulations. *Engineering with Computers*, 20(2):147–156, 2004.
16. João Paulo Peçanha Navarro de Oliveira. Iterative method for edge length equalization. In *International Conference on Computational Science*, pages 481–490, Barcelona,Spain, 2013.
17. João Vitor Hauck, Ramon Nogueira da Silva, Marcelo Bernardes Vieira, and Rodrigo Luis de Souza da Silva. Iterative remeshing for edge length interval constraining. In *Computational Science and Its Applications–ICCSA 2014*, pages 300–312. Springer, 2014.
18. Gabriel Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22nd*

annual conference on Computer graphics and interactive techniques, SIGGRAPH '95, pages 351–358, New York, NY, USA, 1995. ACM.

19. Pierre Alliez, Mark Meyer, and Mathieu Desbrun. Interactive geometry remeshing. *ACM Trans. Graph.*, 21(3):347–354, July 2002.
20. William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.