Error measurements and parameters choice in the GNG3D model for mesh simplification

RAFAEL ALVAREZ LEANDRO TORTOSA JOSE F. VICENT ANTONIO ZAMORA
Universidad de Alicante
Departmento de Ciencia de la Computacion e Inteligencia Artificial
Campus de Sant Vicent, Ap. 99, E-03080
SPAIN
tortosa@dccia.ua.es, jvicent@ dccia.ua.es, zamora@dccia.ua.es

Abstract: In this paper we present different error measurements with the aim to evaluate the quality of the approximations generated by the GNG3D model for mesh simplification. The first phase of this method consists on the execution of the GNG3D algorithm, described in the paper. The primary goal of this phase is to obtain a simplified set of vertices representing the best approximation of the original 3D object. In the reconstruction phase we use the information provided by the optimization algorithm to reconstruct the faces thus obtaining the optimized mesh. The implementation of three error functions, named $E_{avg}, E_{max}, E_{sur}$, allows us to control the error of the simplified model, as it is shown in the examples studied. Besides, from the error measurements implemented in the GNG3D model, it is established a procedure to determine the best values for the different parameters involved in the optimization algorithm. Some examples are shown in the experimental results.

Key–Words: Surface simplification, mesh reconstruction, error approximations, neural networks, growing neural gas, growing cell structures.

1 Introduction

Current computer graphic tools allow design and visualization of more and more realistic and precise 3D models. These models are numerical representations of both the real and imaginary worlds. Acquisition and design techniques of 3D models (modeler, scanner, sensor, etc) usually produce huge data sets containing geometrical and appearance attributes. These devices generate meshes of great complexity to represent the models.

Simplification is mandatory when one has to manage the meshes produced by 3D scanning devices. The sampling resolution of current scanning instruments produces surface meshes composed by 20M-100M faces. Meshes of this size usually have to be reduced to a more easily manageable size to be used in real applications. Therefore, the objective is clear: to produce a model that is visually similar to the original model and contains fewer polygons. Some techniques offer efficient processing but produce simplified meshes which are visually undesirable. Others create more pleasing approximations but are slow and difficult to implement.

The typical surface models handled by contemporary computer graphics applications have millions of triangles. Mesh simplification has emerged as a critical step for handling such huge meshes. The problem of approximating a given input mesh with a less complex but geometrically faithful representation is well-established in computer graphics. Level-of-detail representations figure prominently in real-time applications such as virtual reality, terrain modeling, and scientific visualization.

Over the last decades a tremendous amount of work has been done on mesh simplification. Most of the techniques or algorithms proposed to accomplish this objective are based on reducing the mesh complexity either by merging/collapsing elements or by re-sampling vertices. Simplification strategies may be broadly grouped into two categories: local strategies that iteratively simplify the mesh and global strategies that are applied to the input mesh as a whole.

Local strategies are the most common and some examples are the following ones.

- *Vertex Decimation*, first proposed by Schroeder et al. [20], operates on a single vertex by deleting that vertex and re-tessellating the resulting hole. The algorithm operates by making multiple passes over all the vertices of the model. A Bayesian technique for the reconstruction and subsequent decimation of 3D surface models from noisy sensor data can be seen in [6].

- Re-Tiling Polygonal Surfaces. The paper by

Turk [21] describes a method to simplify arbitrary polyhedral objects and works best on smoothly curved surfaces without sharpe edges or discontinuities.

- *Edge Contraction*, originally proposed by Hoppe et al. [13], is a common simplification operation. An edge contraction operates on a single edge and contracts it to a single vertex, updating all edges previously incident on it. Heckbert and Garland [11] showed that, under the L_2 metric, this strategy produces optimal triangulations in the limit as the number of triangles goes to infinity and their area goes to zero. Garland and Zhou [10] described a generalized version for high dimensions. See [1, 23] for recent papers in this context.

Some representative examples of global simplification strategies are

- *Shape Approximation*, proposed by Cohen-Steiner et al. [4]. They employ a variational partitioning scheme to segment the input mesh into a set of non-overlapping connected regions, and then fit a locally approximating plane to each one.

- *Vertex Clustering*, originally proposed by Rossignac and Borrel [18] to handle meshes of arbitrary topological structure.

In recent years, the problem of mesh simplification has received increasing attention. Several different algorithms have been formulated for simplifying meshes (see, for example, [2, 5, 22]).

In this paper, we define three error functions, named $E_{avg}, E_{max}, E_{sur}$ with the objective to evaluate the approximations generated by the GNG3D method. Such a way, we can control the quality of the simplified models we are generating. Moreover, two particular 3d models are used as examples, showing the numerical results obtained for the different error functions, applied to these models.

2 A brief description of the GNG3D algorithm

The GNG model presents some drawbacks itself when it is applied to the problem of mesh optimization for three-dimensional objects in computer graphics. The model only provides us information about the nodes or vertices and the edges connecting them; there is no information about the faces of the mesh. These drawbacks can be summarized in the existence of inactive nodes that produces the convergence of the algorithm towards a local minimum and the generation of foldovers in the new mesh. See [14] for a more detailed description of these problems.

Consequently, it is not possible to perform a suitable reconstruction of the original object without developing a further step of reconstruction, where the faces are reconstructed following a technique described later.

The method that we propose in this paper, denoted as Growing Neural Gas 3D (GNG3D) has been designed taking as a basis the GNG model, with an outstanding modification consisting on the possibility to remove some nodes or neurons that do not provide us relevant information about the original model. Besides, it has been added a reconstruction phase in order to construct the faces of the optimized mesh. Therefore, the GNG3D model consists on two different phases:

Phase 1 Mesh Optimization.

Phase 2 Mesh Reconstruction.

2.1 Phase 1. Mesh Optimization

The primary objective of this optimization phase is the calculation of the best vertices distribution that shapes the new simplified mesh. The core of this phase is the implementation of an optimization algorithm similar to the GNG algorithm described in Section 2. The details of this algorithm are described in the following:

Optimization algorithm

INIT: Start with two nodes a and b at random positions w_a and w_b in R^n. Initialize the error variable to zero.

1. Generate an input signal ξ according to $P(\xi)$.

2. Find the nearest node s_1 and the second nearest s_2 to the input signal.

3. Increment the age of all edges emanating from s_1. If the age of any edge is greater than a_{max}, then mark it in order to be eliminated afterwards.

4. Increment the local counter variable of the winner node. Add the square distance between the input signal and the nearest node in input space to a local counter variable:

$$\Delta error(s_1) = \|w_{s_1} - \xi\|^2$$

Store the nodes with the highest and lowest value of the local counter variable

5. Move s_1 and its direct topological neighbors towards ξ by fractions ϵ_b and ϵ_n, respectively, of the total distance:

$$\Delta w_{s_1} = \epsilon_b(\xi - w_{s_1}),$$

$$\Delta w_{s_n} = \epsilon_n(\xi - w_n),$$

where n represents all direct neighbors of s_1.

6. If s_1 and s_2 are connected by an edge, set the age of this edge to zero. If such an edge does not exist, create it.

7. Remove edges with an age larger than a_{max}. If this results in nodes having no emanating edges, remove them as well.

8. Decrease the error variables of all the nodes by multiplying with a constant d.

9. Repeat steps 1 to 8 λ times, with λ an integer.

   - If the maximum number of nodes has not been reached then insert a new node as follows:
     - Determine the node q with the maximum accumulated error.
     - Insert a new node r halfway between q and its neighbor f with the largest error variable:

$$w_r = 0.5(w_q + w_f).$$

     - Insert edges connecting the new node r with nodes q and f, and remove the original edge between q and f.
     - Decrease the error variables of q and f by multiplying them with a constant α. Initialize the error variable and the local counter of the node r with the new value of the error variable and local counter of q, respectively.
   - If the maximum number of nodes has been reached then remove a node as follows:
     - Set k the stored node with the lowest error variable.
     - Remove the node k and all the edges emanating from k.

10. If N is the total number of nodes, every $\mu \cdot N$ iterations of steps 1 to 8 remove all the nodes that have not been used (local counter equal to zero) and all the edges emanating from them. Reset the local counter of all the nodes to zero.

Some basic characteristics of this algorithm must be remarked.

- The accumulated error of nodes in step 4 is a quantity that allows us to determine the regions where there is a low density of nodes according to the vertices existing in the original 3D object. Regions where the accumulated error is high are suitable candidates for being covered with new nodes or neurons.
- The local counter variable is useful to eliminate nodes and avoid the problem of local minima.
- Parameters $\epsilon_b, \epsilon_n, d, \lambda, \alpha,$ and μ are not fixed. They are obtained experimentally.
- The parameter λ is just used to determine the moment to insert a new node in the mesh.
- The parameter μ is used to determine when to eliminate a node that has not been referenced in the previous iterations.

The GNG3D algorithm described above presents a very important difference respect to the GNG model. In this algorithm the nodes and the edges emanating from them that have not been referenced along the process of constructing the optimum neural network, are removed. This is carried out because of the introduction of a local counter variable in the step 4 of the algorithm. This local counter gives us the information about the number of times that a node has been referenced as the winner one in the process of determining the closest node to the input signal. The introduction of this counter for each node is related with the step 11 of the algorithm, where the nodes that have never been referenced as the winner ones are removed. The edges emanating from these nodes are also removed. In such a way, we avoid the foldover problems between some edges connecting nodes that appear in practice when executing the GNG model. This is solved because the nodes that generate this problem have in common that they are not referenced; so, they will be removed when the step 11 is performed. Consequently, these nodes do not appear in the final simplified mesh.

2.2 Phase 2. Reconstruction of the 3D object

In general, phase 1 can be seen as a training process based on neural networks. At the end of this process a set of nodes, which represent the new vertices of the optimized mesh, is computed. The edges connecting these nodes show the neighboring relations among the nodes generated by the optimization algorithm. This phase can be run as many times as we want with the aim to obtain the best configuration of the nodes in

the new mesh. Once this process has been completed, then it begins the second phase of reconstruction that is analyzed in the following.

The reconstruction phase constitutes a post-process which uses the information about new nodes provided by the optimization phase and the information about the nodes of the original model. With these sets of nodes, a concordance process can be carried on between the nodes of the original object and the nodes generated by the optimization algorithm. This concordance process allows us to reconstruct the faces of the new optimized mesh. This reconstruction phase can be summarized in three steps:

1. Associate a representative (node) to every node in the original mesh, making groups of nodes with the same representative.

2. Create a string with all the connections among the resulting groups.

3. Reconstruct the faces.

Step 1. Associate a representative (node) to every node in the original mesh, making groups of nodes with the same representative.

In this step, it must be calculated, for each node of the original mesh, which is the node of the optimization set that is closer to it. Suppose that $A = \{n_1, n_2, \ldots, n_N\}$ is the set of nodes (vertices) of the original object and $\kappa = \{k_1, k_2, \ldots, k_M\}$ is the set of nodes obtained by the optimization algorithm. Then, for each $n_i \in A$ we must find the representative of n_i, that is, the node $k_l \in \kappa$ which is closer to n_i. This task must be repeated for every node of the original object.

As the number of vertices or nodes in the original object is generally very high, in order to speed up this step we use an octree with the aim to divide the three-dimensional space in a balanced way and to set bounds to the searching space. An octree is a data structure to represent objects in the three-dimensional space, automatically grouping them hierarchically and avoiding the representation of empty portions of the space. The first octree node is the root cell, which is an array of eight contiguous elements. In our case, we have implemented an octree of one level, where each son keeps an array of vertices or nodes that are placed in its region.

The problem that can arise when working with an octree structure is that dividing the space into regions and searching only inside the region in which the original node is situated can give rise to an undesirable situation. That is, we can chance upon the possibility that the representative node obtained in this step is not the optimum. If it is not the optimum, at least will be close to it. The worst case occurs when all the nodes must be checked to find the optimum; in this situation no benefit will be obtained when using such structure. In the experiments performed with different models and different topologies, these cases only arise in the early iterations of the algorithm, when the number of nodes is small.

Step 2. Create a string with all the connections among the resulting groups.

Accordingly to the operations of labelling carried out in the step 1, the nodes of the original mesh have been arranged in groups and each one of these groups has associate one and only one node belonging to the group of nodes obtained after applying the optimization algorithm. Recall that the number of nodes in the simplified mesh is much smaller than the number of nodes of the original object. This association of nodes is performed minimizing the distance among the two sets of nodes (the original set of nodes and the set provided by the optimization algorithm). In this second step we continue analyzing each of the faces of the original mesh to check if their vertices have different representative. In other words, we are looking for triangles in the original mesh where the representative nodes of the vertices belong to different groups. When a triangle with this property is found, then it is necessary to store the connection among these groups for a further representation of these connections in the optimized mesh.

Step 3. Reconstruct the faces.

In this third step we proceed to create the faces of the optimized mesh. For this purpose, the key point is to scan the list of the representatives and when we find a connection among three neighboring groups, we conclude that this face must be represented.

3 The error measurement

We have developed in Section 2 an algorithm which produces simplified versions of any polygonal model. As we know, the goal of polygonal surface simplification is to take a polygonal model as input and generate an approximation of the original as output.

Therefore, an error measurement is required to evaluate the quality of approximations produced by the GNG3D algorithm. This error measurement will be completely dependent on the choice of error functions, so many such functions have been proposed in the last years. Ronfard and Rossignac [19] proposed an efficient measure of the error. Given a contraction $\{i, j\} \rightarrow \{h\}$ they define the local geometric error to be the maximum squared distance between vertex v_h and the planes defined by the triangles in $C(i) \cup C(j)$, where $C(s)$ are the cofaces of a simplex $s \in K$, with K a simplicial complex representing the connectivity

of the mesh. To avoid error propagation, each new vertex inherits the plane equations from the cofaces of the two merged vertices when a contraction is performed.

Garland and Heckbert [9] developed a surface simplification algorithm based on iterative contraction of vertex pairs to simplify models and maintains surface error approximations using quadric metrics. They observed that, given a simple plain (n, d) one can express the squared distance from the plane to a point x by

$$error(x) = x^T Ax + 2b^T x + c,$$

where $(A, b, c) = (nn^T, dn, d^2)$ is the fundamental quadric of the plane (n, d).

As explained in [9], the error of the approximation is typically measured with respect to L_2 or L_∞ error. The L_2 error between two n-vectors u and v is defined as $\|u - v\|_2 = \left[\sum_{i=1}^{n}(u_i - v_i)^2\right]^{1/2}$. The L_∞ error, also called the *maximum error*, is defined as $\|u - v\|_\infty = max_{i=1}^{n}|u_i - v_i|$. Optimization with respect to the L_2 and L_∞ metrics are called *least squares* and *minimax* optimization, and such solutions are called L_2*-optimal* and L_∞*-optimal*, respectively. Distances can be measured in various ways, e.g., to the closest point on a given polygon, or closest point on the entire surface. Others error measurements can be found in [3] and [12].

We have chosen two methods of error evaluation. For the first one, we use a metric which measures the squared distance between the approximation and the original model as described in [9]. We define the distance $d(v, A) = min_{p\in A}\|v - p\|$ as the minimum distance from v to the closest vertex p in the optimized mesh. This metric provides two error measurements which permits us to evaluate the approximations we are generating. These error measurements are:

- Mean error value of the minimum squared distance, given by

$$E_{avg} = \frac{1}{|M|}\sum_{v\in K} d^2(v, A).$$

- Maximum error value of the minimum squared distance, given by

$$E_{max} = max_{v\in K}\left\{d^2(v, A)\right\}.$$

Remember that K is the set of vertices of the original model, $|M|$ is the number of elements of K, and A is the set of vertices of the simplified object.

The second error measurement method computes the difference between the area comprised by the faces of the original object and the area corresponding to the faces of the simplified object [3]. Taking that the faces of the three-dimensional models considered here are triangular, this metric can be computed in the following way:

$$E_{sur} = S_K - S_A,$$

with

$$S_X = \frac{1}{2}\sum_{f\in X} v_a \cdot v_b \cdot sen\alpha = \frac{1}{2}\sum_{f\in X}|\vec{v}_a \otimes \vec{v}_b|,$$

with X being the set of faces of the original mesh and v_a, v_b the vectors joining the vertices belonging to face f.

The quality of the mesh being generated can be known at any time employing the metric of the distance to the vertices on any iteration during the training of the neural network in phase 1. The area difference metric can only be computed after applying phase 2 because there are no faces in the mesh being optimized during Phase 1.

Using the three error measurements exposed in this section, $E_{avg}, E_{max}, E_{sur}$, we have performed numerical experiments for a variety of models with different geometric characteristics.

We choose two representative examples of 3d objects: *gargoyle* and *horse*. *Gargoyle* is a 3d object composed by 21279 vertices and 40348 edges, while *horse* is a 3d object with 19852 vertices and 37540 edges.

For the *Gargoyle* we show the results in Table 1. We run the algorithm for this model and stop it for different iterations. For each iteration we show in columns two and three the characteristics of the reconstructed model, that is, the number of vertices and faces of the approximation generated by the GNG3D method. At the same time, we obtain from the program the values for $E_{avg}, E_{max}, E_{sur}$ for that particular iteration. Therefore, it is easy to compare the evolution of the error values for each approximation model when the number of iterations grows.

Remark that the GNG3D algorithm has been implemented with a particular characteristic: when half the number of vertices of the original object have been created the algorithm does not add any more vertex to the mesh. However, that does not mean that the training process is finished. We can follow running the optimization algorithm as time as we desire to produce the best mesh. Therefore, we can perform as many iterations as we desire, without taking the stopping criteria into account. In our case, the program is implemented to generate only half of the vertices of the original model.

The numerical results obtained for the horse 3d model are summarized in Table 2. We only show a

Table 1: Error values for the gargoyle 3d model.

Iterations	Vertices	Faces	E_{avg}	E_{max}	E_{sur}
42758	1118	1252	0.707	1.75	16.86
64137	1660	2458	0.324	1.13	9.25
106895	2868	4885	0.102	0.49	3.58
128274	3462	6128	0.068	0.21	3.45
171032	4722	8308	0.034	0.16	2.31
192411	5378	9613	0.025	0.12	2.09
235169	6584	12041	0.015	0.11	1.80
256548	7297	13137	0.011	0.07	1.96
299306	8545	15399	0.007	0.05	0.85
320685	9074	16601	0.006	0.04	1.44
363443	10499	18895	0.004	0.04	1.50
384822	10690	19952	0.003	0.04	1.20
427580	10401	20801	0.004	0.04	1.02
448959	10690	20875	0.003	0.04	1.25
491717	10690	21327	0.003	0.04	1.24
513096	10690	21408	0.002	0.03	1.12
555854	10690	21449	0.002	0.10	1.12
577233	10690	21506	0.002	0.10	0.97

Table 2: Error values for the horse 3d model.

Iterations	Vertices	Faces	E_{avg}	E_{max}	E_{sur}
39702	1226	1239	0.000100	0.000054	0.000241
59553	1848	2498	0.000044	0.000031	0.000150
99255	3097	4934	0.000016	0.000027	−0.000318
119106	3713	6196	0.000011	0.000018	−0.000033
158808	4948	8640	0.000006	0.000015	−0.000251
178659	5610	9901	0.000004	0.000012	−0.000075
218361	6901	12436	0.000003	0.000008	−0.000148
238212	7521	13666	0.000002	0.000008	−0.000154
277914	8794	16162	0.000001	0.000006	−0.000133
297765	9456	17321	0.000001	0.000008	−0.000081
337467	9926	19509	0.000001	0.000006	−0.000077
357318	9926	19614	0.000001	0.000005	−0.000073
397020	9926	19894	0.000001	0.000005	−0.000098
416871	9926	19910	0.000001	0.000006	−0.000057
456573	9926	19905	0.000001	0.000006	−0.000106
476424	9926	19939	0.000001	0.000006	−0.000085
516126	9926	19925	0.000001	0.000006	−0.000007
535977	9926	19940	0.000001	0.000006	−0.000048

part of the error values collected when the number of iterations increase.

In both tables we observe that when the number of iterations increases, the error values that we obtain for $E_{avg}, E_{max}, E_{sur}$ decrease. In other words, when we perform more and more iterations to obtain a simplified model, the approximation to the original one is better, what leads us to conclude that the GNG3D algorithm is efficient in the sense that when more iterations are carried out, better results are obtained for the approximations. That is absolutely agree with the general objective of the GNG3D algorithm, that is, to provide the best set of vertices and edges to reconstruct the simplified model.

4 The parameters choice

Using the three error measurements exposed in Section 3, $E_{avg}, E_{max}, E_{sur}$, we have performed numerical experiments for a variety of models with different geometric characteristics, concluding that we obtain better approximations from the original object specially when the number of iterations is high. However, it is possible to use these error measurements to determine the best possible parameters in the optimization algorithm.

To run our experiments we first need to specify a set of parameters for the optimization algorithm. The parameters involved in the algorithm are the following:

- a_{max}, the maximum age for the edges,
- ϵ_b, related to the displacement of the winner node in the space,
- ϵ_n, related to the displacement of the neighbor nodes in the space,
- d, a constant to decrease the error variables,
- λ, an integer to determine when to create a new node,
- α, a constant to decrease the error variables of the nodes after adding a new one,
- μ, a constant to know when to remove the nodes that have not been referenced in successive iterations.

There is no parameter set that can always produce the best results (lowest error values) for all possible three-dimensional models. The parameter values that minimize error for a certain model can achieve slightly higher error values for other models. This characteristic is common in all training processes based on neural networks. Nevertheless, the simplified models present a very high similarity with the original object regardless of parameter values, since the error is still very small.

As an example, we have taken the 3D model Bull to perform an experimental analysis of the way that the value of the parameters influence the error achieved. We have fixed the values of all the parameters except the one being studied. In Figure 1 (top) we show the error evolution (in this case E_{avg}), changing the parameter ϵ_b. Observing the results achieved, we can conclude that the lowest error is obtained when ϵ_b varies from 0.3 to 0.4. In Figure 1 (middle) we show the error evolution changing the parameter λ; in this case, the lowest error is obtained when λ is equal to 85. In Figure 1 (bottom) the study have been done for the parameter ϵ_n. We conclude that the value of this parameter should be smaller than 0.04 to produce good results. A similar study can be performed for the rest of the parameters.

As we have already remarked, these experimental results are only valid for this model. For any other one, the results may change, as we can see in Figure 2, where the model we have taken in this case is Octopus. Observe that the best results of the E_{avg} for the parameter ϵ_b are exactly at 0.4. It is also remarkable the differences respect to the Figure 1 for the parameters λ and ϵ_n.

5 Conclusion

Using the GNG3D method for mesh simplification, we have implemented three error functions which allow us to evaluate the quality of the approximations generated from the algorithm. With these error measurements we determine if the differences between the original and the simplified object are relevant or not.

In the numerical experiments it is observed that when the number of iterations is high, the error measurements decrease, which means that we are obtaining good simplifications of the original object. The reduction of the error is specially important in the surface error, which provides us a global idea about the quality of the simplified surface. Moreover, the error measurements implemented in the algorithm will allow us to compare with other methods for mesh simplifications.

Besides, we can use these error measurements to determine the best values for the different parameters involved in the running of the optimization algorithm. Fixing all the parameters except the one we want to optimize and measuring the error for different values of the parameter, it is possible to determine the value for which the error is minimum.

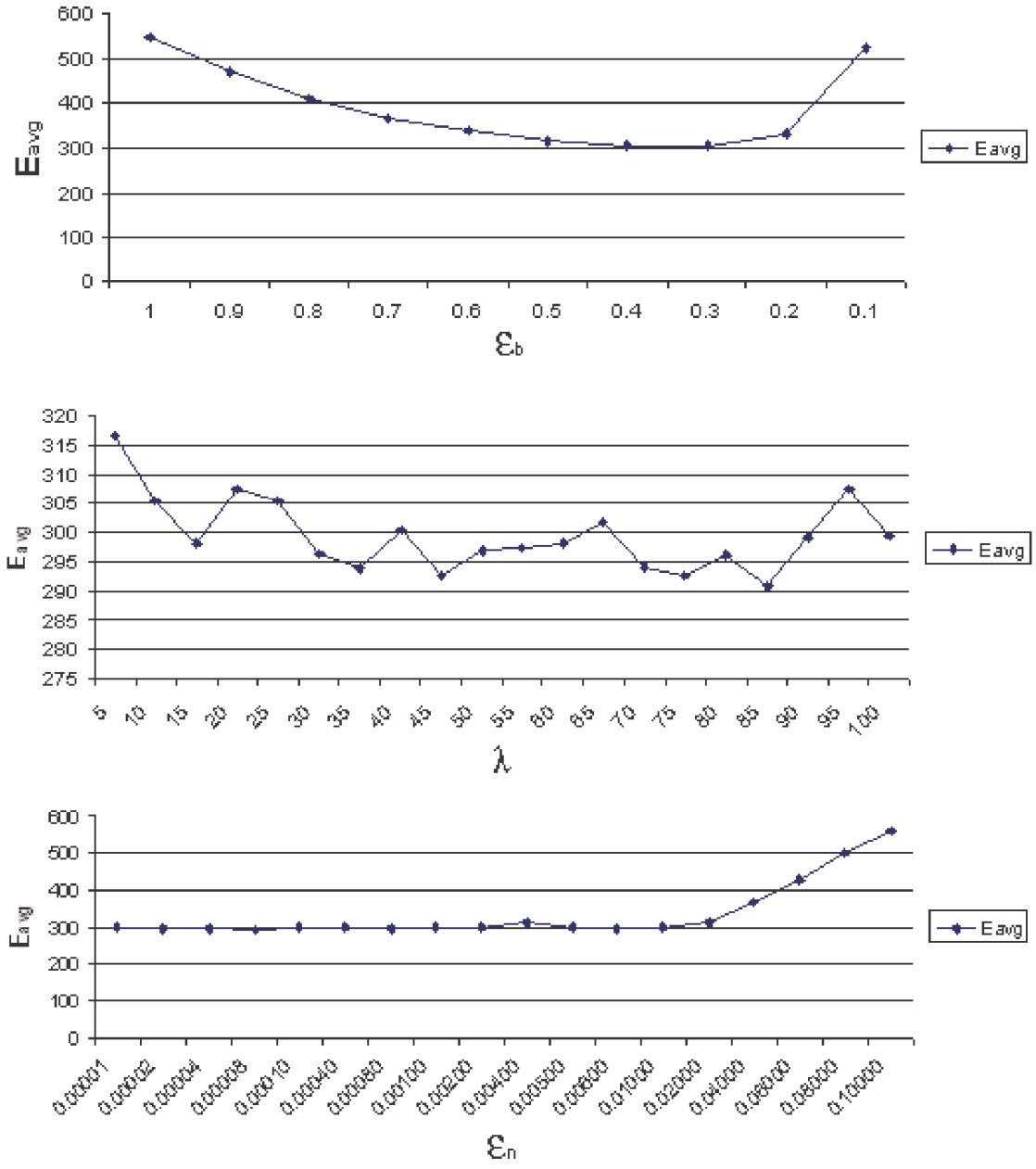

Figure 1: The error depends on the values of the parameters. In this example, we have tested the E_{avg} obtained with the GNG3D method when the parameter ϵ_b varies from 1 to 0.1, the parameter λ varies from 5 to 100, and the parameter ϵ_n varies from 0.00001 to 0.10000 for the 3D model Bull.

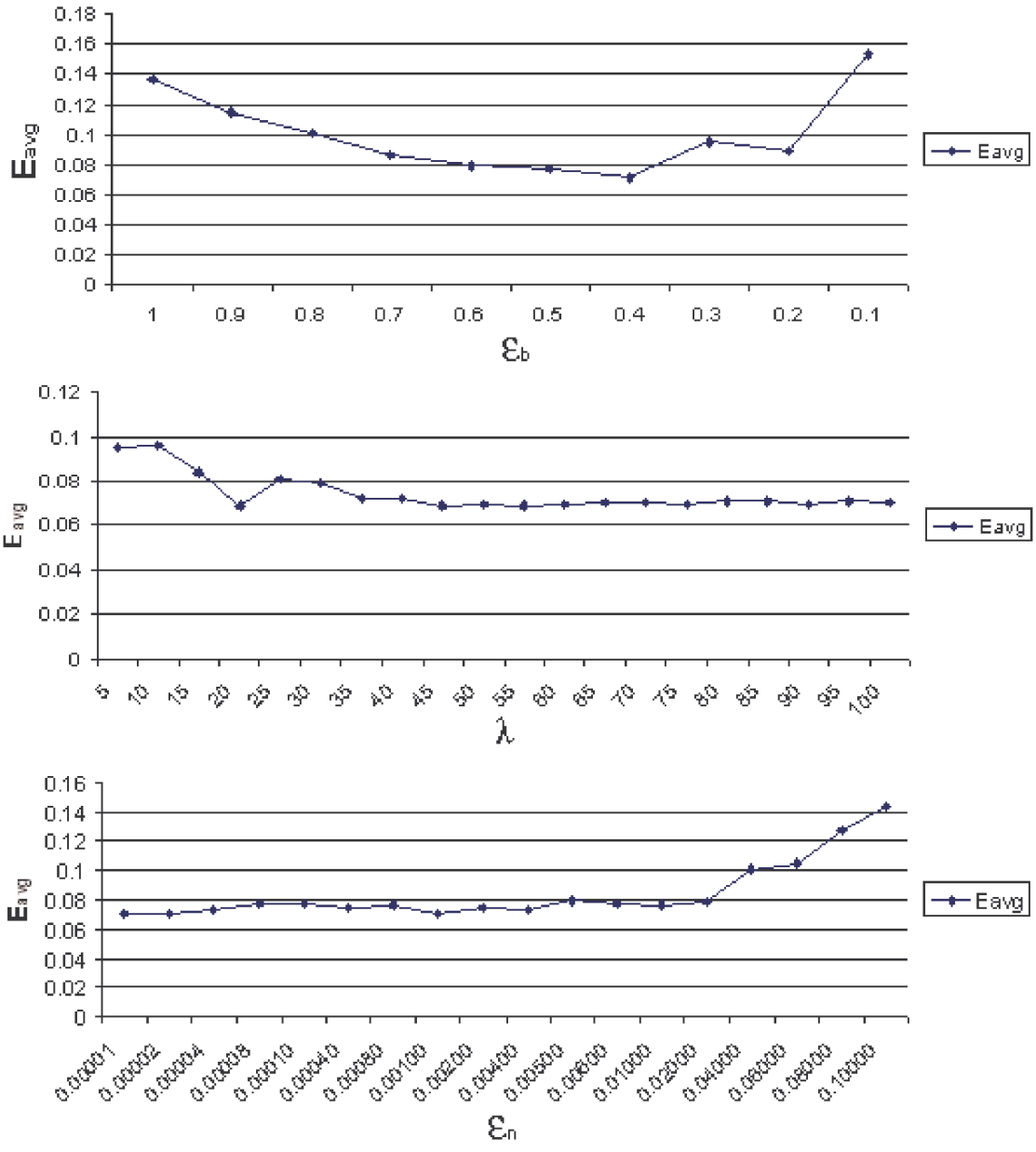

Figure 2: The error depends on the values of the parameters. In this example, we have tested the E_{avg} obtained with the GNG3D method when the parameter ϵ_b varies from 1 to 0.1, the parameter λ varies from 5 to 100, and the parameter ϵ_n varies from 0.00001 to 0.10000 for the 3D model Octopus.

Acknowledgements: The research was supported by the University of Alicante, (GV06/018).

References:

[1] M. Anderson, J. Gudmundsson and C. Levcopoulos, Restricted mesh simplification using edge contractions, *22nd European Workshop on Computational Geometry*, 2006, pp. 121–124.

[2] S. Atlan and M. Garland, Interactive multiresolution editing and display of large terrains, *Computer Graphics Forum* 25(2), 2006, pp. 211–224.

[3] P. Cignoni, C. Rocchini and R. Scopigno, Metro: measuring error on simplified surfaces, *Computer Graphics Forum 17(2), 1998, pp. 167-174.*

[4] D. Coen-Steiner, P. Alliez and M. Desbrun, Variational shape approximation, ACM Transactions on Graphics *23(3), 2004, pp. 905–914.*

[5] L-F. Diachin, P. Knupp, T. Munson and S. Shontz, A comparison of two optimization methods for mesh quality improvement, Engineering with Computers *22(2), 2006, pp. 61–74.*

[6] J-R. Diebel, S. Thrun and M. Brunig, A Bayesian method for probable surface reconstruction and decimation, ACM Transactions on Graphics *25(1), 2006, pp. 39–59.*

[7] B. Fritzke, Growing cell structures - a self-organizing network for unsupervised and supervised learning, Neural Networks *7(9), 1994, pp. 1441–1460.*

[8] B. Fritzke, A growing neural gas network learns topology, in: G. Tesauro, D.S. Touretzky, T. K. Leen, (Eds.), Advances in Neural Information Processing Systems *7, MIT Press, Cambridge MA, 1995, pp. 625–632.*

[9] M. Garland and P. Heckbert, Surface simplification using quadric error metrics, Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley, 1997, pp. 209–216.*

[10] M. Garland and Y. Zhou, Quadric-based simplification in any dimension, ACM Transactions on Graphics *24(2), 2005, pp. 825–838.*

[11] P. Heckbert and M. Garland, Optimal triangulation and quadric-based surface simplification, Journal of Computational Geometry: Theory and Applications *14(1-3), 1999, pp. 49–65.*

[12] H. Hoppe, New Quadric Metric for Simplifying Meshes with Appearance Attributes, in: Proc. IEEE Visualization '99, 1999, pp. 59-66.

[13] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle, Mesh optimization, *Computer Graphics* 27, 1993, pp. 19–26.

[14] I. P. Ivrissimtzis, W-K. Jeong and H.P. Seidel, Using growing cell structures for surface reconstruction, *Proc. International Conference on Shape Modeling and Applications*, 2003, pp. 78–88.

[15] T. Kohonen, Self-Organizing formation of topologically correct feature maps, *Biological Cybernetics* 43, 1982, pp. 59–69.

[16] T. Martinetz and K.J. Schulten, A neural-gas network learns topologies, In T. Kohonen, K. M Okisara, O. Simula (Eds.), *Artificial Neural Networks*, Amsterdam, 1991, pp. 397-402.

[17] T. Martinetz, Competitive Hebbian learning rule forms perfectly topology preserving learning, *Proc. ICANN'93: International Conference on Artificial Neural Networks*, Amsterdam, The Netherlands, Springer-Verlag, 1993, pp. 427–434.

[18] J. Rossignac and P. Borrel, Multi-resolution 3D approximation for rendering complex scenes, *Geometric Modeling in Computer Graphics, Springer Verlag, Genova, Italy, 1993, pp. 455-465.*

[19] R. Ronfard and J. Rossignac, Full-range approximations of triangulated polyhedra, Proc. of Eurographics*, 15(3), 1996, pp. 67–76.*

[20] W.J. Schroeder, J.A. Zarge and W.E. Lorensen, Decimation of triangle meshes, Proc. SIGGRAPH'92: 19th International Conference on Computer Graphics and Interactive Techniques, Chicago IL, 1992, pp. 65–70.

[21] G. Turk, Re-Tiling polygonal surfaces, *Proc. SIGGRAPH'92: 19rd International Conference on Computer Graphics and Interactive Techniques*, Chicago IL, 1992, pp. 55–64.

[22] A.W. Vieira, T. Lewiner, L. Velho, H. Lopes and G. Tavares, Stellar mesh simplification using probabilistic optimization, *Computer Graphics Forum*, 23, 2004, pp. 825–838.

[23] J. Yan, P. Shi and D. Zhang, Mesh simplification with hierarchical shape analysis and iterative edge contraction, *IEEE Transactions on Visualization and Computer Graphics* 10(2), 2004, pp. 142–151.