COMPUTATIONAL COST OF GNG3D ALGORITHM FOR MESH SIMPLIFICATION

Rafael Álvarez, José Noguera, Leandro Tortosa and Antonio Zamora*
Universidad de Alicante *
*Departamento de Ciencia de la Computación e Inteligencia Artificial, Universidad de Alicante,
Ap. Correos 99, E--03080, Alicante, Spain

ABSTRACT

In this paper we present a study of the computational cost of the GNG3D algorithm for mesh optimization. This algorithm has been implemented taking as a basis a new method which is based on neural networks and consists on two differentiated phases: an optimization phase and a reconstruction phase. The optimization phase is developed applying an optimization algorithm based on the Growing Neural Gas model, which constitutes an unsupervised incremental clustering algorithm. 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 computational cost of both phases is calculated, showing some examples.

KEYWORDS

Mesh simplification, polygonal reduction, computational cost, neural networks.

1. INTRODUCTION

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.

Some of the more representative work for a distinct approach related to the problem of mesh simplification can be seen in [1], [5], [7], [13], [14], [15], where different techniques are used to tackle the problem.

In this paper, we study an algorithm, called GNG3D, which is able to simplify any mesh representing a 3D model, regardless of its topological characteristics. The core of the GNG3D algorithm is a mesh optimization method based on artificial neural networks.

We use neural networks, in particular an adaptation of Fritzke's Growing Neural Gas (GNG) algorithm ([4]) for mesh generation, with the aim of approaching the problem of surface optimization and reconstruction. The GNG algorithm has its origin in the neural gas algorithm [11] and the Growing Cell Structure (GCS) algorithm [3]. These algorithms constitute perfect examples of growing or incremental network models, which are generated by successive addition (sometimes occasional deletion) of elements.

The GCS algorithm was introduced as a special type of Self-Organizing Maps (SOM's) [10] with the very distinctive feature of growing incrementally, vertex by vertex. SOM's and GCS's have already found many applications in geometric modeling and visualization problems. In [6] and [16], SOM's are used for surface reconstruction.

2. THE GNG3D ALGORITHM

The GNG3D algorithm has been designed taking as a basis the GNG model, with an outstanding modification consisting on the possibility of removing some nodes or neurons that do not provide relevant information about the original model. Besides, a reconstruction phase has been added in order to construct the faces of the optimized mesh.

2.1 Description of the Algorithm

1. Therefore, the GNG3D algorithm consists of two different phases: a **Mesh Optimization** Phase and a **Mesh Reconstruction** Phase.

A. Phase 1. Mesh Optimization

The primary objective of this optimization phase is the calculation of the best distribution of vertices that shapes the new simplified mesh. To perform this task an optimization algorithm has been implemented.

Optimization algorithm

INIT: Start with two nodes a and b at random positions w_a and w_b in $\Re^n$. 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 eliminate it afterwards.
4) Increment the local activation counter of the winner node. Add the square distance between the input signal and the nearest node in input space to a local error counter:

$$\Delta error(s_1) = \left\| w_{s_1} - \xi \right\|^2 . \tag{1}$$

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

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

$$\begin{aligned} \Delta w_{s_1} &= \varepsilon_b (\xi - w_{s_1}) \\ \Delta w_{s_n} &= \varepsilon_n (\xi - w_{s_n}), \end{aligned} \tag{2}$$

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, being 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:

$$w_r = 0.5\ (w_q + w_f). \tag{3}$$

     - 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 activation counter equal to zero) and all the edges emanating from them. Reset the local activation counter of all the nodes to zero.

Some remarkable points from the above algorithm are:

- 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 ε_b, ε_n, d, λ, α, 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.

Although the optimization algorithm is somewhat similar to the GNG algorithm, it presents a very important difference. In this algorithm, the neurons and the edges emanating from them, which 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 activation counter variable in step 4 of the algorithm. This local counter gives us the information about the number of times that a neuron has been referenced as the winner in the process of determining the closest neuron to the input signal. The introduction of this counter for each neuron is related to step 10 of the algorithm, where the neurons that have never been referenced as the winners are removed. The edges emanating from these neurons are also removed.

B. 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.

The reconstruction phase constitutes a post-process which uses the information on new nodes provided by the optimization phase and the information on the nodes of the original model. With these sets of nodes, a concordance process can be carried out 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. Label the nodes of the original mesh according to their representative.
2. Create a vector with all the connections among the resulting groups.
3. Reconstruct the faces.

Step 1. *Label the nodes of the original mesh according to their representative*. In this step, for each vertex of the original mesh, the vertex of the optimization set that is closer to it must be calculated. 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 $k_i \in \kappa$ which is closer to n_i. And this task must be repeated for every node in the original object.

As the number of vertices or nodes in the original object are normally very high, in order to speed up this step we use an octree with the aim of dividing 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.

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

According to the labelling operations carried out in step 1, the nodes of the original mesh have been arranged in groups and each of these groups has associated one and only one node belonging to the group of nodes obtained after applying the optimization algorithm. This association of nodes is performed minimizing the distance among the two sets of nodes. In this second step, we proceed analyzing each of the faces of the original mesh to check if their vertices have a 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.

2.2 Computational Cost of the Algorithm

The parameters involved in the execution of the GNG3D algorithm are ε_b, ε_n, a_{max}, d, λ, α, and μ. To compute the cost of the *i-th* iteration, we assume that the value of these parameters has been previously fixed. For our objective which is to obtain the computational cost of the algorithm, the modifications introduced in the parameters do not represent a variation in the cost.

The arithmetic cost of the optimization algorithm will be given by the number of operations required to perform K iterations of the algorithm following the steps described in Section 2.1.

In general, let us assume that in the *i-th* iteration we have $n_1, n_2, \ldots, n_l$ neurons or nodes. Therefore, the maximum number of neurons l in the i-th iteration will be $\lceil i/\lambda \rceil + 2$.

Now, let us determine the arithmetic cost of the optimization algorithm in phase 1 of the GNG3D method. Let us begin computing the cost of the *i-th* iteration and, later on, the cost of performing K iterations will be obtained. We distinguish two cases: when i is not a multiple of λ and when i is a multiple of λ.

Let us assume that i is not a multiple of λ and compute the arithmetic cost of this iteration.

- In step 1 of the algorithm, an input signal is generated, so the arithmetic cost in this step is 0.
- In step 2, we must find the nearest node (s_1) and the second nearest node (s_2) to the input signal. To compute the nearest node we use the usual euclidean distance between two points. In our case, with the aim of avoiding square root computations, we compute d^2. The computation of d^2 requires 8 operations. As the maximum number of neurons is $\lceil i/\lambda \rceil + 2$, we conclude that the arithmetic cost of this step is given by $8\lceil i/\lambda \rceil + 16$.
- In step 3, we increment the age of all edges emanating from s_1. This represents an addition, but the problem is that we cannot determine exactly the number of edges emanating from s_1. Therefore, it is necessary to establish an upper bound from the fact that the number of edges will be always lower than the number of neurons in the network minus one. Consequently, the maximum number of edges will be $\lceil i/\lambda \rceil + 1$ and the number of operations (an upper bound) in this step will be $\lceil i/\lambda \rceil + 1$.
- In step 4, we increment the local activation counter variable lc of the winner node, that is, $[lc(s_1)]_i = [lc(s_1)]_{i-1} + 1$, what represents an addition. As well as that, the local error variable of s_1 is increased using the expression (1). The computations required to obtain this error were performed in step 2, so we only perform the addition $[error(s_1)]_i = [error(s_1)]_{i-1} + \Delta error(s_1)$. Then, the cost of this step is 2 additions.
- In step 5, to move the winner node according to the expression (2), requires 9 arithmetic operations. This step also produces the movement of the direct topological neighbors of s_1. The equations we use to obtain the new position of each neighbor neuron are the same as the ones used to move the winner; consequently, the number of operations involved in this computation are 9. As we do not know exactly the number of neighbor neurons, we establish an upper bound using the fact that the maximum number of edges is $\lceil i/\lambda \rceil + 1$. Therefore, an upper bound for the arithmetic cost of this step is $9\lceil i/\lambda \rceil + 18$.
- The arithmetic cost of steps 6 and 7 is zero.
- In step 8, we decrease the error variables of all the nodes, which means that $\lceil i/\lambda \rceil + 2$ operations are performed.

To compute the arithmetic cost of the *i-th* iteration, when i is not a multiple of λ, it is enough to add the cost of the eight steps of the algorithm, that is, $19\left\lceil \frac{i}{\lambda} \right\rceil + 39$ operations.

Once we have computed the arithmetic cost of the *i-th* iteration, when i is not a multiple of λ, we proceed

to study the case of *i-th* iteration when *i* is a multiple of λ. The only difference with the case exposed above is that we need to perform step 9, where we insert a new node, if the maximum number of nodes has not been reached.

Let us assume that the maximum number of nodes has not been reached. The insertion of a node in the position given by the equation (3) requires 6 operations. Apart from this, we have to decrease the error variable of *q* and *f* by multiplying them by a constant. This means two more operations. Therefore, the arithmetic cost of step 9 is 8 operations.

In step 10 of the algorithm, we remove all the nodes that have not been used and all the edges emanating from them; no arithmetic operations are required to do this task.

Now, we can affirm that the arithmetic cost of the *i-th* iteration of the optimization algorithm, when *i* is a multiple of λ, is $19\left\lceil \frac{i}{\lambda} \right\rceil + 47$ operations.

If we perform *K* iterations of the optimization algorithm, the arithmetic cost will be

$$\sum_{i=1}^{K}\left(19\left\lceil \frac{i}{\lambda} \right\rceil + 47\right). \tag{4}$$

Developing expression (4), we have that, given λ and *K*, the arithmetic cost of performing *K* iterations of the optimization algorithm, C_K, is

$$C_K = 47K + \lambda\left(\sum_{i=1}^{j} 19 \cdot i\right) + \omega_K\left(19\left\lceil \frac{K}{\lambda} \right\rceil\right), \tag{5}$$

with

$$j = \left\lceil \frac{K}{\lambda} \right\rceil - 1, \qquad \omega_K = \left(K - \left\lceil \frac{K}{\lambda} \right\rceil \lambda\right) + 1.$$

Once we have obtained the cost of performing *K* iterations of the optimization algorithm, we proceed to compute the arithmetic cost of the reconstruction phase of the GNG3D algorithm. It is necessary to compute the cost of the three steps of the reconstruction phase.

- Step 1: Label the nodes of the original mesh according to their representative.

Let us assume that $A = \{n_1, n_2, \ldots, n_N\}$ and $R = \{m_1, m_2, \ldots, m_M\}$ represent the set of nodes of the 3D original model and the simplified one, respectively. Then, for every $n_i \in A$, with $i = 1, 2, \ldots, N$, we have to find the node $m_j \in R$, for $j = 1, 2, \ldots, M$, which is closer to n_i. Consequently, we need to compute, for a fixed *i*,

$$d^2(n_i, m_1),\ d^2(n_i, m_2),\ \ldots,\ d^2(n_i, m_M),$$

what represents $8 \cdot M$ arithmetic operations. As in our implementation we use an octree to divide the 3D space into 8 regions, we reduce the cost to *M* operations. Consequently, the total cost for the *N* vertices of the original model is given by $M \cdot N$ operations, with $N >> M$.

However, the generation of an octree to divide the 3D space into 8 regions means an additional computational cost in this step. The octree structure uses the well-known quicksort searching algorithm. The average computational cost of the quicksort algorithm is $O(N \cdot logN)$. As well as that, it is necessary to add the cost to include the neurons of *R* in the octree, what represents a cost of $O(M)$. Then, the cost associated with the construction of the octree structure is $O(M + N \cdot logN)$.

Finally, the cost of this step of reconstruction is given by

$$O(M + N \cdot logN + MN). \tag{6}$$

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

In this step, no arithmetic operations are required, so the arithmetic cost is zero.

- Step 3: Reconstruct the faces.

In this step, as well as the previous one, no arithmetic operations are required, so the arithmetic cost is zero.

Consequently, the arithmetic cost of this reconstruction phase is given by expression (6).

2.3 Some Examples

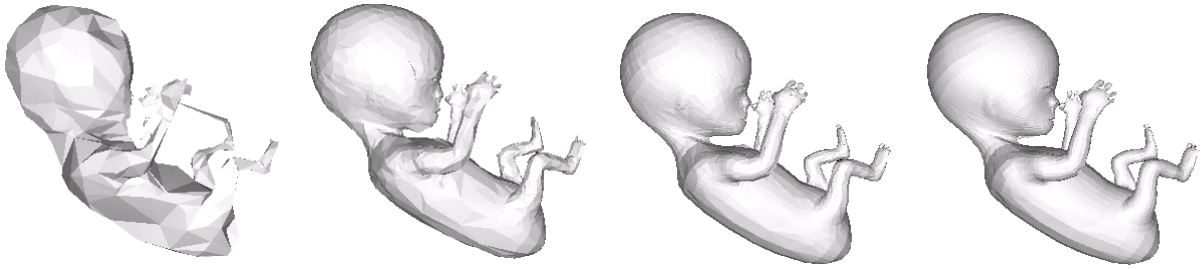

Figure 1. Fetus 3D model incrementally reconstructed using GNG3D method. The original mesh has 12500 vertices. From left to right, the resulting meshes have 500, 2500, 5000 and 6250 vertices after 5000, 25000, 50000 and 100000 iterations respectively.

The algorithm studied in this paper produces a high fidelity reconstruction of any 3D model as we can see in Figure 1. In order to evaluate the cost of the optimization and reconstruction phase we have tested the algorithm to find a simplified mesh with half of the vertices of the original model. As we described before, the optimization phase cost depends on the number of iterations K and the parameter λ, (see expression (4) or (5)). Therefore, the reconstruction phase cost depends on the size (number of vertices) of the original and simplified model, (see expression (6)).

Table 1 shows the increment of the algorithm cost during a complete reconstruction process where we have generated a simplified model with only 50% of the original model vertices. Note that the reconstruction phase is only necessary at the end of the process, when iterations are 300,000. In this case, the reconstruction phase has a constant cost of 200,060,000 operations because the maximum number of vertices has been reached and will not vary. Besides this, we have included in the table the reconstruction cost as if it had been executed using the indicated number of simplified vertices.

Table 1. Optimization and reconstruction cost of the GNG3D algoritm.

Iterations (K)	λ	Original vertices (M)	Simplified vertices (N)	Optimization phase cost	Reconstruction phase cost
100	20	20,000	7	8,595	160,005
1000	20	20,000	52	513,450	1,060,089
10,000	20	20,000	502	47,884,500	10,061,355
25,000	20	20,000	1,252	297,836,250	25,063,878
50,000	20	20,000	2,502	1,189,422,500	50,068,502
100,000	20	20,000	5,002	4,753,845,000	100,078,503
150,000	20	20,000	7,502	10,693,267,500	150,089,071
200,000	20	20,000	10,000	19,007,690,000	200,060,000
250,000	20	20,000	10,000	28,510,040,000	200,060,000
300,000	20	20,000	10,000	38,012,390,000	200,060,000

From Table 1 we can extract some important conclusions. The first one is that the optimization phase cost is much bigger than the reconstuction phase cost. Another one is that the optimization phase cost grows with the number of iterations, while the reconstructions phase cost grows until the desired number of simplified vertices is reached and remains constant from that moment on. These conclusions are shown in Figures 2, 3 and 4.

In Figure 2 and 3 we can see how parameter λ affect to cost of each phase. λ is used in GNG3D to determine when a new vertice has to be inserted. In consequence, a lower value of λ implies a quick insertion frequency of new vertices and a quick increase of the cost. A high value of λ implies a lower frequency insertion, so that with the same number of iterations, less vertices have been inserted and therefore, it has a smaller cost.

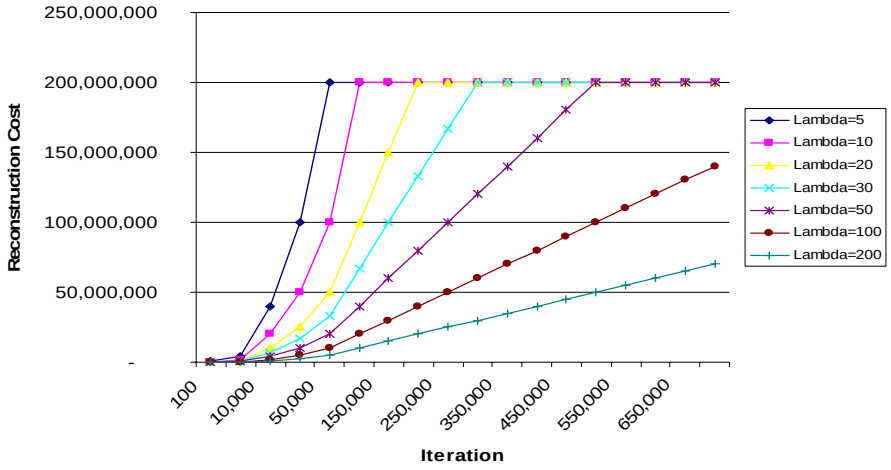

Figure 2. The optimization phase cost depends on the value of parameter λ.

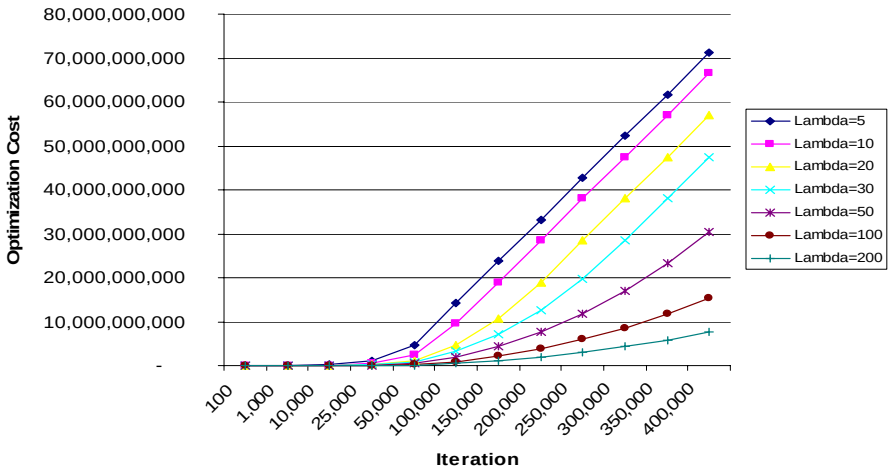

Figure 3. The optimization phase cost is much bigger than the reconstruction phase cost.

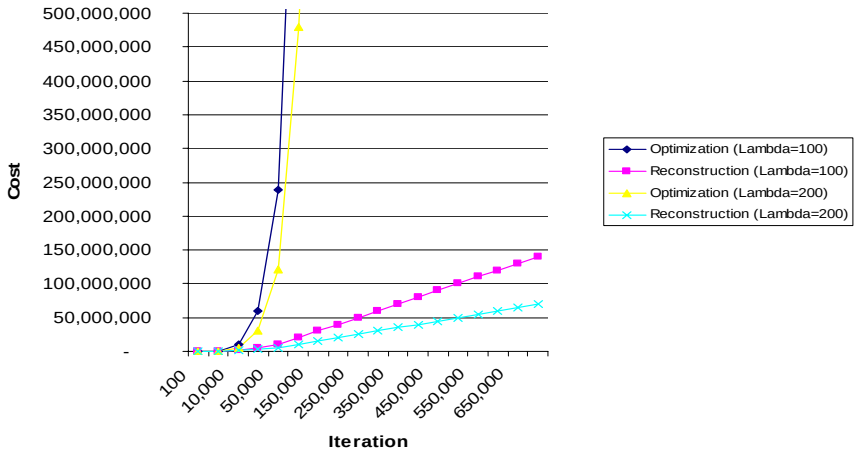

Figure 4. The reconstruction phase cost grows until maximun number of simplified vertices is reached.

3. CONCLUSION

In this paper we have performed a detailed study of the computational cost of the GNG3D algorithm. The GNG3D algorithm presents very good characteristics for 3D mesh simplification, as well as constitutes a new application of neural networks. This study confirms the high computational power needed for these tasks and also can be used to compare the results with other mesh optimization methods.

REFERENCES

Algorri, M.E, and Schmitt, F., 1996. Mesh simplification, *Computer Graphics Forum* (Eurographics'96 Proc.), Vol. 15, No. 3, pp. 78-86.

Bishop, C.M., 1995. *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press, England.

Fritzke, B., 1994. Growing cell structures - a self-organizing network for unsupervised and supervised learning, *Neural Networks*, Vol. 7, No. 9, pp. 1441-1460.

Fritzke, B., 1995. A growing neural gas network learns topology, in *Advances in Neural Information Processing Systems 7*, Edited by G. Tesauro, D.S. Touretzky and T. K. Leen, Cambridge, MA: MIT Press, pp. 625-632.

Gross, M., Staadt, O., and Gatti, R., 1996. Efficient triangular surface approximations using wavelets and quadtree structures, *IEEE Transactions on Visual and Computer Graphics*, Vol. 2, No.2, pp. 130-144.

Hoffmann, M., and Varady, L., 1998. Free-form modeling surfaces for scattered data by neural networks, *Journal of Geometry and Graphics*, Vol. 1, pp. 1-6.

Hoppe, H., 1996. Progressive meshes, *Proceedings of SIGGRAPH'96: 23rd International Conference on Computer Graphics and Interactive Techniques*, New Orleans, Louisiana, pp. 99-108.

Ivrissimtzis, I.P., Jeong, W-K., and Seidel, H.P., 2003. Using growing cell stuctures for surface reconstruction, *Proceedings of International Conference on Shape Modeling and Applications*, pp. 78-88.

Kohonen, T., Self-Organizing formation of topologically correct feature maps, *Biological Cybernetics*, Vol. 43,

pp. 59-69.

Kohonen, T., The Self-Organizing Map, *Proceedings of the IEEE*, Vol. 76, No.9, pp. 1464-1480.

Martinetz, T., and Schulten, K. J., 1991. A neural-gas netwok learns topologies, in *Artificial Neural Networks*, Edited by T. Kohonen, K. M Okisara, and O. Simula, Amsterdam, Netherlands, pp. 397-402.

Martinetz, T., 1993. Competitive Hebbian learning rule forms perfectly topology preserving learning, *Proceedings of the ICANN'93: International Conference on Artificial Neural Networks*, Amsterdam, Netherlands, Springer-Verlag, pp. 427-434.

Rossignac, J., and Borrel, P., 1993. Multi-resolution 3D Approximation for rendering complex scenes, in *Geometric Modeling in Computer Graphics*, Edited by B. Falcidieno, and T. Kunii, Springer-Verlag, Genova, Italy,

pp. 455-465.

Schroeder, W. J., Zarge, J.A. , and Lorensen, W.E., 1992. Decimation of triangle meshes, *Proceedings of the SIGGRAPH'92: 19th International Conference on Computer Graphics and Interactive Techniques*, Chicago IL,

pp. 65-70.

Turk, G., 1992. Re-Tiling polygonal surfaces , *Proceedings of the SIGGRAPH'92: 19rd International Conference on Computer Graphics and Interactive Techniques*, Chicago IL, pp. 55-64.

Yu, Y., 1999. Surface reconstruction from unorganized points using self-organizing neural networks, *Proceedings of IEEE Visualization 99*, pp. 61-64.