A self-organized neural network for 3D surface reconstruction

AGOSTINHO DE MEDEIROS BRITO JÚNIOR
ADRIÃO DUARTE DÓRIA NETO
JORGE DANTAS DE MELO
Departmento de Engenharia de Computação e Automação
Universidade Federal do Rio Grande do Norte
DCA - CT - UFRN, 59056-270, Natal, RN, Brasil

Abstract: We present a multiresolution surface reconstruction method from point clouds in 3D space based on Kononen's self-organizing neural networks. It uses a set of mesh operators and simple rules for selective mesh refinement. Experimental results shows the method is very sucessfull to reconstruct forms of varied geometry.

1 Introduction

The surface reconstruction of 3D objects has a wide range of applications such as CAD design, virtual reality, medical imaging and movie industries.

According to the scanning device used, the sample of points used for reconstruction can be classified as structured and unstructured, based on to the connectivity information among the points [1].

Given a real surface and a set of points sampled over it, the goal is to create a surface model that approximate the features of the real model. A good surface reconstruction algorithm must be able to recover both geometry and topology in order to fit the data correctly.

One of the main trouble with surface reconstruction is about topology recovery. In some algorithms, if the distribution of the samples in the space is not dense or uniform enough, wrong holes may appear on undesirable places over the surface. Sometimes, the algorithm must pre-establish the mesh topology, just as in the case of deformable models [2].

We address the surface reconstruction problem using a self-organizing neural network to position the vertices of a triangle mesh. Given an initial mesh, we use the Kohonen learning algorithm to move its vertices coordinates toward the data sampled over the surface.

In fact, neural networks have been already used for surface reconstruction[3, 4, 5]. However, in this work, we introduce an improved mesh refinement method to create a better multi-resolution surface reconstruction algorithm based on neural networks. The major advantage of our approach is that the refinement method proposed leads to surface meshes with a user-selective increase of elements, while still maintaining the same topology pre-defined by the initial mesh.

2 Previous work

The surface reconstruction methods can be roughly classified as static methods, based on geometric techniques, and dynamic, based on the evaluation of energy or force functions.

One of most famous static method was proposed by Hoppe [1]. The sample points are used to define a signaled distance function in $\mathbb{R}^3$. The zero set is interpolated and polygonized using the Marching Cubes [6] algorithm to create an initial approximation for the surface. After steps of surface optimization and smoothing, excellent models are achieved.

Another static method, the Crust [7], is based on the Voronoi diagram to find an approximation for the Medial Axis Transform (MAT) of the object. The inverse transformation of the MAT is used to obtain the reconstructed surface mesh. However, if the sample is not uniformly spaced, some undesirable holes may appear on the surface.

The other methods are usually based on the triangulation of the points in the sample. The establishment of geometry constraints such as the curvature [8], each method obtain the solution according to the desired application.

Among the dynamic methods, the most common ones are based on deformable surfaces and balloons [9, 10]. Deformable surfaces are polygonal meshes that are modified according to the data and some shape constraints, starting from a given initial state. The reconstruction is obtained by the equilibrium of internal and external forces. On the balloons, the vertices movement is governed by internal pressure forces that make the model (balloon) inflate in order to fit the data [11].

Most of these approaches have been widely studied in literature. Some of them are known to be susceptible to problems with local minima, may result on surface meshes with wrong geometry or topology and still are very time consuming, even for small datasets.

On the other hand, neural network research in this field is still very incipient. Most of the previous works with neural nets treats simple cases, usually only with elevation data and open topology meshes [3, 4, 5].

Some of the important stochastic characteristics of the self-organizing neural networks may lead to surface meshes with very good geometry properties that were still not investigated.

3 Self-organizing Maps: Kohonen algorithm

Kohonen [12] proposed a model of self-organizing neural network that has the ability to approximate the input space used for training, while still maintaining the topological ordering.

In the Kohonen model, the training objective is to transform an incoming signal pattern of arbitrary dimension into a two-dimensional discrete map (the Self-Organizing Map - SOM), and to perform this transformation adaptively in a topologically ordered fashion [13]. The map is usually a bidimensional lattice (see Fig. 1) of neurons whose weights store the map approximation for the input space.

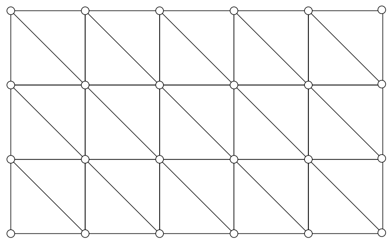
Figure 1: Kohonen self-organizing map with triangular lattice.

The input patterns of the neural network are vectors denoted by

$$\mathbf{x} = [\ x_1 \quad x_2 \quad \dots \quad x_m\ ]^T \tag{1}$$

and the synaptic weights are vectors denoted by

$$\begin{aligned} \mathbf{w}_j = [\ w_{j1} \quad w_{j2} \quad \dots \quad w_{jm}\ ]^T, \\ j = 1, 2, \dots, l, \end{aligned} \tag{2}$$

where m is the dimension of the input space and l is the number of neurons in the network.

The training algorithm is composed of three main steps for each input pattern presented to the network: **Competition**, where the value of a discriminant function is calculated for each neuron. The Euclidean distance is usually the normal choice. The neuron with the lowest value for the discriminant function

$$i(\mathbf{x}) = \arg\min_i \|\mathbf{x} - \mathbf{w_j}\| \tag{3}$$

is considered to be the winner; **Cooperation**, where each neuron establishes a topological neighborhood of neurons to be excited. Such neurons will be excited according to an neighborhood function, usually a Gaussian function:

$$h_{j,i(\mathbf{x})} = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2}\right) \tag{4}$$

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right) \tag{5}$$

where σ_0 is the size of the neighborhood function at the beginning of the training,

$\tau_1 = niter/\log(\sigma_0)$ is a time constant, $n = 0, 1, \ldots, niter$ is the time step and $niter$ is the number of iterations used for training; and **Adaptation**, where the synaptic weights of each neuron is modified in order to enhance the response to similar input patterns. Such modification can be *sequential*, if the weights are modified as the input patterns are presented to the neural net, or *in batch*,

$$\mathbf{w}_j(n+1) = \frac{\sum_{k=0}^{m} h_{j,i(\mathbf{x}(k))}\mathbf{x}(k)}{\sum_{k=0}^{m} h_{j,i(\mathbf{x})}}, \qquad (6)$$

if the weights are modified only once, after all patterns are presented.

We choosed to use the batch SOM approach because it has a lot of advantages when compared to the sequential SOM. The weights update does not depend on the order the input patterns is presented to the network. There is no learning parameter, avoiding a potential source of error and still there exists the advantage of data partitioning, allowing the network to be trained on parallel architectures.

4 Mesh representation and operation

We use triangle meshes for surface representation. Each surface S is 2-manifold without borders, represented by a polygonal mesh composed of: a set of vertices; a set of edges, interconnecting pairs of vertices; and a set of triangle faces.

Despite the impressive self-organization of the SOM algorithm, it has some difficulties when reconstructing concave regions. Some vertices or triangles of the reconstructed surface may not be stable, "*dangling*" among regions populated by the data, such as those placed between the bunny's ear and loin in figure 2.

In order to make such improvements, we define a set of mesh operators that will allow to perform improvements on the surface geometry, in order to better fit the data: edge swap, edge collapse, vertex split and triangle subdivision. Three of these operations are presented in the figure 3. In this work, the position of the new vertex in an edge collapse operation will assume the position of one of the two vertices of the edge.

In the vertex split operation used, only two new triangles will be created, although other strategies are possible [1].

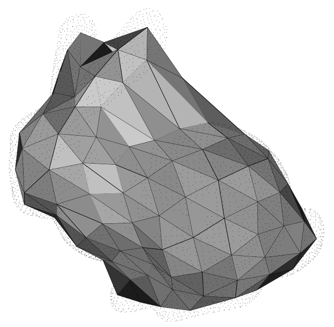
Figure 2: Vertices and triangles dangling among regions of the sample. See detail at bunny's ear.

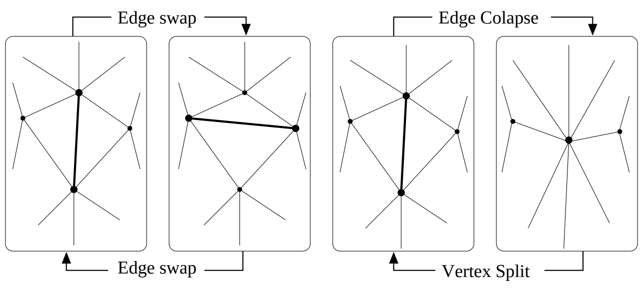

Figure 3: Edge swap, edge colapse and vertex split operations.

The triangle split is done for the whole mesh in a single step. First, the edges of all triangles desired to be subdivided are marked. Then, for each triangle, we verify how many of its three edges are marked and perform subdivision for this triangle according to the subdivision rules presented in figure 4. With this procedure, we will ensure that the mesh will be composed only of triangles after this refinement. An example of this subdivision procedure is presented in figure 5.

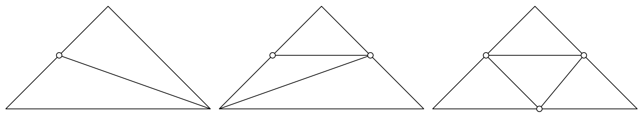
Figure 4: Triangle subdivision rules.

5 The algorithm

We use the self-organizing network described in section 3 to move the vertices of a initial mesh. In the case of surface reconstruction, the positions of

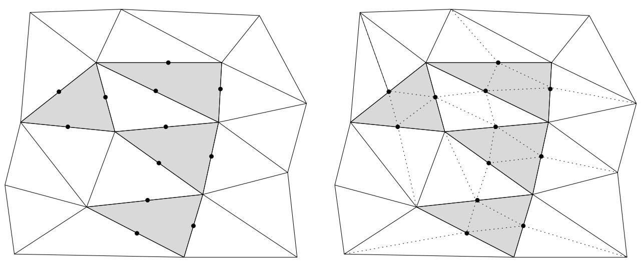

Figure 5: Triangle subdivision example. On the left, the dark triangles represent the desired to be subdivided. On the right, the result of the subdivision is presented.

the vertex in the mesh will be represented by neurons weights and the lattice interconnection structure for the network will be constructed in order reflect the geometry and topology of the initial mesh.

The size of the initial neighborhood function, σ_0, described in equation (4), is equal to mesh diameter divided by two. The data sample of the surface to be reconstructed is used as training set to the neural network. We use N iteration steps for training and more N training steps for and additional convergence phase, where only the nearest neurons of the winner neuron are maintained in the neighborhood. This additional steps are used to compensate some adaptive capability that is lost in the batch version of SOM [14].

Once finished the training phase, the resulting mesh is modified in order to better fit the data. We first calculate the Euclidean distance of every vertex in the mesh to the closest point in the training set. The mean and the associated standard deviation for such distance are calculated and those vertices which distance is larger than *mean + standard deviation* are considered unstable.

For each unstable vertex, we search through its neighbour vertices for any that is not dangling and we collapse the edge connecting these two vertices. The coordinates of the new vertex resulted from the edge collapse will be same of the chosen adjacent neighbour.

After the edge collapse operation, we now perform a set of edge swaps in the resulting mesh using the algorithm proposed by [5]. In this step, we obtain the minimum Euclidean distances from the triangle baricenters (B_T) to the the training set:

$$MD(T) = \min_{p \in \mathbf{x}} Distance(p, B_T) \qquad (7)$$

The set of distances with its respective triangles ids is stored into a sorted list, with the largest distance values at the top. We define a initial flip threshold equal to the distance stored at the top of the sorted list. We now iterate over the list and we perform swaps on the edges of the triangles with values of $MD(T)$ largest than the specified threshold. To choose which edges should be swapped, a edge *deviation* measure is defined:

$$Dev(E) = MD(T_0) + MD(T_1) \qquad (8)$$

where T_0 and T_1 are the two triangles adjacent to the edge E. For each problematic triangle, we first try a single swap on its edge with the largest value of $Dev(E)$. If the value of $Dev(E)$ for the new edge is smaller than the old one and the value of $MD(T)$ is smaller for at least one of the two triangles adjacents to this edge, the single swap is accepted. Otherwise, we try a double swap, maintaining this first swap and swapping the edge with the second largest deviation. If this new swap satisfies the above condition, the double swap is accepted. Otherwise, the original triangle configuration is maintained. The single and double edge swap operations are presented in figure 6.

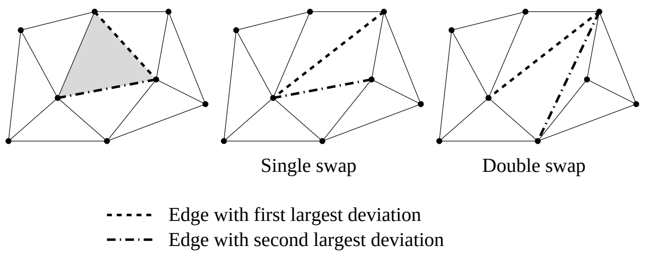
Single swap
Double swap
Edge with first largest deviation
Edge with second largest deviation

Figure 6: Single and double edge swaps.

Once all triangles with minimum distance largest than the threshold are tested, we search the list for the largest distance that is smaller than the previous threshold. Using this value as the new threshold, we restart the scan through the list and the edge swap operations. When the threshold becomes smaller than the mean value for $MD(T)$ the edge swap step is finished.

The reconstruction stops at the end of the edge swap step. However, for a multi-resolution learning, we must refine the resulting mesh in order to obtain a new mesh with more elements. This refined mesh is now used to build a new neural network to be trained with the same process described above.

In the refinement step, we calculate the values of $MD(T)$ for all triangle in the mesh and obtain the mean and standard deviation values for this measure. The triangles with $MD(T)$ largest than *mean + standard deviation* are marked to be subdivided. Using the triangle subdivision rules presented in section 4, we create the triangles that will compose the new mesh. And, at least, we search for vertices with excessive vertices connected to it. Vertex with more than eight adjacents are submitted to a vertex split operation, with half part of the consecutive adjacents belonging to the new created vertices.

6 Results

The whole method was implemented in C++, running on a Pentium III 500 Linux machine. We are assuming that the surfaces to be reconstructed are homeomorphic to a sphere, but meshes with different topology are possible. The initial mesh is a octahedron with 8 faces. We trained the neural network with 50 iterations for each resolution level. The method was tested with the Stanford bunny (Fig.7), with 35947 points. The experimental results show the algorithm capabilities.

When we compared the Stanford bunny reconstruction with other multi-resolution method available in literature [5], the amount of triangles in equivalent resolution levels are smaller. The first presented resolution level has 306 triangles against 320 of the other method; In the second, third and fourth resolution level, we get 988, 2822 and 7468 triangles, against 1280, 5120 and 20480 triangles of the other method.

7 Conclusion

We developed a surface reconstruction method based on the unsupervised capabilities of the Ko-

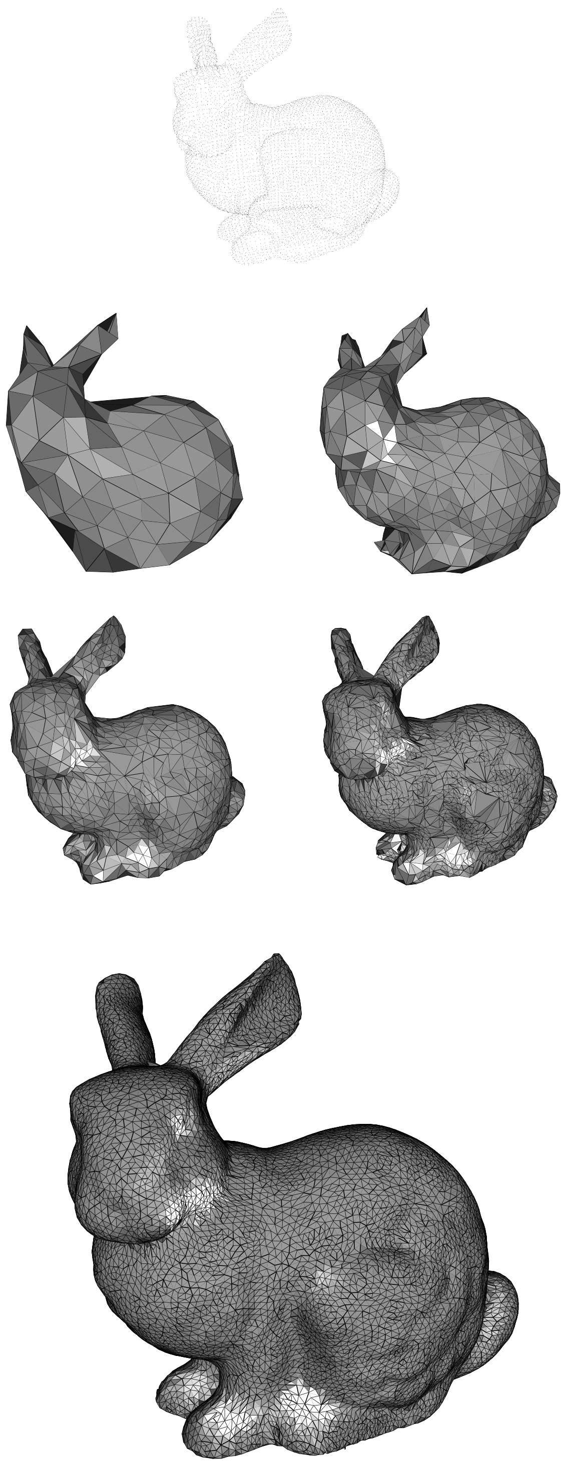

Figure 7: The Stanford bunny points and the surface reconstruction from the coarser to the finest resolution level, with 306, 988, 2822, 7468 and 25016 triangles, respectively.

honen algorithm. The positions of the vertices in a initial mesh are modified according to the sample data used as the training set. The developed algorithm is able to create 3D meshes with variated geometry, in a multi-resolution fashion.

Some used heuristics allow a selective refinement, from coarser resolutions to finest resolutions, according a threshold value based on its minimum distance from the mesh triangles to the training set. The major advantage of this approach is that this threshold can be set by the user in order to produce meshes with more or less triangles, allowing a better refinment control, when compared with other multi-resolution methods.

The algorithm was tested with well known datasets available in the literature, but is extensible for others input sets.

References

[1] Hugues Hoppe, *Surface reconstruction from unorganized points*, Ph.D. thesis, University of Washington, 1994.

[2] J. Montagnat, H. Delingette, and N. Ayache, “A review of deformable surfaces: topology, geometry and deformation,” *Image and Vision Computing*, vol. 19, 2001.

[3] D. S. Chen, R. C. Jain, and B. G. Schunck, “Surface reconstruction using neural networks,” in *Computer Vision and Pattern Recognition conference proceedings*. Junho 1992, pp. 815–817, IEEE Computer Society.

[4] S. Alfonzetti, S. Coco, S. Cavalieri, and M. Malgeri, “Automatic mesh generation by the let-it-grow neural network,” *IEEE transactions on magnetics*, vol. 32, no. 3, Maio 1996.

[5] Yizhou Yu, “Surface reconstruction from unorganized points using self-organizing neural networks,” in *Proc. of IEEE Visualization’99 LBHT*, 1999.

[6] W. E. Lorensen and H. E. Cline, “Marching cubes: a high resolution 3d surface construction algorithm,” *Computer Graphics*, vol. 21, no. 4, pp. 163–169, Julho 1987.

[7] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri, “The power crust,” in *Proceedings of the sixth ACM symposium on Solid modeling and applications*. 2001, pp. 249–266, ACM Press.

[8] L. Alboul, G. Kloosterman, C.R. Traas, and R.M.J van Damme, “Best data-dependent triangulations,” Tech. Rep. 1487, Faculty of Mathematical Sciences, University of Twente, Junho 1999.

[9] Laurent D. Cohen, “On active contour models and ballons,” *Computer Vision, Graphics, and Image Processing: Image Understading*, vol. 2, no. 53, pp. 211–218, Maro 1991.

[10] Demetri Terzopoulos and Andrew Witkin, “Deformable models,” *IEEE Computer Graphics and Applications*, vol. 8, no. 6, pp. 41–51, 1988.

[11] Y. Chen and G. Medioni, “Description of complex objects from multiple range images using an inflating balloon model,” *Computer Vision and Image Understanding*, vol. 61, no. 3, pp. 325–334, May 1995.

[12] T. Kohonen, *Self-organization and associative memory*, Springer-Verlag, New York, 1984.

[13] Simon Haykin, *Neural Networks: a comprehensive foundation*, Prentice-Hall, New Jersey, 2 edition, 1999.

[14] T. Kohonen, “Things you haven’t heard about the self-organizing map,” in *Proceedings of the IEEE Internation Conference on Neural Networks*, San Francisco, pp. 1147–1156.