Surface Reconstruction from Unorganized Points Using Self-Organizing Neural Networks

Yizhou Yu

Computer Science Division
University of California at Berkeley

ABSTRACT

We introduce a novel technique for surface reconstruction from unorganized points by applying Kohonen's self-organizing map. The topology of the surface is predetermined, and a neural network learning algorithm is carried out to obtain correct 3D coordinates at each vertex of the surface. Edge swap and multiresolution learning are proposed to make the algorithm more effective and more efficient. The whole algorithm is very simple to implement. Experimental results have shown our techniques are successful.

Keywords: Surface Reconstruction, Point Set, Neural Networks, Self-Organizing Maps, Geometry, Modeling, Visualization, Laser Scanner

1 Introduction

The problem of reconstructing a surface from scattered sample points arises in many applications such as geometric model acquisition in graphics, visualization of data from medical imaging, and cartography where the input devices can only obtain the 3D positions of sample points with no information of connectivity among them.

The difficulty involved in the problem of surface reconstruction from unorganized points is in obtaining correct connectivity among the sample points. Correct connectivity can give us a reconstructed surface mesh that faithfully represent the shape and topology of the original object from which the set of sample points were drawn. Usually this difficulty is tackled in a bottom-to-top way. That is, previous work[7, 3, 2] tried to build connections among nearest points and finally build a mesh out of the input point set. This approach requires that the sample points be dense enough; otherwise, holes may appear at undersampled areas, which changes the topology of the surface. We are pursuing a totally different approach here, which we call the top-to-bottom approach. We assume a mesh with correct topology(connectivity) has already been given in advance with some help from the user, e.g. the user can tell from the sample points whether the original object is topologically equivalent to a sphere or a torus. However, in the beginning, the coordinates at each vertex of the mesh are unknown and initialized in a naive way. Instead of building connectivity, the major challenge here is to work out a method of moving the vertices so that they are close enough to the input point set to manifest its shape. A learning procedure, which was originally used for training self-organizing neural networks[1], is applied to the vertices which obtain meaningful coordinates at the end so that the shape of the mesh approximates that of the input point set. We can summarize the difference of these two approaches as learning connectivity given coordinates versus learning coordinates given connectivity. The major advantage of our approach is that no holes will appear at unexpected places even for sparse datasets because of the predetermined mesh topology.

This paper is organized as follows. Section 2 describes previous work on surface reconstruction. Section 3 introduces Kohonen's Self-Organizing Map and its training algorithm. Section 4 gives our surface reconstruction algorithm using the above-mentioned training algorithm. Section 5 describes how to assign texture coordinates in our approach. Section 6 shows some example results. And Section 7 has the conclusions.

2 Previous Work

There are two types of previous work on surface reconstruction: interpolation and approximation. The first type has combinatorial algorithms. The second type is similar to data fitting by using a piecewise linear surface and minimizing the distance between the point set and the approximating surface.

Hoppe et al[7, 6], Curless and Levoy[3] presented algorithms in the second type. They exploited the fact that a surface can be reconstructed from its normal orientations because we can get tangent planes from normals and the area around the normal on a tangent plane is a first order approximation of the local surface. These algorithms need to estimate normals from the point set very accurately.

The algorithm in the first type includes α-shape by Edelsbrunner et al[5], and the "crust" by Amenta and Bern[2]. α-shape is a heuristic. It works well for uniform sampling. The major drawback of using α-shapes for surface reconstruction is that the optimal value of α depends on the sampling density, which often varies over different parts of the surface. The "crust" is provably correct if the sampling density is enough everywhere. But the local topology may be changed and holes may appear due to undersampling.

3 Kohonen's Self-Organizing Map

The basic configuration of Kohonen's Self-Organizing Map [1] is a two-dimensional network of cells denoted by $\{C_1, C_2, ..., C_m\}$ with a kind of spatial arrangement so that each cell in the network has a few neighboring cells. The spatial arrangement and definition of neighborhood may be different for different applications. For instance(Fig. 1), if the cells are arranged as a rectangular array,

*Email: yizhouy@acm.org, yyz@cs.berkeley.edu,
Website: http://www.cs.berkeley.edu/∼yyz

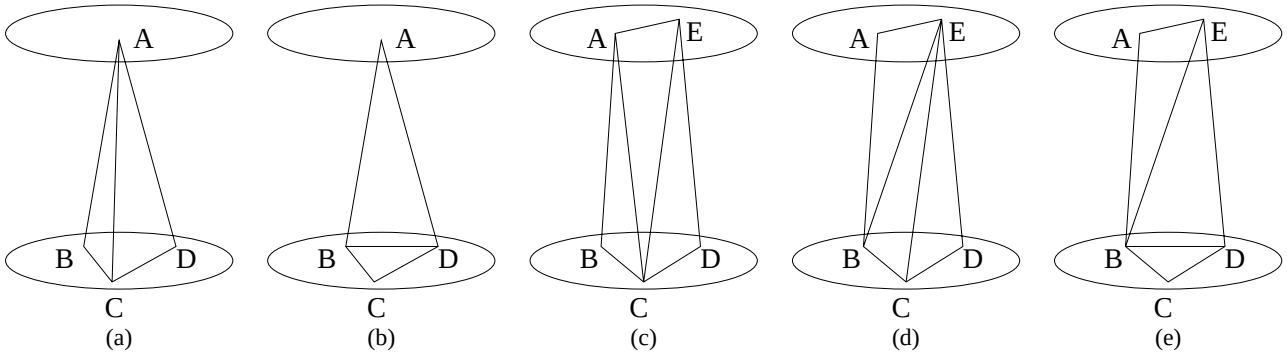


Figure 2: (a)-(b) If the local mesh structure is like the one shown in (a) where vertex A belongs to one part of the surface and vertices B, C, D belong to some other part of the surface, we can use a single swap at the middle edge to reduce the number of problematic triangles as shown in (b); (c)-(e) If the local mesh structure is like the one shown in (c) where vertices A, E belongs to one part of the surface and vertices B, C, D belong to some other part of the surface, we can use a double swap at the middle triangle to reduce the number of problematic triangles. (d) shows the 0 after the first single swap, and (e) shows the structure after the second single swap.

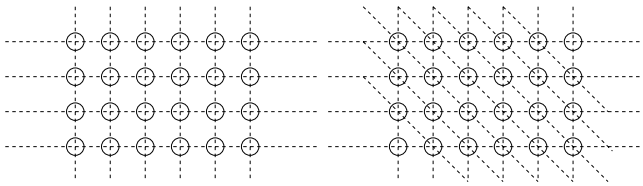


Figure 1: Kohonen's Self-Organizing Maps with rectangular and triangular spatial arrangement, respectively. Each cell in the rectangular arrangement has four immediate neighbors. Each cell in the triangular arrangement has six immediate neighbors.

each cell has four immediate neighbors. In computer graphics, triangular meshes are very common geometric representations. So we may arrange the cells as the vertices in a triangulation. Two cells are neighbors to each other if there is an edge between them. The distance between two neighboring cells is usually defined to be one, and the distance between any two cells in the network is defined to be the length of the shortest path connecting them.

The input to the network is a set of vectors denoted by $\{x_1, x_2, ..., x_n\}$. Each cell C_j in the network has a weight vector w_j whose dimensionality is the same as that of the input vectors. Given a particular input vector, the response of a cell can be either the inner product between the input vector and the cell's weight vector or the Euclidean distance between these two vectors. The winner cell is defined to be the one with the largest(inner product) or smallest(Euclidean distance) response.

Kohonen's iterative training procedure is used for obtaining a proper weight vector for every cell so that winner cells corresponding to two input vectors are close to each other in the network if the input vectors are close to each other in their vector space. In his training algorithm, if cell $C(t)$ is the winner cell corresponding to input vector $x(t)$ at time step t, the updating rule may read

$$w_k(t+1) = \begin{cases} w_k(t) + K(t,d)[x(t) - w_k(t)], \\ \qquad\qquad d = \text{Distance}(C_k, C(t)) \leq \delta(t), \\ w_k(t), \qquad\quad \text{otherwise.} \end{cases} \tag{1}$$

where $K(t,d)$ is a scalar-valued function $0 < K(t,d) < 1$, and $\delta(t)$ is a distance threshold, both of which vary as the training proceeds. $K(t,d)$ is actually the product of a gain function $\alpha(t)$ and a hat function $h(t,d)$ where $\alpha(t)$ becomes smaller and smaller as the training proceeds, and $h(t,d)$ is usually a bell-shaped function with its width gradually decreased. A Gaussian function like $\exp(-d^2/\sigma^2(t))$ is appropriate for $h(t,d)$ where $\sigma(t)$ controls the width. Once $\sigma(t)$ is defined, we can see it is appropriate to set $\delta(t) = c\sigma(t)$ where c is a positive constant. So there are only two functions $\alpha(t)$ and $\sigma(t)$ that need fine-tuning.

The above training algorithm can effectively adapt a network to learn the implicit order in the input vectors and adjust the weight vectors of the cells so that they respond to the input vectors in an orderly manner, i.e. the winner cell will move continuously in the network as the input vector changes smoothly. The achieve this result, $\alpha(t)$ should be initialized to a large value close to 1 and gradually decrease to a small value of the order of or less than 0.01. Meanwhile, $\sigma(t)$ should be initialized to a value of the order of the radius of the network and gradually decrease to be less than 1.

4 Surface Reconstruction

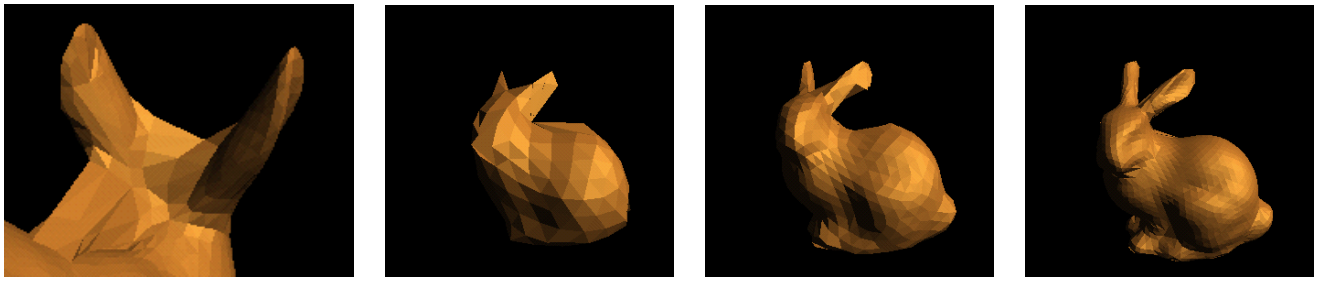
Kohonen's training algorithm for self-organizing maps is suitable for our approach. We consider the vertices in a mesh as the cells in a neural network, the coordinates at each vertex as the three dimensional weight vector at each cell, the input unorganized point set as the set of input vectors. Therefore, a geometric mesh defines an equivalent self-organizing map. The winner cell corresponding to an input point is chosen such that its coordinates are closest to the input point in 3D Euclidean space. This can be formulated as

$$||C_w - x_{in}|| = \min_i ||C_i - x_{in}|| \tag{2}$$

where C_w is the winner cell, x_{in} is the input point. Because of the algorithm's ability to learn the implicit order of the input dataset, neighboring vertices in the final mesh should be close to each other geometrically and the final mesh should look smooth and not have any discontinuities. If we imagine the initial mesh as a rubber sheet, the procedure of learning coordinates for the vertices is like stretching the rubber sheet and wrapping it around the point set so that there are no big gaps between the point set and the rubber sheet. The number of vertices in the mesh is usually different from the number of input points. And the coordinates of a certain vertex in the mesh do not necessarily coincide with those of a particular point from the input.

4.1 Edge Swap

Here we assume a triangular mesh is used. If the real object surface has concave structures, simply applying Kohonen's learning algorithm sometimes has difficulty to approximate those parts well. The reconstructed surface may have some thin elongated faces connecting separate parts and filling up concave structures (Fig. 3(a)).



(a) (b) (c) (d)

Figure 3: (a) Simply applying Kohonen's learning algorithm sometimes has difficulty to approximate concave structures well. The reconstructed surface may have some thin elongated faces connecting separate parts and filling up concave structures. It can be removed by edge swaps. (b)-(d) Intermediate reconstruction with edge swaps at three consecutive resolutions with 320, 1280, 5120 triangles, respectively.

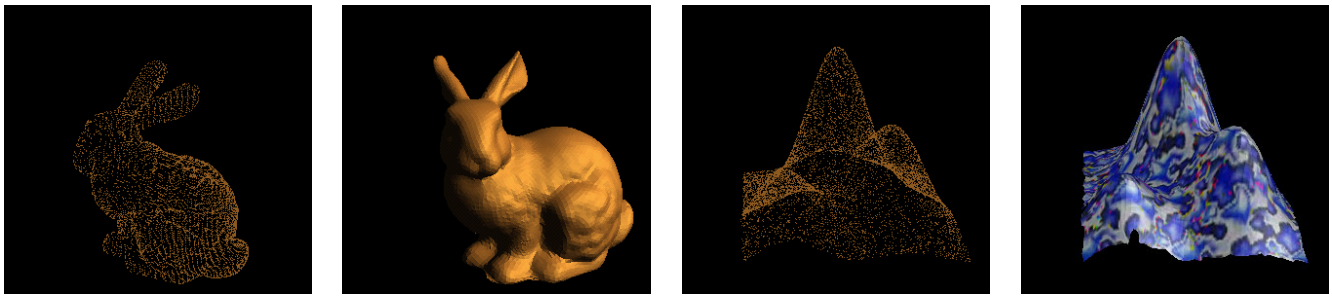


Figure 4: (a) The original dataset of the Stanford bunny with 35947 points, (b) The final reconstruction of the Stanford bunny with 20480 triangles. (c) The original dataset of a height function with 10000 points, (d) The reconstructed surface(2209 rectangles) of the point set in (c) with texture-mapping.

They are not part of the real object surface. It is relatively easy to detect these triangles by calculating the minimum distance between the centroid of a triangle and the input point set. This *minimum distance* for a triangle T is defined as follows.

$$MD(T) = \min_{p \in S} \text{Distance}(p, \text{centroid of } T) \tag{3}$$

where S is the input point set. After applying Kohonen's learning algorithm, we calculate $MD(T)$ for every triangle T in the mesh, and obtain the mean and standard deviation over all triangles. Those elongated triangles that do not belong to the real object surface should have large minimum distance compared to the mean minimum distance.

To remove these problematic triangles, we adopt one mesh operation: *edge swap*. Edge swap can preserve the original topology of the mesh. In [6], Hoppe et al. discussed three elementary operations, *edge collapse*, *edge swap*, *edge split*. They are the only operations needed to transform two arbitrary triangular meshes to each other. However, if we want to transform between two meshes which have the same number of vertices and edges and both of which are 2D manifolds, edge swap is the only operation needed. A simple and classic example of this is using edge swap to transform two different triangulations of a convex polygon.

Theorem: Edge swap can transform two manifold triangular meshes with the same number of vertices and edges to each other.

Proof: There exists a particular sequence of edge swaps that can realize the 0. Choose a vertex V in the first mesh and suppose its corresponding vertex in the second mesh is V'. Swap the edges in the two meshes in such a way that in the end, all the edges in the first mesh are incident to V and all the edges in the second mesh are incident to V'. The desired sequence is obtained by concatenating the sequence of edge swaps for the first mesh with the reversed sequence for the second mesh. The details about how to swap the edges are left out here.

Now that we know edge swap is enough for changing a mesh if the number of vertices and edges, therefore the faces, are kept the same. However, the sequence of edge swaps described in the above proof is not useful in practice. To remove those problematic triangles, we actually use two variants of edge swaps, *single swap* and *double swap*. A single swap is the same as the original edge swap (Fig. 2(a)-(b)). After a single swap, the two triangles sharing the old edge are deleted and two new triangles sharing the new edge are generated. A double swap is two consecutive single swaps but the two swapped edges belong to the same triangle in the original mesh (Fig. 2(c)-(e)).

To choose which edge of a problematic triangle should be swapped, we define a quantity called the *deviation* of an edge as follows.

$$Dev(e) = MD(T_1) + MD(T_2) \tag{4}$$

where e denotes some edge in the mesh, T_1 and T_2 are the two triangles sharing edge e. For a problematic triangle whose minimum distance is larger than a threshold, we choose to swap the edge with the largest deviation. First, a single swap is tried and accepted if and only if the deviation of the new edge is smaller than that of the old one and the minimum distance of at least one of the two new triangles is smaller than the prescribed threshold. If a single swap fails, we continue to try a double swap which means that we keep the first swap and try to swap another edge with the second largest deviation. A double swap is accepted if and only if the second swap satisfies the above conditions. Otherwise, we recover the original mesh and no edge swap happens to the considered triangle.

For the whole mesh, we first obtain the minimum distance of each triangle, then gradually decrease a threshold on the minimum distance from a large value to the mean. At each step of the decrement, single out the list of triangles whose minimum distance is larger than the threshold and try a single (and double swap) on each of them.

There is a minor problem that needs some attention before edge

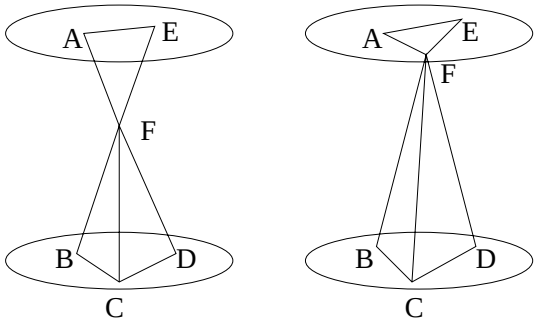


Figure 5: If there is a dangling vertex in the middle of two different parts of the surface, we need to move it close to one part of the surface before an edge swap can happen. The left figure shows the structure before the move, the right one shows the structure after the move.

swap is carried out. There may be a small number of vertices far away from the whole input point set(Fig. 5). Edge swap can not move the triangles incident to them closer to the input points because these vertices pull the centroids of their incident triangles away from the input point set. We need to move these vertices first. We go through each of the vertices which has a nearby point from the input, and move towards it any of its neighboring vertices in the mesh that are far away from the input points. This process is repeated a few times until all the vertices have some nearby point from the input.

4.2 Multiresolution Learning

Overall performance of the algorithm can be improved by applying Kohonen's learning algorithm at multiple resolutions of the mesh. Most low-frequency features can be learned at lower resolutions with much less computational cost. When it comes to the desired resolution, only limited high-frequency details need to be learned. Therefore, the number of iterations can be fairly small at the highest resolution. We do edge swap at each intermediate resolution after the learning algorithm has been applied. So concave structures can also be learned gradually at multiple resolutions. We use either triangular or rectangular meshes. Every time when both learning and edge swap are finished at a certain resolution and a higher resolution is desired, each face in the mesh needs to be split into four smaller faces, each edge needs to be split into two with a new vertex inserted in-between. The total number of faces is quadrupled. And so is the computational cost at the next level.

5 Texture Mapping

Given an arbitrary mesh, the assignment of texture coordinates to every vertex could be painful. In our approach, texture-mapping can be carried out in a natural way for a point set whose topology is equivalent to a disk. We start with a flat rectangular or triangular mesh which is initially embedded in the texture space. The initial 2D coordinates at each vertex can be considered as its pre-assigned texture coordinates. Then Kohonen's learning algorithm is applied on the mesh to learn the geometry of the input point set. When the learning is done, every vertex has correct 3D coordinates as well as texture coordinates. Texture-mapping the final mesh becomes straightforward.

6 Results

We implemented the above surface reconstruction algorithm on a Pentium II 300MHz processor running Linux. We are currently considering two different kind of mesh topology, open surface with one continuous boundary which is homeomorphic to a disk, and closed surface without holes which is homeomorphic to a sphere. We use rectangular mesh for the open surface topology, and triangular mesh for the spherical topology which is initialized as an icosahedron with twenty faces. Each subdivision splits each face into four smaller ones. The number of iterations at each resolution is between 50 and 100. We have tested our algorithm on two datasets, a 10000 point dataset sampled from a height function defined by a mixture of three Gaussian functions (Fig. 4(c)-(d)), and a 35947 point dense dataset of the well-known Stanford bunny (Fig. 3(b)-(d) and 4(a)-(b)). We use the open surface topology for the first dataset, and the closed spherical topology for the other. From the results, we can see our algorithm is quite effective to generate the correct surface reconstruction. The running time of our algorithm is less than half an hour for the smaller dataset, but a little more than one hour for the larger bunny dataset.

7 Conclusions and Future Work

In this paper, we introduced a novel technique for surface reconstruction from unorganized points by applying Kohonen's self-organizing map. A learning procedure is carried out to obtain correct 3D coordinates at each vertex of the surface. Edge swap and multiresolution learning are introduced to make the algorithm more effective and more efficient. The whole algorithm is very simple to implement. Experimental results have shown our techniques are successful. In future, we would like to extend this work to accommodate more sophisticated topology and try different distance metrics, such as geodesic distance, for cells in the self-organizing map. Space subdivision schemes, such as octrees, could be used for accelerating the search for the nearest input point to a mesh vertex.

Acknowledgments

This research was supported by a Multidisciplinary University Research Initiative on three dimensional direct visualization from ONR and BMDO, grant FDN00014-96-1-1200, and Microsoft Graduate Fellowship. The author wishes to thank Marshall Bern for providing the bunny point set, and the reviewers for their valuable comments.

References

[1] Kohonen, T., "The Self-Organizing Map", in *Proceedings of the IEEE*, vol. 78, No. 9, pp.1464-1480, 1990.

[2] Amenta, N., Bern, M., and Kamvysselis, M., " A New Voronoi-Based Surface Reconstruction Algorithm ", in *Proc. of SIGGRAPH'98*, pp.415-421.

[3] Curless, B., and Levoy, M., " A volumetric method for building complex models from range images", in *Proc. of SIGGRAPH'96*, pp.303-312, 1996.

[4] Turk, G., and Levoy, M., " Zippered polygon meshes from range images," in *Proc. of SIGGRAPH'94*, pp.311-318, 1994.

[5] Edelsbrunner, H., and Mücke, D.P., "Three-dimensional Alpha Shapes", in *ACM Transactions on Graphics*, 13, pp.43-72, 1994.

[6] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W., "Mesh Optimization", in *Proc. of SIGGRAPH'93*, pp.19-26.

[7] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W., "Surface reconstruction from unorganized points", in *Proc. of SIGGRAPH'92*, pp.71-78.

[8] Stander, B.T., and Hart, J.C., "Guaranteeing the Topology of an Implicit Surface Polygonization for Interactive Modeling", in *Proc. of SIGGRAPH'97*, pp.279-286.

[9] Yu, Y., "Efficient Visibility Processing for Projective Texture-Mapping", in Journal of *Computers & Graphcs*, Vol. 23, No. 2, (1999), pp. 245-253..

[10] YU, Y., AND WU, H., A Rendering Equation for Specular Transfers and its Integration into Global Illumination. Eurographics'97, in Journal of *Computer Graphics Forum*, 16(3), (1997), pp. 283-292.