Quadrilateral mesh generation from point clouds by a Monte Carlo method

Ágoston Róth
Chair of Numerical Calculus and Statistics, Babeş - Bolyai University
Str. Mihail Kogălniceanu nr. 1
Romania,RO-400084, Cluj-Napoca
agoston_roth@yahoo.com

Imre Juhász
Department of Descriptive Geometry, University of Miskolc
Egyetemváros
Hungary, H3515, Miskolc
agtji@uni-miskolc.hu

ABSTRACT

We present a Monte Carlo method that generates a quadrilateral mesh from a point cloud. The proposed algorithm evolves an initial quadrilateral mesh towards the point cloud which mesh is constructed by means of the skeleton of the input points. The proposed technique proves to be useful in case of relatively complex point clouds that describe smooth and non-self-intersecting surfaces with junctions/branches and loops. The resulted quadrilateral mesh may be used to reconstruct the surfaces by means of tensor product patches such as B-spline or NURBS.

Keywords: Quadrilateral mesh, point cloud, Monte Carlo method.

1 INTRODUCTION

Surface reconstruction from point clouds (scattered data or unorganized set of points in space/plane) is a key issue in several fields of geometric modeling, such as reverse engineering or medical applications. Most often, tensor product surfaces (e.g. NURBS surfaces) are used to describe the surfaces, that require organized point sets, i.e. points arranged into rows and columns. Therefore, quadrilateral meshes has to be extracted from the point cloud, or the cloud has to be approximated by a quadrilateral mesh, then the quadrilateral mesh can be interpolated or approximated by tensor product surface patches.

The usual way to produce a quadrilateral mesh consists of two steps. At first a triangular mesh is generated from the point cloud, then the quadrilateral mesh from the triangular one. There are numerous publications dealing with triangular mesh generation from scattered data, cf. [EM94], [HG97], [HDD+92], [Kós01] and references therein. Concerning the triangular mesh to quadrilateral mesh generation process the reader is referred to [TACSD06] and references therein.

Our objective is to generate quadrilateral mesh from a point cloud without a previous triangular mesh generation. There are some papers (cf. [BF02], [VHK99], [GY95]) that generate quadrilateral mesh directly from unorganized set of points that represent surfaces of simple form (and topology). All of them use neural networks.

Our proposed Monte Carlo method allows more complex geometry. We assume that a point cloud and its topological graph (skeleton) are given. The method consists of the following main steps:

- build an initial rough quadrilateral mesh around the skeleton;
- refine the mesh;
- adjust the mesh to the point cloud using a Monte Carlo method.

The resulted quadrilateral mesh can be interpolated or approximated by tensor product surfaces.

The rest of the paper is organized as follows. In Section 2 there is a short review of skeleton generation methods of spatial point clouds, Section 3 contains our proposed MC method for quadrilateral mesh generation, in Section 4 some test results and examples are presented and a section on future work concludes the paper.

2 SKELETON OF SPATIAL POINT CLOUDS

A skeleton (topological graph) is an abstraction of a 2D or 3D object that represents the shape of the object. It is a graph structure that represents the substantial parts of the object and their relations. The edges of the graph are the representation of such parts of the objects that are roughly of the same thickness/diameter. Skeleton can be considered as an approximation of the centerline of the medial axis. A comprehensive study on skeletons can be found in [CMS07].

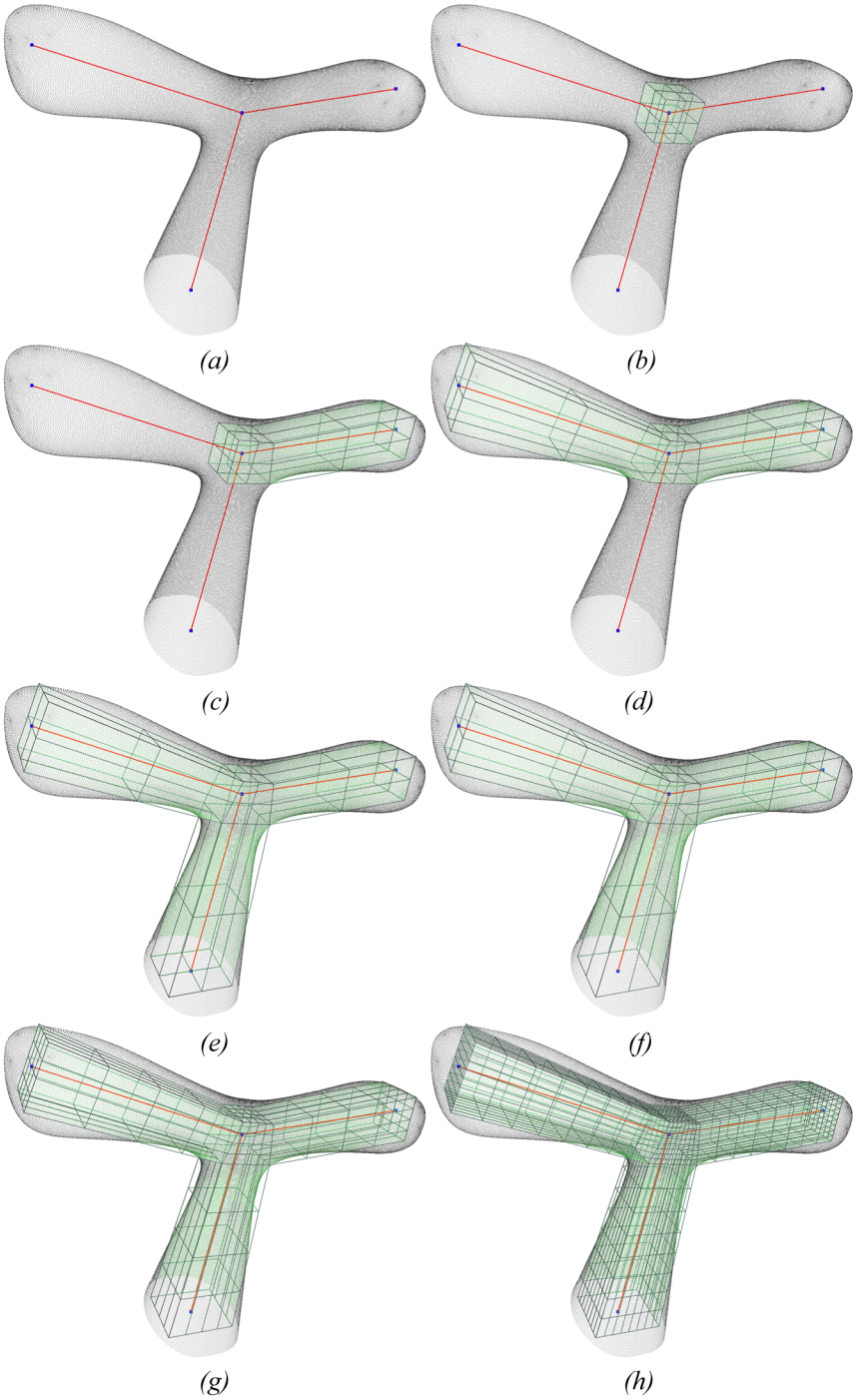

Figure 1: *(a)* A T-shaped scattered point data set is presented together with its skeleton. *(b)* A prism is placed in the T-junction. The prism consists of six face groups. *(c)* The right face group is extruded along its corresponding bone. *(d)* The left face group is also extruded along its corresponding bone. *(e)* The bottom face group is extruded along the T-leg, too. *(f)* The face group at the bottom of the T-leg is deleted. *(g)* All faces in face groups are split in four smaller faces that form new face groups which replace the old ones. *(h)* The previous smoothing step is repeated once again.

There are several methods for the determination of skeletons of 2D or 3D objects. These methods fall into one of the following categories: voxel topology ([GS99], [BND99]), geometric ([DZ04], [AM97], [WML+03], [DS06]), implicit ([BKS01], [CSYB05a], [GG00]) and deformable model evolution ([SLSK07]). A detailed survey on these methods can be found in [CSYB05b].

This paper does not deal with the computation of the skeleton. We assume that the skeleton of the point cloud has been produced already.

3 PROPOSED MONTE CARLO METHOD

Our approach to reconstruct a surface from an unorganized (scattered) point set

$$\mathbf{P} = \left\{ \mathbf{p}_i \in \mathbb{R}^3 : i = 1,2,\ldots,N \right\}$$

is based on a Monte Carlo (MC) method and a flexible quadrilateral mesh data structure that consist of:

- *types* like vertices, oriented halfedges, counterclockwise oriented faces and face groups that consist of four adjacent faces that have a common vertex (i.e. a face group is a fan of quadrilateral faces);
- *methods* for fast neighborhood access, mesh smoothing, face group scaling, rotating, extruding, deleting and vertex merging.

MC methods are computational algorithms that provide approximate solutions to a variety of mathematical problems by performing statistical sampling experiments on a computer. These methods are successfully applied in computer graphics, e.g. in global illumination computing, a comprehensive study of which can be found in [SK99]. Each MC method follow a particular pattern:

- first, we need to define a domain of possible inputs;
- second, we generate inputs randomly from the domain and perform a deterministic computation on them;
- finally, we aggregate the results of the individual computations into the final result.

In what follows, we assume that the skeleton

$$\mathbf{S} = \left\{ (\mathbf{b}_j, A_j) : j = 1,2,\ldots,M \right\}$$

of the point cloud $\mathbf{P}$ is known, where $\mathbf{b}_j \in \mathbb{R}^3$ represents one of the two endpoints of a bone, while $A_j \in \mathscr{P}(\{1,2,\ldots,M\} \setminus \{j\})$ is the adjacency list of the bone node $\mathbf{b}_j$. Using adjacency lists A_j $(j = 1,2,\ldots,M)$, one is also able to create an adjacency matrix

$$A = [a_{kl}]_{k,l=1,2,\ldots,M} \in \mathscr{M}_{M,M}(\{0,1\}),$$

where

$$a_{kl} = \begin{cases} 1, \text{ if bone node } \mathbf{b}_k \text{ is connected to } \mathbf{b}_l, \\ 0, \text{ otherwise.} \end{cases}$$

Naturally, the point cloud $\mathbf{P}$ corresponds to the domain of possible inputs. Based on the adjacency matrix A of the skeleton, we are able to build a rough quadrilateral mesh $\mathbf{Q}$ that will be evolved (i.e. spanned/stretched to the point cloud) by the proposed MC method. Figure 1 presents the steps of the creation process of the rough quadrilateral mesh that is based on the skeleton of a T-shaped point cloud. The vertices $\mathbf{q}_i$ $(i = 1,2,\ldots,n)$ of the mesh $\mathbf{Q}$ can be considered as control points. After the mesh $\mathbf{Q}$ has been created, one can evaluate the average unit normal vectors $\mathbf{n}_i$ $(i = 1,2,\ldots,n)$ that are associated with control points $\mathbf{q}_i$. Later, after every deformation of mesh $\mathbf{Q}$, these unit normal vectors will be reevaluated. The initial unit normal vectors of the quadrilateral mesh associated with the previous T-shaped point cloud are illustrated in Figure 2.

Note, in general $\mathbf{Q}$ is not a fixed sized quadrilateral mesh, however, most of its parts can be decomposed in smaller regular control nets that determine smooth interpolating or approximating patches. The decomposition is not possible around extraordinary points which by definition are control points connected to other than four edges. In the neighborhood of extraordinary points one can use some kind of merging method, e.g. the bicubic T-spline patch merging algorithm provided by [SZBN03]. Figure 3 illustrates two types of extraordinary points.

Points of the cloud $\mathbf{P}$ will be organized in cells using a (balanced) 3-dimensional *kd*-tree $T_{\mathbf{P}}$ which is a space-partitioning data structure useful in case of applications that involve range or nearest neighbor searches.

Consider the notations of Figure 4. The proposed algorithm in each iteration step tends to span the vertex $\mathbf{q}_i = \left(q_x^i, q_y^i, q_z^i\right)^T$ $(i = 1,2,\ldots,n)$ along its unit normal vector $\mathbf{n}_i$ to an optimal point $\mathbf{o}_i \in \mathbb{R}^3$ which approximates the position of the yet unknown nearest cloud point to $\mathbf{q}_i$ as follows.

Let us denote by

$$\alpha_l = l \cdot \frac{\alpha_{\max}}{c},\ l = 1,2,\ldots,c$$

the half angles of a right circular cone sequence

$$\begin{cases} \mathbf{c}_l : [0,\infty) \times [0,2\pi] \to \mathbb{R}^3, \\ \mathbf{c}_l(u,v) = \left(c_l^x(u,v), c_l^y(u,v), c_l^z(u,v)\right)^T, \end{cases}$$

that share the common axis $\mathbf{n}_i$ and apex $\mathbf{q}_i$. The number c of cones and the maximal half angle $\alpha_{\max}$ are user-defined parameters. If $\mathbf{k}' = \mathbf{n}_i = \left(k_x', k_y', k_z'\right)^T$, $\mathbf{j}' = \mathbf{k}' \times (1,0,0)^T = \left(j_x', j_y', j_z'\right)^T$ and $\mathbf{i}' = \mathbf{j}' \times \mathbf{k}' = \left(i_x', i_y', i_z'\right)^T$,

then this cone sequence is defined by the matrix equation

$$\begin{bmatrix} \mathbf{c}_l(u,v) \\ 1 \end{bmatrix} = \begin{bmatrix} c_l^x(u,v) \\ c_l^y(u,v) \\ c_l^z(u,v) \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} i_x' & j_x' & k_x' & q_x^i \\ i_y' & j_y' & k_y' & q_y^i \\ i_y' & j_y' & k_y' & q_z^i \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u\cos v \\ u\sin v \\ u\tan\left(\frac{\pi}{2}-\alpha_l\right) \\ 1 \end{bmatrix}.$$

In case of each cone $\mathbf{c}_l$ $(l = 1,2,\ldots,c)$, let $r+1$ $(r \geq 0)$ be the number of fixed and uniformly distributed generators

$$\mathbf{c}_{lg}(u) = \mathbf{c}_l\left(u, \frac{2g\pi}{r}\right),\ g = 0,1,\ldots,r,\ u \in [0,\infty).$$

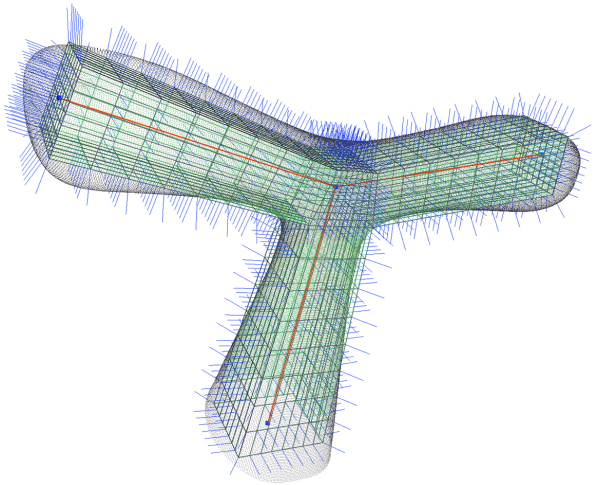

Figure 2: Initial average unit normal vectors of a quadrilateral mesh are presented

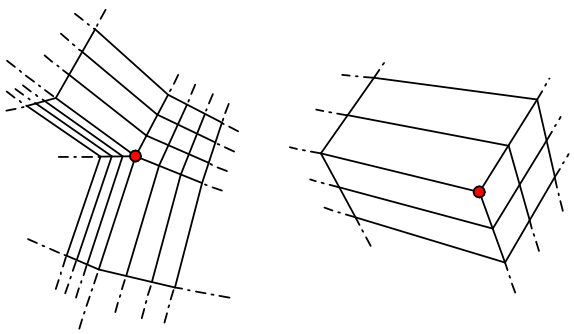

Figure 3: Two parts of a quadrilateral mesh are shown. The highlighted vertices of the mesh are extraordinary points

Parameter r is also user-definable. We also introduce the set of rays

$$\begin{aligned} \mathbf{R} = \{&\mathbf{r}_k(u) : u \in [0,\infty), \\ &k = 1,2,\ldots,c\cdot(r+1)+1\} \\ = \{&\mathbf{q}_i + u\cdot\mathbf{n}_i,\ \mathbf{c}_{lg}(u) : \\ &l = 1,2,\ldots,c; \\ &g = 0,1,\ldots,r,\ u \in [0,\infty)\} \end{aligned}$$

the elements of which may intersect several adjacent point cells around control point $\mathbf{q}_i$.

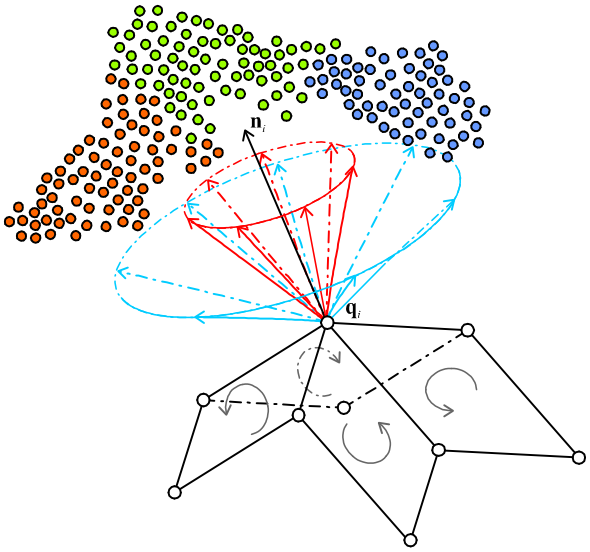

Figure 4: The image depicts a set of uniformly distributed rays (generators) of a sequence of right and circular cones that share the common axis $\mathbf{n}_i$ and apex $\mathbf{q}_i$. The colored dots represent different cells of the 3-dimensional kd-tree that was associated with the point cloud

Let $m \geq 1$ either be a user defined or cell based number of random point samples and consider the set of indices $K = \{k_1,k_2,\ldots,k_z\} \subseteq \{1,2,\ldots,c\cdot(r+1)+1\}$ for which the ray $\mathbf{r}_{k_j}$ $(j = 1,2,\ldots,z)$ intersects a point cell. Note, m is much less than the number of possible input points within a cell. Also, it is possible that different rays intersect the same cell of input points. Each ray $\mathbf{r}_{k_j}$ generates a minimal projection

$$d_j = \min_{l=1,2,\ldots,m}\left\{\mathbf{r}_{k_j}^0 \cdot \mathbf{a}_l\right\},$$

where the vector $\mathbf{r}_{k_j}^0$ is the unit direction of ray $\mathbf{r}_{k_j}$ and points $\mathbf{a}_l$ $(l = 1,2,\ldots,m)$ are random sample points from the cell which is intersected by the ray $\mathbf{r}_{k_j}$.

Let

$$D_\varepsilon = \left\{d_j : |d_j| < \varepsilon,\ j = 1,2,\ldots,z\right\}$$

be the set of feasible minimal projections, where $\varepsilon > 0$ is also a user defined parameter which determines the

maximal distance of the acceptable point cells from control point $\mathbf{q}_i$. The average of feasible minimal projections

$$\bar{\delta}_\varepsilon = \frac{1}{|D_\varepsilon|} \sum_{\delta \in D_\varepsilon} \delta$$

determines the current optimal point

$$\mathbf{o}_i = \mathbf{q}_i + \bar{\delta}_\varepsilon \cdot \mathbf{n}_i$$

that will be the new position of $\mathbf{q}_i$. After repositioning $\mathbf{q}_i$ its average unit normal vector $\mathbf{n}_i$ will also be reevaluated. Such an iterative process is presented in Figure 5.

This generational process is repeated until a termination condition has been reached. Common terminating conditions are: a fixed number of iterations is reached or the error of the current solution (i.e. the quadrilateral mesh) is at such a level that successive iterations no longer produce better results.

The error of the evolving mesh $\mathbf{Q}$ can be calculated as follows. Suppose, the 3-dimensional *kd*-tree $T_{\mathbf{P}}$ consists of point cells

$$\mathbf{Z}_l = \left\{ \mathbf{p}_{lw} = (x_{lw}, y_{lw}, z_{lw})^T : w = 1,2,\ldots,W_l \right\},$$
$$l = 1,2,\ldots,L.$$

Naturally, the point cells $\mathbf{Z}_l$ ($l = 1,2,\ldots,L$) are disjoint sets and the union of them results the point cloud $\mathbf{P}$.

Let us denote by

$$\mathbf{g}_l = \frac{1}{|\mathbf{Z}_l|} \sum_{\mathbf{p} \in \mathbf{Z}_l} \mathbf{p} = \left(g_x^l, g_y^l, g_z^l \right)^T \in \mathbb{R}^3,$$

the center of gravity of the cell $\mathbf{Z}_l$ ($l = 1,2,\ldots,L$) and by

$$\mathbf{t}_l = \left(t_x^l, t_y^l, t_z^l \right)^T \in \mathbb{R}^3$$

the unit normal vector of the plane π_l that contains the point $\mathbf{g}_l$ and it is parallel to least square plane determined by the cell $\mathbf{Z}_l$. Components t_x^l, t_y^l and t_z^l are obtained by solving the symmetric homogeneous system

$$\begin{bmatrix} \beta_{00}^l & \beta_{01}^l & \beta_{02}^l \\ \beta_{01}^l & \beta_{11}^l & \beta_{12}^l \\ \beta_{02}^l & \beta_{12}^l & \beta_{22}^l \end{bmatrix} \begin{bmatrix} t_x^l \\ t_y^l \\ t_z^l \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad (1)$$

with the constraint

$$\left(t_x^l \right)^2 + \left(t_y^l \right)^2 + \left(t_z^l \right)^2 = 1,$$

where

$$\beta_{00}^l = \sum_{w=1}^{W_l} \left(x_{lw} - g_x^l \right)^2,$$
$$\beta_{11}^l = \sum_{w=1}^{W_l} \left(y_{lw} - g_y^l \right)^2,$$
$$\beta_{22}^l = \sum_{w=1}^{W_l} \left(z_{lw} - g_z^l \right)^2,$$
$$\beta_{01}^l = \sum_{w=1}^{W_l} \left(x_{lw} - g_x^l \right) \left(y_{lw} - g_y^l \right),$$
$$\beta_{02}^l = \sum_{w=1}^{W_l} \left(x_{lw} - g_x^l \right) \left(z_{lw} - g_z^l \right),$$
$$\beta_{12}^l = \sum_{w=1}^{W_l} \left(y_{lw} - g_y^l \right) \left(z_{lw} - g_z^l \right).$$

With this constraint, it is readily seen that the solution to the system (1) is an eigenvalue problem. The three eigenvectors are mutually orthogonal and define three sets for components t_x^l, t_y^l and t_z^l. These three sets represents the best, intermediate and worst planes. In our case, we want to choose for $\mathbf{t}_l$ the eigenvector associated with the smallest eigenvalue. The cyclic Jacobi numerical method can be used to find the eigenvalues and the eigenvectors of the symmetric matrix appeared in (1).

If $\mathbf{Z}_l$ ($l \in \{1,2,\ldots,L\}$) is the nearest cell to the control point $\mathbf{q}_i$ ($i \in \{1,2,\ldots,n\}$), then

$$e_i = |\mathbf{t}_l \cdot (\mathbf{q}_i - \mathbf{g}_l)|$$

denotes the distance between $\mathbf{q}_i$ and the plane π_l. Now, the average value

$$\bar{e} = \frac{1}{n} \sum_{i=1}^{n} e_i$$

is the error level of the current quadrilateral mesh $\mathbf{Q}$.

In what follows, we examine the complexity of the proposed algorithm. Building a static kd-tree from point cloud $\mathbf{P} = \{\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_N\}$ takes $O\left((N+1)\log^2(N+1)\right)$ time if an $O((N+1)\log(N+1))$ sort is used to compute the median at each level. The complexity is $O((N+1)\log(N+1))$ if a linear median-finding algorithm such as the one described in [CLR90] (Chapter 10) is used. A ray-shooting/tracing can be done by an $O(\log(N+1))$ operation.

Thus, the runtime complexity of the algorithm is

$$\begin{aligned} & O((N+1)\log(N+1)) \\ & + O\Big((c \cdot (r+1)+1) \cdot \log(N+1) \cdot m \\ & + \max_{l=1,2,\ldots,L} W_l + 3^2 \Big) \cdot n \cdot I \Big) \\ & = O((N+1+n \cdot I)\log(N+1)), \end{aligned}$$

 ISBN 978-80-86943-93-0

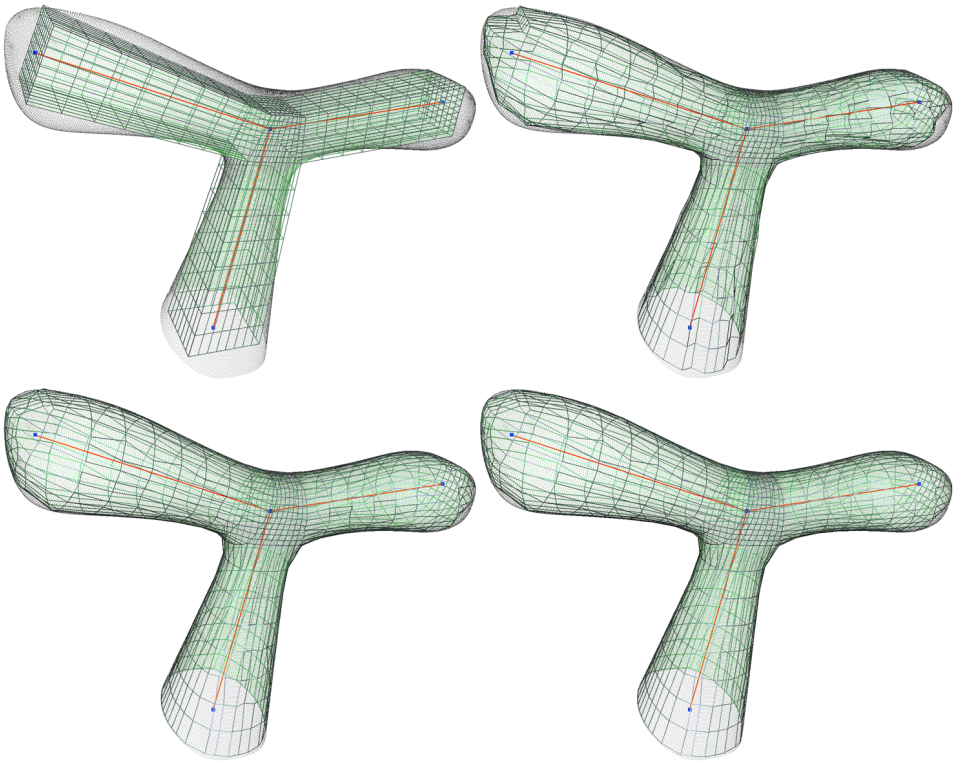

Figure 5: Here we can follow-up the evolution of the quadrilateral mesh after successive iterations of the proposed MC algorithm

where:

- I is the number of iterations/steps performed by the method;
- n is the number of control points within control net $\mathbf{Q}$;
- $c \cdot (r+1)+1$ is the number of rays associated with an evolving control point;
- m is the number of random point samples from a point cell;
- $\max_{l=1,2,\ldots,L} W_l < N+1$ is the maximum number of cloud points within an arbitrary point cell;
- $O\left(3^2\right)$ is the complexity of Jacobi cyclic method that calculates the eigenvectors of a symmetric matrix of dimension 3×3.

4 RESULTS AND EXAMPLES

Figure 6 shows the main steps of the algorithm for a complex geometry. The parameters of the algorithm were set as follows:

- the number of input points is 182274, the bounding box of the model is $[-6.8, 6.8] \times [-7.83, 7.83] \times [-7.83, 7.0]$;
- the maximal half angle is $\alpha_{\max}=\pi/18$;
- the number of cones is 4 while the number of fixed generators/rays on each cone is 6;
- the maximal distance of acceptable point cells is $\varepsilon = 2$;
- for each vertex of the mesh the number of random samples is 10;
- the final error level is 0.056.

In case of the point cloud depicted in Figure 7 most of the parameters of the algorithm were set as in the previous example. The differences consist in:

- the number of input points is 413696, while the bounding box of the model is $[-24.88, 4.43] \times [-17.25, 10.91] \times [-1.71, 7.41]$;
- the maximal distance of acceptable point cells is $\varepsilon = 0.8$;
- after approximately 20 iterations the error level of the mesh is 0.051.

5 CONCLUSION AND FUTURE WORK

As it is illustrated in Section 4 the proposed algorithm is able to generate a quadrilateral mesh from point clouds that represent quite complex geometry. However, there

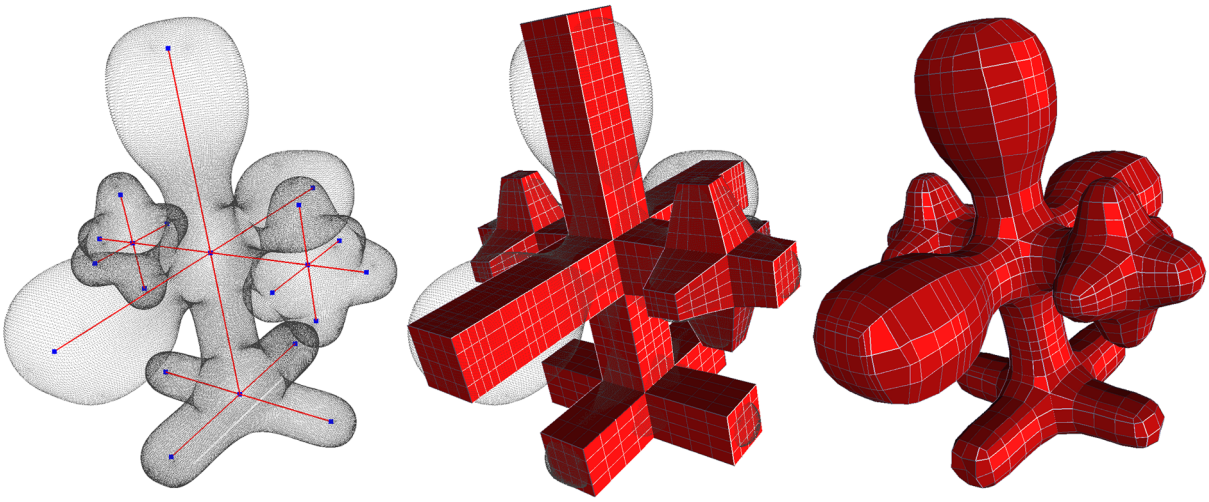

Figure 6: The main steps of the algorithm in the case of a complex model

are several problems to be solved. At the moment, the proposed algorithm is not able to generate a correct quadrilateral mesh in case of point clouds that either describe very detailed surfaces or have junctions the branches of which form very small acute angles. In such cases, the resulted quadrilateral mesh will have self intersections in the neighborhoods of arms due to the fact that adjacent vertices of the evolving mesh in vicinities of junctions are moved along their average unit normal vectors. In this phase the initial rough quadrilateral mesh is constructed manually by means of the given skeleton. It is highly probable that an automatic initial mesh generation will cause undesired effects like twists, self intersections, etc.

In the immediate future we plan to improve the presented algorithm by introducing more constraints and penalty functions to handle such disadvantages. Unfortunately/naturally, we have performance hits in case of point clouds that belong to very detailed surfaces, when the construction of the initial rough quadrilateral meshes needs a large number of (local) smoothing/refining steps that will raise exponentially the size of the allocated memory and the duration of the evolution.

ACKNOWLEDGEMENTS

The second author would like to thank the Hungarian Scientific Research Fund (OTKA T 048523) for its support.

REFERENCES

[AM97] D. Attali and A. Montanvert. Computing and simplifying 2d and 3d skeletons. *Comp. Vision and Image Underst.*, 67(3):156–169, 1997.

[BF02] J. Barhak and A. Fischer. Adaptive reconstruction of freeform objects with 3d som neural network grids. *Computers & Graphics*, 26:745–751, 2002.

[BKS01] I. Bitter, A. Kaufman, and M. Sato. Penalizeddistance volumetric skeleton algorithm. *TVCG, IEEE*, 7(3):195–206, 2001.

[BND99] G. Borgefors, I. Nystrom, and G. Dibaja. Computing skeletons in three dimensions. *Pattern Recognition*, 32(7):1225–1236, 1999.

[CLR90] Th. H. Cormen, Ch. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1990.

[CMS07] N. D. Cornea, P. Min, and D. Silver. Curve-skeleton properties, applications and algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):530–548, 2007.

[CSYB05a] N. Cornea, D. Silver, X. Yuan, and R. Balasubramanian. Computing hierarchical curveskeletons of 3d objects. *The Visual Computer*, 21(11):945–955, 2005.

[CSYB05b] N. Cornea, D. Silver, X. Yuan, and R. Balasubramanian. Curve-skeleton applications. *Visualization, IEEE*, pages 95–102, 2005.

[DS06] T. K. Dey and J. Sun. Defining curve-skeletons with medial geodesic function. *SGP*, pages 143–152, 2006.

[DZ04] T. K. Dey and W. Zhao. Approximate medial axis as a voronod' subcomplex. *Computer-Aided Design*, 36(2):195–202, 2004.

[EM94] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.

[GG00] P. Golland and W. Grimson. Fixed topology skeleton. *Comp. Vision and Pattern Recog., IEEE*, pages 1010–1017, 2000.

[GS99] N. Gagvani and D. Silver. Parameter-controlled volume thinning. *Graph. Models and Image Proc.*, 61(3):149–164, 1999.

[GY95] P. Gu and X. Yuan. Neural network approach to the reconstruction of freeform surfaces for reverse engineering. *Computer-Aided Design*, 27(1):59–64, 1995.

[HDD+92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics (SIGGRAPH'92)*, pages 19–26, 1992.

[HG97] P. S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. Course notes, course

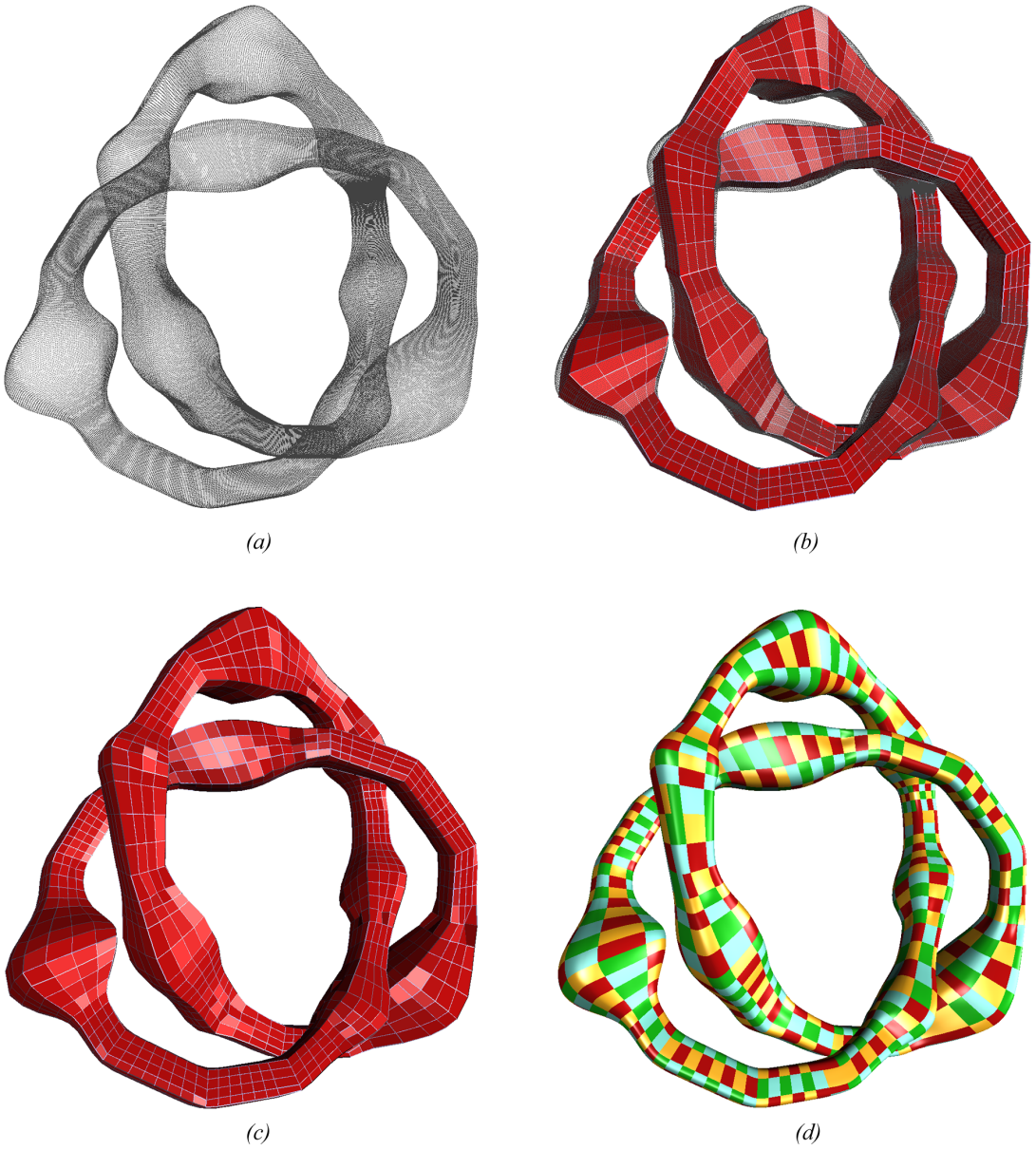

Figure 7: *(a)* Point cloud of a translational surface the directrix of which is a torus knot *(b)* The initial quadrilateral mesh *(c)* The evolving mesh after approximately 20 iteration of the proposed MC method *(d)* The quadrilateral mesh (c) has no extraordinary points, thus it can be decomposed into smaller control nets of the size 4×4 that can be used to generate a C^2-continuous bicubic uniform periodic B-spline surface

25, SIGRAPH 97, 1997.

[Kós01] G. Kós. An algorithm to triangulate surfaces in 3d using unorganized point clouds. *Computing Suppl.*, 14:219–232, 2001.

[SK99] L. Szirmay-Kalos. Monte-carlo methods in global illumination. Technical report, Vienna University of Technology, 1999.

[SLSK07] A. Sharf, T. Lewiner, A. Shamir, and L. Kobbelt. Adaptive subdivision and the length and energy of bézier curves. *Computer Graphics Forum*, 26(3):323–328, 2007.

[SZBN03] T. W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri. T-splines and T-NURCCs. 22(3):477–484, 2003.

[TACSD06] Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun. Designing quadrangulations with discrete harmonic forms. Technical report, Symposium on Geometry Processing, Eurographics, 2006.

[VHK99] L. Várady, M. Hoffmann, and E. Kovács. Improved free-form modelling of scattered data by dynamic neural networks. *Journal for Geometry and Graphics*, 3(2):177–81, 1999.

[WML+03] F. C. Wu, W. C. Ma, P. C. Liou, R. H. Laing, and M. Ouhyoung. Skeleton extraction of 3d objects with visible repulsive force. pages 409–413, 2003.