Curve Reconstruction from Unorganized Points [*]

In-Kwon Lee

Institute of Geometry, Vienna University of Technology
Wiedner Hauptstrasse 8–10, A–1040 Wien, Austria
iklee@geometrie.tuwien.ac.at

Abstract

We present an algorithm to approximate a set of unorganized points with a simple curve without self-intersections. The moving least-squares method has a strong power to reduce a point cloud to a thin curve-like shape which is a near-best approximation of the point set. In this paper, an improved moving least-squares technique is suggested using minimum spanning tree, region growing and refining iteration. After thinning a given point cloud using the improved moving least-squares technique, we can easily reconstruct a smooth curve. As an application, a pipe surface reconstruction algorithm is presented.

Keywords: curve reconstruction, reverse engineering, moving least-squares, unorganized points, pipe surface

1 Introduction

Reconstructing a curve or a surface from a point set is one of the most important problems in reverse engineering of geometric models. In some cases, the curve reconstruction plays an important role in the surface reconstruction problem [3, 17, 18, 19]. In this paper, we focus on the reconstruction of a curve from an *unorganized* point cloud having no ordering of the point elements. An attractive feature of our solution is that the algorithm can be extended to any dimension.

The motivation of this work comes from a series of recent researches to reconstruct surfaces of revolution, helical surfaces, spiral surfaces, profile surfaces and ruled surfaces from a set of points [3, 17, 18, 19]. In these papers, principal motion parameters such as motion axis and pitch are computed using line geometry. Using the motion trajectories, the data points are projected onto an appropriate plane and approximated with a curve which is the profile curve of the reconstructed surface. Figure 1 shows an example of the reconstruction of a surface of revolution. After a rotation axis of the surface of revolution is computed, a profile curve can be constructed by: i) rotating all points into a plane through the rotation axis, ii) approximating the projected point set with a curve.

Pottmann and Randrup [17] used a method based on *image thinning* to approximate the profile curve. After defining an appropriate grid on the plane, the pixels including one or more points are filled with black. Using conventional image thinning algorithms, the medial axis of this binary image can be computed easily. Finally, we can approximate the medial axis with a smooth curve. However, this method has some disadvantages:

- It is not easy to determine the size of pixel of the image. If the pixel is too large, the medial axis may not represent the best approximation curve of the point set. If the pixel is too small, it is possible for the point set to be separated into several different components.
- For open curves, the medial axis does not represent the shapes of the point set near the end points of the curve.

[*]Technical Report No. 55, Institute of Geometry, Vienna University of Technology, August 1998. The research was supported in part by '97 overseas postdoctoral fellowship program of KOSEF (Korea Science and Engineering Foundation).

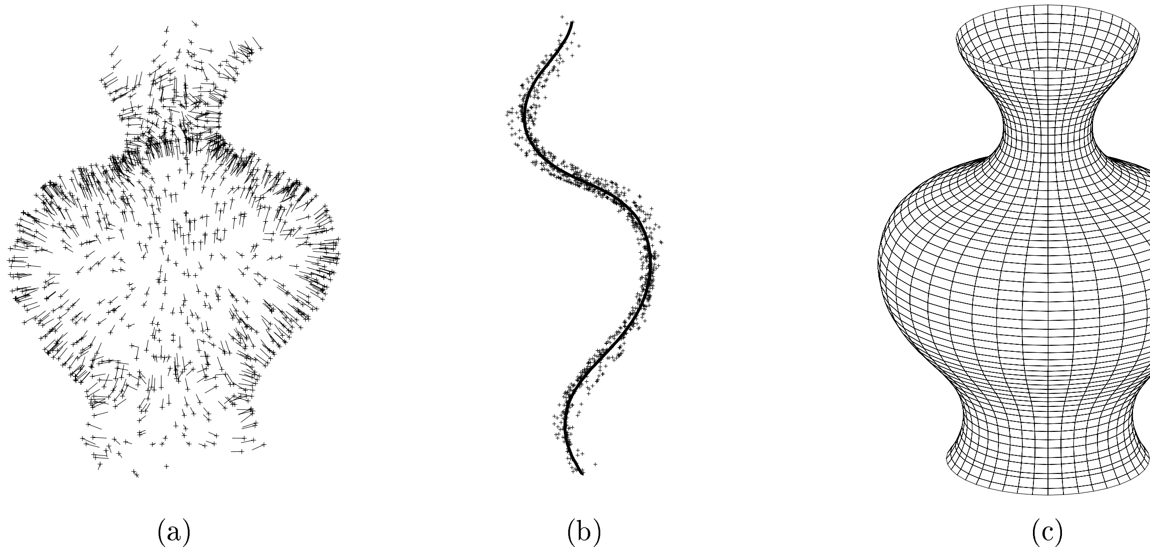


Figure 1: Reconstruction of surface of revolution: (a) data points and estimated normal vectors, (b) projected points and approximating curve, (c) reconstructed surface of revolution

Several approaches have been suggested for the reconstruction of surfaces from unorganized point sets [2, 7, 8, 15]. For curve reconstruction, there are many solutions in case the ordering of points is known. However, there have been very little researches for curve reconstruction from an unorganized point set. Fang et al. [6] used a method based on spring energy minimization to approximate the unorganized point set with a curve, which needs a good initial guess of the solution. Dedieu et al. [4] presented an algorithm for ordering unorganized points assuming that all points are on the reconstructed curve; thus, this method is not appropriate for the point cloud.

One simple solution is finding a polygonal boundary such as α-shape [5] which represents the shape of the point set. Then, any algorithm to compute the medial axis of this polytop can be applied to find a polygonal skeleton of the point set which can be easily approximated a smooth curve. However, as we saw in many situations, the computation of the medial axis of a polytop is troubled with the numerical problems especially when the polytop is very complex such as an α-shape in higher dimension. Furthermore, like image thinning, this approach cannot reflect the shape nearby the end points of the open curves.

To solve the curve reconstruction problem, we focus on the problem of making the point cloud as thin as possible with respect to the density and the shape of the point set. After the original point set is reduced to a sufficiently thin cloud, ordering the points in this thin cloud is not hard as we will show later. The method of *moving least-squares* [11, 12] is a very powerful tool for the purpose of thinning the point set. The basic idea of moving least-squares is to compute a simple regression curve/surface $\mathbf{C}_i$ for each data point $\mathbf{P}_i$ which locally fits a certain neighborhood of $\mathbf{P}_i$ using a weighted regression scheme. Then $\mathbf{P}_i$ is moved to a new position $\mathbf{P}'_i$ on $\mathbf{C}_i$. Clearly, moving least-squares is near-best in the sense that the local error is bounded with the error of a local best approximation. This simple idea works well in many cases. For a thin cloud of points, moving least-squares generates very good results (see Figure 2). However, there are some serious difficulties in moving least-squares technique for some cases such as varying thickness of the point set and the effects from unwanted neighboring points (see Figure 3). In this paper, we suggest an improved version of moving least-squares to overcome these difficulties. Experimental results show that great improvements are achieved by our simple modification of moving least-squares. In this paper, we assume that the point cloud may represent a simple smooth curve without self-intersections.

This paper is organized as follows. In Section 2 we introduce moving least-squares technique and describe some difficulties in this method. Section 3 presents an improved version of the moving least-squares to overcome the difficulties. The extension to 3D or higher dimension is presented in Section 4. In Section 5, an algorithm is described to compute an approximation curve of a thin point cloud which is generated by moving least-squares. In Section 6, we present an interesting application of the curve

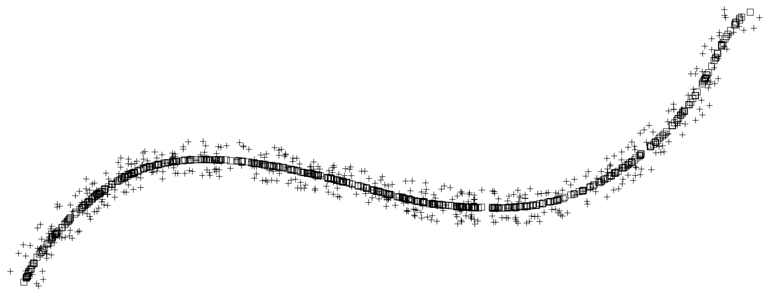


Figure 2: Thinning a 2D point cloud using moving least-squares: the points represented with boxes are the result of moving least-squares

approximation – the reconstruction of pipe surfaces. Finally, in Section 7 we describe the conclusion of this work and suggest some future research directions.

2 Background

The concept of moving least-squares has been used in many applications such as scattered data interpolation and smoothing [10, 11, 12, 14, 21]. For each data point, a simple curve or surface is computed which fits some neighborhood of the data point using a weighted regression scheme. Then, the data point is moved to a new position on this approximated curve or surface. The moving least-squares method is near-best in the sense that the local error is bounded with the error of a local best polynomial approximation (see [11, 12] for the detailed error analysis of moving least-squares).

Let $\mathcal{S} = \{\mathbf{P}_i = (x_i, y_i) \mid i = 1, ..., N\}$ be a point set in R^2. For a point $\mathbf{P}_*$, a local regression line, $\mathbf{L}_* : y = ax + b$, can be computed by minimizing a quadratic function:

$$D_l = \sum_{i=1}^{N} (ax_i + b - y_i)^2 w_i, \tag{1}$$

where w_i is a nonnegative weight for each point $\mathbf{P}_i$ computed by an appropriate weighting function. (One can compute a regression line which minimizes the sum of the squares of orthogonal distances between the data points and the line, see Appendix.) We can choose any weighting function which generates larger penalty for the points far from $\mathbf{P}_*$. One of our choices is

$$w_i = e^{-r^2/H^2}, \tag{2}$$

where $r = \|\mathbf{P}_i - \mathbf{P}_*\|^2$, and H is a prescribed real constant. From this weighted regression, we can compute the local best regression line $\mathbf{L}_*$ for $\mathbf{P}_*$. Consider a transformation M which transforms the whole point set into a new coordinate system where the x axis is parallel to the line $\mathbf{L}_*$ and $\mathbf{P}_*$ is a new origin. Let $\hat{\mathcal{S}} = \{\hat{\mathbf{P}}_i = (\hat{x}_i, \hat{y}_i) \mid i = 1, ..., N\}$ be the transformed point set. The local quadratic regression curve

$$\mathbf{Q}_* : \hat{y} = a\hat{x}^2 + b\hat{x} + c \tag{3}$$

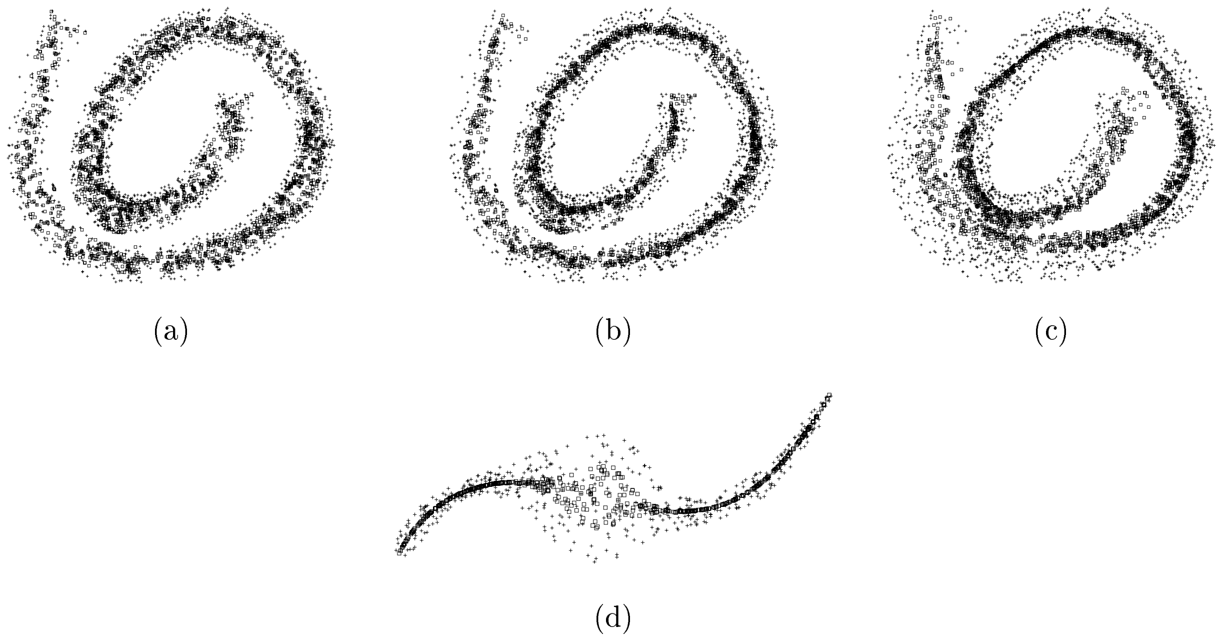
for $\hat{\mathbf{P}}_*$ can be computed by minimizing

$$D_q = \sum_{i=1}^{N} (a\hat{x}_i^2 + b\hat{x}_i + c - \hat{y}_i)^2 w_i. \tag{4}$$

Note that the projection of $\hat{\mathbf{P}}_*$ onto $\mathbf{Q}_*$ is $(0, c)$. Finally, $\mathbf{P}_*$ is moved to a new position computed by inverse-transformation M^{-1} of $(0, c)$.

Instead of using the whole point set for each local regression, we can restrict the neighborhood of each point by introducing a cubic function [22]:

$$w_i = \begin{cases} 2\dfrac{r^3}{H^3} - 3\dfrac{r^2}{H^2} + 1 & \text{if } r < H \\ 0 & \text{if } r \geq H \end{cases} \tag{5}$$



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

Figure 3: Difficulties in moving least-squares

where $r = \|\mathbf{P}_i - \mathbf{P}_*\|^2$. The weighting function in Equation (5) enforces the weights of the points to vanish which are outside of the open circle of radius H with the center $\mathbf{P}_*$. Thus, we can compute a regression line or quadratic curve using only the set of points whose distances from $\mathbf{P}_*$ are less than H.

This simple method works well in many cases to reduce a point set to a thin cloud as we saw in Figure 2. However, the pure moving least-squares method does not work well in some cases:

- If H is too small, the local regressions do not reflect the thickness of the point cloud. Thus, the resulting points are scattered (Figure 3(a)).
- Increasing H may help to make the thin point cloud. However, with a large H, the local regression may include some unwanted points, which can be a reason of failure of the algorithm (Figure 3(b)(c)).
- Fixed H cannot be applied to the whole point cloud which has varying thickness (Figure 3(d)).

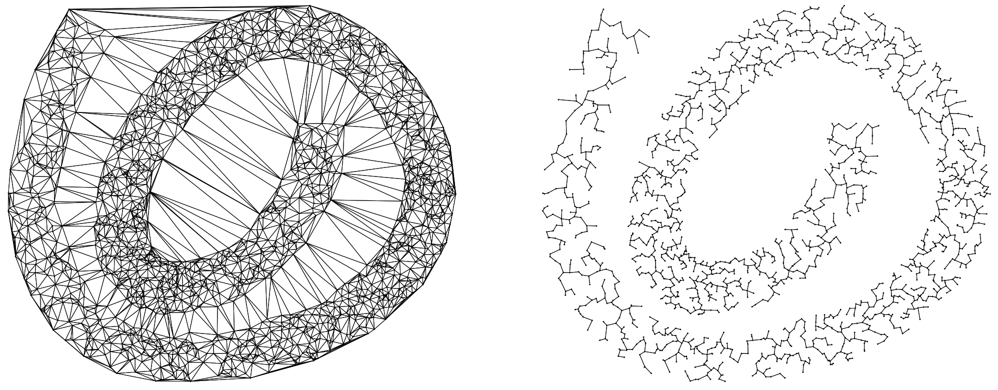
3 Improved moving least-squares

To prevent the effects from unwanted points in the local regressions, we need to make a certain structure (as simple as possible) for the point set to define the connectivity of the point elements. The *minimum spanning tree* (MST) is a good candidate. Let $\mathcal{S} = \{\mathbf{P}_i = (x_i, y_i) \mid i = 1, ..., N\}$ be a point set. Consider a graph $\mathcal{G} = (\mathcal{S}, \mathcal{E})$ having total connectivity among all point elements, i.e., $\mathcal{E} = \{(\mathbf{P}_i, \mathbf{P}_j) \mid i, j = 1, ..., N,\ i \neq j\}$. The MST of $\mathcal{G}$ is a tree (thus, having no cycle) connecting all points in $\mathcal{S}$ so that the sum of its edge lengths is minimum. MST can be computed by well-known Kruskal's or Prim's algorithm [1] in $O(N^2)$ time.

Instead of using total connectivity graph, we can use *Delaunay triangulation* (DT) [20] to compute MST due to the well-known fact that MST is a subgraph of DT. Figure 4 shows an example of DT and MST of a point set in 2D.

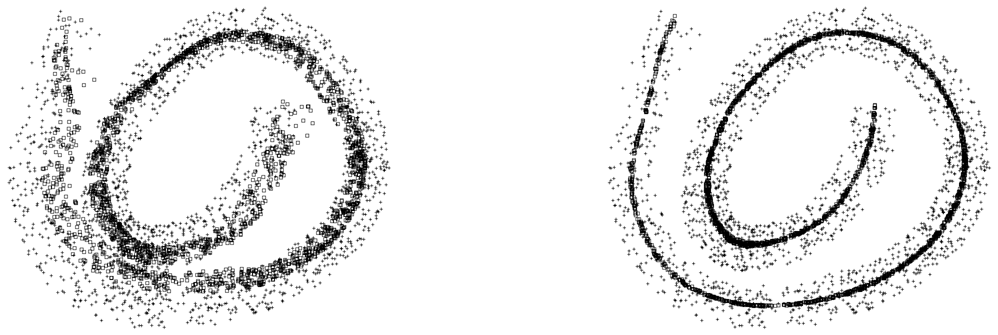
Now, in the local regression for $\mathbf{P}_*$, we use the MST to collect the neighboring points. Let $\mathcal{A}$ denote a set of neighboring points of $\mathbf{P}_*$ with respect to a distance H. The following recursive procedure is used to compute $\mathcal{A}$ by initially assigning an empty set to $\mathcal{A}$ and calling **Collect**($\mathbf{P}_*$, H, $\mathcal{A}$).

```
**Collect**($\mathbf{P}$, $H$, $\mathcal{A}$)
    **begin**
```



(a) (b)

Figure 4: (a) Delaunay triangulation, (b) minimum spanning tree



(a) (b)

Figure 5: (a) Moving least-squares without MST, (b) moving least-squares with MST

```
        $\mathcal{A} \Leftarrow \mathcal{A} \cup \{\mathbf{P}\}$
        **for** each edge $(\mathbf{P}, \mathbf{P}_j)$ in MST **do**
            **if** $\|\mathbf{P} - \mathbf{P}_j\| < H$
                **then Collect**$(\mathbf{P}_j, H, \mathcal{A})$
    **end**
```

Figure 5 shows the effect of the collecting algorithm applied to moving least-squares by comparing two results with and without the MST structure.

The size of H must be determined to reflect the thickness of the point cloud. *Correlation*, developed in probability theory, provides a measure of the degree of linear dependence between two variables [16]. Let X and Y be two random variables. The *covariance* of X and Y is defined by

$$Cov(X, Y) = E[(X - E(X))(Y - E(Y))] = E(XY) - E(X)E(Y), \tag{6}$$

where $E[\circ]$ denotes an expectation (average) of a random variable $\circ$. Correlation is a standardized covariance which is easier to interpret:

$$Corr(X, Y) = \frac{Cov(X, Y)}{SD(X)SD(Y)}, \tag{7}$$

where $SD(\circ)$ represents a standard deviation of a random variable $\circ$. A correlation has a value between -1 and $+1$ representing the degree of linear dependence between X and Y. For example, if $Y = X + b$ (b is a constant) exactly, then $Corr(X, Y) = 1$. Conversely, if $Y = -X + b$ exactly, then $Corr(X, Y) = -1$. Figure 6 shows some correlations $\rho = Corr(X, Y)$ of various sets of points.

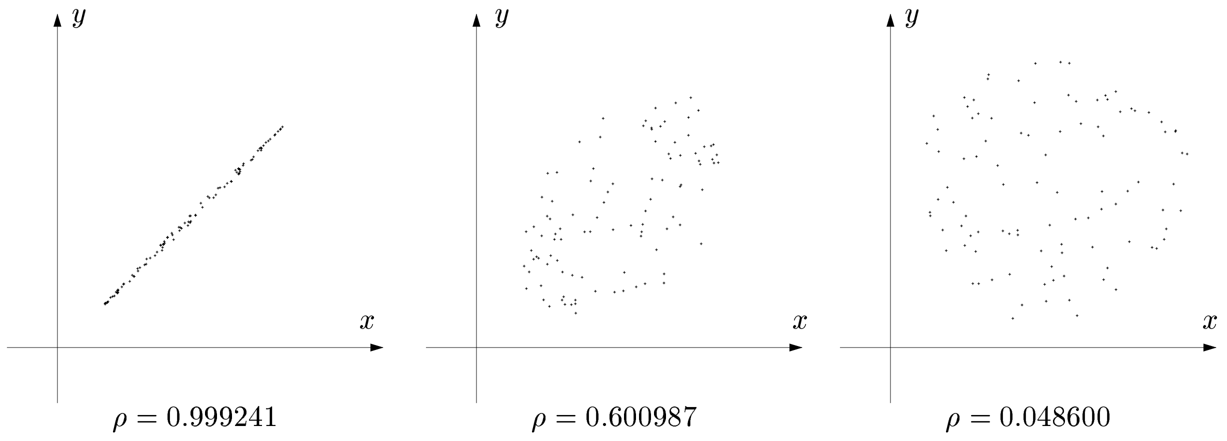


Figure 6: Correlations of various point sets

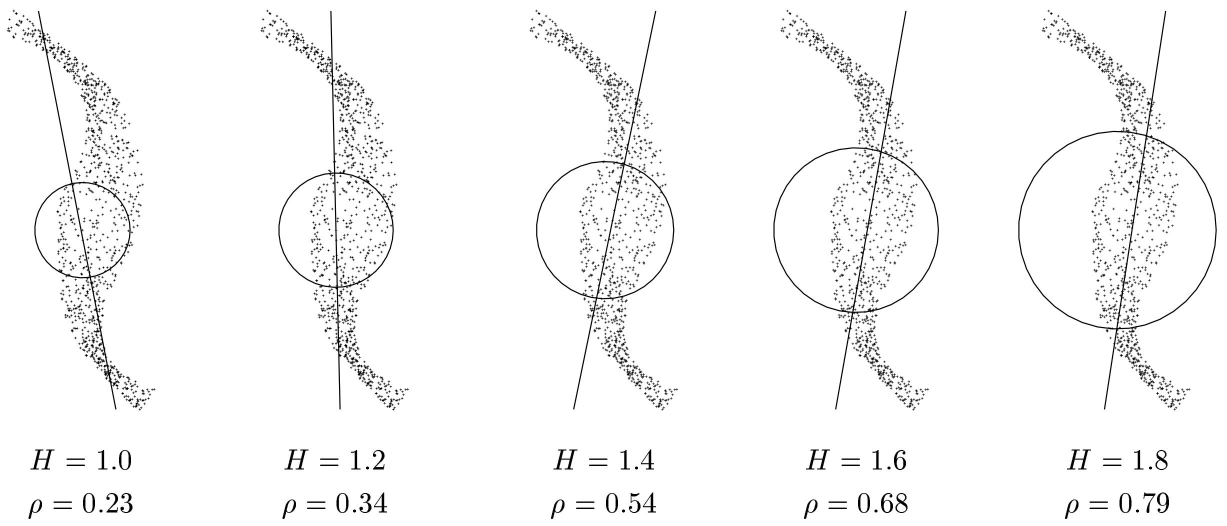


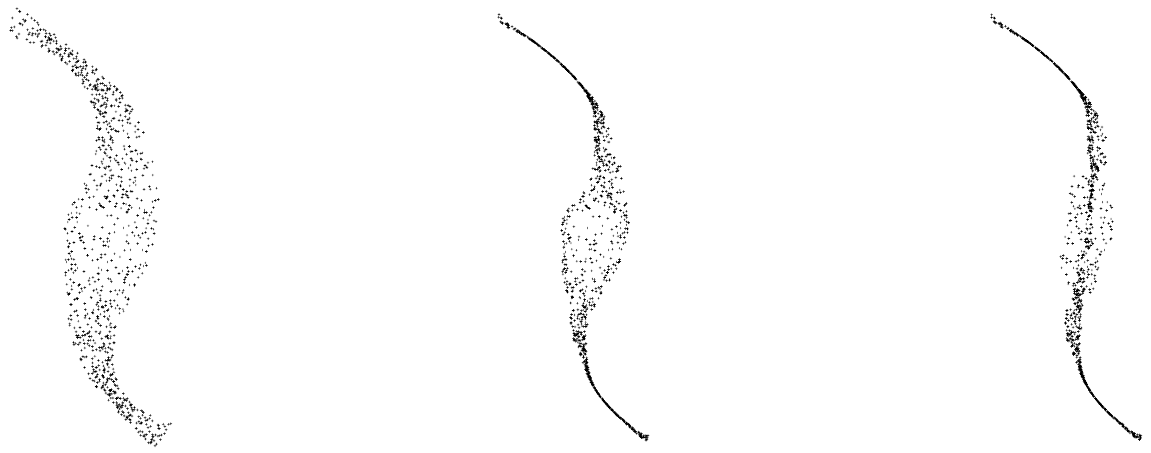
Figure 7: Growing a region to find a H using correlation computation: lines represent the local regression lines computed from the sets of collected points

The basic idea to determine the size of a region is initially taking a small H and growing the region until the region has a certain large ρ, which means the region includes enough points for a local regression. Figure 7 shows the sequence of the region growing to find an appropriate H for a local regression of a point. Note that, before the computation of each ρ, the set of points in the current region must be rotated by an angle $\frac{\pi}{4} - \theta$, where θ is an angle between a local regression line of the current region and a positive x axis. Figure 8(c) shows the result of moving least-squares with variable H determined from correlation computation, which is better than the case using fixed H (Figure 8(b)).

However, as we see in Figure 8(c), the difficulty from the varying thickness of the point cloud still remains. We suggest to use an iteration scheme for refining the point set. Let $\mathbf{Q}_*$ be a quadratic regression curve (in Equation (3) and $\mathcal{A}$ be a set of neighboring points of a point $\mathbf{P}_*$. During the computation of the local quadratic regression curve, we can easily compute the average approximation error

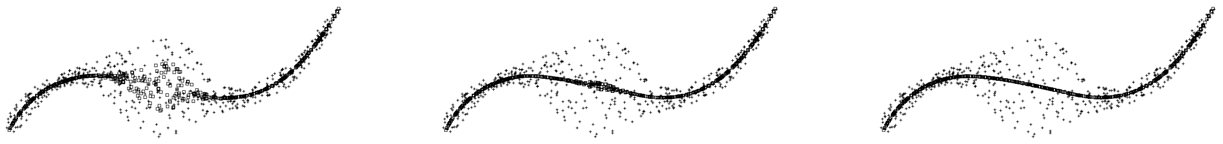
$$\epsilon_* = \frac{\sum_{\mathbf{P}_j \in \mathcal{A}} ||\mathbf{P}_j - \mathbf{Q}_*||}{|\mathcal{A}|}, \tag{8}$$

where $|\mathcal{A}|$ is the number of points in $\mathcal{A}$. If ϵ_* is larger than the prescribed tolerance, we apply moving least-squares for the point $\mathbf{P}_*$ repeatedly. Figure 9 shows an example of the iteration.



(a) (b) (c)

Figure 8: (a) Original point set, (b) using fixed size region ($H = 1.0$), (c) using region growing until $\rho > 0.7$ (initially $H = 1.0$)



(a) (b) (c)

Figure 9: H = 2, tolerance = 0.1 (a) first iteration (500 points moved), (b) second iteration (500 points moved), (c) third iteration (487 points moved)

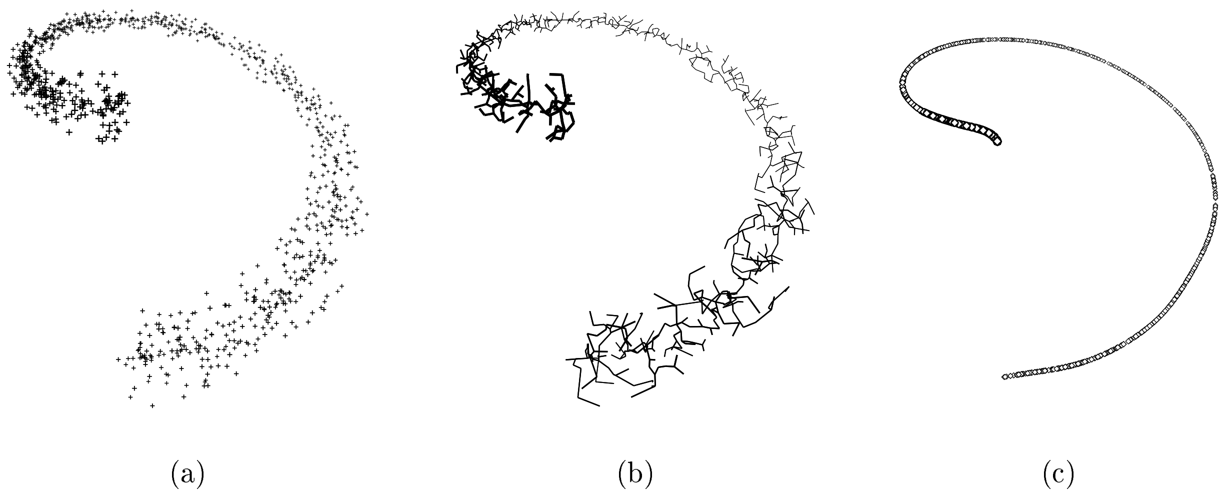


Figure 10: 3D curve reconstruction: (a) 3D point set, (b) MST of the point set, (c) thin point cloud using $H = 2$ after two iterations

4 3D extension

The moving least-squares technique described in Section 2 cannot be extended into three dimension directly. Although we can easily compute a 3D regression line (see Appendix), computing the 3D quadratic regression curve is not easy. We suggest a local regression algorithm for each point $\mathbf{P}_*$ in 3D as follows:

1. Collect a set of neighboring points, $\mathcal{A}$, using MST.
2. Compute a regression plane $\mathbf{K} : z = Ax + By + C$ by minimizing the quadratic function:

$$D_k = \sum_{\mathbf{P}_j \in \mathcal{A}} (Ax_j + By_j + C - z_j)^2 w_j$$

3. Project the points in $\mathcal{A}$ onto the plane $\mathbf{K}$.
4. Solve the 2D moving least-squares problem on the plane $\mathbf{K}$.

The above algorithm is based on the fact that a point on a regular space curve locally has an osculating plane. Basically, the above algorithm can be extended to any higher dimension. Note that the weight values computed for 3D data in step 1 and 2 can be used to solve the 2D problem in step 4 without any extra computations. Figure 10 shows moving least-squares method applied to a 3D point set.

5 Approximating a thin point cloud with a curve

Once moving least-squares technique generates a sufficiently thin point cloud, we can approximate the thin point cloud with a smooth curve by ordering the points in the cloud after reducing the number of points if needed.

First, we take a random point $\mathbf{P}_*$ from the thin point cloud $\mathcal{S}$. The neighboring points of $\mathbf{P}_*$ are nearly on a line (having a direction $\mathbf{L}_*$) if $\mathcal{S}$ is sufficiently thin. Let h be a prescribed small value and $\mathcal{B}$ be a set of points having distance less than h from $\mathbf{P}_*$. We can easily divide $\mathcal{B}$ into two sets by inspecting the direction of vectors $\vec{v}_j$ which are from $\mathbf{P}_*$ to $\mathbf{P}_j \in \mathcal{B}$:

$$\begin{aligned} \mathcal{B}_1 &= \{\mathbf{P}_j \mid \mathbf{P}_j \in \mathcal{B},\ \langle \vec{v}_j, \mathbf{L}_* \rangle \geq 0\}, \\ \mathcal{B}_2 &= \{\mathbf{P}_j \mid \mathbf{P}_j \in \mathcal{B},\ \langle \vec{v}_j, \mathbf{L}_* \rangle < 0\}, \end{aligned}$$

where $\langle \circ, \circ \rangle$ denotes the scalar product of two vectors. Recall that we already computed the local regression line $\mathbf{L}_*$ for the point $\mathbf{P}_*$ in the previous stage; thus we can use $\mathbf{L}_*$ without any extra computation.

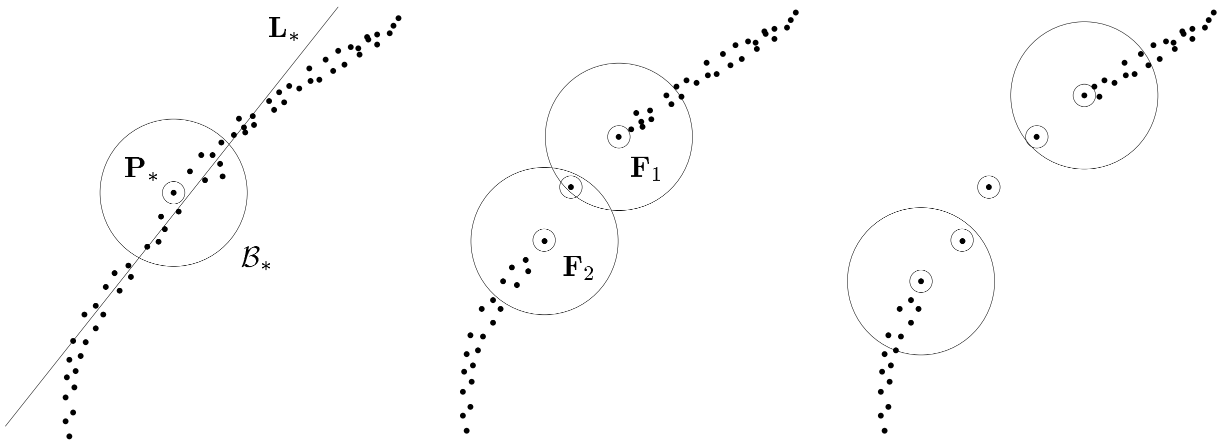


Figure 11: Ordering points

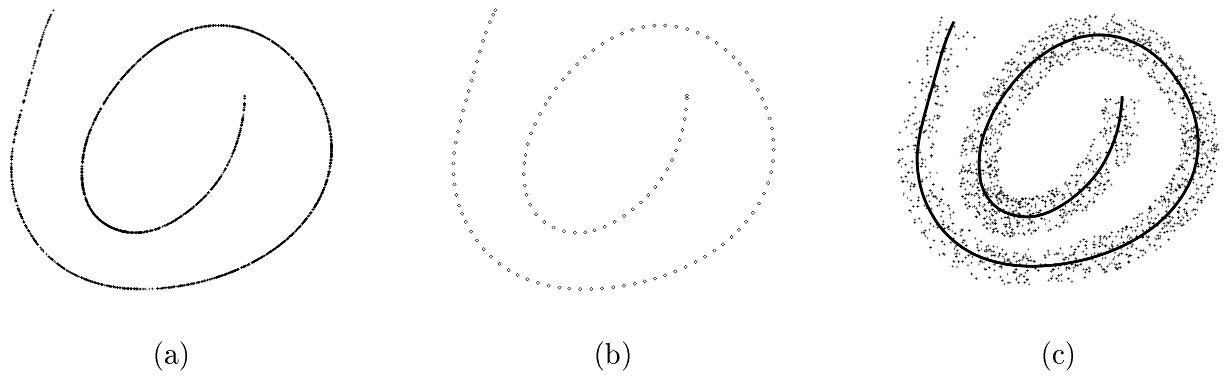


Figure 12: (a) Thin point cloud with 2000 points, (b) 125 ordered points, (c) quadratic B-spline curve approximation

However, for a 3D point set, it is necessary to compute a 3D regression line (see Appendix). We take two points $\mathbf{F}_1$ and $\mathbf{F}_2$ which are most far from $\mathbf{P}_*$ in both two sets $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively. From these two points $\mathbf{F}_1$ and $\mathbf{F}_2$, the step is repeated in both directions until we cannot continue. See Figure 11 which shows the ordering process of the thin point cloud.

After ordering the points, we can apply conventional curve approximation methods [9] to the ordered points. Figure 12 shows an example of the curve approximation from a thin point cloud with 2000 points generated from the moving least-squares technique with H = 1 and two iterations. To order the thin point cloud, we used h = 0.1 and reduced the point set to 125 ordered points (Figure 12(b)). Finally, the ordered points are approximated with a quadratic B-spline curve with 50 control points (Figure 12(c)) using chord length parametrization scheme [9].

Figures 13-16 illustrate some results of the curve reconstruction using the algorithm in the paper. 2D random data points were sampled from the area between two variable radius offset curves, $\mathbf{C}(t)+\mathbf{N}(t)r(t)$ and $\mathbf{C}(t)-\mathbf{N}(t)r(t)$, where $\mathbf{N}(t)$ is a unit normal vector of a curve $\mathbf{C}(t)$ in the plane. Similarly, 3D random data points were sampled from the inside volume of a sweep surface $\mathbf{S}(u,v) = \mathbf{C}(u) + \mathcal{F}(u)r(u)\mathbf{U}(v)$, where $r(v)$ is a variable radius, $\mathbf{U}(v)$ is a unit circle, and $\mathcal{F}(u)$ is a Frenet frame of a 3D spine curve $\mathbf{C}(u)$.

6 Pipe surface reconstruction

A *pipe surface* is generated as envelope of a sphere with a constant radius whose center runs along a spine curve. Using the curve reconstruction algorithm, we can easily reconstruct the pipe surface from a given

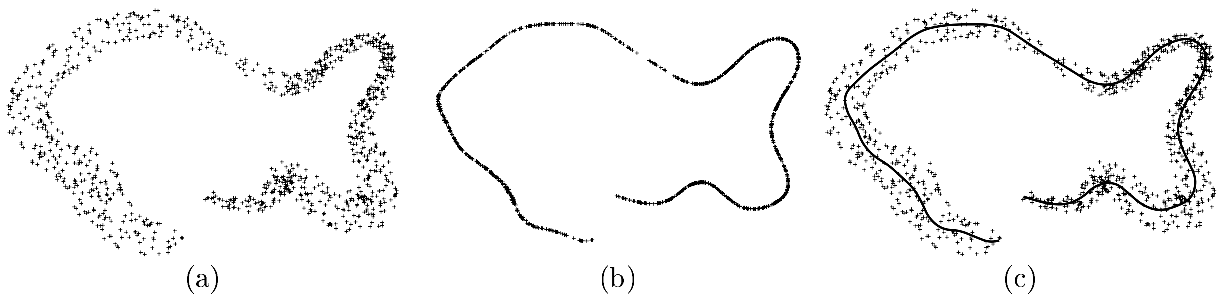


Figure 13: 2D curve reconstruction: (a) 1000 points, (b) thin point cloud after two iteration, (c) quadratic B-spline curve approximated from 76 ordered points

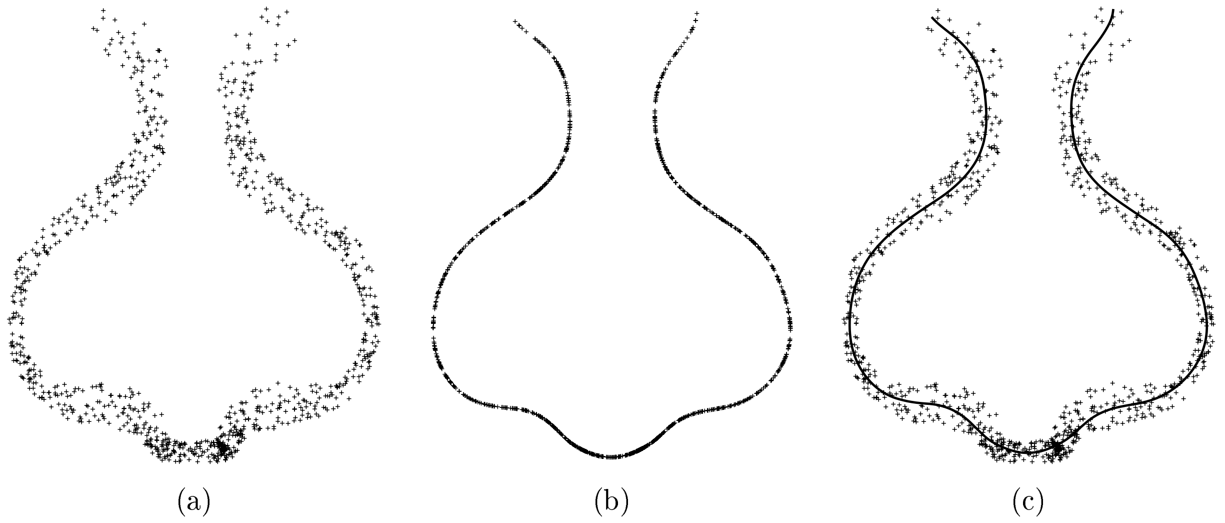


Figure 14: 2D curve reconstruction: (a) 1000 points, (b) thin point cloud after two iteration, (c) quadratic B-spline curve approximated from 87 ordered points

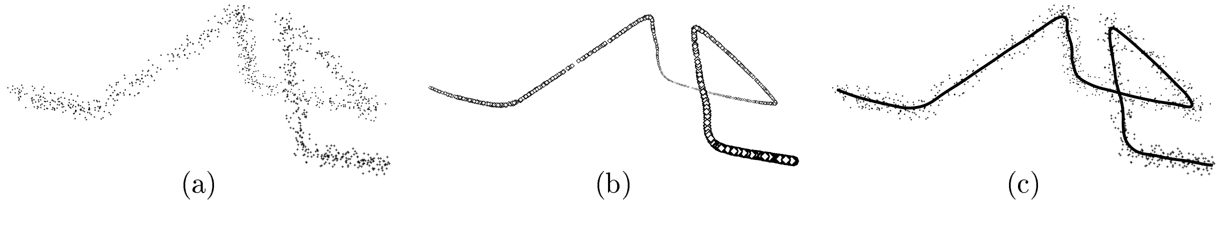


Figure 15: 3D curve reconstruction: (a) 1000 points, (b) thin point cloud after two iteration, (c) quadratic B-spline curve approximated from 83 ordered points

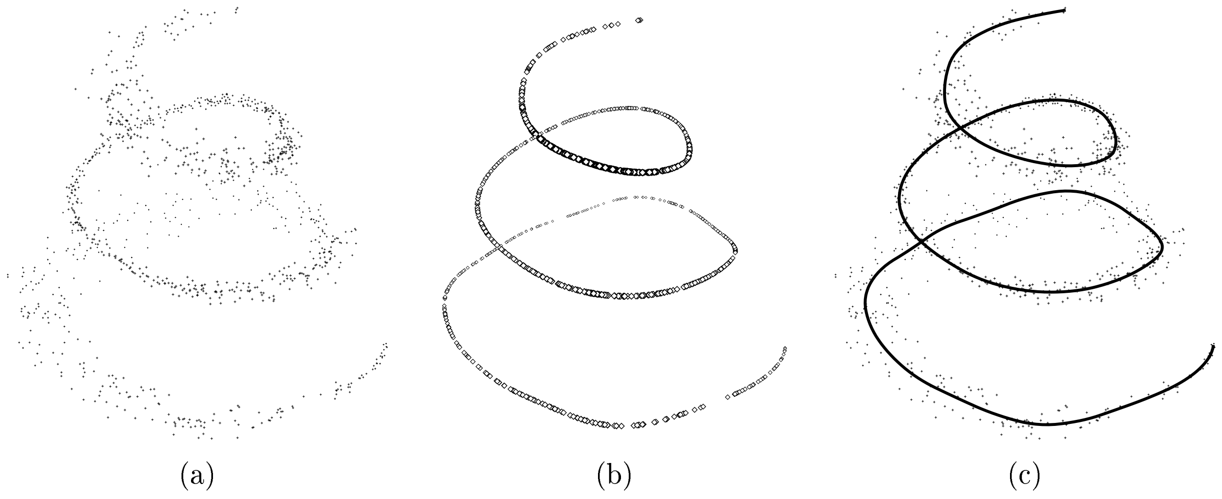


Figure 16: 3D curve reconstruction: (a) 1000 points, (b) thin point cloud after two iteration, (c) quadratic B-spline curve approximated from 67 ordered points

set of points. We assume that the (unit) normal vector at each data point was already estimated [9].

A pipe surface is defined by a *spine curve* and a *constant radius* of the swept sphere. The radius of the swept sphere can be computed by several different ways. A simple method is to collect some points from a point set, and computing a torus which locally fits the region. One can use the torus least-squares fitting (Lukács et al. [13]) or surface of revolution reconstruction (Pottmann et al. [19]).

Once the radius r is found, each data point $\mathbf{P}_i$ is translated by a vector $r\mathbf{N}_i$, where $\mathbf{N}_i$ is a unit normal vector of $\mathbf{P}_i$. If a given point set is likely to represent a pipe surface, the translations generate a reasonably thin point cloud corresponding to a spine curve. Using the curve reconstruction algorithm that we presented in this paper, we can easily compute the spine curve.

Figure 17 shows an example of the pipe surface reconstruction. After adding some perturbation factor $e(u,v)$ to an exact pipe surface $\mathbf{S}(u,v)$ (Figure 17(a)), 1000 sample points and normal vectors are taken from the perturbated pipe surface (Figure 17(b)). In Figure 17(c), the original data points are translated towards the spine curve using unit normal vectors and the radius which is computed by the method in [19]. The moving least-squares technique is applied to the point set (Figure 17(d)), and the spine curve is approximated (Figure 17(e)).

7 Conclusion

In this paper, we presented a method to reconstruct a curve from an unorganized point set. Using the power of the moving least-squares technique, the point set can easily be reduced to a thin point cloud which is a near-best approximation in the sense of the local regression. We suggested some techniques to overcome the difficulties arising in the pure moving least-squares, from which we could achieve great improvements of the results.

In this paper, we used a simple method based on the correlation of the point set to find the weighting parameter H. For estimating an optimal weighting parameter H for each local regression, it is necessary to measure the width of the point cloud locally using some polygonal structures such as α-shape [5] as well as the curvature analysis of the point set. It will also be interesting to consider non radial penalty functions. After computing the local regression line, we can modify the weighting function to give less penalties to the points in the tangent direction of the curve and more penalties to the points in the orthogonal direction. Nevertheless, we must keep in mind that the local regression must be as simple as possible because the local regression will be applied to every point in the set.

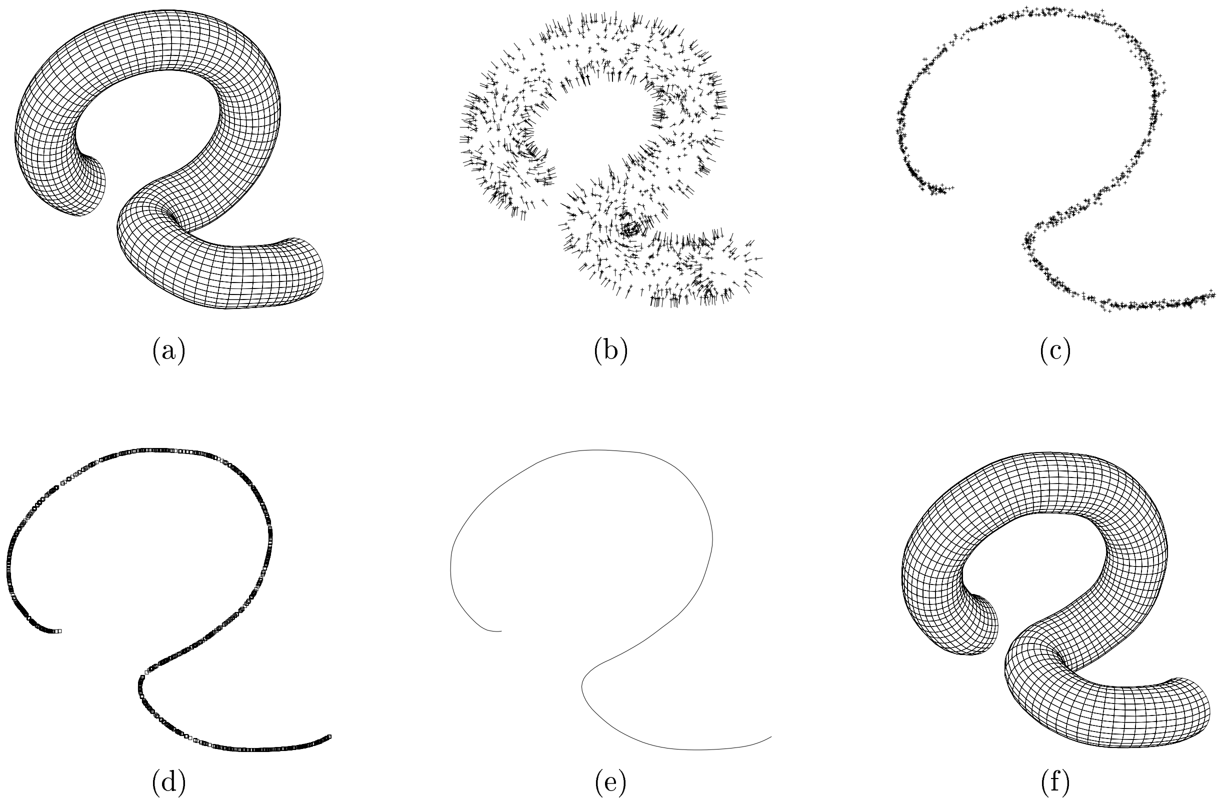


Figure 17: Pipe surface reconstruction: (a) original pipe surface, (b) 1000 points and normal vectors, (c) data points are translated by the radius of the swept sphere, (d) thin point cloud after moving least-squares, (e) spine curve reconstructed, (f) reconstructed pipe surface

Acknowledgement

The author would like to thank Professor David Levin at Tel-Aviv University and Professor Helmut Pottmann at Vienna University of Technology for pointing the author to the moving least-squares with many pieces of valuable advice.

References

[1] A. Aho, J. Hopcroft, and J. Ullman, The Design and Analysis of Computer Algorithms (Addison-Wesley, 1974).

[2] C. Bajaj, F. Bernardini, and G. Xu, Reconstructing surfaces and functions on surfaces from unorganized 3D data, Algorithmica 19 (1997) 243–261.

[3] H.-Y. Chen, I.-K. Lee, S. Leopoldseder, H. Pottmann, T. Randrup, and J. Wallner, On surface approximation using developable surfaces, Technical Report 54, Institute of Geometry, Vienna University of Technology, 1998.

[4] J.-P. Dedieu and Ch. Favardin, Algorithms for ordering unorganized points along parametrized curves, Numerical Algorithms 6 (1994) 169–200.

[5] H. Edelsbrunner and E. P. Mücke, Three-Dimensional Alpha Shapes, ACM Transactions on Graphics 13 (1994) 43–72.

[6] L. Fang and D. C. Gossard, Fitting 3D curves to unorganized data points using deformable curves, in Visual Computing (Proceedings of CG International '92) (Springer-Verlag, 1992) 535–543.

[7] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, Surface reconstruction from unorganized points, Computer Graphics 26 (1992) 71–78.

[8] H. Hoppe, T. DeRose, Duchamp T., J. McDonald, and W. Stuetzle, Mesh optimization, Computer Graphics 27 (1993) 19–26.

[9] J. Hoschek and D. Lasser, Fundamentals of Computer Aided Geometric Design (A.K.Peters, 1993).

[10] P. Lancaster, Moving weighted least-squares methods, in Polynomial and Spline Approximation (Reidel Publishing Company, 1979) 103–120.

[11] D. Levin, The approximation power of moving least-squares, Mathematics of Computation 67 (1998) 1517–1531.

[12] D. Levin, Mesh-Independent Surface Interpolation, Private communication

[13] G. Lukács, A. D. Marshall, and R. R. Martin, Geometric least-squares fitting of spheres, cylinders, cones, and tori, in RECCAD Deliverable Documents 2 and 3 Copernicus Project No. 1068, Eds. R. R. Martin, T. Varady, Report GML 1997/5, Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, 1997.

[14] D. McLain, Two dimensional interpolation from random data, The Computer Journal 19 (1976) 178–181.

[15] R. Mencl, A graph-based approach to surface reconstruction, Computer Graphics Forum 14 (1995) 445-456.

[16] J. Pitman, Probability (Springer-Verlag 1992).

[17] H. Pottmann and T. Randrup, Rotational and helical surface approximation for reverse engineering, Computing (to appear).

[18] H. Pottmann, I.-K. Lee, and T. Randrup, Reconstruction of kinematic surface from scattered data. in Proceedings of Symposium on Geodesy for Geotechnical and Structural Engineering, Eisenstadt-Austria, (1998) 483–488.

[19] H. Pottmann, H.-Y. Chen, and I.-K. Lee, Approximation by profile surfaces, in A. Ball et al. Eds., The Mathematics of Surfaces VIII (Information Geometers, 1998).

[20] F. P. Preparata and M. I. Shomos, Computational Geometry: An Introduction (Springer-Verlag, 1985).

[21] D. Shepard, A two dimensional interpolation function for irregularly spaced data, in Proceedings of 23th ACM National Conference (1968) 517–524.

[22] G. Wyvill, C. McPheeters, and B. Wyvill, Data structure for soft objects. Visual Computer 2 (1986) 227–234.

Appendix: Regression line in 3D

Let $\mathbf{G} = (\bar{x}, \bar{y}, \bar{z})$ be the center of gravity of a set of points,

$$\mathcal{S} = \{\mathbf{P}_i = (x_i, y_i, z_i) \mid i = 1, 2, ..., N\}. \tag{9}$$

It can be easily proved that a regression line passes through the center of gravity of the point set. The regression line has an equation:

$$\mathbf{R} = \mathbf{G} + t\mathbf{L}, \quad (t \in R), \tag{10}$$

where $\mathbf{L}$ is a unit vector. The distance between a point $\mathbf{P}_i$ and the line $\mathbf{R}$ is defined by

$$\|(\mathbf{P}_i - \mathbf{G}) \times \mathbf{L}\|. \tag{11}$$

Therefore the direction vector $\mathbf{L}$ can be computed by minimizing a positive semidefinite quadratic form,

$$F(\mathbf{L}) := \sum_{i=1}^{n} \|(\mathbf{P}_i - \mathbf{G}) \times \mathbf{L}\|^2 =: \mathbf{L}^T \cdot N \cdot \mathbf{L}, \tag{12}$$

under the normalization condition,

$$\|\mathbf{L}\|^2 = \mathbf{L} \cdot I \cdot \mathbf{L} = 1, \tag{13}$$

where I is an identity matrix and

$$N = \begin{bmatrix} \sum_{i=1}^n (\tilde{y}_i^2 + \tilde{z}_i^2) & -\sum_{i=1}^n \tilde{x}_i\tilde{y}_i & -\sum_{i=1}^n \tilde{x}_i\tilde{z}_i \\ -\sum_{i=1}^n \tilde{x}_i\tilde{y}_i & \sum_{i=1}^n (\tilde{z}_i^2 + \tilde{x}_i^2) & -\sum_{i=1}^n \tilde{y}_i\tilde{z}_i \\ -\sum_{i=1}^n \tilde{x}_i\tilde{z}_i & -\sum_{i=1}^n \tilde{y}_i\tilde{z}_i & \sum_{i=1}^n (\tilde{x}_i^2 + \tilde{y}_i^2) \end{bmatrix}, \tag{14}$$

where $\tilde{x}_i = x_i - \bar{x}$, etc.

By introducing a Lagrangian multiplier λ, we have to minimize the function

$$\tilde{F}(\mathbf{L}, \lambda) := \mathbf{L}^T \cdot N \cdot \mathbf{L} - \lambda(\mathbf{L}^T \cdot I \cdot \mathbf{L} - 1) \tag{15}$$

Using vector notation, the gradient of positive semidefinite quadratic form, $F = \mathbf{L}^T \cdot N \cdot \mathbf{L}$, can be represented by $\bigtriangledown F = 2N \cdot \mathbf{L}$. Thus, the minimization of $\tilde{F}$ is reduced to solving the system,

$$\frac{\partial \tilde{F}}{\partial \mathbf{L}} \quad : \quad (N - \lambda I) \cdot \mathbf{L} = 0, \tag{16}$$

$$\frac{\partial \tilde{F}}{\partial \lambda} \quad : \quad \mathbf{L}^T \cdot I \cdot \mathbf{L} - 1 = 0. \tag{17}$$

Equation (16) represents a simple eigenvalue problem. The eigenvalues can be computed by solving a cubic equation

$$\det(N - \lambda I) = 0. \tag{18}$$

Furthermore, the eigenvalues satisfy

$$N \cdot \mathbf{L} = \lambda I \cdot \mathbf{L}, \tag{19}$$

thus,

$$F(\mathbf{L}) = \mathbf{L}^T \cdot N \cdot \mathbf{L} = \mathbf{L}^T \cdot \lambda I \cdot \mathbf{L} = \lambda \mathbf{L}^T \cdot I \cdot \mathbf{L} = \lambda. \tag{20}$$

That is, the problem is finally reduced to finding the smallest eigenvalue λ in Equation (18). The corresponding eigenvector of the smallest eigenvalue is the direction vector $\mathbf{L}$ of the regression line.