Computational Geometry 44 (2011) 427–441

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Non-manifold surface reconstruction from high-dimensional point cloud data

Shawn Martin *, Jean-Paul Watson

Sandia National Laboratories, Albuquerque, NM 87185, USA

ARTICLE INFO

Article history:
Received 1 June 2009
Accepted 3 May 2011
Available online 6 May 2011
Communicated by G. Rote

Keywords:
Non-manifold surface
High-dimensional point cloud data
Incremental triangulation
Fitting intersecting planes
Degenerate quadratic surfaces

ABSTRACT

We present an algorithm capable of reconstructing a non-manifold surface embedded as a point cloud in a high-dimensional space. Our algorithm extends a previously developed incremental method and produces a non-optimal triangulation, but will work for non-orientable surfaces, and for surfaces with certain types of self-intersection. The self-intersections must be ordinary double curves and are fitted locally by intersecting planes using a degenerate quadratic surface. We present the algorithm in detail and provide many examples, including a dataset describing molecular conformations of cyclo-octane.

1. Introduction

There are many algorithms available that will reconstruct surfaces from point cloud data in $\mathbb{R}^3$. Surface reconstruction algorithms are used in applications such as computer graphics, medical imaging, and laser range scanning. Examples of these algorithms include Hoppe's level set algorithm [1,2]; Bernardini's ball-pivoting algorithm [3,4]; Amenta's crust algorithm [5,6]; Dey's co-cone algorithm [7–10]; and Edelsbrunner's wrap method [11] (available commercially at www.geomagic.com). Surface reconstruction is well-studied and there are many additional algorithms.

In certain applications, however, we may wish to triangulate surfaces which are not realizable in $\mathbb{R}^3$. Two examples are the Klein bottle and the real projective plane, both of which are non-orientable and cannot be embedded in $\mathbb{R}^3$. In addition, there are many application datasets which consist of high-dimensional point cloud data, presumed to lie on low-dimensional manifolds. Application areas which produce these datasets include computer vision, text analysis, biology, and chemistry.

Unfortunately, surface reconstruction methods almost always assume a point cloud in $\mathbb{R}^3$. Although some of the algorithms may generalize to higher dimensions (e.g. [11,10]), many of the methods use the Delaunay triangulation as a starting point. In the worst case the Delaunay triangulation has exponential complexity [12], limiting its application to relatively low dimensions. To our knowledge, an incremental algorithm due to Freedman [13] is the only practical method specifically targeted at triangulating a surface from high-dimensional ($\mathbb{R}^{>3}$) point cloud data.

To further complicate matters, most surface reconstruction methods explicitly assume a manifold surface. To our knowledge, there are only two algorithms that will reconstruct non-manifold surfaces [14,15], and both of these again assume data in $\mathbb{R}^3$. Thus, there are no algorithms available for reconstructing non-manifold surfaces from high-dimensional point cloud data. In this paper we present an algorithm designed to fill this gap.

* Corresponding author.
E-mail addresses: smartin@sandia.gov (S. Martin), jwatson@sandia.gov (J.-P. Watson).

0925-7721/$ – see front matter
doi:10.1016/j.comgeo.2011.05.002

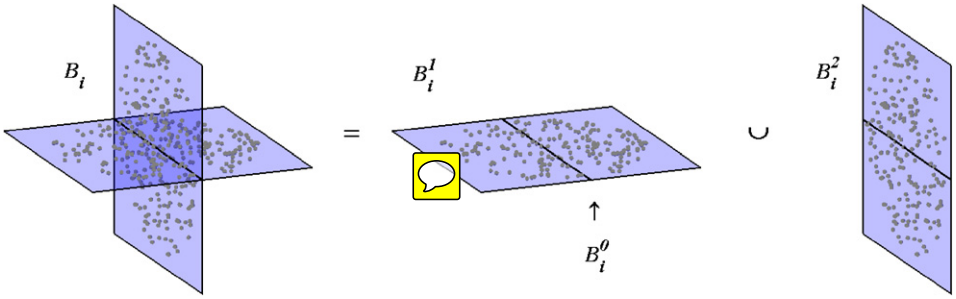

Fig. 1. Non-manifold neighborhood. Here we show the decomposition of a 3D non-manifold neighborhood of the type accepted by our algorithm. On the left we show a 3D non-manifold neighborhood, denoted by B_i. In the center and on the right, we show the 3D non-manifold neighborhood decomposed into two 2D manifold neighborhoods, denoted by B_i^1 and B_i^2. The intersection $B_i^0 = B_i^1 \cap B_i^2$ occurs at the intersection of the two planes.

We describe an extension of Freedman's algorithm [13] which can reconstruct certain high-dimensional non-manifold surfaces. The development of this algorithm was largely motivated by our desire to construct a surface from a point cloud dataset describing molecular conformations of cyclo-octane, known from first principles to have two degrees of freedom [16]. Of course the algorithm is not restricted for use on molecular conformation data, and we have tested the approach on a variety of other examples.

This paper is organized as follows: in Section 2 we describe our algorithm; in Section 3 we demonstrate its use on several examples; in Section 4 we apply the algorithm to the cyclo-octane dataset; and in Section 5 we discuss the algorithm and possible generalizations.

2. Algorithm

Our algorithm can reconstruct both manifold and non-manifold surfaces. In the case of a manifold surface, the method reduces to a slightly modified version of Freedman's incremental triangulation algorithm [13]. Our modification is the use of multi-dimensional scaling (MDS) for neighborhood projection. This modification allows us to use Freedman's algorithm with carefully selected neighborhoods and distances, thereby reducing our treatment of surface self-intersections to a pre-processing step (described next). Owing to this change, however, we lose angle information (used by Freedman for neighborhood projection and incremental triangle selection) so that we now select triangles using circumradius. The triangulation algorithm will be discussed in Section 2.2.

In the case of a non-manifold surface, we perform several pre-processing steps prior to calling the triangulation algorithm. In the first pre-processing step, we identify the non-manifold regions by local dimension estimation. The manifold regions of the surface are assumed to the two-dimensional (2D) and the non-manifold regions are assumed to be three-dimensional (3D). The 3D regions are then fitted locally with two intersecting planes using a degenerate quadratic surface. Using the two planes, we split the 3D neighborhood into two 2D neighborhoods, as shown in Fig. 1. These calculations will be described in Section 2.1.

We should note that our algorithm is experimental, and we offer no guarantees of correctness. As discussed by Freedman [13], we add triangles incrementally using local tangent space approximations. Thus, depending on how well the tangent space approximations agree locally, we may produce a triangulation with self-intersections. Nevertheless, the algorithm works well in practice, and most failures occur in non-manifold regions or regions of high curvature. We investigate the performance of the algorithm, including failures, in more detail in Section 3.

2.1. Non-manifold calculations

The first step in our algorithm is the identification of non-manifold regions of the surface. We perform this identification by partitioning the dataset according to local dimension, which we expect to be either 2D (manifold) or 3D (non-manifold). This step also helps identify valid neighborhoods and verify that our dataset indeed describes a surface. In order to estimate local dimension we use principal component analysis (PCA) locally at each point in the dataset. The various properties of PCA that we exploit are described in greater detail in [17].

Suppose our dataset is given by $X = \{\mathbf{x}_i\} \subset \mathbb{R}^N$. We denote an ϵ neighborhood of the point $\mathbf{x}_i$ by the set $B_i = \{\mathbf{x}_j : \|\mathbf{x}_j - \mathbf{x}_i\| < \epsilon\}$. (An alternative would be to use nearest neighbors to select B_i.) To perform PCA on B_i, we first mean subtract to obtain $\bar{B}_i = \{\mathbf{x}_j - \bar{\mathbf{x}}_i : \mathbf{x}_j \in B_i\}$, where $\bar{\mathbf{x}}_i = \frac{1}{n_i}\sum_{\mathbf{x}_j \in B_i} \mathbf{x}_j$ is the mean of the n_i points in B_i. Abusing notation, we use $\bar{B}_i$ to denote the matrix of points (as columns) in the set $\bar{B}_i$. We can use the singular value decomposition (SVD) to get the decomposition $\bar{B}_i = U\Sigma V^T$, where Σ is a diagonal matrix containing the singular values σ_j.

The singular values give us information about the local dimension of B_i. If we use a rank d approximation to B_i then the mean square error of our approximation is given by $\sum_{j>d}\sigma_j^2$. Thus we say that B_i is d-dimensional within tolerance d_t if $\sum_{j>d}\sigma_j^2 / \sum_j \sigma_j^2 < d_t$ (we normalize by $\sum_j \sigma_j^2$ so that we can specify d_t as a fraction between 0 and 1). Using this

definition we compute the dimension d_i of the neighborhood B_i as $d_i = \min\{d\colon\ B_i \text{ is } d\text{-dimensional with tolerance } d_t\}$. For our examples and for the cyclo-octane application, we often use a d-dimensional tolerance of $d_t = 0.05$.

The above approach is fairly standard [18], but there are many other methods for estimating local dimension. These methods include the use of fractal dimension [19], packing numbers [20], maximum likelihood estimates [21], and geodesic minimum spanning trees [22]. We use the PCA-based approach because it is fast, simple and also yields a low-dimensional projection (necessary for our non-manifold neighborhood modeling effort described next). However, some of the alternatives mentioned here could be used to improve the accuracy of our local dimension estimates.

2.1.1. Modeling self-intersections

Suppose a given neighborhood B_i of our surface is identified as non-manifold (3D) using local dimension estimation. An assumption in our method is that such a neighborhood contains a double curve. By a double curve we mean that the surface has a self-intersection, and that points on the self-intersection belong to two different tangent spaces. Practically speaking, this means that the neighborhood can be modeled by two intersecting planes, as shown in Fig. 1 (left).

Our goal is to fit these two planes using a degenerate quadratic surface. Since B_i has been identified as 3D using PCA, we obtain a projection $P_i = [\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3]^T \bar{B}_i$ of B_i onto a 3D (affine) subspace, where $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$ are the first three left singular vectors of $\bar{B}_i$. For the purposes of our exposition (and again abusing notation) we denote P_i by the set $\{(x_j, y_j, z_j)\}$. We want to fit a quadratic in the variables x, y, z to the points (x_j, y_j, z_j) such that the quadratic factors into two linear terms.

Following Dresden's exposition of quadratic surfaces [23], we denote a quadratic by

$$f(x, y, z) = a_{11}x^2 + 2a_{12}xy + 2a_{13}xz + 2a_{14}x + a_{22}y^2 + 2a_{23}yz + 2a_{24}y + a_{33}z^2 + 2a_{34}z + a44 = 0. \tag{1}$$

Our goal is to solve for $\mathbf{a} = [a_{11}\ a_{12}\ a_{13}\ a_{14}\ a_{22}\ a_{23}\ a_{24}\ a_{33}\ a_{34}\ a_{44}]^T$ such that the surface $f(x, y, z) = 0$ models our projected neighborhood $P_i = \{(x_j, y_j, z_j)\}$. To solve for $\mathbf{a}$, we use a constrained least squares fit. Suppose $\mathbf{m}_j^T = [x_j^2\ \ 2x_jy_j\ \ 2x_jz_j\ \ 2x_j\ \ y_j^2\ \ 2y_jz_j\ \ 2y_j\ \ z_j^2\ \ 2z_j\ \ 1]$ and $M = [\mathbf{m}_j^T]$, i.e. M is a matrix with the jth row given by $\mathbf{m}_j^T$. Now $f(x_j, y_j, z_j) = \mathbf{m}_j^T\mathbf{a}$ and $\sum_j f^2(x_j, y_j, z_j) = \mathbf{a}^T M^T M\mathbf{a}$. Since we want $f(x_j, y_j, z_j) = 0$ for each (x_j, y_j, z_j), it is logical to minimize $\sum_j f^2(x_j, y_j, z_j) = \mathbf{a}^T M^T M\mathbf{a}$. In the unconstrained case, this minimization is equivalent to solving $M\mathbf{a} = \mathbf{0}$, or finding the nullspace of M.

Depending on the data $\{(x_j, y_j, z_j)\}$, the minimization of $\mathbf{a}^T M^T M\mathbf{a}$ may result in a solution $\mathbf{a}$ such that the polynomial $f(x, y, z)$ does not factor into two linear terms. We must impose constraints to guarantee this factorization. Fortunately, such constraints are known and can be described as follows [23]. Suppose

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{12} & a_{22} & a_{23} & a_{24} \\ a_{13} & a_{23} & a_{33} & a_{34} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{bmatrix},$$

where the entries of A are components of the vector $\mathbf{a}$ of coefficients of the polynomial $f(x, y, z)$. Denote by A_3 the first principal (upper left) 3×3 submatrix of A. Then $f(x, y, z)$ can be factored as two linear terms with real coefficients when $\operatorname{rank}(A) = \operatorname{rank}(A_3) = 2$, and the quantity $T_2 = (a_{11}a_{22} - a_{12}^2) + (a_{11}a_{33} - a_{13}^2) + (a_{22}a_{33} - a_{23}^2) \leqslant 0$.

Now we can model our neighborhood as two intersecting planes if we can solve the optimization problem given by

$$\begin{aligned} &\min_{\mathbf{a}} \mathbf{a}^T M^T M\mathbf{a} \\ &\quad \text{s.t. } \operatorname{rank}(A) = \operatorname{rank}(A_3) = 2, \\ &\qquad T_2 \leqslant 0, \qquad \|A_3\|_F = 1. \end{aligned} \tag{2}$$

The last constraint $\|A_3\|_F = 1$ (Frobenius norm of A_3 is 1) restricts the freedom to scale the equation $f(x, y, z) = 0$ by an arbitrary constant. (We chose A_3 instead of A for compatibility with results to follow.)

Unfortunately, the minimization in (2) is non-trivial. Although the objective $\mathbf{a}^T M^T M\mathbf{a}$ is quadratic, the constraints are nonlinear and difficult to express as relationships between the variables in $\mathbf{a}$. However, we can simplify matters by parameterizing the constraint matrix A using an eigenvalue decomposition. We claim that

$$\left\{ A \;\middle|\; \begin{array}{l} \operatorname{rank}(A) = 2, \\ \operatorname{rank}(A_3) = 2, \\ \|A_3\|_F = 1 \end{array} \right\} = \left\{ \lambda_1\mathbf{q}_1\mathbf{q}_1^T + \lambda_2\mathbf{q}_2\mathbf{q}_2^T \;\middle|\; \begin{array}{l} \lambda_1, \lambda_2 \neq 0, \\ \lambda_1^2 + \lambda_2^2 = 1, \\ \|\tilde{\mathbf{q}}_1\| = \|\tilde{\mathbf{q}}_2\| = 1, \\ \tilde{\mathbf{q}}_1^T\tilde{\mathbf{q}}_2 = 0 \end{array} \right\}, \tag{3}$$

where $\tilde{\mathbf{q}}_1$, $\tilde{\mathbf{q}}_2$ are vectors in $\mathbb{R}^3$ given by the first three components of $\mathbf{q}_1$, $\mathbf{q}_2$. The inclusion ($\supseteq$) in (3) is straightforward. To check the opposite inclusion ($\subseteq$), we suppose that A is a symmetric matrix with $\operatorname{rank}(A) = \operatorname{rank}(A_3) = 2$ and $\|A_3\|_F = 1$.

Since $\text{rank}(A_3) = 2$, we can use an eigenvalue decomposition to obtain $A_3 = \lambda_1 \tilde{\mathbf{q}}_1 \tilde{\mathbf{q}}_1^T + \lambda_2 \tilde{\mathbf{q}}_2 \tilde{\mathbf{q}}_2^T$, with $\lambda_1, \lambda_2 \neq 0$, $\|\tilde{\mathbf{q}}_1\| = \|\tilde{\mathbf{q}}_2\| = 1$, and $\tilde{\mathbf{q}}_1^T \tilde{\mathbf{q}}_2 = 0$. We also know that $\lambda_1^2 + \lambda_2^2 = 1$ since $\|A_3\|_F = 1$. Further, we can solve for r, s such that $\mathbf{q}_1 = [\tilde{\mathbf{q}}_1 \; r]^T$, $\mathbf{q}_2 = [\tilde{\mathbf{q}}_2 \; s]^T$. This is done using the last column of A to obtain

$$[\lambda_1 \tilde{\mathbf{q}}_1 \quad \lambda_2 \tilde{\mathbf{q}}_2] \begin{bmatrix} r \\ s \end{bmatrix} = \begin{bmatrix} a_{14} \\ a_{24} \\ a_{34} \end{bmatrix}. \tag{4}$$

Using the fact that $\text{rank}(A) = 2$ we know that r, s exist and are uniquely determined by (4), and that the entry a_{44} of A is given by $\lambda_1 r^2 + \lambda_2 s^2$.

We can now re-write the optimization problem in (2) in a slightly more tractable form. If we denote the matrix

$$X_j = \begin{bmatrix} x_j^2 & x_j y_j & x_j z_j & x_j \\ x_j y_j & y_j^2 & y_j z_j & y_j \\ x_j z_j & y_j z_j & z_j^2 & z_j \\ x_j & y_j & z_j & 1 \end{bmatrix}$$

then $\sum_j f^2(x_j, y_j, z_j) = \sum_j \text{tr}^2(X_j A) = \sum_j (\lambda_1 \mathbf{q}_1^T X_j \mathbf{q}_1 + \lambda_2 \mathbf{q}_2^T X_j \mathbf{q}_2)^2$ so that (2) becomes

$$\begin{aligned} \min_{\mathbf{q}_1, \mathbf{q}_2, \lambda_1, \lambda_2} & \sum_j \left(\lambda_1 \mathbf{q}_1^T X_j \mathbf{q}_1 + \lambda_2 \mathbf{q}_2^T X_j \mathbf{q}_2\right)^2 \\ \text{s.t. } & \|\tilde{\mathbf{q}}_1\| = \|\tilde{\mathbf{q}}_2\| = 1, \qquad \tilde{\mathbf{q}}_1^T \tilde{\mathbf{q}}_2 = 0, \\ & \lambda_1, \lambda_2 \neq 0, \quad \lambda_1^2 + \lambda_2^2 = 1, \\ & T_2 = \sum_{k<l\leqslant 3} \det\left(\left[\lambda_1 \mathbf{q}_1 \mathbf{q}_1^T + \lambda_2 \mathbf{q}_2 \mathbf{q}_2^T\right]_{kl}\right) \leqslant 0, \end{aligned} \tag{5}$$

where $[\lambda_1 \mathbf{q}_1 \mathbf{q}_1^T + \lambda_2 \mathbf{q}_2 \mathbf{q}_2^T]_{kl}$ denotes the principal submatrix given by the kth and lth rows and columns of $\lambda_1 \mathbf{q}_1 \mathbf{q}_1^T + \lambda_2 \mathbf{q}_2 \mathbf{q}_2^T$.

The main advantage of the revised formulation (5) over the original formulation (2) is the simplified constraints. The new constraints are easier to enforce and provide an important insight into the computation of an initial estimate $\mathbf{a}_0$ of the true solution $\mathbf{a}^*$ to the optimization problem. Although the new objective $\sum_j (\lambda_1 \mathbf{q}_1^T X_j \mathbf{q}_1 + \lambda_2 \mathbf{q}_2^T X_j \mathbf{q}_2)^2$ is more complicated, it is in fact equal to $\mathbf{a}^T M^T M \mathbf{a}$, so that we may obtain a first estimate $\mathbf{a}_{-1}$ of $\mathbf{a}^*$ by finding the nullspace of M. The best way to estimate the nullspace is to set $\mathbf{a}_{-1}$ to the right most singular vector of M. For ideal data, describing two intersecting planes without error, $\mathbf{a}_{-1}$ would be equal to $\mathbf{a}^*$. For data still describing two intersecting planes, but with error and/or curvature, M will be full rank, so that $\mathbf{a}_{-1}$ is our best estimate of the (non-existent) nullspace of M.

Now using the new constraints (3), we form the matrix A from $\mathbf{a}_{-1}$ and normalize it such that $\|A_3\|_F = 1$. We compute an eigenvalue decomposition of A_3, truncating to the first two largest magnitude eigenvalues to obtain λ_1, λ_2, $\tilde{\mathbf{q}}_1$, $\tilde{\mathbf{q}}_2$. We solve Eq. (4) to obtain $\mathbf{q}_1$, $\mathbf{q}_2$. The resulting values give us our initial estimate for the optimization in (5), and correspond to our initial estimate $\mathbf{a}_0$ of the true solution $\mathbf{a}^*$. Perhaps surprisingly, this estimate is very good. On all of our examples, including our application to the cyclo-octane dataset, we never needed to further improve the initial estimate, instead just taking $\mathbf{a}^* = \mathbf{a}_0$.

Given that we can now obtain a coefficient vector $\mathbf{a}^*$ such that $f(x, y, z)$ fits our data $\{(x_j, y_j, z_j)\}$, we must still factor $f(x, y, z)$ into two linear terms. This can be done following Dresden, [23, p. 207]. Now use $\mathbf{a}^*$ to form A. Since $\text{rank}(A_3) = 2$, it follows that one of the three terms in $T_2 = (a_{11}a_{22} - a_{12}^2) + (a_{11}a_{33} - a_{13}^2) + (a_{22}a_{33} - a_{23}^2)$ is non-zero. Suppose that the first term $a_{11}a_{22} - a_{12}^2 \neq 0$ (the other cases are the same). Since $T_2 \leqslant 0$, we know that $a_{11}a_{22} - a_{12}^2 \leqslant 0$. Now the quadratic $a_{11}t^2 - 2a_{12}t + a_{22} = 0$ has two real solutions α, β. Let $g(x, y, z) = 2(a_{11}x + a_{12}y + a_{13}z + a_{14})$ and $h(x, y, z) = 2(a_{12}x + a_{22}y + a_{23}z + a_{24})$. We can factor our solution as $f = (h - \alpha g)(h - \beta g)$.

We can assess the quality of the fit of the two intersecting planes by evaluating the objective function $\sum_j f^2(x_j, y_j, z_j)$. However, this sum may not be a true reflection of the distance from the points $\{(x_j, y_j, z_j)\}$ to the planes $(h - \alpha g)$ and $(h - \beta g)$. As an alternative, we can compute the mean squared error $\frac{1}{n_i} \sum_j \min\{d((x_j, y_j, z_j), h - \alpha g), d((x_j, y_j, z_j), h - \beta g)\}$ of the n_i points in our neighborhood, where $d((x_j, y_j, z_j), h - \alpha g)$ is the distance from (x_j, y_j, z_j) to the plane $h - \alpha g$ and similarly for $d((x_j, y_j, z_j), h - \beta g)$. In our algorithm, we use a threshold d_p on the mean squared error to determine whether or not a given fit is acceptable.

Fitting two planes to a dataset is a difficult and somewhat specific problem. In our approach, we have taken advantage of the particular structure of the problem to avoid nonlinear optimization, when possible. Our approach is simple and efficient, but should it fail, we can also employ more complicated methods. Another approach based on nonlinear optimization is given in [24], and more general alternatives can be found in the field of computer vision, where the problem is known as subspace segmentation. Perhaps the method most relevant to our approach is generalized PCA [25], which can fit not only two planes, but more generally different subspaces of different dimension (such as a plane and a line), and might therefore be useful for extending our method to other types of singularities.

Table 1
The five pre-processing parameters. Successful triangulation requires appropriate selection of the parameters ϵ, d_t, d_p, ϵ_p, and d_l. Selection is simplified by considered the parameters d_t, d_p, and ϵ_p to be tolerances and choosing ϵ to satisfy those tolerances. The choice of parameter d_l is influenced by ϵ but is less critical.

Param.	Description	Comments
ϵ	Neighborhood size	Small enough to satisfy tolerances d_t, d_p, ϵ_p, but as large as possible.
d_t	Dimension threshold	Fraction between 0 and 1, small as possible with the constraint that neighborhoods must be 2D or 3D.
d_p	Plane fitting threshold	As small as possible to ensure good fits for intersecting planes in non-manifold 3D neighborhoods.
ϵ_p	Intersection tolerance	As small as possible to ensure good representation of self-intersections.
d_l	Landmark separation	Min. distance between landmarks, select for triangulation size, quality.

2.2. Surface reconstruction

By modeling non-manifold (3D) neighborhoods using intersecting planes, we can decompose our dataset into locally planar (2D) neighborhoods. Using these neighborhoods, we can proceed with surface reconstruction as if our data were manifold. This approach can be described as a series of pre-processing steps we apply prior to the actual triangulation.

2.2.1. Pre-processing

Suppose we have decomposed a non-manifold (3D) neighborhood into locally planar (2D) neighborhoods, as described in Section 2.1. This is shown in Fig. 1 for a neighborhood B_i. After fitting B_i with two planes, we simply split B_i into three subsets. The first of these subsets $B_i^0 = \{(x_j, y_j, z_j) \mid d((x_j, y_j, z_j), h - \alpha g) < \epsilon_p, d((x_j, y_j, z_j), h - \beta g) < \epsilon_p\}$ consists of points near the intersection of the two planes. The next two subsets B_i^1 and B_i^2 consist of points closest to one plane or another. The subset $B_i^1 = \{(x_j, y_j, z_j) \mid d((x_j, y_j, z_j), h - \alpha g) \leqslant d((x_j, y_j, z_j), h - \beta g)\} \cup B_i^0$ consists of points closest to the first plane (plus intersections), and the subset $B_i^2 = \{(x_j, y_j, z_j) \mid d((x_j, y_j, z_j), h - \beta g) \leqslant d((x_j, y_j, z_j), h - \alpha g)\} \cup B_i^0$ consists of points closest to the second plane. The parameter ϵ_p specifies our tolerance for error and/or curvature in the characterization of the intersection. In general ϵ_p should be on the order of the parameter d_p used to determine how well the two planes fit the data. For uniformity of notation, we use similar notation in case B_i is determined to be manifold (2D) from local dimension estimation: we set $B_i^0 = \emptyset$, $B_i^1 = B_i$, $B_i^2 = \emptyset$.

The next step in our pre-processing reduces computation and improves the quality of our triangulation. Borrowing an idea from the field of dimension reduction, we subsample our data $X = \{\mathbf{x}_i\}$ to obtain a set of *landmark* points $X_L = \{\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_L}\}$. We select landmarks by randomly sampling from our dataset with the constraint that each point in X_L must be at least (Euclidean) distance d_l from any other point in X_L. To accommodate surface self-intersections, we first sample from the set of intersections $\bigcup_i B_i^0$, then we sample from the remainder of the data.

Landmark points have been used in combination with dimension reduction methods to improve performance and scale algorithms to larger problems. For surface reconstruction, subsampling also tends to improve the quality of triangles obtained. In particular, by requiring a minimum distance between points, we are prohibiting small edges in the triangulation. In turn, we reduce our chances of obtaining long skinny triangles, thus improving the quality of our results. In addition to our approach, more sophisticated methods can be used for selecting landmarks [26,27].

The final step in the pre-processing stage of our algorithm is to calculate neighborhood distances. These distances are computed between each pair of points in a given neighborhood, where a neighborhood is one of B_i^0, B_i^1, B_i^2. We keep only the distances between landmarks, and use the neighborhoods $B_{i,L}^0 = B_i^0 \cap X_L$, $B_{i,L}^1 = B_i^1 \cap X_L$, $B_{i,L}^2 = B_i^2 \cap X_L$ as input to our triangulation algorithm.

The inputs to our triangulation algorithm are landmark neighborhoods $B_{i,L}^0$, $B_{i,L}^1$, $B_{i,L}^2$ and pairwise distances between points within those neighborhoods. As such, the triangulation cannot be tuned except by altering the various pre-processing parameters. Before describing the triangulation algorithm, we summarize these parameters and their effects. For convenience, we include Table 1 in addition to the following discussion.

There are five pre-processing parameters: ϵ, d_t, d_p, ϵ_p, and d_l. Although these parameters are all related, the values of d_t, d_p, and ϵ_p largely determine acceptable values for ϵ and in turn d_l. We consider d_t, d_p and ϵ_p to be tolerances and we choose ϵ to satisfy those tolerances. The parameter d_l is less critical but is somewhat limited by ϵ.

The dimension threshold parameter d_t determines how closely our neighborhood can be approximated by a 2D or 3D projection. For low or no-error data (as is the case for our application and examples), we use a low value of d_t, such as 0.01 or 0.05. (Recall that d_t assumes values between 0 and 1.) Thus the neighborhood size parameter ϵ must be small enough for every neighborhood to be either 2D or 3D within this tolerance.

The plane distance parameter d_p specifies how accurately we expect to model a 3D non-manifold neighborhood using two intersecting planes. Since this model is critical to our method, we again use a low value for d_p, such as 0.01. (Note that d_p is not a fraction and will vary depending on the scale and distribution of X.) The neighborhood size parameter ϵ must be small enough that every non-manifold neighborhood can be modeled to the tolerance given by d_p.

The parameter ϵ_p is related to d_p and tells us how close a point must be to both intersecting planes in order to be considered a point representing a surface self-intersection. In our application and examples, we choose ϵ_p to be on the order of d_p. Often we let $\epsilon_p = d_p$.

Based on the tolerances d_t, d_p, and ϵ_p, we must choose neighborhood size ϵ to be relatively small. On the other hand, it is important that each ϵ neighborhood B_i have enough points (e.g. $\geqslant 50$) so that we may successfully apply local dimension estimation, neighborhood projection, landmark sampling, and of course our local triangulation algorithm. Thus ϵ must be chosen small enough to satisfy the tolerances d_t, d_p, and ϵ_p but otherwise as large as possible.

The choice of landmark parameter d_l is less critical, and mostly determines the number and distribution of points that will be input into the triangulation. While d_l must still be chosen so that the neighborhoods $B^0_{i,L}$, $B^1_{i,L}$, $B^2_{i,L}$ are large enough, we have considerable control over d_l in determining the size and quality of the final triangulation.

2.2.2. Reconstruction

Although our reconstruction algorithm is based on Freedman's algorithm [13], we have adopted a modular approach which can accommodate the various pre-processing steps that we employ. In particular, Freedman inputs vector data and relies on vector based calculations such as angle and projection. In contrast, we input distance based data and rely on calculations such as neighborhoods and area. To be precise, our algorithm takes as input a sparse pairwise distance matrix which contains pairwise distances within neighborhoods. This approach is flexible and allows us to decouple the triangulation algorithm from any of the pre-processing steps, such as dimension estimation, neighborhood projection, modeling the self-intersections, decomposing the neighborhoods, subsampling, and so on. In this manner we can in the future adapt the algorithm to other types of singularities, or employ different distance metrics (e.g. geodesic distance).

Like Freedman's algorithm [13], our method is incremental and non-optimal. Unlike Freedman's algorithm, we use distance based calculations instead of vector based quantities. This difference necessitates a few other changes. In particular, Freedman's algorithm uses tangent plane projection to provide local bases while we used MDS (multi-dimensional scaling); Freedman's algorithm adds one triangle at a time regardless of neighborhood while we triangulate each neighborhood fully before continuing to the next neighborhood; finally, Freedman's algorithm computes angles between tangent planes to select new triangles, while we select triangles according to minimum circumradius (a distance and area based calculation). A full description of our algorithm follows.

Suppose we are given neighborhoods $B^0_{i,L}$, $B^1_{i,L}$, $B^2_{i,L}$ and previously computed pairwise distances within each neighborhood. There are two steps in our surface reconstruction. First, we triangulate the surface self-intersections $\bigcup_i B^0_{i,L}$. Second, we triangulate the entire surface using the 2D neighborhoods $B^1_{i,L}$, $B^2_{i,L}$. In the second step we preserve the edges created in the first step.

Our method visits each point in the landmark dataset X_L one and only one time. For a given point $\mathbf{x}_i \in X_L$, we use one of the previously computed neighborhoods $B^0_{i,L}$, $B^1_{i,L}$, or $B^2_{i,L}$ along with the pairwise distance matrix for that neighborhood to obtain a local coordinate system via MDS (multi-dimensional scaling). MDS is related to PCA and can be used to compute a coordinate system from a pairwise distance matrix [28]. Very briefly, the coordinate system obtained from MDS is given by the eigenvectors of a matrix HBH^T. In this matrix, $H = I - \frac{1}{n}\mathbf{1}\mathbf{1}^T$, where I is the identity matrix and $\mathbf{1}$ is a vector of ones, and $B = (-\frac{1}{2}d^2_{rs})$, where d_{rs} is the distance between points r and s.

In the local coordinate system, we construct a local triangulation such that $\mathbf{x}_i$ is not on a boundary, and that the local triangulation is geometrically compatible with any previously obtained edges and triangles. For the 1D neighborhoods $B^0_{i,L}$ this is simply a matter of connecting the point $\mathbf{x}_i$ to it two nearest neighbors. For the 2D neighborhoods $B^1_{i,L}$, $B^2_{i,L}$ this is done using the "dangling edge" approach employed by Freedman [13]. If $\mathbf{x}_i$ is already contained in an edge then we select one such edge which serves as the border of only one triangle (a dangling edge). If $\mathbf{x}_i$ is not contained in any edge, we add an initial edge connecting $\mathbf{x}_i$ to it's nearest neighbor (that edge then serves as the dangling edge). We add new triangles containing $\mathbf{x}_i$ and the dangling edge according to minimum circumradius, computed with the formula $uvw/4K$, where u, v, w are the lengths of the sides of a triangle (i.e. they are distances in the original space), and K is the area of the triangle (note that the area K can be computed using Heron's formula from u, v, w). Before we add a new triangle, we check that it doesn't intersect any previously obtained triangle using Freedman's linear program [13]

$$\begin{aligned} \max_{\alpha,\beta} \left(F = \sum_{j \in t' - t} \beta_j \right) \\ \text{s.t.} \sum_{i \in t} \alpha_i \mathbf{u}_i = \sum_{j \in t'} \beta_j \mathbf{v}_j, \\ \sum \alpha_i = \sum \beta_j = 1, \\ \alpha_i, \beta_j \geqslant 0, \end{aligned} \tag{6}$$

where t, t' are two triangles (t is the existing triangle and t' is the proposed addition), $\mathbf{u}_i$, $\mathbf{v}_j$ are the vertices of t, t' respectively in the local MDS coordinates, and α, β are the barycentric coordinates for t, t'. This linear program can be used to check if t, t' intersect. If there is no feasible solution then $t \cap t' = \emptyset$; if $F = 0$ then $t \cap t' \neq \emptyset$, but only along an edge or vertex (this is allowed); and if $F > 0$ then $t \cap t'$, with some interior intersection (this is not allowed).

The order that we visit the points in X_L is important for a successful triangulation. The problem arises from the self-intersections. If $\mathbf{x}_i \in B^0_{i,L}$ then $\mathbf{x}_i$ is on a self-intersection. In this case $\mathbf{x}_i \in B^0_{i,L} \cap B^1_{i,L} \cap B^2_{i,L}$ so that there is no unique way

to obtain a local (<3D) coordinate system containing $\mathbf{x}_i$. For this reason, we first visit all the points in the set of self-intersections $\bigcup_i B^0_{i,L}$. We use one-dimensional (1D) local coordinate systems for these points and connect the points with edges only. When these points are exhausted, we continue using points in $X_L \setminus \bigcup_i B^0_{i,L}$. Since we never re-visit previously visited points, the remainder of our neighborhoods are uniquely determined, and we can obtain 2D local coordinate systems. Finally, we use an advancing front type method to determine the order that we visit points in X_L, i.e. we always visit neighbors of previously visited points. After visiting each point $\mathbf{x}_i$, we obtain a triangulation of our surface. If at any stage of the algorithm we encounter local conflicts, such as edge crossings or overlapping triangles, the algorithm terminates unsuccessfully.

2.2.3. Complexity

An analysis of the complexity of our algorithm can be divided into two parts: an analysis of the complexity of the pre-processing stage and an analysis of the complexity of the surface reconstruction algorithm. Suppose our dataset $X = \{\mathbf{x}_i\}_{i=1}^n \subset \mathbb{R}^N$, that k is the average size of a neighborhood, and that L is the number of landmark points.

For the pre-processing stage, computation of the neighborhood sets B_i is $O(n^2N)$; computation of local dimension and projection using the reduced order SVD is $O(nkN^2 + nN^3)$; computation of the least squares two plane fit without nonlinear optimization is $O(kn)$ (actually less because most neighborhoods are manifold); computation of the landmark points is $O(L^2N)$; and computation of pairwise neighborhood distances is $O(Lk^2N)$. Adding each of these steps together, we get a total of

$$O\left(n^2N + nkN^2 + nN^3 + kn + L^2N + Lk^2N\right)$$

for the pre-processing stage. Generally, we assume $k, N \ll L \ll n$. Grouping factors of n gives us

$$O\left(Lk^2N + L^2N + \left(kN^2 + N^3 + k\right)n + Nn^2\right).$$

If we treat the smallest terms k, N as constant we get $O(L + L^2 + n^2)$. This term is dominated by $O(n^2)$ due to the subsampling algorithm.

For the triangulation, we use Freedman's analysis. Freedman gives the worst case complexity for the triangulation to be $O(L^3)$ and the typical complexity to be $O(L^2)$. Freedman mentions that the triangulation algorithm complexity is independent of N. It is interesting to note that for our version of the algorithm, N has in fact been completely removed from the calculation, due to the fact that we use a sparse pair-wise distance matrix as input.

According to this analysis, we expect the overall running time of the algorithm to be dominated by either $O(n^2)$ due to the subsampling or the term $O(L^3)$ due to the triangulation. In practice, subsampling is relatively fast, unless we are required to obtain a dense landmark sampling in order to accurately describe the non-manifold regions, i.e. if L is relatively large compared to n. Otherwise the reconstruction algorithm dominates, due in practice mainly to the repeated efforts required to solve the linear program in (6). In Section 3 we provide running times for the various examples that we use, listed according to algorithm stage.

3. Examples

In this section we validate our method using easily visualized examples. Each example is generated by sampling from a parameterization. The less common parameterizations are provided, and all of the parameterizations can be found in [29].

To check our results, we use our triangulations to compute Betti numbers for each example. Betti numbers are topological invariants which quantify large scale features of a space [30]. The first Betti number counts the connected components of the space; the second Betti number counts the non-contractible loops; and the third Betti number counts enclosed volumes. We compute Betti numbers using the Plex toolbox (www.comptop.stanford.edu) and Linbox (www.linalg.org). We use Plex to compute boundary operator matrices and Linbox to compute matrix ranks. The Betti numbers are obtained from the matrix ranks. For more information on Betti numbers and how they are computed, see [30].

3.1. Orientable surfaces

We first validated our method using some simple orientable manifold surfaces in 3D, namely a sphere, a torus and a double-torus. The datasets for each surface were generated using minimal pre-processing, assuming in all cases 2D projections (not using local dimension estimation): 10,000 points on a unit sphere were generated at random and subsampled using a landmark threshold $d_l = 0.1$ to obtain an 886 point dataset; 10,000 points on a torus with unit tube and inner radius were generated at random and subsampled using a landmark threshold $d_l = 0.3$ to obtain a 667 point dataset; and 20,000 points on a double torus (both tori with unit inner and tube radii) were generated at random and subsampled using a landmark threshold $d_l = 0.3$ to obtain an 813 point dataset.

To reconstruct these surfaces we used neighborhood sizes $\epsilon = 0.4$ for the sphere; $\epsilon = 1$ for the torus; and $\epsilon = 1.2$ for the double-torus. The resulting triangulations are shown in Fig. 2. Using Plex we computed the Betti numbers of the sphere to be $1, 0, 1$; the Betti numbers of the torus to be $1, 2, 1$; and the Betti numbers of the double-torus to be $1, 4, 1$. All of these values correctly reflect the topologies of the underlying surfaces.

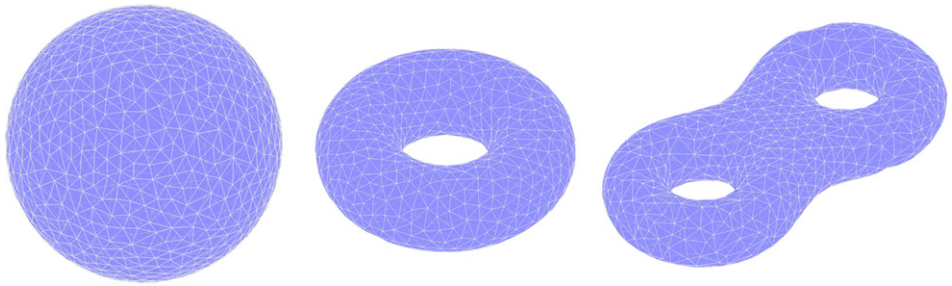

Fig. 2. Triangulations of orientable manifold surfaces. Here we show the triangulations obtained from point clouds representing a sphere (left), a torus (center), and a double-torus (right). Betti numbers for the sphere were computed to be $1, 0, 1$; Betti numbers for the torus were computed to be $1, 2, 1$; and Betti numbers for the double-torus were computed to be $1, 4, 1$.

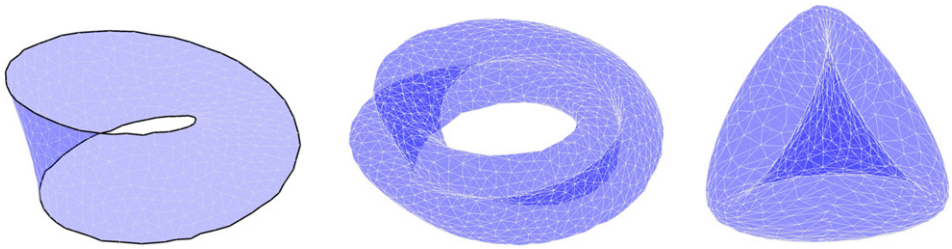

Fig. 3. Triangulations of non-orientable manifold surfaces. Here we show the triangulations obtained from point clouds representing the Mobius strip (left), the Klein bottle (center), and the real projective plane $\mathbb{R}P^2$ (right). In the case of the Mobius strip the boundary was treated as an intersection and is shown using a black curve. In the case of the Klein bottle, we show the figure 8 immersion, and for $\mathbb{R}P^2$ we show the Roman surface. Betti numbers for the Mobius strip were computed to be $1, 1, 0$; Betti numbers for the Klein bottle were computed to be $1, 1, 0$; and Betti numbers for $\mathbb{R}P^2$ were computed to be $1, 0, 0$.

3.2. Non-orientable surfaces

Next we tested our method using three standard non-orientable manifolds: the Mobius strip, the Klein bottle, and the real projective plane. The Mobius strip was generated using a two variable parameterization, where $0 \leqslant \theta < 2\pi$, $-1 \leqslant \rho \leqslant 1$, and

$$
\begin{aligned}
x &= \big(1 + (\rho/2)\cos(\theta/2)\big)\cos(\theta),\\
y &= \big(1 + (\rho/2)\cos(\theta/2)\big)\sin(\theta),\\
z &= (\rho/2)\sin(\theta/2).
\end{aligned}
$$

The Klein bottle was embedded in $\mathbb{R}^5$ as described by Freedman [13], using a parameterization of the figure 8 immersion in $\mathbb{R}^3$ given by

$$
\begin{aligned}
x &= \big(3 + \cos(\theta/2)\sin(\rho) - \sin(\theta/2)\sin(2\rho)\big)\cos(\theta),\\
y &= \big(3 + \cos(\theta/2)\sin(\rho) - \sin(\theta/2)\sin(2\rho)\big)\sin(\theta),\\
z &= \sin(\theta/2)\sin(\rho) + \cos(\theta/2)\sin(2\rho),
\end{aligned}
\tag{7}
$$

with two additional coordinates given by $\sin(\theta)$ and $\cos(\rho)$, where now $0 \leqslant \theta$, $\rho < 2\pi$. Finally, we embedded the real projective plane ($\mathbb{R}P^2$) into $\mathbb{R}^6$ using $(xy, xz, yz, x^2, y^2, z^2)$, where x, y, z are restricted to lie on the unit sphere in $\mathbb{R}^3$. The first three components of this embedding (xy, xz, yz) give a representation of $\mathbb{R}P^2$ known as the Roman surface.

The point clouds representing these non-orientable manifolds were again generated with minimal pre-processing, assuming 2D projections (no use of local dimension estimation). The Mobius strip dataset containing 416 points was obtained by subsampling from 10,000 randomly generated points using a landmark threshold of $d_l = 0.1$; the Klein bottle dataset containing 1940 points was obtained by subsampling from 10,000 randomly generated points using a landmark threshold of $d_l = 0.25$, and the $\mathbb{R}P^2$ dataset containing 753 points was obtained by subsampling from 100,000 randomly generated points using a landmark threshold of $d_l = 0.1$.

Neither our method nor Freedman's method for surface reconstruction handles manifolds with boundary. However, our method does handle intersections. Therefore to triangulate the Mobius strip, we considered the boundary to be an intersection. As discussed in Section 2.2, we visited each point in the boundary (intersection) first, using 1D projections and connecting points by edges. Then we proceeded with the full triangulation, never re-visiting the points on the boundary.

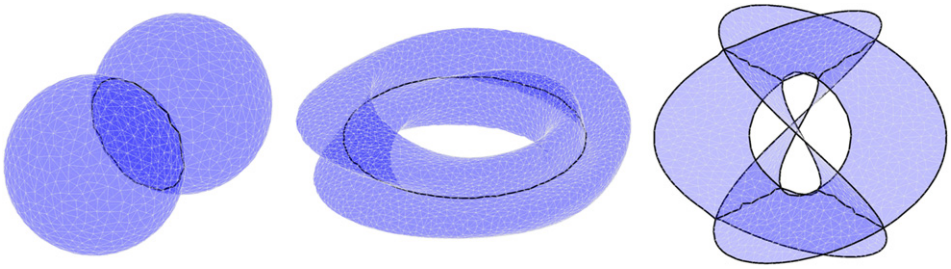

Fig. 4. Triangulations of non-manifold surfaces. Here we show triangulations obtained from point clouds representing two intersecting spheres (left), the figure 8 immersion of the Klein bottle (center), and Henneberg's surface (right). For the intersecting spheres and the figure 8 immersion, the circles of intersection are shown in black. In the case of the Henneberg surface, the intersections and boundaries are shown in black. Betti numbers for the two spheres were computed to be $1, 0, 3$; Betti numbers for the figure 8 immersion of the Klein bottle were computed to be $1, 2, 1$; and Betti numbers for Henneberg's surface were computed to be $1, 5, 0$.

We used a neighborhood size of $\epsilon = 0.4$. For the Klein bottle and $\mathbb{R}P^2$, we used neighborhood sizes of $\epsilon = 1$ and $\epsilon = 0.35$ respectively. The triangulations for the Mobius strip, Klein bottle, and $\mathbb{R}P^2$ are shown in Fig. 3.

For the three triangulations we used Plex and Linbox to compute the Betti numbers. For the Mobius strip we obtained $1, 1, 0$; for the Klein bottle we obtained $1, 1, 0$; and for $\mathbb{R}P^2$ we obtained $1, 0, 0$. In all cases these Betti numbers validate our triangulations.

3.3. Non-manifold surfaces

To test the ability of our algorithm to reconstruct non-manifold surfaces, we considered three additional examples: two intersecting spheres, the figure 8 immersion of the Klein bottle in $\mathbb{R}^3$, and Henneberg's minimal surface. The intersecting spheres were both unit radius, separated by a distance of $\sqrt{2}$. The figure 8 immersion was obtained using the (x, y, z) coordinates in (7). Henneberg's minimal surface was generated using the parameterization

$$\begin{aligned} x &= \frac{2(\rho^2-1)\cos(\theta)}{\rho} - \frac{2(\rho^6-1)\cos(3\theta)}{3\rho^3}, \\ y &= -\frac{6\rho^2(\rho^2-1)\sin(\theta) + 2(\rho^6-1)\sin(3\theta)}{3\rho^3}, \\ z &= \frac{2(\rho^4+1)\cos(2\theta)}{\rho^2}, \end{aligned} \tag{8}$$

where $0 \leqslant \theta \leqslant 2\pi$. Henneberg's minimal surface is an immersion of $\mathbb{R}P^2$ in $\mathbb{R}^3$ and in fact has a triple point (all immersions of $\mathbb{R}P^2$ have a triple point [31]). Since our algorithm cannot handle triple points, we remove the triple point by restricting $0.4 \leqslant \rho \leqslant 0.6$.

A dataset of 83,646 points representing the two intersecting spheres was generated by sampling at random from 100,000 points on each sphere, maintaining a minimum distance of 0.01 between any two points. Local dimension was determined using a neighborhood size of $\epsilon = 0.25$ and a dimension threshold of $d_t = 0.05$. For 3D neighborhoods we used a plane fitting threshold $d_p = 0.05$ and an intersection tolerance $\epsilon_p = 0.05$. After discovering these intersections, we re-sampled using a new minimum distance $d_l = 0.1$. This resulted in a final dataset of 1588 points, whose triangulation is shown in Fig. 4 (left).

Next, a dataset of 61,440 points representing the Klein figure 8 immersion was similarly generated by sampling at random from 100,000 points on the immersion, maintaining a minimum distance of 0.025 between any two points. Local dimension was determined using a neighborhood size of $\epsilon = 0.2$ and a dimension threshold of $d_t = 0.075$. For 3D neighborhoods we used a plane fitting threshold $d_p = 0.05$ and an intersection tolerance $\epsilon_p = 0.025$. This resulted in a final dataset of 4566 points. Due to the high curvature of the immersion, the neighborhoods using $\epsilon = 0.2$ were necessary to estimate the local dimension, but were too small for the triangulation algorithm. Thus, we had to re-compute our neighborhoods using a larger neighborhood size of $\epsilon = 0.4$, while maintaining the dimension estimates using the previous neighborhood size of $\epsilon = 0.2$. The $\epsilon = 0.4$ neighborhoods were used in the triangulation, shown in Fig. 4 (center).

In our last example, a dataset of 13,637 points representing Henneberg's minimal surface was generated by sampling at random from 100,000 points on the surface, maintaining a minimum distance of 0.25 between any two points. Local dimension was determined using a neighborhood size of $\epsilon = 4$ and a dimension threshold of $d_t = 0.05$. For 3D neighborhoods, a plane fitting threshold of $d_p = 1$ and an intersection tolerance of $\epsilon_p = 0.25$ were used. We next re-sampled the dataset using a landmark distance of $d_l = 0.75$, obtaining a final dataset of 1463 points.

The boundaries of Henneberg's surface are complicated and required special treatment. As in the case of the Mobius strip, we treated them as intersection points. Unlike the Mobius strip, however, the boundaries of Henneberg's surface have self-intersections. Due to this complication we had to use the original (ρ, θ) parameterization to identify the boundary points, whose coordinates were then re-computed using exact values of $\rho = 0.4$ or $\rho = 0.6$. We also had to ensure that

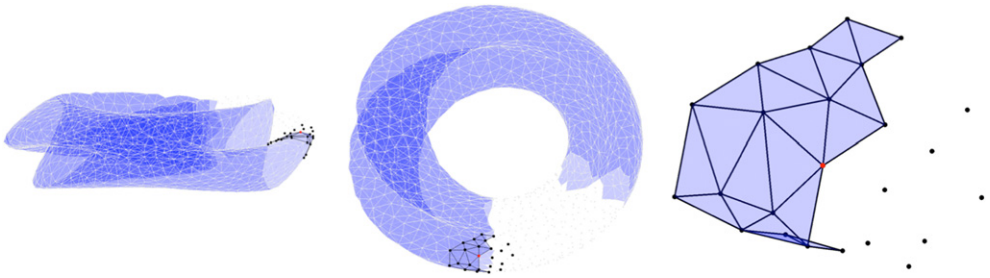

Fig. 5. Triangulation failure due to high curvature. Here we show an instance where the incremental triangulation fails due to a highly curved manifold. On the left a partially triangulated neighborhood is shown using black points surrounding a red neighborhood center. In the middle we show the same neighborhood from a different viewpoint. On the right we show the local approximation of the neighborhood obtained using MDS. In the local representation, the vertices of the overlapping triangles correspond to the vertices on the opposite side of the manifold from the neighborhood center, as seen in the viewpoint shown on the left. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

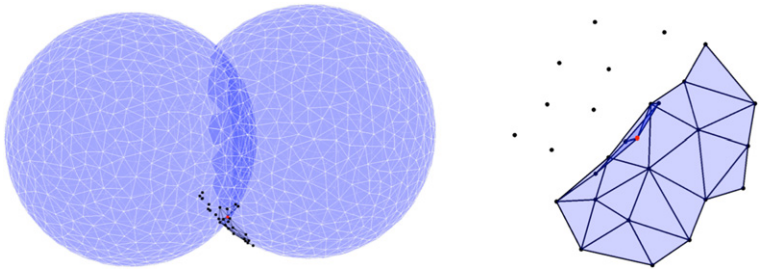

Fig. 6. Triangulation failure in the non-manifold case. Here we show how the algorithm can fail to characterize the non-manifold neighborhoods correctly. On the left, we show a partially triangulated neighborhood using black points surrounding a red neighborhood center. On the right we show the local approximation of the same neighborhood. Notice that the original neighborhood includes points from both spheres. This is because we allowed a poor fit for the two intersecting planes and thus failed to decompose the non-manifold neighborhood. As a result, the local approximation includes triangles on both spheres, resulting in overlapping triangles. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the sampling near the boundary intersections was sufficient by moving interior points closest to the boundary intersection (there are 32 such points – two for each intersecting plane) to within a distance of 1.25 of the boundary intersection. These steps ensured that the triangulation algorithm would correctly fill in the corners near the boundary intersections (a situation it was never designed to address). After accounting for the boundary of the Henneberg surface, the triangulation algorithm produced the result shown in Fig. 4 (right).

Finally, we used our triangulations with Plex and Linbox to compute Betti numbers for the two intersecting spheres, the figure 8 immersion, and Henneberg's surface. The Betti numbers for the two spheres were $1, 0, 3$; the Betti numbers for the figure 8 immersion were $1, 2, 1$; and the Betti numbers for Henneberg's surface were $1, 5, 0$. These numbers are verified theoretically in Appendix A.

3.4. Failures

As mentioned in Section 2, our algorithm is experimental, and we do not investigate guarantees of correctness (theoretical guarantees are also not considered by Freedman [13]). On the other hand, the algorithm works well in practice and seems to fail in relatively predictable ways. The most common mode of failure occurs when the underlying structure has high curvature, as illustrated using the Klein bottle example in Fig. 5. In this case the local approximation becomes inaccurate so that the projected triangulation is non-simplicial, i.e. that triangles overlap. The second, but less frequent mode of failure occurs when non-manifold neighbors are modeled incorrectly, as shown using the two intersecting spheres in Fig. 6. In this case the local approximation includes points perpendicular to the actual tangent space, again resulting in overlapping triangles.

Although we have included these examples as failures, they are in fact only failures because we have used the wrong parameters in the algorithm. In the case of the Klein bottle we used a neighborhood size that was too large relative to the curvature of the manifold ($\epsilon = 0.3$ instead of $\epsilon = 0.25$). In the case of the two intersecting spheres, we used a looser tolerance for fitting the two planes to a singular neighborhood ($d_p = 0.25$ instead of $d_p = 0.05$). In both cases the correct triangulations are obtained by lowering the tolerances of the algorithm. Nevertheless, these failures provide insight into the workings of the algorithm.

Table 2
Example run times. Here we show the run times obtained for the different examples investigated in this section. For each example we provide the number of points n, number of landmarks L, neighborhood size k, time in seconds for pre-processing, and time in seconds for reconstruction.

Example	n	L	k	Pre-proc.	Recon.
Sphere	10,000	886	36	1.7	368.2
Torus	10,000	667	28	1.1	220.5
Double torus	20,000	813	26	3.7	263.1
Mobius strip	10,000	416	23	0.9	123.7
Klein figure 8	10,000	1940	33	3.8	778.0
$\mathbb{R}P^2$	100,000	753	35	11.7	302.0
Two spheres	83,646	1588	13	446.4	344.4
Klein immersion	61,440	4566	14	295.7	1183.3
Henneberg	13,637	1463	39	40.9	723.4

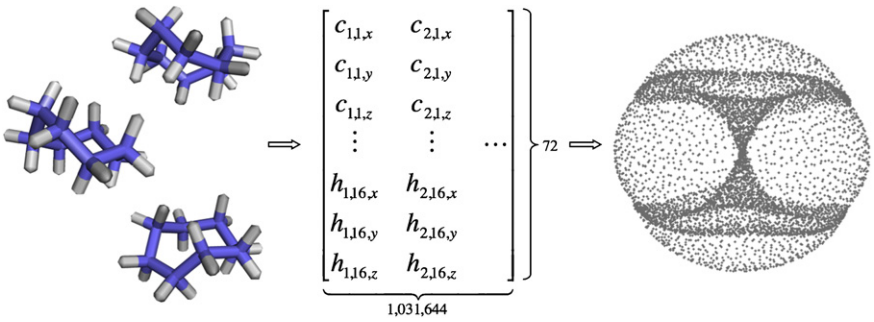

Fig. 7. Conformation space of cyclo-octane. Here we show how the set of conformations of cyclo-octane can be represented as a surface in a high-dimensional space. On the left, we show various conformations of cyclo-octane as drawn by PyMol (www.pymol.org). In the center, these conformations are represented by the 3D coordinates of their atoms. The coordinates are concatenated into vectors and shown as columns of a data matrix. As an example, the entry $c_{1,1,x}$ of the matrix denotes the x-coordinate of the first carbon atom in the first molecule. On the right, the Isomap method is used to obtain a lower-dimensional visualization of the data.

3.5. Run times

The run times for the nine examples we have investigated are shown in Table 2. These times were obtained on a 2.26 GHz Intel Xeon dual quadcore workstation with 16 GB of RAM. The algorithm was implemented in Matlab (www.mathworks.com) using the optimization toolbox to solve the linear program in (6). Table 2 shows that pre-processing is negligible except for the non-manifold examples. In the case of the non-manifold examples, the pre-processing is generally faster than the triangulation.

4. Application

Cyclo-octane is a saturated eight-member cyclic compound with chemical formula C_8H_{16}. Cyclo-octane has received attention in computational chemistry because it has multiple conformations of similar energy, a complex potential energy surface, and significant (steric) influence from the hydrogen atoms on preferred conformations [32–34]. Cyclo-octane is also interesting because there are enumerative algorithms available which can provide a dense sampling of the conformation space [35,36]. These algorithms show from first principles that the resulting conformation space has two degrees of freedom, suggesting that the space is a surface (but not necessarily a manifold).

Using dimension reduction methods, we have previously analyzed the cyclo-octane conformation space [16]. In our analysis, we used a dataset of 1,031,644 cyclo-octane conformations, enumerated using the triaxial loop closure algorithm of Coutsias et al. [35]. Each conformation is placed in Cartesian space via the 3D position coordinates of each atom in the molecule. The conformations are then aligned to a reference conformation such that the Eckart conditions are satisfied [37]. The final positions of a given conformation are concatenated to obtain a vector in $\mathbb{R}^{72}$. The resulting collection is a dataset $\{\mathbf{x}_i\}_{i=1}^{1,031,644} \subset \mathbb{R}^{72}$ which is presumed to describe a surface. In Brown et al. [16] we applied a variety of dimension reduction methods to the cyclo-octane dataset, one of which was Isomap [38]. A summary of our analysis using the Isomap reduction is shown in Fig. 7.

Beyond dimension reduction, the next step in our analysis is surface reconstruction. Unfortunately, the Isomap representation of the cyclo-octane conformation space is only a visualization, and is not accurate enough for use with a 3D surface reconstruction methods. Therefore we applied Freedman's algorithm for surface reconstruction in the original high-dimensional conformation space. Freedman's method failed because the surface had self-intersections of the type discussed in this paper. Thus we developed our method for non-manifold surface reconstruction and applied it to the cyclo-octane dataset.

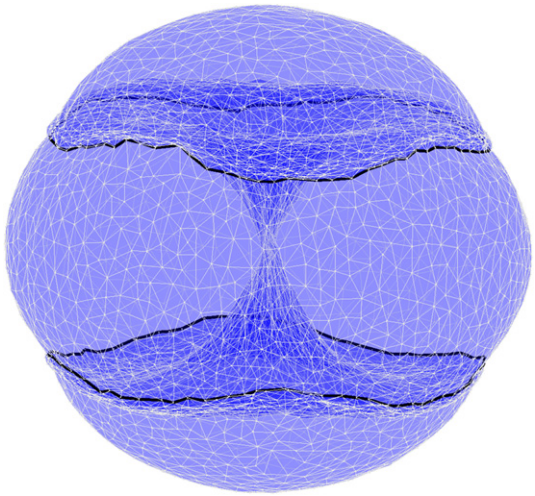

Fig. 8. Triangulation of cyclo-octane conformation space. Here we show the triangulation obtained by our surface reconstruction algorithm on the cyclo-octane conformation space. The triangulation was carried out in 24 dimensions, but is shown using the reduced-dimensional representation provided by Isomap. Self-intersections are shown in black.

To reduce complexity and avoid potential error due to hydrogen placement, we used only ring atoms to obtain a dataset $\{\mathbf{x}_i\}_{i=1}^{1,031,644} \subset \mathbb{R}^{24}$. We applied our algorithm to this dataset using parameters $\epsilon = 0.23$, $d_t = 0.05$, $d_p = 0.01$, and $\epsilon_p = 0.02$. We used five different values of d_l, given by 0.08, 0.09, 0.10, 0.11, and 0.12. We produced five different triangulations with 6040; 7114; 8577; 10,503; and 13,194 vertices.

We used the Plex and Linbox toolboxes to check the accuracy of the triangulations. For each of the five triangulations, we verified that every neighborhood B_i (before decomposition) had Betti numbers $1, 0, 0$. This is an accuracy check because any neighborhood B_i should be homotopic to a point and should therefore have Betti numbers $1, 0, 0$. We also computed Betti numbers for each of the five full triangulations. In all cases we found the Betti numbers to be $1, 1, 2$. This consistency strongly suggests that the triangulations are all representative of the actual conformation space. A visualization of the triangulation with 6044 vertices using the Isomap coordinate representation is shown in Fig. 8. Further analysis of the cyclo-octane conformation space using this triangulation will be described in a future publication.

5. Discussion and conclusions

Non-manifold surface reconstruction from high-dimensional point cloud data is a difficult problem. Building on Freedman's incremental algorithm for manifold surface reconstruction [13], we identify and treat the non-manifold portions of a surface using a pre-processing step. In our pre-processing step we employ an algorithm for fitting two planes to a neighborhood of points.

Since our algorithm is based on Freedman's work, we can make many of the same observations about its properties and performance [13]. First, our algorithm is experimental and offers no theoretical guarantees of accuracy or correctness. Second, we must choose various parameters to ensure a successful result. While Freedman's parameters were largely based on angle, ours are based on distance. Of course we also have additional parameters for our plane fitting method. Third and last, the algorithm performs well in practice, as we have shown using many examples and one application.

Not considered in this work is any unusual distribution of data. In fact, we have expended extra effort to ensure uniform sampling, exploiting the idea of landmark points from dimension reduction (Section 2.1) to achieve this ideal. Freedman makes no such assumption, and neither is such an assumption necessary in our plane fitting algorithm. Thus our algorithm should still work in the case of unusual data distribution. However, we have not tested this assertion. If achievable, a uniform distribution is desirable for many reasons, including more accurate identification of non-manifold intersections, fewer samples to triangulate, and a higher likelihood of equilateral ("good") triangles in the final triangulation.

Also not considered in this work is dataset error. This was also not examined by Freedman, but presumably dataset error would have a large effect on the triangulation algorithm. Points not on the surface would likely be projected on top of other points, resulting in many overlapping triangles. Error might not affect our plane fitting algorithm, since we are performing a least squares fit, but it would certainly cause difficulties in decomposing a non-manifold neighborhood. Perhaps the best suggestion for avoiding these types of problems would be aggressive application of landmark sampling. If the minimal distance d_l was greater than the error, then we would expect to avoid the projection of points on top of other points. Indeed, this is how we avoid error in our identification of the surface self-intersections.

Finally, there are a few extensions of this work that might be considered. In addition to ϵ neighborhoods, we could of course use k nearest neighbors. We could use adaptive neighborhoods for different regions of the surface, or for more

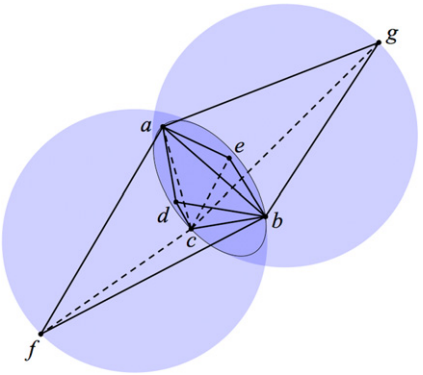

Fig. 9. Complex for two intersecting spheres. Here we show a 7 vertex complex which has the same topology as the two intersecting spheres. The vertices a, b, c, d, e, f, g are connected as shown, where a, b, c are on the ring of intersection, e, f are on the left-hand sphere, and d, g are on the right-hand sphere. The complex consists of faces abd, acd, bcd, abe, ace, bce, abf, acf, bcf, abg, acg, bcg along with associated edges.

accurate identification of self-intersections. (This was done in the example using the Klein bottle figure 8 immersion in Section 3.)

Perhaps more interesting would be the use of geodesic distances for the triangulation. Geodesic distances can be easily computed via the method proposed in the Isomap algorithm [38]. These distances could be substituted for the Euclidean distances used in the current implementation to avoid problems with high surface curvature.

Lastly, we consider only self-intersections which can be modeled by two intersecting planes (double curves). By using a method such as generalized PCA [25] (mentioned in Section 2.1), we might be able to model other types of singularities, including triple points, or lines intersecting planes. Such models would allow us to decompose additional types of non-manifold neighborhoods and apply our approach to even more complicated surfaces.

Acknowledgements

We would like to thank Mark D. Rintoul, Chris Baker, and Scott Mitchell at Sandia for discussions of the algorithm described in this paper. This project was supported by the Sandia Computer Science Research Fund. Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

Appendix A

Here we compute the Betti numbers for the non-standard surfaces used as examples in Fig. 4. As mentioned in Section 3, the Betti numbers are obtained via the matrix ranks of boundary operators [30]. For a surface we have three relevant boundary operators: the operator ∂_0 which takes vertices to the null set; the operator ∂_1 which takes edges to vertices; and the operator ∂_2 which takes triangles to edges. These operators can be expressed as matrices such that

$$\beta_0 = \dim\ker\partial_0 - \dim\operatorname{im}\partial_1,$$
$$\beta_1 = \dim\ker\partial_1 - \dim\operatorname{im}\partial_2,$$
$$\beta_2 = \dim\ker\partial_2.$$

Note that $\dim\ker\partial_0$ is just the number of vertices in the complex, since ∂_0 maps to the null set.

A.1. Two intersecting spheres

The two intersecting spheres are topologically equivalent to the 7 vertex complex shown in Fig. 9. In this complex we have 7 vertices and 15 edges so that ∂_1 is a 7×15 matrix as shown in Table 3. There are 12 faces so that ∂_2 is a 15×12 matrix as shown in Table 4. Now using the boundary matrices we can compute

$$\beta_0 = \dim\ker\partial_0 - \dim\operatorname{im}\partial_1 = 7 - 6 = 1,$$
$$\beta_1 = \dim\ker\partial_1 - \dim\operatorname{im}\partial_2 = 9 - 9 = 0,$$
$$\beta_2 = \dim\ker\partial_2 = 3.$$

Table 3
Matrix ∂_1 for the seven vertex complex representing the two intersecting spheres.

∂_1	ab	ac	bc	ad	bd	cd	af	bf	cf	ae	be	ce	ag	bg	cg
a	−1	−1	0	−1	0	0	−1	0	0	−1	0	0	−1	0	0
b	1	0	−1	0	−1	0	0	−1	0	0	−1	0	0	−1	0
c	0	1	1	0	0	−1	0	0	−1	0	0	−1	0	0	−1
d	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
f	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
g	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Table 4
Matrix ∂_2 for the seven vertex complex representing the two intersecting spheres.

∂_2	abd	acd	bcd	abe	ace	bce	abf	acf	bcf	abg	acg	bcg
ab	1	0	0	1	0	0	1	0	0	1	0	0
ac	0	1	0	0	1	0	0	1	0	0	1	0
bc	0	0	1	0	0	1	0	0	1	0	0	1
ad	−1	−1	0	0	0	0	0	0	0	0	0	0
bd	1	0	−1	0	0	0	0	0	0	0	0	0
cd	0	1	1	0	0	0	0	0	0	0	0	0
af	0	0	0	0	0	0	−1	−1	0	0	0	0
bf	0	0	0	0	0	0	1	0	−1	0	0	0
cf	0	0	0	0	0	0	0	1	1	0	0	0
ae	0	0	0	−1	−1	0	0	0	0	0	0	0
be	0	0	0	1	0	−1	0	0	0	0	0	0
ce	0	0	0	0	1	1	0	0	0	0	0	0
ag	0	0	0	0	0	0	0	0	0	−1	−1	0
bg	0	0	0	0	0	0	0	0	0	1	0	−1
cg	0	0	0	0	0	0	0	0	0	0	1	1

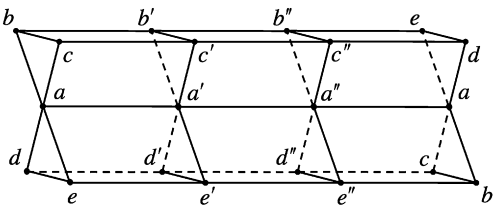

Fig. 10. Complex for figure 8 immersion. The figure 8 immersion is made from a double tube having cross-section shaped like an "8" which is twisted once and connected in a loop. The "8" is identified in our complex using vertices a, b, c, d, e. As the loop is twisted we use a', b', c', d', e', a'', b'', c'', d'', e''. The twist is shown as an equivalence on the right end of the figure. Also included in the complex, but not shown here are 36 triangular faces which fill in the rectangular faces shown. For example, rectangle $aa'c'c$ is made of triangles $aa'c'$ and acc'.

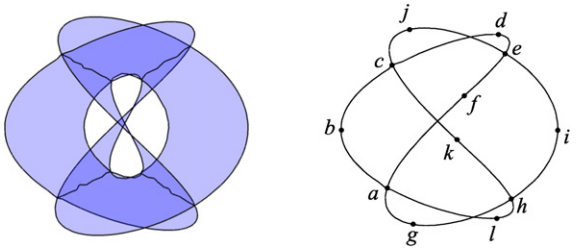

Fig. 11. Complex for Henneberg's surface. Our restriction of Henneberg's surface in Eq. (8) using $0.4 \leqslant \rho \leqslant 0.6$ is shown on the left. This restriction removes the triple point at the origin, and is therefore topologically equivalent to the boundary when $\rho = 0.6$, shown on the right. Thus to compute the Betti numbers for our restriction of Henneberg's surface, we simply label the intersections and each edge using a, b, c, d, e, f, g, h, i, j, k, l as shown on the right. This forms a complex with 12 vertices and 16 edges.

A.2. Figure 8 immersion

The figure 8 immersion consists of a double tube having a cross-section shaped like an "8" which is twisted once and connected in a loop. As such it can be represented using the 15 vertex complex shown in Fig. 10. This complex has vertices a, b, c, d, e, a', b', c', d', e', a'', b'', c'', d'' and e''. There are 36 faces $aa'b'$, abb', $aa'c'$, acc', $bb'c'$, bcc', $aa'e'$, aee', $aa'd'$, add', $dd'e'$, dee', $a'a''b''$, $a'b'b''$, $a'a''c''$, $a'c'c''$, $b'b''c''$, $b'c'c''$, $a'a''e''$, $a'e'e''$, $a'a''d''$, $a'd'd''$, $d'd''e''$, $d'e'e''$, $a''ae$, $a''b''e$, $a''ad$, $a''c''d$, $b''ed$, $b''c''d$, $a''ac$, $a''d''c$, $a''ab$, $a''e''b$, $d''cb$, $d''e''b$. Also included in the complex are the edges associated with the faces

listed. Although tedious, it is again straightforward to compute the Betti numbers of the figure 8 immersion by calculating the ranks of the boundary maps. The Betti numbers are $1, 2, 1$.

A.3. Henneberg's minimal surface

Our restriction of Henneberg's minimal surface using $0.4 \leqslant \rho \leqslant 0.6$ in Eq. (8) is topologically equivalent to the boundary using $\rho = 0.6$, as shown in Fig. 11. After labeling each intersection and edge in the $\rho = 0.6$ boundary, we obtain a complex with 12 vertices described by edges ab, bc, cd, de, ef, fa, ag, gh, hi, ie, ej, jc, ck, kh, hl, la. This complex has Betti numbers $1, 5, 0$.

References

[1] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, Surface reconstruction from unorganized points, in: Proc. 25th Special Interest Group on Graphics SIGGRAPH'92, 1992, pp. 71–78.
[2] H. Hoppe, Surface reconstruction from unorganized points, PhD thesis, Dept. of Computer Science and Engineering, University of Washington, 1994.
[3] F. Bernardini, C.L. Bajaj, Sampling and reconstructing manifolds using alpha-shapes, in: Proc. 9th Canad. Conf. Comput. Geom., 1997, pp. 193–198.
[4] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, G. Taubin, The ball-pivoting algorithm for surface reconstruction, IEEE Trans. Visual. Comput. Graph. 5 (4) (1999) 349–359.
[5] N. Amenta, M. Bern, Surface reconstruction by Voronoi filtering, Discrete Comput. Geom. 22 (1999) 481–504.
[6] N. Amenta, S. Choi, R.K. Kolluri, The power crust, in: Proc. 6th ACM Symposium on Solid Modeling, 2001, pp. 249–260.
[7] N. Amenta, S. Choi, T.K. Dey, N. Leekha, A simple algorithm for homeomorphic surface reconstruction, Internat. J. Comput. Geom. Appl. 12 (1–2) (2002) 125–141.
[8] T.K. Dey, J. Giesen, Detecting undersampling in surface reconstruction, in: Proc. 17th Symposium on Comput. Geom. SCG'01, 2001, pp. 257–263.
[9] T.K. Dey, S. Goswami, Tight cocone: a water-tight surface reconstructor, in: Proc. 8th Symp. on Solid Modeling SM'03, 2003, pp. 127–134.
[10] S.-W. Cheng, T.K. Dey, E.A. Ramos, Manifold reconstruction from point samples, in: SODA'05: Proc. 16th ACM–SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, PA, USA, 2005, pp. 1018–1027.
[11] H. Edelsbrunner, Surface reconstruction by wrapping finite point sets in space, in: B. Aronov, S. Basu, J. Pach, M. Sharir (Eds.), Ricky Pollack and Eli Goodman Festschrift, Springer-Verlag, 2003, pp. 379–404.
[12] P. McMullen, The maximum number of faces of a convex polytope, Mathematika 17 (1970) 179–184.
[13] D. Freedman, An incremental algorithm for reconstruction of surfaces of arbitrary codimension, Comput. Geom. Theory Appl. 36 (2) (2007) 106–116.
[14] Y. Duan, Q. Hong, 2.5d active contour for surface reconstruction, in: Proc. 8th Int. Workshop on Vision, Modeling and Visualization VMV'03, 2003, pp. 431–439.
[15] J. Wang, M. Oliveira, A. Kaufman, Reconstructing manifold and non-manifold surfaces from point clouds, in: Proc. of IEEE Visualization, VIS'05, 2005, pp. 415–422.
[16] W.M. Brown, S. Martin, S.N. Pollock, E.A. Coutsias, J.-P. Watson, Algorithmic dimensionality reduction for molecular structure analysis, J. Chem. Phys. 129 (6) (2008) 064118.
[17] M. Kirby, Geometric Data Analysis: An Empirical Approach to Dimensionality Reduction and the Study of Patterns, John Wiley & Sons, Inc., 2001.
[18] K. Fukunaga, D.R. Olsen, An algorithm for finding intrinsic dimensionality of data, IEEE Trans. Comput. 20 (1971) 176–183.
[19] F. Camastra, A. Vinciarelli, Estimating the intrinsic dimension of data with a fractal-based method, IEEE Trans. Pattern Anal. Mach. Intell. 24 (10) (2002) 1404–1407.
[20] B. Kégl, Intrinsic dimension estimation using packing numbers, in: S. Becker, S. Thrun, K. Obermayer (Eds.), Advances in Neural Information Processing Systems (NIPS), vol. 15, MIT Press, 2003, pp. 681–688.
[21] E. Levina, P.J. Bickel, Maximum likelihood estimation of intrinsic dimension, in: L.K. Saul, Y. Weiss, L. Bottou (Eds.), Advances in Neural Information Processing Systems (NIPS), vol. 17, MIT Press, 2004, pp. 777–784.
[22] J. Costa, A. Hero, Geodesic entropic graphs for dimension and entropy estimation in manifold learning, IEEE Trans. Signal Process. 52 (8) (2004) 2210–2221.
[23] A. Dresden, Solid Analytical Geometry and Determinants, Dover Publications, New York, 1964.
[24] R. Unnikrishnan, M. Hebert, Denoising manifold and non-manifold point clouds, in: 18th British Machine Vision Conference (BMVC), 2007.
[25] R. Vidal, Y. Ma, S. Sastry, Generalized principal component analysis (gpca), IEEE Trans. Pattern Anal. Mach. Intell. 27 (12) (2005) 1945–1959.
[26] V. de Silva, J.B. Tenenbaum, Global versus local methods in nonlinear dimensionality reduction, in: S.T.S. Becker, K. Obermayer (Eds.), Advances in Neural Information Processing Systems, vol. 15, MIT Press, Cambridge, MA, 2002, pp. 705–712.
[27] J. Silva, J. Marques, J. Lemos, Selecting landmark points for sparse manifold learning, in: Y. Weiss, B. Schölkopf, J. Platt (Eds.), Advances in Neural Information Processing Systems (NIPS), vol. 18, MIT Press, Cambridge, MA, 2005, pp. 1241–1248.
[28] T.F. Cox, M.A.A. Cox, Multidimensional Scaling, CRC Press, 2001.
[29] A. Gray, Modern Differential Geometry of Curves and Surfaces with Mathematica, 2nd ed., CRC Press, Inc., Boca Raton, FL, USA, 1997.
[30] G. Rote, G. Vegter, Computational topology: An introduction, in: J.-D. Boissonnat, M. Teillaud (Eds.), Effective Computational Geometry for Curves and Surfaces, Mathematics and Visualization, Springer-Verlag, 2006, pp. 277–312.
[31] P. Henrique, R.F.V. Viana, S. Northrup, Self intersections in immersions of the projective plane, in: Proc. of the International Research Experience for Students in Mathematics (IRES), 2007.
[32] J.B. Hendrickson, Molecular geometry. V. Evaluation of functions and conformations of medium rings, J. Amer. Chem. Soc. 89 (26) (1967) 7036–7043.
[33] W.R. Rocha, J.R. Pliego, S.M. Resende, H.F.D. Santos, M.A.D. Olivera, W.B. de Almeida, Ab initio conformational analysis of cyclooctane molecule, J. Comput. Chem. 19 (1998) 524–534.
[34] B.R. K, Conformational properties of cyclooctane: a molecular dynamics simulation study, Mol. Phys. 98 (2000) 211–218.
[35] E.A. Coutsias, C. Seok, M.J. Wester, K.A. Dill, Resultants and loop closure, Int. J. Quantum Chem. 106 (2006) 176–189.
[36] J.M. Porta, L. Ros, F. Thomas, F. Corcho, J. Cantó, J.J. Pérez, Complete maps of molecular-loop conformational spaces, J. Comput. Chem. 28 (13) (2007) 2170–2189.
[37] A.Y. Dymarsky, K.N. Kudin, Computation of the pseudorotation matrix to satisfy the Eckart axis conditions, J. Chem. Phys. 122 (12) (2005) 124103.
[38] J.B. Tenenbaum, V. de Silva, J.C. Langford, A global geometric framework for nonlinear dimensionality reduction, Science 290 (5500) (2000) 2319–2323.