https://doi.org/10.1016/j.cag.2017.07.003

Computers & Graphics (2017)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/results-in-physics

Efficiently computing feature-aligned and high-quality polygonal offset surfaces

Wenlong Meng[a], Shuangmin Chen[a,*], Zhenyu Shu[b], Shi-Qing Xin[c], Hongbo Fu[d], Changhe Tu[c]

[a] *Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo 315211, China*
[b] *School of Information Science and Engineering, Ningbo Institute of Technology, Zhejiang University, Ningbo 315100, China*
[c] *School of Computer Science and Technology, Shandong University, Jinan 250101, China*
[d] *School of Creative Media, the City University of Hong Kong, Hongkong, China*

ARTICLE INFO

Article history:
Received July 12, 2017

Keywords: Offsetting, Particle system, Feature alignment

ABSTRACT

3D surface offsetting is a fundamental geometric operation in CAD/CAE/CAM. In this paper, we propose a super-linear convergent algorithm to generate a well-triangulated and feature-aligned offset surface based on particle system. The key idea is to distribute a set of moveable sites as uniformly as possible while keeping these sites at a specified distance away from the base surface throughout the optimization process. In order to make the final triangulation align with geometric feature lines, we use the moveable sites to predict the potential feature regions, which in turn guide the distribution of moveable sites. Our algorithm supports multiple kinds of input surfaces, e.g., triangle meshes, implicit functions, parametric surfaces and even point clouds. Compared with existing algorithms on surface offsetting, our algorithm has significant advantages in terms of meshing quality, computational performance, topological correctness and feature alignment.

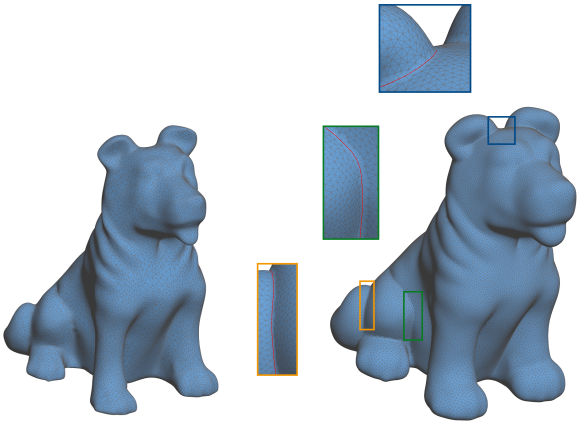
1. Introduction

An *offset* surface (Maekawa, 1999), also called a parallel surface, consists of all the points that are at a constant distance d to an input surface. The computation of surface offsets is a common and fundamental operation in various applications in CAD/CAE/CAM (Kim et al., 2004; Pham, 1992; Kim and Yang, 2005), e.g., hollowed or shelled solid model generation for rapid prototyping.

There is a large body of literature on computing offset surfaces. Existing methods can be roughly divided into three categories depending on the specific representation form of the input surface. For parametric curves or surfaces, a commonly used approach (Farouki, 1985, 1986; Martin and Stephenson, 1990) is to generate parametric offsets first, followed by carefully handling tangent discontinuities, cusps and self-intersections. When the input is a polygonal surface or implicit surface (Maekawa, 1999; Pavi and Kobbelt, 2008; Chen and Wang, 2011), one has to build a volumetric scalar field with a dense resolution and then extract the iso-surface at the specified distance. However, such an approach has at least two disadvantages including (1) it requires a huge time/space cost since the total number of voxels is $O(1/\epsilon^3)$, where ϵ is the accuracy tolerance, and (2) the final offset surface does not have a desirable triangulation quality. Finally, it seems that offset surfaces can be obtained by a series of mesh boolean operations (Zhou et al., 2016) across a sufficiently large number of spheres centered at the base surface, but experimental results show that it cannot work well in practice due to the fact that the meshing quality gets worse and worse after many boolean operations. This motivates us to develop an easy-to-use tool for generating a well-triangulated and feature-aligned offset for an input surface that can be a polygonal surface, a parametric surface, an implicit surface, or even a point cloud.

In this paper, we propose a super-linear convergent algorithm

*Corresponding author.
e-mail: chenshuangmin@nbu.edu.cn (Shuangmin Chen)



(a) Input (b) Offset surface

Fig. 1. Our algorithm is able to produce a feature-aligned and high-quality offset surface (b) for the input surface (a); See the close-up views.

to generate polygonal offsets. The key idea is to distribute a set of moveable sites as uniformly as possible while keeping these sites at a specified distance from the original surface throughout the optimization process. Because of the uniform distribution of these sites, an additional quick step of simply connecting sites is sufficient for producing the final triangle mesh. An example is shown in Figure 1.

Our main contributions are at least threefold:

1. Taking the uniformity of sites as the objective function whereas the specified distance to the base surface as the hard constraint, we formulate the offsetting problem using particle system, which can be efficiently solved due to the closed-form formula of the gradients of the objective function.
2. Throughout the optimization process, we use the moveable sites to predict the potential feature regions of the final offset surface, which is in turn enforced on the objective function to guide the distribution of the moveable sites, leading to a feature-aligned triangulation.
3. The algorithm framework is powerful and supports various kinds of input surfaces, including polygonal surfaces, parametric surfaces, implicit surfaces and even point clouds.

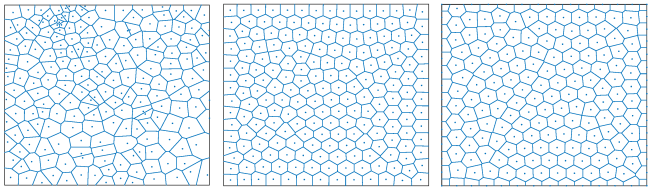
2. Related Work

At least three kinds of works are related to the theme of this paper, including surface offsetting, particle system, and remeshing.

Surface Offsetting. Existing offset algorithms assume that the input surface has a specific representation form. When the input surface has a parametric form, it is quite often to represent the offset surface as a parametric form as well. Existing algorithms of this kind focus on seeking a polynomial/rational alternative to approximate the exact parametric form, and handling tangent discontinuities, cusps and self intersections. For example, Filip et al. (Filip et al., 1986) developed a theorem on approximation accuracy using the bounds of second derivatives of the original curves and surfaces. Piegl and Tiller (Piegl and Tiller, 1999) proposed to approximate the offset surface with the fewest number of control points. Kumar et al. (Kumar et al., 2002) developed a set of trimming techniques to handle invalid local intersections. The above-mentioned methods, whose input and output are both in parametric form, are different from the goal in this paper, i.e., generating a high-quality polygonal offset surface.

When the input is a polygonal or implicit surface, one can build a volumetric scalar field to encode signed distances to the base surface and then extract the offset surface based on the marching cube technique (Liu and Wang, 2011; Pavi and Kobbelt, 2008; Varadhan and Manocha, 2004). However, the resolution of voxelization is hard to set. Coarse voxelization may lead to a topologically incorrect reconstructed offset surface but an over-dense voxelization requires a huge time/space cost. What's important is that it cannot produce a high-quality triangle mesh to represent the offset surface.

Theoretically speaking, mesh boolean operations (Zhou et al., 2016) seem to be able to compute the offsets individually for each face, edge, and vertex and then return the union of the basic offset elements as the final offset surface. However, experimental results show that mesh boolean operations cannot work well in practice. First, these basic offset elements highly overlap, causing a notorious difficulty in unionizing a large number of such objects. Second, performing mesh boolean operations across a large number of objects is inefficient and cannot guarantee a desirable meshing quality. Similarly, point based reconstruction algorithms (Chen and Wang, 2011; Lien, 2008), based on point shifting and filtering operations, cannot guarantee the meshing quality either.



(a) Input (b) CVT (c) Particle system

Fig. 2. For 200 input sites (a), CVT requires about 0.45 seconds and 91 iterations to get the distribution in (b), while the particle system requires only 0.01 seconds and 48 iterations to achieve (c). Note that the distribution in (c) is sufficient for the triangulation purpose in practice.

Particle System v.s. CVT. There are many application occasions where we need to distribute a set of sites as uniformly as possible. Both centroidal Voronoi tessellations (CVT) (Du et al., 1999; Okabe et al., 2009) and particle systems (Turk, 1992; Witkin and Heckbert, 1994; Meyer et al., 2007; Bronson et al., 2012) can serve for this purpose. Du and Wang (Du and Wang, 2003) introduced the Lloyd method to compute CVT and apply it into optimal tetrahedral mesh generation, while Liu et al. (Liu et al., 2009) proposed a quasi-Newton method to compute CVT and demonstrated the extraordinary ability in surface remeshing. Particle system, by contrast, has a sound basis in

physics and can serve for the same purpose by minimizing the global inter-particle forces to make the particles (sites, or vertices) keep the optimal balanced state, leading to a collection of uniformly distributed particles. Generally speaking, particle system is able to generate a desirable site distribution with less computational cost (Zhong et al., 2014) in contrast to CVT. As Figure 2 shows, particle system runs about many times faster than CVT in producing a uniform distribution of almost the same quality. Therefore, in this paper, we adopt particle system to iteratively optimize the distribution of sites (serving as vertices of the final offset surface).

Remeshing. A wide range of applications require meshes with high-quality triangulation to facilitate numerical computation, and thus remeshing is an important research topic in computer graphics. Roughly speaking, there are three kinds of remeshing depending on various purposes. The first kind targets at uniform triangulation, which seeks for an as-uniform-as-possible vertex distribution (Liu et al., 2009). The second kind of remeshing algorithms aims at isotropic or anisotropic triangulation assuming that the base surface is equipped with a density function or an anisotropic metric to encode the underlying distance. For example, Chen et al. (Chen et al., 2012) developed an isotropic remeshing method based on constrained centroidal Delaunay mesh(CCDM), while Zhong et al. (Zhong et al., 2013) introduced a particle-based approach for anisotropic surface meshing. The third kind is to align triangulation with geometric features. For example, Lai et al. (Lai et al., 2010) presented an algorithm which turns an unstructured triangle mesh into a quad dominant mesh with mesh edges well aligned to the principal directions of the underlying surface.

3. Problem Formulation

3.1. *Conventional formulation*

Suppose that the input surface is closed and orientable, and has a parametric form $S = S(u,v), (u,v) \in \Omega \subset \mathbb{R}^2$, for this moment. Let $\mathbf{n}(u,v)$ be the unit normal vector at each point $(x(u,v), y(u,v), z(u,v)) \in S$. Then the offset surface can be represented by

$$O_d(u,v) = \{(x(u,v), y(u,v), z(u,v)) + d \cdot \mathbf{n}(u,v) \mid (u,v) \in \Omega \subset \mathbb{R}^2\}. \quad (1)$$

It offsets the original surface outward if $d > 0$ and inward otherwise. However, such a formulation has at least two disadvantages. First, there may be redundant parts and an additional trimming operation is required. To our knowledge, the trimming operation is tedious and highly non-trivial. Second, it only supports a parametric surface as the input. Therefore, a better formulation of this problem is badly needed.

3.2. *Particle system based formulation*

Suppose that there is a collection of moveable sites $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$ to serve as the vertices of the final polygonal offset surface. On the one hand, we hope that $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$ is as uniform as possible. This can be achieved by minimizing the following energy function:

$$E(\mathbf{X}) = \sum_i^n \sum_j^n e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{4\sigma^2}}, \quad (2)$$

where σ, called the kernel width, is used to adjust the influence region for each site. On the other hand, we have to set a hard constraint that each $\mathbf{x}_i$ must be lying on the offset surface, i.e.,

$$\|\mathbf{x}_i - \mathbf{x}_i^S\| = d, i = 1, 2, \cdots, N, \quad (3)$$

where $\mathbf{x}_i^S$ is the projection (or closest point) of $\mathbf{x}_i$ onto the primitive surface S and it can be determined depending on specific situations.

Parametric surface. When the input surface has a parametric form, $\mathbf{x}_i^S$ can be found by solving an optimization problem, i.e., seeking for a pair of parameters (u^*, v^*) such that the squared distance

$$\|\mathbf{x}_i - S(u^*, v^*)\|^2 = (\mathbf{x}_i - S(u^*, v^*))^{\mathrm{T}}(\mathbf{x}_i - S(u^*, v^*)) \quad (4)$$

is minimized.

Implicit surface. When the input surface is an implicit surface $F(x) = 0$, $\mathbf{x}_i^S$ can be found by considering the following constrained optimization problem

$$\begin{aligned} \text{Minimize} \quad & \|\mathbf{x}^* - \mathbf{x}_i\|^2 = (\mathbf{x}^* - \mathbf{x}_i)^T(\mathbf{x}^* - \mathbf{x}_i) \\ \text{subject to} \quad & F(\mathbf{x}^*) = 0. \end{aligned} \quad (5)$$

In implementation, we compute $\mathbf{x}_i^S$ by an iterative scheme. Let $\mathbf{x}_i^{(0)} := \mathbf{x}_i$. Then $\mathbf{x}_i^{(j+1)}$ is updated from $\mathbf{x}_i^{(j)}$ by repeatedly updating $\mathbf{x}^{(i)}$ according to

$$F(\mathbf{x}_i^{(j)}) + \frac{\partial F}{\partial \mathbf{x}} \mid_{\mathbf{x}=\mathbf{x}_i^{(j)}} \cdot(\mathbf{x}_i^{(j+1)} - \mathbf{x}_i^{(j)}) = 0$$

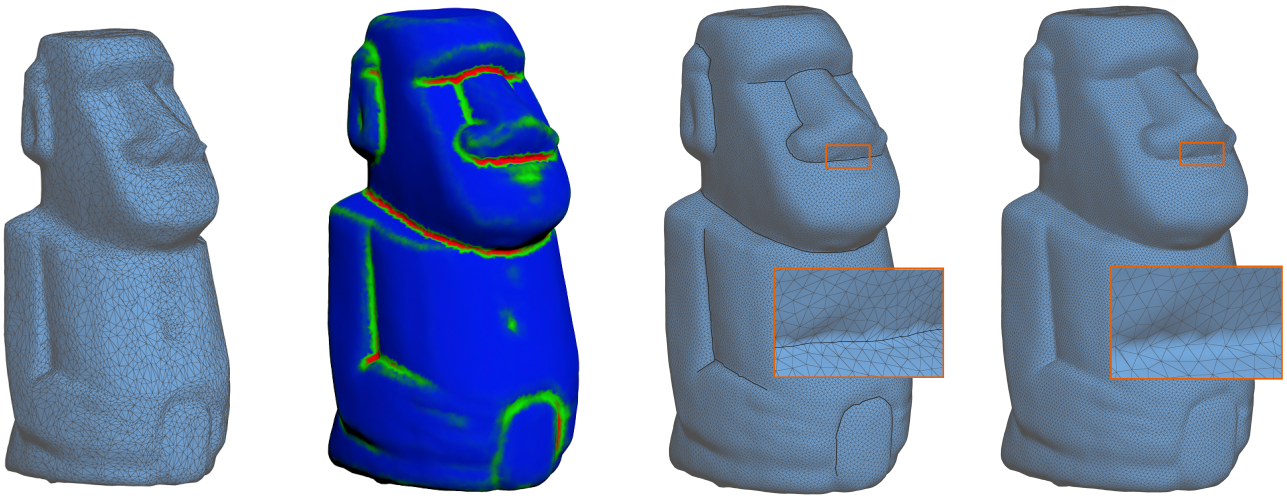
and $\mathbf{x}_i^{(j+1)} - \mathbf{x}_i^{(j)}$ is parallel to $\frac{\partial F}{\partial \mathbf{x}} \mid_{\mathbf{x}=\mathbf{x}_i^{(j)}}$. The iterative algorithm terminates until $\|\mathbf{x}_i^{(j+1)} - \mathbf{x}_i^{(j)}\| < \epsilon$.

Polygonal mesh. When the input is a polygonal mesh, the nearest point $\mathbf{x}_i^S$ can be quickly found by bounding box tree techniques, e.g., directly calling the proximity query package (PQP) (Larsen et al., 1999).

Point clouds. Based on the Moving Least Square (MLS) technique (Lim et al., 2007), we can define a point-set surface approximated locally for a certain neighborhood by a polynomial, and then project the test point $\mathbf{x}_i$ near the point set onto this surface, obtaining the projection point $\mathbf{x}_i^S$.

3.3. *Feature Alignment*

Feature alignment is to require edges follow feature lines, which is very helpful to many computer graphics occasions especially mesh quadrangulation. Generally speaking, the quality of feature alignment depends on the accuracy of feature detection (Hildebrandt et al., 2005; Weinkauf and Günther, 2009),



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

Fig. 3. In order to compute the offset surface of (a), we predict the feature regions (b) of the potential offset surface using the moveable sites. The feature regions are able to trap the nearby sites into feature lines and finally lead to a feature-aligned triangulation (c), which is significantly different from the uniform triangulation (d).

which is also a difficult problem. For example, Kalogerakis et al. (Kalogerakis et al., 2007) proposed robust estimation of smoothed curvature directions that encode feature lines.

In this paper, we predict the feature regions of the potential offset surface using the moveable sites and further use the feature regions to guide the moves of the sites. We measure to what degree the site $\mathbf{x}_i$ is on feature line using the following formula:

$$\tau_{\mathbf{x}_i} = \frac{\int_{\mathbf{x}\in\Omega(\mathbf{x}_i)} \|\mathbf{n}_\mathbf{x} - \overline{\mathbf{n}}\|^2 \mathrm{d}\mathbf{x}}{\int_{\Omega(\mathbf{x}_i)} \mathrm{d}\mathbf{x}}, \tag{6}$$

where $\mathbf{n}_\mathbf{x}$ is the normal vector at the point $\mathbf{x}$ in $\mathbf{x}_i$'s neighborhood $\Omega(\mathbf{x}_i)$, while $\overline{\mathbf{n}}$ is the average normal vector of $\Omega(\mathbf{x}_i)$. In the discrete setting, $\tau_{\mathbf{x}_i}$ can be also written in the following form:

$$\tau_{\mathbf{x}_i} = \frac{\sum_{\mathbf{x}_j\in\Omega(\mathbf{x}_i)} \|\mathbf{n}_{\mathbf{x}_j} - \overline{\mathbf{n}}\|^2}{K}, \tag{7}$$

where K is the number of moveable sites in $\mathbf{x}_i$'s neighborhood $\Omega(\mathbf{x}_i)$. Intuitively, $\tau_{\mathbf{x}_i}$ represents the disorder of the normal vectors around $\mathbf{x}_i$. It is able to well capture the feature regions of the potential offset surface (see Figure 3(b)), in spite of the dynamic sites, throughout the optimization process. Then we enforce $\{\tau_{\mathbf{x}_i}\}$ on the particle system as follows.

$$E(\mathbf{X}) = \sum_i^n \sum_j^n e^{-(\tau-\tau_{\mathbf{x}_i}-\tau_{\mathbf{x}_j})\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{4\sigma^2}}, \tag{8}$$

where τ is set to 1.3 in our experiments. Compared to the uniform triangulation in Figure 3(d) that is computed from Eq. (2), we find that Eq. (8) is able to trap the nearby sites into feature lines, leading to a feature aligned triangulation shown in Figure 3(c). Note that our technique to achieve feature alignment (Chen et al., 2012; Fuhrmann et al., 2010) is quite different from existing approaches on this side that usually require a feature-line detection step.

4. Super-linear Convergent Algorithm

In this section, we summarize the particle-based method and detail the components of the algorithm including the computation of the objective function and the algorithmic pseudo-code.

4.1. Objective function

The objective function is shown in Eq. (8), which is independent of the specific representation form of the input surface and has infinite-order smoothness. In order to achieve a better convergence rate, we use the L-BFGS solver to optimize the sites $\mathbf{X}$. The gradients of the objective function $\frac{\partial E}{\partial \mathbf{x}_i}$ are:

$$\sum_{j\neq i} -\frac{(\tau-\tau_{\mathbf{x}_i}-\tau_{\mathbf{x}_j})(\mathbf{x}_i-\mathbf{x}_j)}{2\sigma^2} e^{-(\tau-\tau_{\mathbf{x}_i}-\tau_{\mathbf{x}_j})\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{4\sigma^2}}, \quad i = 1, 2, \cdots, N, \tag{9}$$

where N is the user-specified number of sites.

Initialization. We need to initialize a user-specified number of sites for further optimization. It is natural that a set of uniformly distributed sites are helpful for reducing the number of iterations. In our implementation, a user-specified number of initial seeds $\mathbf{X}$ are generated on S with a uniform random distribution according to original surface areas.

Choice of σ. Basically, σ is related to the influence range of each particle. If the distance between two particles is larger than $5\sqrt{2}\sigma$, then the force between them is negligible. In this work, we set $\sigma = c\sqrt{|S|/N}$, where $|S|$ denotes the total area of the entire surface, and c is an empirical coefficient and typically set to 0.25.

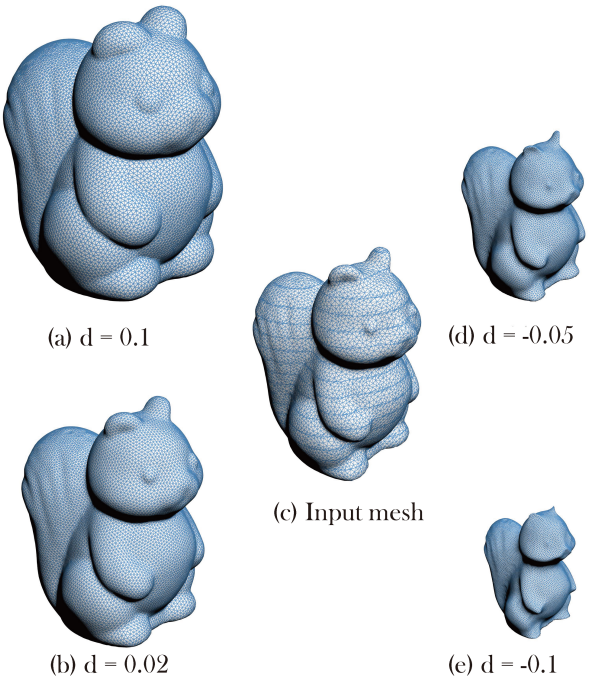


Fig. 4. The offset surfaces at various distances. When N=10K, it requires about 2.5 seconds for generating the offset surface.

Computation of the objective function. Obviously, computing the objective function E, as well as its gradients, requires $O(N^2)$ time. To reduce the computational cost, we need to ignore those terms that almost do not contribute to E. In our implementation, we consider only the particle pairs that have a distance less than 5σ. To quickly filter out those redundant particle pairs, we use the Approximate Nearest Neighbor (ANN) library (Mount and Arya, 1998) to achieve this purpose.

Hard constraint of the offset distance. During each iteration of optimization, we need to adjust the sites such that they are constrained on the offset surface. Let $\mathbf{x}_i$ be one site and $\mathbf{x}_i^S$ be its projection on the primitive surface S. Then $\mathbf{x}_i$ should be updated to $\mathbf{x}_i'$ as follows.

$$\mathbf{x}_i' = \mathbf{x}_i^S + d \times \frac{\mathbf{x}_i - \mathbf{x}_i^S}{\|\mathbf{x}_i - \mathbf{x}_i^S\|}. \tag{10}$$

Termination condition. The termination condition is set to

$$\max_i |\frac{\partial E}{\partial \mathbf{x}_i} \cdot \frac{\mathbf{x}_i - \mathbf{x}_i^S}{d}| < 10^{-6}. \tag{11}$$

That is to say, even if we perturb these sites along the tangent plane, the objective function will not decrease any more.

Site connection. When the optimization converges, the sites are uniformly distributed on the offset surface, and thus it is very easy to build connection between them. In our implementation, the final mesh is extracted as the restricted Delaunay triangulation (RDT) (Yan et al., 2009) restricted on the surface.

Algorithm 1 shows the pseudo-code. Fig. 4 shows an example of offsetting the Squirrel model with various distance settings. When N = 10K, it requires about 2.5 seconds for generating the offset surface.

Algorithm 1: Generating high-quality polygonal offset surface

Input: A surface S, a user-specified offset distance d, and N initial sites.
Output: A polygonal offset surface.
while $\max_i |\frac{\partial E}{\partial \mathbf{x}_i} \cdot \frac{\mathbf{x}_i - \mathbf{x}_i^S}{d}| \geq 10^{-6}$ **do**
  Update the ANN data structure for the current site collection $\mathbf{X}$;
  for *each site* $\mathbf{x}_i$ **do**
    Get neighboring sites from ANN;
    for *each neighboring site* $\mathbf{x}_j$ **do**
      Compute E^{ij} using $E^{ij} = e^{-(\tau - \tau_{\mathbf{x}_i} - \tau_{\mathbf{x}_j})\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{4\sigma^2}}$;
      Compute F^{ij} using $F^{ij} = -\frac{(\tau - \tau_{\mathbf{x}_i} - \tau_{\mathbf{x}_j})(\mathbf{x}_i - \mathbf{x}_j)}{2\sigma^2} e^{-(\tau - \tau_{\mathbf{x}_i} - \tau_{\mathbf{x}_j})\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{4\sigma^2}}$;
    end
    Sum F^{ij} with regard to j to get F^i;
  end
  Sum E^{ij} to get the total energy E;
  Feed the scalar E and the vector F into the L-BFGS solver to get updated locations of $\mathbf{X}$;
  Adjust $\mathbf{X}$ such that the sites are located on the offset surface; See Eq. (10);
end
Connect the sites to output the polygonal offset surface.

5. Experimental Results

We implemented our algorithm in Microsoft Visual C++ 2013. All the experiments were conducted on a computer with Intel(R) Core(TM) i7-6700QM CPU 2.60GHz and 4GB memory. All the models are scaled into a bounding box with a unit-length diagonal. In the following, we will evaluate our algorithm in meshing quality, performance, topological correctness and accuracy.

5.1. *Scalability*

Figure 5 shows four examples where the inputs are respectively a polygonal mesh (Figure 5(a)), an implicit function ((Figure 5(c))

$$(x^2 + 9/4 \times y^2 + z^2 - 1)^3 - x^2 \times z^3 - 9/80 \times y^2 \times z^3 = 0,$$

a parametric form (Figure 5(e)):

$$x = (1 - \cos u) \times \sin u \times \cos v,$$

$$y = (1 - \cos u) \times \sin u \times \sin v,$$

$$z = \cos u. \qquad u, v \in [0, 2\pi]$$

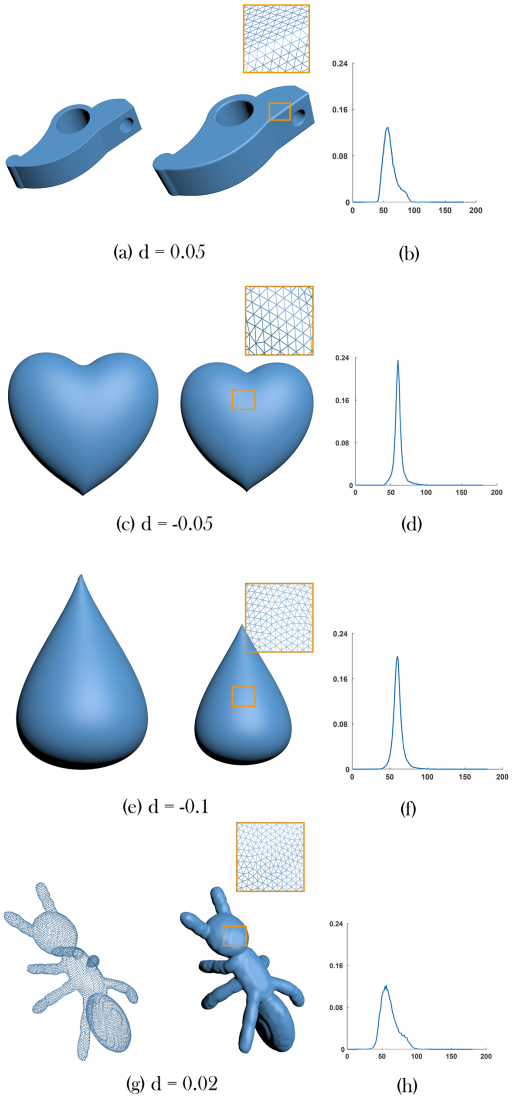


Fig. 5. Our algorithm supports multiple kinds of inputs: polygonal surfaces (a), implicit functions (c), parametric surfaces (e) and point clouds (g). The middle column shows the offset surfaces, while the right column shows the angle histograms.

and a point cloud (Figure 5(g)). The four examples show that our algorithm supports various kinds of input models. The target number of vertices is set to 10K. It requires about 2.5 seconds to compute each offset surface. The general adaptability distinguishes itself from the existing approaches for offsetting surfaces.

5.2. *Meshing quality*

We use the angle regularity to measure the meshing quality. Let θ_{min} be the smallest angle, and θ_{avg} be the average angle. In Figure 5, we give the statistics plot of angles in (b,d,f,h). It can be seen that most of the angles are very close to 60^o, which shows our resulting meshes have an overall desirable meshing quality.

In order to compare our algorithm with the signed distance field based method (Liu and Wang, 2011) and the famous software Rhino, we compute the offset surface for the Teddy model shown in Figure 6. Table 1 gives the statistics of Lo values (Lo, 1985): $G(\triangle ABC) = \frac{S_{\triangle ABC}}{|AB|^2+|BC|^2+|CA|^2}$. Let G_{min} be the lowest quality value, and G_{avg} be the average value. Table 1 shows that our algorithm has a significant advantage of meshing quality. Note that G_{avg} is very close to $\sqrt{3}/12$, which implies that our algorithm is able to generate a high-quality mesh with most of the triangles being approximately equilateral. Furthermore, our algorithm exhibits the merit of feature alignment, which is the second advantage.

Table 1. Comparison of meshing quality on the Teddy surface (Fig. 6).

Method	G_{avg}	G_{min}	θ_{avg}	θ_{min}
Ours	0.144337	0.041400	56.21	23.02
Liu & Wang, 2011	0.105587	0.010841	47.03	1.4699
Rhino	0.089958	0.009124	39.87	0.9021

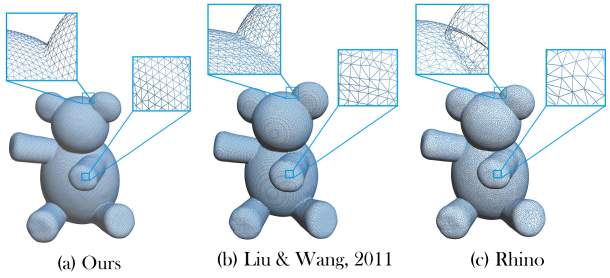


Fig. 6. Our algorithm has a significant advantage of yielding a high-quality mesh. Here the target number of vertices is 17K and the offset distance is set to 0.05.

For most of the existing approaches, they have to deal with the self-intersection issue when the offset distance is relatively large; See the close-up views in Figure 6(c). Although a number of techniques (Liu and Wang, 2011) have been proposed for obtaining intersection-free offset surfaces, it is not easy to accomplish this step in a robust and efficient manner. However, our algorithmic framework has no need to deal with the self-intersection issue since every moveable site is required to keep the offset distance away from the base surface, as shown in Figure 6(a). This is another advantage of our algorithm.

In addition, it seems to work well by performing a series of mesh boolean operations (Jacobson et al., 2016) across a sufficiently large number of spheres centered at the original surface. We use the Bird model in Figure 7(a) for test. If we use 1K spheres to generate the offset surface, it requires about 2K seconds to get a poorly triangulated result; see Figure 7(c). If we

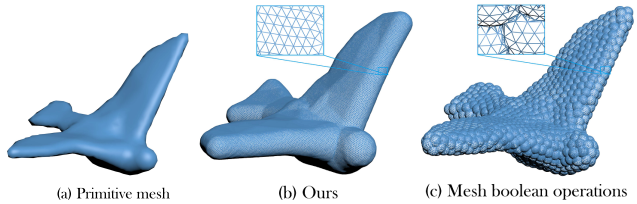


Fig. 7. Comparison of meshing quality between ours (b) and the boolean operation based method (c). The latter cannot work for offsetting in practice.

Table 2. Time statistics with various inputs.

Model	Figure	#Tri	Offset	$T_d{}^*$(s)	$T_c{}^\#$(s)	T_{tol}(s)
Heart	5(c)	10k	−0.05	0.851	0.8898	1.7408
		20k		1.983	1.7479	3.7309
		30k		3.088	2.5609	5.6489
Squirrel	4(e)	20k	−0.1	2.184	1.8278	4.0118
		30k		3.657	2.6977	6.3547
		40k		5.037	3.658	8.695
Dog	1	40k	0.03	5.670	3.9529	9.6229
		80k		13.988	5.3058	19.2938
		100k		25.173	10.3076	35.4806
Moai	3	60k	0.02	14.501	4.8149	19.3159
		100k		27.016	9.8513	36.8673
		200k		86.99	20.0724	107.0624

$^*T_d$–optimization time $^\#T_c$–polygonal mesh generation time

use more spheres, a numerical issue occurs due to the fact that the meshing quality becomes worse and worse. By contrast, our method requires only 30 seconds to generate a high-quality triangulated offset surface with 100K faces; see Figure 7(b).

5.3. *Efficiency*

Recall that we use the L-BFGS solver to optimize the sites, which exhibits a super-linear convergence in our experiments. Furthermore, we use the Approximate Nearest Neighbor (ANN) library to filter out those pairs that contribute little to the objective function. In detail, we omit the site pairs whose distances exceed 5σ. The two techniques are central to guarantee the high performance. In this subsection, we respectively show the overall performance and the convergence rate.

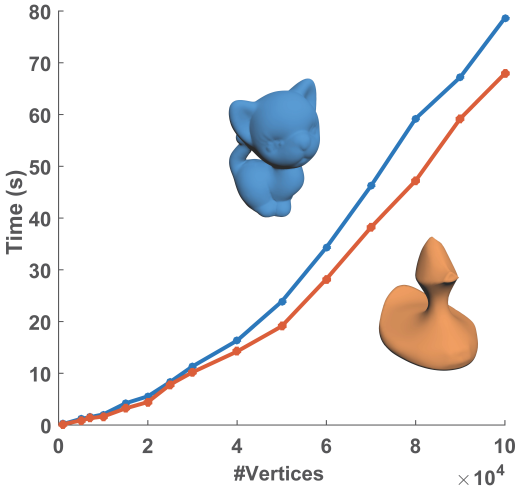
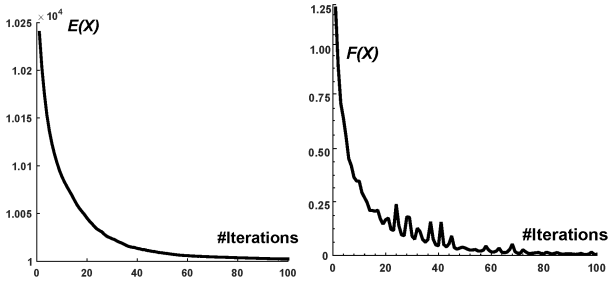


Fig. 8. Performance plots.

In order to test the overall performance, we set the target number of the offset surfaces of the Kitty and Duck model to vary from 1K to 100K and show the performance plots in Figure 8. In Table 2, we respectively give the timing costs spent in the optimization of particle system T_d and those in generating the final mesh T_c, which shows the high performance of our algorithm. For example, generating the offset surface of 80K triangles for the Dog model requires about 19 seconds while the voxelization-based method (Pavi and Kobbelt, 2008), generating a polygonal offset with poor triangulation, needs about 25 seconds.

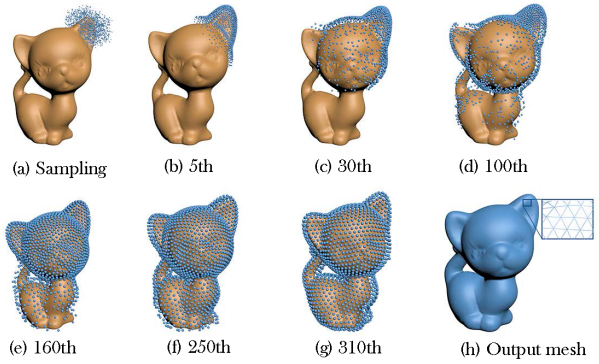


(a) Energy change (b) Gradient change

Fig. 9. The energy and gradient change plots of the Duck model during the optimization process. The total number of iterations is 102.

In order to observe the convergence rate of our algorithm, we use Figure 9 to show the energy decreasing plot and the gradient decreasing plot for the Duck model (the target offset surface has 10K vertices). From the plot, we can clearly see that the objective function, as well its gradient norm, decreases very sharply, which implies that our algorithm has a super-linear convergence rate.

5.4. *Robust to bad initialization*



(a) Sampling (b) 5th (c) 30th (d) 100th
(e) 160th (f) 250th (g) 310th (h) Output mesh

Fig. 10. Our algorithm is robust to bad initialization.

In order to test if our algorithm is robust to initialization, we make 5K sites gathering around the right ear of the Kitten model, as shown in Figure 10(a). From Figure 10(b-g), we can see that the sites gradually spread over the surface until they become uniform, which is due to the repulsion between sites. This shows that our algorithm is able to get a high-quality offset mesh even if the initial sites are not well distributed. For this example, our algorithm requires 320 iterations and about 20 seconds to compute the offset surface. It's worth noted that an as-uniform-as-possible initial site distribution is helpful to reduce the number of iterations.

5.5. *Topological correctness*

It's easy to know that the topology of an offset surface may be different from that of the base mesh, especially when we

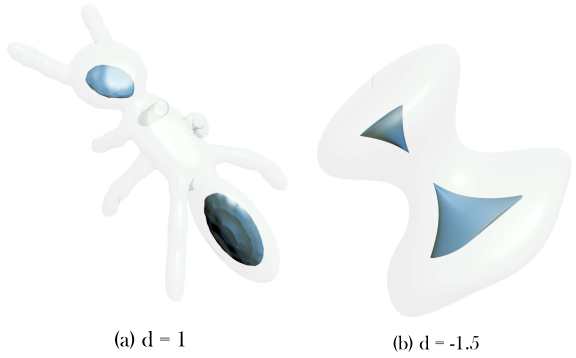


Fig. 11. Our algorithm has the ability to deal with topology changes, where *d* is set to -1.0 in (a) while -1.5 in (b).

Table 3. Error analysis.

Model	Figure	#Tri	Offset()	E_{avg}	E_{max}
Heart	5(c)	20K	-0.02	3.626×10^{-6}	0.02007
			0.02	2.151×10^{-6}	0.02100
			0.05	2.118×10^{-6}	0.05006
Squirrel	4	40K	-0.1	4.456×10^{-6}	0.010001
			-0.05	3.212×10^{-6}	0.09000
			0.1	1.110×10^{-6}	0.010006
Bear	12	60K	-0.02	6.403×10^{-7}	0.02000
			0.02	3.501×10^{-7}	0.02160
			0.05	4.342×10^{-7}	0.05004
Dog	1	100K	-0.01	9.491×10^{-7}	0.0591
			0.05	2.453×10^{-5}	0.04120
			0.1	1.24×10^{-5}	0.07005

compute an inward offset surface. Most of existing approaches have to perform an extra step of removal of redundant parts, which is tedious and highly non-trivial. Our algorithm, however, can naturally guarantee the topological correctness. The key lies in that we keep those sites at a constant distance d from the base surface during the optimization process. So it is impossible to have sites on the redundant parts since the corresponding distance in between is not d.

In Figure 11, we show two examples of inward offset surfaces. It can be seen that the resulting offset surfaces have different topological structures from the base surfaces and consist of multiple connected components, which demonstrates that our algorithm has the ability to deal with topology changes.

5.6. *Error analysis*

Generally speaking, our resulting polygonal mesh is not absolutely accurate. We perform error analysis in this way: for an arbitrary point p on the offset surface, we find its projection point q on the primitive surface, and then keep down $E(p) = |\|p - q\| - d|/d$. We use average and maximum errors (Wang and Manocha, 2013) to measure the accuracy:

$$E_{\text{avg}} = \frac{1}{n}\sum_{i}^{n} E(p_i), \qquad E_{\text{max}} = \max_{1\le i\le n}\{E(p_i)\}. \tag{12}$$

Figure 12 visualizes the errors of a family of the Teddy's offsets. Table 3 shows the detailed error statistics. From the statistics we

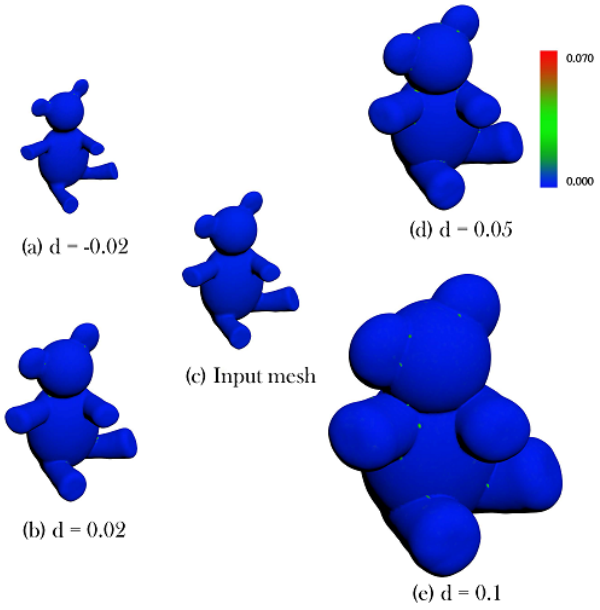


Fig. 12. Error analysis. (c) is the input model. From (a), (b), (d) and (e), we visualize the errors across various offset distances in a color-coded style.

can clearly see that the results are very accurate.

We use Figure 13(a) to compare our algorithm with the existing voxelization-based method. It can be seen that our result is much more accurate if measured at the same level of computational cost. Generally speaking, it is hard to set the voxelization resolution for the voxelization-based method. If the voxelization resolution is too low, the resulting offset surface will have a topological error. But if the resolution is too high, it will take much more time since the computational time is at least cubic to the accuracy parameter.

6. Limitations

In spite of the significant advantages in terms of meshing quality, computational performance, topological correctness and feature alignment, our algorithm, in its current form, still has a couple of limitations including:

- **Accuracy.** In Section 5.6, we show that the largest errors often occur in the concave regions of the offset surface (assuming offsetting outward). The reason behind lies in the fact that the real offset surface is generally non-smooth around these areas.
- **Performance.** Currently our algorithm cannot compute offset surfaces in real time, which limits its use in some interactive applications.

7. Conclusions and Future Works

In this paper, we propose a general and fast algorithm to generate a feature-aligned and high-quality triangle mesh of an offset surface based on particle system. The algorithm takes

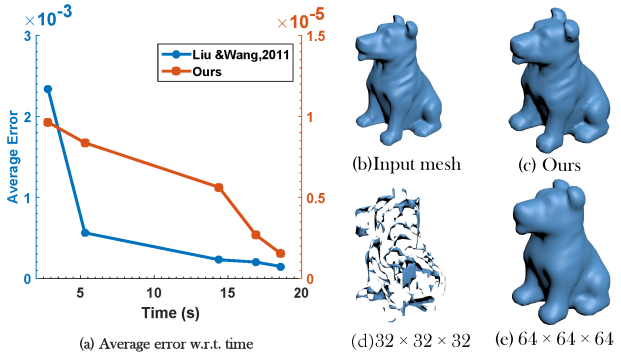


Fig. 13. Accuracy comparison. (a) shows the plots of the average error with regard to the computational cost on the Dog model (b). (c) is our result. For the voxelization-based method, if the voxelization resolution is too low (32×32×32), the resulting offset surface is broken (d). When the the voxelization resolution amounts to 64×64×64, it will takes much longer time. If measured at the same level of time cost, our algorithm is much more accurate than the voxelization-based method.

L-BFGS as the solver and thus has a super-linear convergence rate. Our algorithm supports multiple kinds of inputs, e.g., triangle meshes, implicit functions, parametric surfaces or even point clouds. In the future, we shall give a GPU-based speedup implementation.

Acknowledgments

We are grateful to the editors and anonymous reviewers for their insightful comments and suggestions. The work described in this paper was partially supported by grants from K.C. Wong Magna Fund in Ningbo University, NSF of Zhejiang(LY17F020018), NSF of Ningbo(2017A610115), the Open Project Program of the State Key Lab of CAD&CG (A1702), Zhejiang University, and the Research Grants Council of the Hong Kong Special Administrative Region, China (CityU11237116 and CityU11300615).

References

Maekawa, T. An overview of offset curves and surfaces. Computer-Aided Design 1999;31(31):165–173.

Kim, SJ, Lee, DY, Yang, MY. Offset triangular mesh using the multiple normal vectors of a vertex. Computer-Aided Design and Applications 2004;1(1-4):285–291.

Pham, B. Offset curves and surfaces: a brief survey. Computer-Aided Design 1992;24(4):223–229.

Kim, SJ, Yang, MY. Triangular mesh offset for generalized cutter. Computer-Aided Design 2005;37(10):999–1014.

Farouki, RT. Exact offset procedures for simple solids. Computer Aided Geometric Design 1985;2(4):257–279.

Farouki, RT. The approximation of non-degenerate offset surfaces. Computer Aided Geometric Design 1986;3(1):15–43.

Martin, RR, Stephenson, P. Sweeping of three-dimensional objects. Computer-Aided Design 1990;22(4):223–234.

Pavi, D, Kobbelt, L. High-Resolution Volumetric Computation of Offset Surfaces with Feature Preservation. Computer Graphics Forum 2008;27(2):165–174.

Chen, Y, Wang, CC. Uniform offsetting of polygonal model based on layered depth-normal images. Computer-aided design 2011;43(1):31–46.

Zhou, Q, Grinspun, E, Zorin, D, Jacobson, A. Mesh arrangements for solid geometry. ACM Transactions on Graphics 2016;35(4):1–15.

Filip, D, Magedson, R, Markot, R. Surface algorithms using bounds on derivatives. Computer Aided Geometric Design 1986;3(4):295–311.

Piegl, LA, Tiller, W. Computing offsets of NURBS curves and surfaces. Computer-Aided Design 1999;31(2):147–156.

Kumar, GVVR, Shastry, KG, Prakash, BG. Computing non-self-intersecting offsets of NURBS surfaces. Computer-Aided Design 2002;34(3):209–228.

Liu, S, Wang, CC. Fast intersection-free offset surface generation from freeform models with triangular meshes. IEEE Transactions on Automation Science and Engineering 2011;8(2):347–360.

Varadhan, G, Manocha, D. Accurate Minkowski sum approximation of polyhedral models. Graphical Models 2004;68(4):343–355.

Lien, JM. Covering Minkowski sum boundary using points with applications. Computer Aided Geometric Design 2008;25(8):652–666.

Du, Q, Faber, V, Gunzburger, M. Centroidal Voronoi tessellations: Applications and algorithms. SIAM review 1999;41(4):637–676.

Okabe, A, Boots, B, Sugihara, K, Chiu, SN. Spatial tessellations: concepts and applications of Voronoi diagrams; vol. 501. John Wiley & Sons; 2009.

Turk, G. Re-tiling polygonal surfaces. ACM SIGGRAPH Computer Graphics 1992;26(2):55–64.

Witkin, AP, Heckbert, PS. Using particles to sample and control implicit surfaces. In: Proceedings of the 21st annual conference on Computer graphics and interactive techniques. ACM; 1994, p. 269–277.

Meyer, M, Kirby, RM, Whitaker, R. Topology, accuracy, and quality of isosurface meshes using dynamic particles. IEEE Transactions on Visualization and Computer Graphics 2007;13(6):1704–1711.

Bronson, JR, Levine, JA, Whitaker, RT. Particle systems for adaptive, isotropic meshing of CAD models. Engineering with Computers 2012;28(4):279–296.

Du, Q, Wang, D. Tetrahedral mesh generation and optimization based on centroidal Voronoi tessellations. International journal for numerical methods in engineering 2003;56(9):1355–1373.

Liu, Y, Wang, W, Lévy, B, Sun, F, Yan, DM, Lu, L, et al. On centroidal Voronoi tessellationłenergy smoothness and fast computation. ACM Transactions on Graphics (ToG) 2009;28(4):101.

Zhong, Z, Shuai, L, Jin, M, Guo, X. Anisotropic surface meshing with conformal embedding. Graphical Models 2014;76(5):468–483.

Chen, Z, Cao, J, Wang, W. Isotropic surface remeshing using constrained centroidal Delaunay mesh. Computer Graphics Forum 2012;31(7):2077–2085.

Zhong, Z, Guo, X, Wang, W, Levy, B, Sun, F, Liu, Y, et al. Particle-based anisotropic surface meshing. ACM Transactions on Graphics 2013;32(4):99–99.

Lai, YK, Kobbelt, L, Hu, SM. Feature aligned quad dominant remeshing using iterative local updates. Computer-Aided Design 2010;42(2):109–117.

Larsen, E, Gottschalk, S, Lin, MC, Manocha, D. Fast proximity queries with swept sphere volumes. Tech. Rep.; Technical Report TR99-018, Department of Computer Science, University of North Carolina; 1999.

Lim, JH, Im, S, Cho, YS. MLS (moving least square)-based finite elements for three-dimensional nonmatching meshes and adaptive mesh refinement. Computer methods in applied mechanics and engineering 2007;196(17):2216–2228.

Hildebrandt, K, Polthier, K, Wardetzky, M. Smooth feature lines on surface meshes. In: Eurographics Symposium on Geometry Processing. 2005, p. 85.

Weinkauf, T, Günther, D. Separatrix persistence: Extraction of salient edges on surfaces using topological methods. In: Computer Graphics Forum; vol. 28. Wiley Online Library; 2009, p. 1519–1528.

Kalogerakis, E, Simari, P, Nowrouzezahrai, D, Singh, K. Robust statistical estimation of curvature on discretized surfaces. In: Eurographics Symposium on Geometry Processing, Barcelona, Spain, July. 2007, p. 13–22.

Fuhrmann, S, Ackermann, J, Kalbe, T, Goesele, M. Direct Resampling for Isotropic Surface Remeshing. In: VMV. Citeseer; 2010, p. 9–16.

Mount, DM, Arya, S. ANN: A library for approximate nearest neighbor searching. The 14th annual ACM-SIAM Symposium on Discrete Algorithms 1998;.

Yan, DM, Lévy, B, Liu, Y, Sun, F, Wang, W. Isotropic remeshing with fast and exact computation of restricted Voronoi diagram. In: Computer graphics forum; vol. 28. Wiley Online Library; 2009, p. 1445–1454.

Lo, SH. A new mesh generation scheme for arbitrary planar domains. International Journal for Numerical Methods in Engineering 1985;21(8):1403–1426.

Jacobson, A, Panozzo, D, Schüller, C, Diamanti, O, Zhou, Q, Pietroni, N, et al. Libigl: A simple c++ geometry processing library. 2016.

Wang, CCL, Manocha, D. GPU-based offset surface computation using point samples. Computer-Aided Design 2013;45(2):321–330.