Delaunay Mesh Simplification with Differential Evolution

RAN YI, Tsinghua University, China
YONG-JIN LIU*, Tsinghua University, China
YING HE, Nanyang Technological University, Singapore

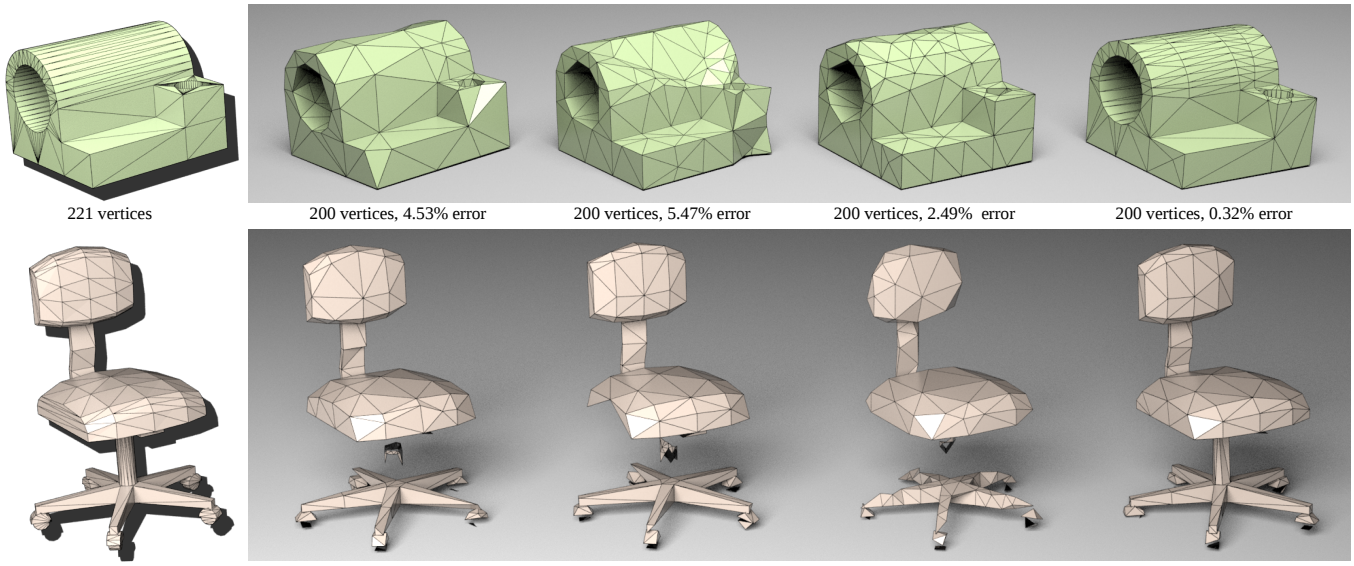


Fig. 1. Given an arbitrary manifold triangle mesh M (not necessarily a Delaunay mesh), our method automatically computes the simplified Delaunay mesh with the user-specified resolution that has a small Hausdorff distance to M. Error (%) is measured by the two-sided Hausdorff distance between M and the simplified DMs, with respect to the diagonal length of model's bounding box. It preserves sharp features well and can deal with models with multiple connected components, whereas the existing methods often fail.

Delaunay meshes (DM) are a special type of manifold triangle meshes — where the local Delaunay condition holds everywhere — and find important applications in digital geometry processing. This paper addresses the general DM simplification problem: given an arbitrary manifold triangle mesh M with n vertices and the user-specified resolution m ($< n$), compute a Delaunay mesh M^* with m vertices that has the least Hausdorff distance to M. To solve the problem, we abstract the simplification process using a 2D Cartesian grid model, in which each grid point corresponds to triangle meshes with a certain number of vertices and a simplification process is a monotonic path on the grid. We develop a novel differential-evolution-based method to compute a low-cost path, which leads to a high quality Delaunay mesh. Extensive evaluation shows that our method consistently outperforms the existing methods in terms of approximation error. In particular, our method is highly effective for small-scale CAD models and man-made objects with sharp features but less details. Moreover, our method is fully automatic and can preserve sharp features well and deal with models with multiple components, whereas the existing methods often fail.

CCS Concepts: • **Computing methodologies → Mesh geometry models**;

Additional Key Words and Phrases: Delaunay mesh, mesh simplification, 2D Cartesian grid model, differential evolution

ACM Reference Format:
Ran Yi, Yong-Jin Liu, and Ying He. 2018. Delaunay Mesh Simplification with Differential Evolution. *ACM Trans. Graph.* 37, 6, Article 263 (November 2018), 13 pages. https://doi.org/10.1145/3272127.3275068

*Corresponding author: liuyongjin@tsinghua.edu.cn (Yong-Jin Liu)

Authors' addresses: Ran Yi, Tsinghua University, BNRist, Department of Computer Science and Technology, Beijing, 100084, China; Yong-Jin Liu, Tsinghua University, China; Ying He, Nanyang Technological University, 50 Nanyang Ave, 639798, Singapore.

0730-0301/2018/11-ART263 $15.00
https://doi.org/10.1145/3272127.3275068

1 INTRODUCTION

Delaunay meshes (DM) [Dyer et al. 2007a] are a special type of manifold triangle meshes where the local Delaunay condition is satisfied: for each internal edge, the sum of the opposite angles in adjacent triangles is less than π. DM has proven useful for many geometry modeling tasks, such as discrete geodesics, manifold harmonics,

and parameterization [Liu et al. 2015]. Unfortunately, most models (scanned or manually created) are not Delaunay meshes. There are two methods [Dyer et al. 2007a; Liu et al. 2015] that are able to convert arbitrary manifold triangle meshes to Delaunay meshes. They are theoretically sound and can guarantee the resulting DM has exactly the same geometry of the input mesh. However, since both methods add a large number of vertices and edges, the price to pay is the significantly higher space complexity, which in turn increases the computational cost for the follow-up applications.

In fact, finding a good tradeoff between mesh resolution and approximation error is a common practice in many graphics applications (e.g., [Hu et al. 2017; Mandad et al. 2015]). In this paper, we focus on the *optimal* DM simplification problem: *given a manifold triangle mesh M (not necessary a Delaunay mesh) with n vertices and a user-specified number $m \leq n$, compute a Delaunay mesh M^* with m vertices that has the least Hausdorff distance to M.*

Observing that any mesh optimization (in our case, including Delaunay mesh construction and simplification) can be realized as a sequence of edge operations, namely, split and collapse [Hoppe et al. 1993], we abstract the optimal DM simplification process using a 2D Cartesian grid model: each grid point corresponds to triangle meshes with a certain number of vertices and each type of edge operation corresponds to one principal direction. The input mesh is the origin and all simplified DMs with m vertices lie on the line $x - y = n - m$. Two adjacent points are two triangle meshes whose vertex counts differ by 1 and their topological realizations can be transformed into each other by an edge operation. Using this model, we can view a DM simplification process as a monotonic path in a trapezoidal subset of the grid. It is worth noting that the existing methods [Dyer et al. 2007a; Liu et al. 2015] compute two different *fixed* paths due to their pre-defined strategies. These methods cannot produce optimal DM due to the fixed, geometry-independent strategies.

Although there are finite monotonic paths in the trapezium, the conventional graph traversal or path finding algorithms (e.g., breadth-first-search, Dijkstra's algorithm and A* search, etc.) would not work. This is because there are $O(n^2)$ nodes in the graph and when visiting a node v, one needs to compute the Hausdorff distance between the original mesh and the simplified mesh corresponding to v. The whole process is too computationally expensive even for small-scale meshes.

In this paper, we develop a simple yet highly effective method to compute a low-cost path. We first encode a path in the solution space (i.e., a sequence of edge split and collapse operations) by a vector of real values, then formulate the objective function that measures the Hausdorff distance between the input mesh and the simplified DM obtained from the vector. Next, we apply differential evolution (DE) [Das and Suganthan 2011] for finding a solution with a small objective function value.

Our method is fully automatic and can preserve sharp features well. Computational results on a wide range of 3D meshes show that our method consistently outperforms the existing methods [Dyer et al. 2007a; Liu et al. 2015] in terms of approximation error. In particular, our method is highly effective for small-scale[1] CAD models and man-made objects that have sharp features but less details. Moreover, thanks to the two-sided Hausdorff distance metric used in our method, it can deal with models with multiple connected components, whereas the other methods often fail.

We make the following contributions in this paper:

- A novel computational framework that formulates the problem as optimal path-finding in a 2D Cartesian grid and explains that the existing methods [Dyer et al. 2007a; Liu et al. 2015], based on pre-defined strategies, compute two fixed paths; and
- A novel stochastic algorithm to compute a low-cost path in the entire solution space. We observe that our method consistently outperforms the existing methods.

2 RELATED WORK

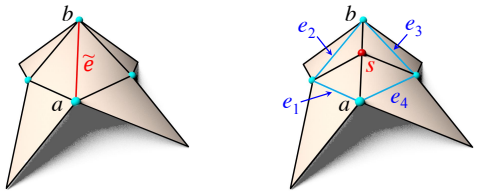
2.1 Intrinsic Delaunay Structures

Rivin [1994] defined *intrinsic Delaunay triangulation* (IDT) of a simplicial surface, and the triangular edges of IDT are geodesic paths. Bobenko and Sprinborn [2007] proved that the edge flipping algorithm is guaranteed to terminate after a finite number of steps, thereby implying the existence of IDT. They also defined Delaunay tessellation via a global empty circle criterion and proved its existence and uniqueness. IDT can then be obtained by triangulating all non-triangular faces in Delaunay tessellation. Boissonnat et al. [2013] further extended the IDT construction to smooth closed submanifolds of Euclidean spaces. On piecewise linear surfaces, Fisher et al. [2007] proposed a practical implementation of the edge flipping algorithm in [Bobenko and Springborn 2007], which is efficient for real-world meshes. However, it has no known time complexity and the resulting IDTs may not be proper, i.e., containing self-loops and/or faces with only two edges.

Dyer et al. [2008] proposed adaptive sampling criteria based on the strong convexity radius and the injectivity radius, and show that in smooth manifolds, if an intrinsic Voronoi diagram satisfies the closed ball property [Edelsbrunner and Shah 1997], its dual IDT exists and is proper. However, their Voronoi cells are restricted in a convex neighborhood, which is an extremely small region around a point in smooth manifold. Based on the intrinsic property of discrete geodesics [Mitchell et al. 1987], Liu et al. [2017] developed a practical algorithm that constructs intrinsic Voronoi diagrams on manifold triangle meshes satisfying the closed ball property with very few auxiliary samples and output proper dual IDTs. Their algorithm has a worst-case theoretical time complexity $O(n^2 + tn\log(n))$, where t is the number of obtuse angles in the input mesh. IDT is compact and preserves the intrinsic geometry, however it is often difficult to use, since the geodesic-path-based edges are incompatible with conventional mesh data structures.

Delaunay meshes (DMs) are a special triangle mesh whose IDT is the mesh itself. The term was coined by Dyer et al. [2007b], who also developed an algorithm to convert an arbitrary manifold triangle mesh into a DM by edge splitting and refinement [Dyer et al. 2007a]. Their algorithm is theoretically sound, however it often adds too many splitting points which significantly increases the complexity of the resulting DM. Considering local geometry in edge refinement, Liu et al. [2015] proposed a simple yet effective

[1] Our method does not scale to large models due to its high computational cost.



(a) (b)

Fig. 2. Geometry-preserving NLD edge split. (a) Consider an unflippable NLD edge $\widetilde{e}$. (b) We can split $\widetilde{e} = (a, b)$ at some point s so that both the new edges (a, s) and (s, b) are LD. However, the four edges $\{e_i\}_{i=1}^4$ may become NLD because of the new vertex s. Liu et al. [2015] computed the optimal position of s so that splitting $\widetilde{e}$ at s preserves the most LD edges in $\{e_i\}_{i=1}^4$.

algorithm for constructing DM with much fewer vertices than Dyer et al's method.

2.2 Mesh Simplification

There are three typical ways to simplify triangle meshes: vertex decimation, vertex clustering and edge contraction [Cignoni et al. 1998a; Luebke 2001]. Compared to the first two, the last one is easy to implement with effective measure of approximation errors. The quadric error metric (QEM), developed by Garland and Heckbert [1997], is a representative work of this category, which iteratively collapses edges prioritized by a quadric error. This algorithm is efficient and can maintain high fidelity to the original model [Heckbert and Garland 1999].

Delaunay mesh simplification is considered as a *constrained* mesh simplification, since it must ensure the Delaunay condition for all internal edges. Dyer et al. [2007a] adopted the QEM framework and for each edge collapse, they determined an allowable region to enclose the resulting vertex so that all the affected edges remain locally Delaunay. Then the optimal position is chosen to minimize the standard quadric error. This method can collapse edges to optimal positions, however, computing this allowed region is time consuming.

Liu et al.' method [2015] is also based on the QEM framework, but it takes a different strategy for edge collapse: it only considers removable vertices that can be safely removed without violating local Delaunay condition. This method is two orders of magnitudes faster than Dyer et al.'s method while maintaining a similar level of accuracy.

In this paper, we denote the edge collapse operations with Delaunay constraints in [Dyer et al. 2007a] and [Liu et al. 2015] as *Dyer QEM* and *Liu QEM*, respectively.

3 PRELIMINARIES

Our optimal DM simplification is motivated by the existing algorithms of DM construction [Dyer et al. 2007a; Liu et al. 2015] and mesh optimization [Hoppe et al. 1993]. In this section, we briefly present the necessary background knowledge.

Denote by M a manifold triangle mesh. We call an internal edge $e \in M$ *locally Delaunay* (LD) if the sum of the two angles facing e does not exceed π, *non-locally Delaunay* (NLD) otherwise. An NLD edge is *flippable* if it has zero dihedral angle, i.e., its two incident

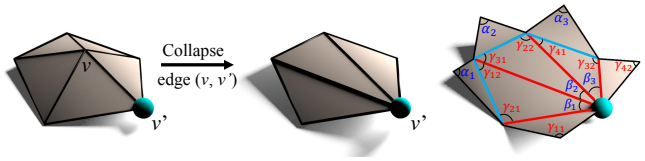


Fig. 3. Collapsing a removable edge (v, v'). An edge (v, v') is *removable* if the collapse $(v, v') \to v'$ does not break the local Delaunay condition: for each edge opposite to v' (blue), $\alpha_i + \beta_i \le \pi$; for each edge incident to v' (red), $\gamma_{j1} + \gamma_{j2} \le \pi$.

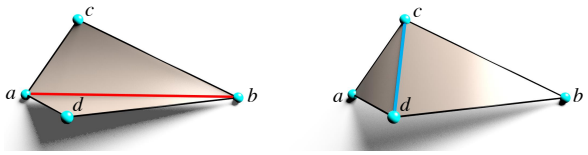


Fig. 4. Flipping an unflippable NLD edge (a, b) alters the geometry. Dyer et al. [2007a] showed that the flipped edge (c, d) must be locally Delaunay. If the new triangles $\triangle acd$ and $\triangle bdc$ intersect other triangular faces, the NLD edge (a, b) is called *physically unflippable*.

faces are coplanar. A boundary edge is NLD if its opposite angle is greater than $\pi/2$. A Delaunay mesh (DM) is a special manifold triangle mesh, in which all the edges satisfy the local Delaunay condition.

There are two methods [Dyer et al. 2007a; Liu et al. 2015] to convert an arbitrary mesh M into a DM with exactly the same geometry. They first flip all the flippable NLD edges. Then, they split each unflippable NLD edge $\widetilde{e}$ followed by geometry-preserving remeshing. Since splitting an edge may turn some neighboring LD edges into NLD (e.g., see Figure 2), they recursively process NLD edges until all edges are locally Delaunay. Both methods are guaranteed to terminate within finite steps and they differ in the local refinement strategy. Dyer et al.'s strategy is purely combinatorial and does not consider the local geometry. As a result, it often adds too many splitting points to the mesh. Liu et al.'s method is shape aware and the resulting DM has bounded space complexity $O(Kn)$, where K is a model-dependent constant.

Geometry-preserving algorithms inevitably increase the mesh complexity due to their upsampling strategies. For highly anisotropic meshes, the vertex numbers may differ in two orders of magnitudes [Liu et al. 2017], making the DMs unpractical for follow-up applications. To balance space complexity and accuracy, Liu et al. [2015] developed an efficient DM simplification algorithm using the QEM framework [Garland and Heckbert 1997]. It iteratively collapses removable edges[2] until the mesh resolution reaches the user-specified target value.

Dyer et al. [2007a] also proposed a geometry-altering algorithm for constructing DM. Unlike the above-mentioned geometry preserving algorithms that flip only the flippable NLD edges, this algorithm flips all NLD edges, thereby the resulting DM alters the geometry of M (Figure 4). Throughout the paper, we use *Dyer GA* and *Dyer GP* to distinguish Dyer et al's geometry-altering and geometry-preserving[3] algorithms, respectively.

[2] An internal edge $e = (v_i, v_j)$ is called *removable* if it has an incident vertex, say v_i, satisfying that the contraction $(v_i, v_j) \to v_j$ does not violate the local Delaunay condition (Figure 3).

[3] *Dyer QEM* is followed if the number of mesh vertices is reduced.

To ease reading, we list the main notations in Table 1.

Table 1. Main notations.

M	input triangle mesh (not necessarily a DM)
M^*	optimal Delaunay mesh
$\widetilde{M}$	DM generated by Liu et al's method
n	number of vertices of M
m	number of vertices of M^*
$\widetilde{n}$	number of vertices of $\widetilde{M}$
$v, v_i, v_j, \cdots$	vertices
e, e_{ij}	edges
$D(,)$	Hausdorff distance
E_c	edge collapse operation
E_s	edge split operation
$S = (\cdots, E_i, \cdots)$	sequence of edge operations $E_i \in \{E_c, E_s\}$
M_S	resulting mesh after applying S to M
Q_{NLD}	priority queue of NLD edges
Q_{REM}	priority queue of removable edges
d	maximal length of the sequence
$X \in \mathbb{R}^d$	real vector
$T(X)$	integer vector after directly rounding X
$S(X)$	sequence of edge operations corresponding to X
M_i	applying the i-th operation to M_{i-1}, $M_0 = M$
$\mathbf{X}_k$	k-th population
$X_{k,j}$	j-th agent of X_k
N_p	population size
$F \in (0, 1)$	differential weight
$C_r \in [0, 1]$	crossover probability
$X'_{k,j}$	mutative agent
$X''_{k,j}$	competitor
$f(X)$	objective function
rand$[0, 1]$	random number between 0 and 1

4 GENERAL DELAUNAY MESH SIMPLIFICATION

4.1 Problem Statement

Given two 3D models A and B, we can quantitatively measure their *difference* using two-sided Hausdorff distance, defined as the mean distance between them:

$$D(A, B) = \max\left\{\max_{p\in A} d(p, B), \max_{p'\in B} d(p', A)\right\} \tag{1}$$

where $d(p, M) = \min_{p'\in M} d(p, p')$ is the shortest distance from point p to mesh M and $d(p, p')$ is the Euclidean distance between points p and p'.

The general Delaunay mesh simplification problem is as follows:

Problem 1. *Given a 2-manifold triangle mesh M (unnecessarily a Delaunay mesh) of n vertices and a user-specified target vertex number m ($< n$), find a Delaunay mesh M^* with m vertices that minimizes the two-sided Hausdorff distance,*

$$M^* = \arg\min_{M'} D(M, M'). \tag{2}$$

Note that the existing methods [Dyer et al. 2007a; Liu et al. 2015] can produce simplified DMs with the user-specified resolution. However, since the global approximation error is not taken into consideration, they do not solve the optimal DM simplification problem.

4.2 Solution Space

To convert a triangle mesh M into a DM, we must process all NLD edges using one of the three edge operations: split E_s, collapse E_c and flip E_f. Hoppe et al. [1993] showed that given any two manifold triangle meshes A and B with the same topology, the topological realization of B can be obtained from A by a finite sequence of these three operations, and vice versa. This means the topological realizations of all possible DMs can be obtained from M by a finite sequence of edge operations. Since an edge flip is equivalent to an edge split followed by an edge collapse, E_s and E_c are the most fundamental operations in mesh optimization and simplification [Heckbert and Garland 1999; Hoppe 1996; Hoppe et al. 1993].

Let $\mathbb{S}$ be the set of all possible finite sequences consisting of E_s and E_c, which can be applied to the input mesh M. For any sequence $S \in \mathbb{S}$, we denote M_S the mesh obtained by applying S to M. To solve the optimal DM simplification problem, we compute an optimal sequence $S^* \in \mathbb{S}_{DM}$, such that

$$M_{S^*} = \arg\min_{S\in\mathbb{S}_{DM}} D(M, M_S) \tag{3}$$

where the subset $\mathbb{S}_{DM} \subset \mathbb{S}$ satisfies that for any sequence $S \in \mathbb{S}_{DM}$, M_S is a DM of m vertices.

We propose a novel 2D Cartesian grid model to characterize the solution space $\mathbb{S}_{DM}$. Then finding the optimal sequence of edge operations equals to finding an optimal path in $\mathbb{S}_{DM}$.

Notice that edge split E_s can be applied to any *unflippable*[4] NLD edges, and edge collapse E_c can be applied to any removable edges in M. To reduce the search space, we sort all NLD edges and removable edges in two priority queues Q_{NLD} and Q_{REM}, respectively.

NLD edges $\widetilde{e}$ are assigned with a key defined by the discrete Laplacian (i.e., the sum of cotangent of the opposite angles of $\widetilde{e}$), which is a simple measure of the local Delaunay condition. The smaller the key, the farther the edge $\widetilde{e}$ is away from local Delaunay. As the first rule of thumb, when an edge split operation is necessary, we always apply it to the NLD edge $\widetilde{e}_{min}$ with the smallest key. After splitting $\widetilde{e}_{min}$, we locally refine the mesh and update the keys for its neighboring edges.

Removable edges $\widehat{e}$ are assigned with a key which measures the approximation error by collapsing $\widehat{e}$. We compute this key using the classic quadratic error metric [Garland and Heckbert 1997]. The smaller the key, the smaller the error, and the better approximation we obtain. As the second rule of thumb, when we apply an E_c operation, we extract the removable edge $\widehat{e}_{min}$ with smallest key value from Q_{REM} and apply E_c to $\widehat{e}_{min}$. After collapsing $\widehat{e}_{min}$, Q_{REM} is locally updated to reflect the local geometry change around $\widehat{e}_{min}$.

Based on the above-mentioned rules, a finite sequence of operations E_s and E_c applied to M is deterministic, which is denoted by

$$S = (E_1, E_2, \cdots, E_i, \cdots),\ E_i \in \{E_s, E_c\} \tag{4}$$

[4] If an NLD edge is flippable, we simply flip it and no further operations are required. Therefore, we assume that all NLD edges are unflippable.

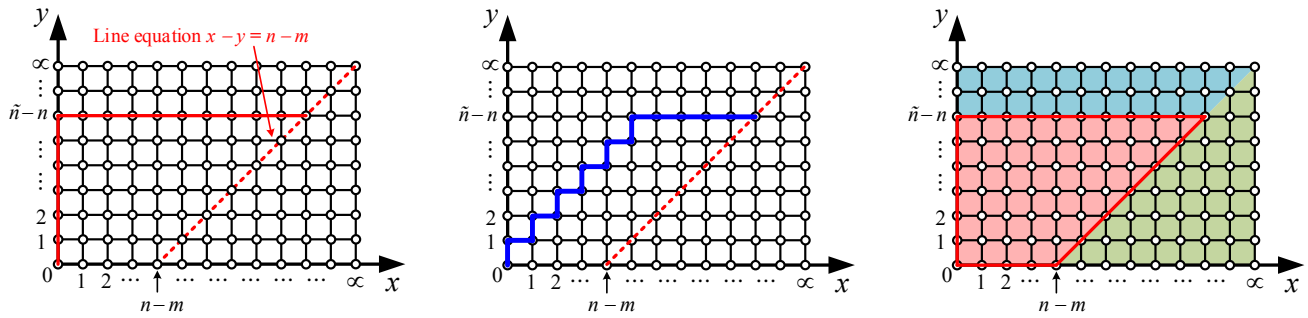


Fig. 5. 2D Cartesian grid model $G(V, E)$ is a non-negative integer grid (x, y) in $\mathbb{R}^2$, $x \geq 0$ and $y \geq 0$. Any finite sequence of E_s and E_c applied to the input mesh M can be mapped to a monotonic path in G. All the meshes with m vertices lie on the line $L : x - y = n - m$ (red dashed line in the left). Both Liu et al.'s method and Dyer et al.' method (*Dyer GP*) start from the origin (0, 0), first go up (along $+y$ direction) to the node $(0, \widetilde{n} - n)$, where $\widetilde{n}$ is the number of vertices in constructed geometry-preserving DM from M, and then go horizontally (along $+x$ direction) to the node $(\widetilde{n} - m, \widetilde{n} - n)$ (solid red line in the left). *Dyer GA* method corresponds to the zigzag path shown in blue lines in the middle. Our restricted solution space $\widetilde{G}(V, E)$ is shown in the red shaded region (bounded by red lines) in the right; see text for details.

To characterize the solution space, we map a sequence S into a 2D Cartesian grid (Figure 5):

(1) The horizontal and vertical axes represent edge collapse and split, respectively.
(2) Every grid node (x, y) maps to triangle meshes $M_{(x,y)}$ with a fixed vertex count. Except for the origin $M_{(0,0)} \triangleq M$, such a mapping is one to many.
(3) Two adjacent nodes are linked by an edge operation. For example, applying E_c (resp. E_s) to $M_{(x,y)}$ produces $M_{(x+1,y)}$ (resp. $M_{(x,y+1)}$).

Using the Cartesian grid model, any sequence of edge operations is a *monotonic*[5] path. Therefore, simplifying DM is equivalent to computing a monotonic path in the grid. Although the solution space is infinite, we have the following observations to reduce the size of solution space.

(1) Increasing x (or y) by 1 removes (or adds) a vertex. Therefore, all the meshes with m vertices must lie on the line $L \triangleq \{(x, y) | x - y = n - m\}$ (the red dashed line in Figure 5 left).
(2) Any simplification process transforming M to a DM of m vertices is simply a monotonic path from the origin to a node on L. In particular, each of the existing methods corresponds to a special path. Liu et al's method [2015] and Dyer et al's method [2007a] (*Dyer GP*) first compute a geometry-preserving DM by upsampling (edge split) and then simplify the mesh by edge collapse. Therefore, it first travels along the y-axis and then goes horizontally until reaching L (the red line in Figure 5 left). Dyer et al's method [2007a] (*Dyer GA*) alternatively splits and collapses edges and its path is hereby zigzag (the blue line in Figure 5 middle).
(3) Once a mesh becomes a DM, there is no need to keep upsampling it, since these split edges will soon be collapsed by the downsampling process. Therefore, we set an upper bound[6] of the number of split edges to be $\widetilde{n} - n$, where $\widetilde{n}$ is the number of vertices in the geometry-preserving DM generated by Liu et al's method.
(4) Any monotonic path that *passes through* line L is *not* an optimal path. This is because the nodes on the right side of L correspond to meshes with less than m vertices. A path goes through L means some intermediate meshes have resolution lower than the target m. For non-planar models, the fewer the vertices, the larger the approximation error.

Putting it all together, we define the solution space as a trapezium, bounded by $y = \widetilde{n} - n$, $x - y = n - m$, $y = 0$ and $x = 0$ (the red shaded region in Figure 5 right). The top-right corner of the trapezium corresponds to a Delaunay mesh, which is the output of Liu et al's method.

5 ALGORITHM

The trapezoidal solution space can be naturally viewed as a restricted graph model $\widetilde{G} = (V, E)$ and each valid solution corresponds to a monotonic path. Given a $k \times l$ Cartesian grid, there are $O(\frac{(k+l)!}{k!l!})$ monotonic paths. Therefore, it is not feasible (even for low-resolution meshes) to enumerate all the possible paths and then compare their costs. A slightly better solution is to adopt graph traversal or path finding algorithms, such as breadth-first search (BFS) and A* search. The former traverses all the graph edges, hereby is time consuming. The latter, guided by a problem-specific heuristic function that estimates the cost of the cheapest path from the current node to the goal, sweeps only a subset of the graph edges. However, it is not easy to find such a heuristic in our problem.

Rather than solving the optimal path problem from a *combinatorial* optimization standpoint, we consider it from a *numerical* optimization perspective, which itself is a well-studied field and many efficient and powerful computational tools are available [Horst et al. 2000; Spall 2003]. We propose a novel method (Section 5.1) to map a monotonic path (i.e., a sequence of edge operations) to a fixed-length array so that the objective function can be written as a real-valued function. Then we minimize it by differential evolution (DE) [Das and Suganthan 2011], which is one of the state-of-the-art global

[5] A path is monotonic if it always goes upward and rightward.
[6] Note that each edge split operation increases the vertex count by one.

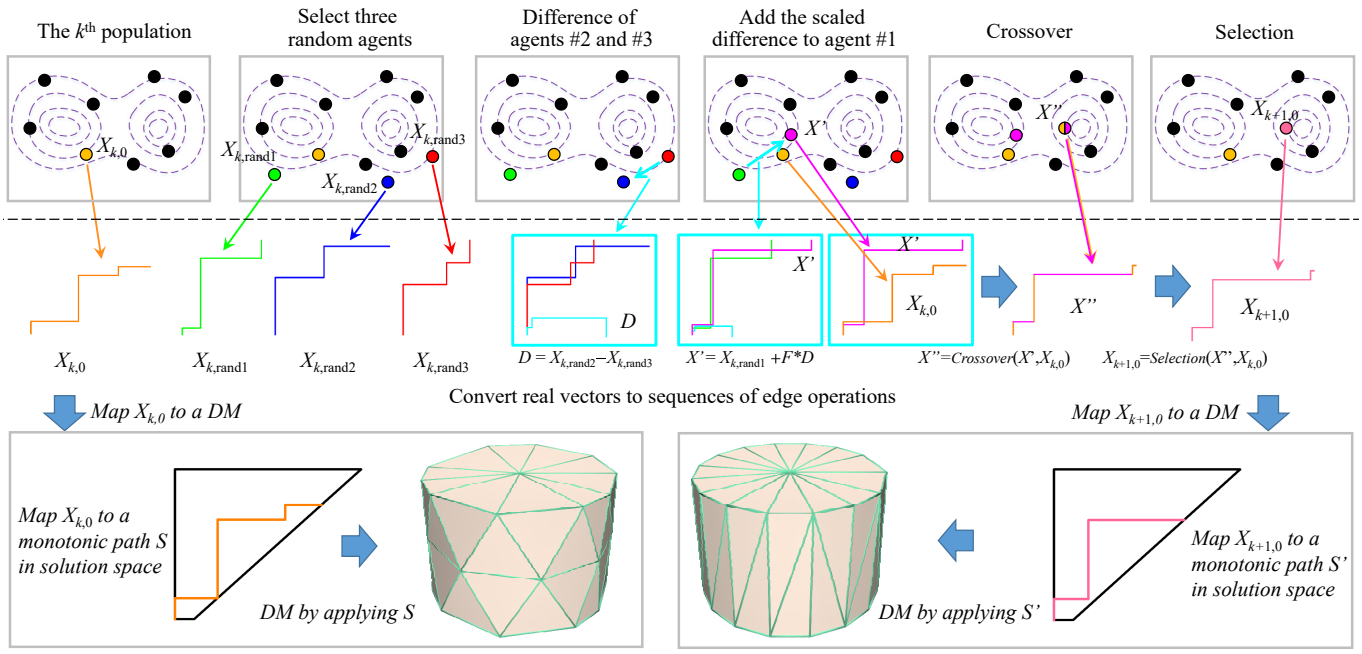


Fig. 6. Illustration of one iteration in differential evolution (DE). Our algorithm maps a real-vector $X \in \mathbb{R}^d$ to a sequence of edge operations, which in turn corresponds to a monotonic path in the solution space $\widetilde{G}$. Then using the canonical *DE/rand/1/bin* form of DE, our method iteratively evolves the population $\mathbf{X}_k = \{X_{k,j}\}_{j=1}^{N_p}$, $k = 1, 2, \cdots$, where $X_{k,j}$ is the j-th vector (*agent*), until the termination condition is met; see text for details.

optimization algorithm (Section 5.2). We illustrate our algorithmic pipeline in Figure 6 and present the pseudo code in Algorithm 1.

5.1 Mapping Scheme

We denote the sets of all real numbers, non-negative real numbers, integers, non-negative integers and positive integers as $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, $\mathbb{Z}$, $\mathbb{Z}_{\geq 0}$ and $\mathbb{Z}_+$, respectively.

Since DE optimizes real-valued functions with real variables, we need to encode the monotonic paths in $\widetilde{G}$ into real-vectors $X \in \mathbb{R}^d$ and vice versa. Since the paths have varying lengths, we need to map them to a d-variate function where $d \in \mathbb{Z}$ is a constant. We adopt the run-length coding by partitioning an arbitrary operation sequence $\{\cdots, E_i, \cdots\}$, $E_i \in \{E_s, E_c\}$, into segments, where all operations are of the same type in each segment. Without loss of generality, we assume that the sequence begins with an E_s and represent it as $(n_1, n_2, n_3, \cdots)$, where n_i is number of operations in the i-th segment and $n_1 \in \mathbb{Z}_{\geq 0}$ and $n_i \in \mathbb{Z}_+$, $i \geq 2$. For example, the sequence $(E_c, E_c, E_c, E_s, E_s)$ is simply encoded as $(0, 3, 2)$.

Observe that the shortest[7] sequence is $(\widetilde{n} - n, \widetilde{n} - m)$ and the longest[8] sequence is $(1, \cdots, 1, n - m + 1)$. So we set the maximal length

$$d = 2(\widetilde{n} - n) \tag{5}$$

For any sequence shorter than d, we append zeros to its end.

For any real vector $X \in \mathbb{R}^d_{\geq 0}$, we round each element in X to the closest integer and denote the resulting vector by $T(X) \in \mathbb{Z}^d_+$. Lampinen and Zelinka [1999] showed that this rounding operation can be intuitively viewed as building leveled regions into a landscape (akin to 1D step functions) and differential evolution with self-adaptive reproduction (see Section 5.2) can move *across* these regions.

Denote by $T(X) = \{I_i\}_{i=1}^d$, $I_i \in \mathbb{Z}_{\geq 0}$, the integer vector after direct rounding. Let $M_0 \triangleq M$ be the input mesh and denote by M_i $(i \geq 1)$ the mesh after applying the I_i operations of E_c or E_s (depending on the parity of i) to M_{i-1}. We define a *dummy* sequence, S_{dummy}, and any sequence that cannot turn M into a DM of m vertices will be set to S_{dummy}. See Figure 7 for an example.

We apply the following rules to convert $T(X)$ to a sequence $S(X)$ of edge operations:

- *Rule 1.* If after or during processing I_i the mesh M_i becomes DM, $i \neq d$, then only removable edge collapse (E_c) is applied to M_i until it becomes a DM of m vertices; i.e., the elements $\{I_{i+1}, \cdots, I_d\}$ are skipped (see M_3 in Figure 7b for an example).
- *Rule 2.* During processing I_i (i.e., only n_i operations are processed, $n_i \leq I_i$), the mesh M_i reaches the upper boundary of $\widetilde{G}$ (i.e., the line $y = \widetilde{n} - n$, which is upper red line in Figure 5 left) and still is not a DM, then this sequence is assigned to S_{dummy} (see Figure 7c for an example); if the mesh M_i reaches the right boundary of $\widetilde{G}$ (i.e., the line $x - y = n - m$, which is the right dashed red line in Figure 5 left and middle) and still is not DM, continue to process I_{i+1}, i.e. the remaining $(I_i - n_i)E_c$ operations are skipped.

[7]The shortest sequence corresponds to Liu et al's method.
[8]The longest sequence corresponds to *Dyer GA* method.

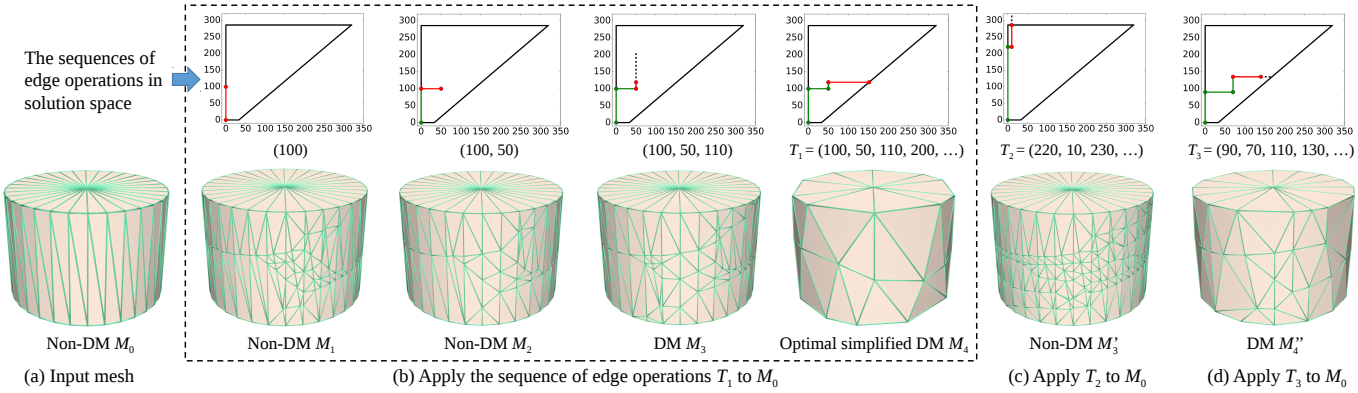


Fig. 7. Illustration of the mapping scheme using a toy model. (a) The input non-DM mesh M_0 is going through three sequences of edge operations T_1, T_2 and T_3, respectively. (b) The mapping process of applying T_1 = (100, 50, 110, 200, $\cdots$) to M_0. M_1 (apply 100 E_s to M_0) and M_2 (apply 50 E_c to M_1) are still non-DMs. When applying 110 E_s to M_2, we obtain a DM after 19 edge splits, and thus, skip the remaining E_s operations (dashed lines). When applying 200 E_c to M_3, we obtain the DM M_4 with the desired number of vertices after 102 E_c. (c) The sequence T_2 = (220, 10, 230, $\cdots$) is invalid and assigned to S_{dummy}, since during processing I_3 = 230 E_s, the mesh M_3' reaches the upper bound y = 286, however it is not a DM yet. (d) The sequence T_3 = (90, 70, 110, 130, $\cdots$) is also invalid: when processing I_4 = 130 E_c, both Q_{REM} and Q_{NLD} become empty, however the vertex number of M_4'' does not reach the target m.

ALGORITHM 1: Delaunay mesh simplification using DE

Input: A 2-manifold triangle mesh M of n vertices, a user-specified target vertex number m, the population size N_p
Output: A high-quality simplified DM M^* of m vertices

Initialize the first population $\mathbf{X}_0$ of N_p agents, initialize $\mathbf{D}_0$ as the set of initial objective function values of N_p agents
$k \leftarrow 0$
while the termination conditions are not met **do**
  $\mathbf{X}_{k+1} \leftarrow \emptyset, \mathbf{D}_{k+1} \leftarrow \emptyset$
  foreach agent $X_{k,j} \in \mathbf{X}_k$ **do**
    Generate a mutative agent $X'_{k,j}$ by mutation
    Generate a trial agent $X''_{k,j}$ by crossover
    Compute the mapped sequence $S(X''_{k,j})$ and the candidate mesh $M_{S(X''_{k,j})}$ (see Algorithm 2)
    if $D(M, M_{S(X''_{k,j})}) < D_{k,j}$ **then**
      $\mathbf{X}_{k+1} \leftarrow \mathbf{X}_{k+1} \cup \{X''_{k,j}\}$
      $\mathbf{D}_{k+1} \leftarrow \mathbf{D}_{k+1} \cup \{D(M, M_{S(X''_{k,j})})\}$
    else
      $\mathbf{X}_{k+1} \leftarrow \mathbf{X}_{k+1} \cup \{X_{k,j}\}$
      $\mathbf{D}_{k+1} \leftarrow \mathbf{D}_{k+1} \cup \{D_{k,j}\}$
    end
    $k \leftarrow k + 1$
  end
end
Find the minimum element $D_{k,j} \in \mathbf{D}_k$ and its corresponding agent $X_{k,j}$
Output the simplified DM $M^* = M_{S(X_{k,j})}$

- *Rule 3.* During processing I_i, if Q_{REM} becomes empty,
  - if Q_{NLD} is empty at the same time, failure will be caused and the sequence is assigned to S_{dummy} (see Figure 7d for an example);
  - otherwise, continue to process I_{i+1} (edge split afterwards may create more removable edges to fill Q_{REM}).

To raise the success rate, we require $I_{odd} \in [0, \widetilde{n} - n]$ and $I_{even} \in [0, \widetilde{n} - m]$. Algorithm 2 presents the pseudo-code of mapping a real vector $X \in \mathbb{R}^d_{\geq 0}$ into a sequence of edge operations and then computing the corresponding simplified DM. We denote the mapped sequence by $S(X)$.

5.2 Differential Evolution

Stochastic methods have been widely studied for solving global optimization problems [Spall 2003]. Among them, differential evolution (DE) is a popular evolutionary computing method for optimizing multi-variate real-valued functions without using the function's gradient. Compared with other evolutionary algorithms, DE is arguably competitive due to its simplicity, robustness and better convergence rate [Vesterstrom and Thomsen 2004]. Although the probabilistic convergence and global optimality of DE are proven only for a restricted class of objective functions (i.e., real-valued second-order continuous functions that possess a unique global optimum) [Ghosh et al. 2012], DE has been recognized as a a highly effective tool for a wide range of optimization problems, including constrained, multi-objective, multi-modal and dynamic optimization even with non-differentiable functions [Das et al. 2016; Das and Suganthan 2011]. Many of these problems do not satisfy the above-mentioned convergence condition, DE still outperforms the other global optimizers. In this paper, we adopt DE to look for a global solution to the general DM simplification problem.

DE starts from a randomly chosen population that samples the searching space $\widetilde{G}$, then iteratively evolves the population for searching the global optimal solution in $\widetilde{G}$ [Das et al. 2016; Das and Suganthan 2011]. In each iteration, it creates new candidates by performing simple mutation, crossover and selection operations, and the one with the best score or fitness is kept in the population. In addition to its great success in real-parameter, large-scale global optimization, DE has also proven effective in geometric modeling [Liu et al. 2016].

ALGORITHM 2: Generating simplified DM from a given real vector

Input: The mesh M of n vertices and a real vector $X \in \mathbb{R}^d_{\geq 0}$
Output: The DM after applying the edge operations $S(X)$

Round each element X_i in X to the closest integer $I_i \in \mathbb{Z}_{\geq 0}$, $1 \leq i \leq d$
$S(X) \leftarrow \emptyset$;
$M_0 \leftarrow M$;
foreach operation $I_i \in T(X)$ **do**
  Set $E = E_s$ if i is odd, E_c otherwise
  Apply I_i operations of E to M_{i-1}
  if *During processing I_i operations of E, DM is reached or m vertices is reached* **then**
    Skip the remaining operations of E;
  end
  if *During processing I_i operations of E, upper boundary of $\widetilde{G}$ is reached but still is not a DM* **then**
    Skip the remaining operations of E;
    Set $S(X) = S_{dummy}$ and return $\emptyset$;
  end
  if *During processing I_i operations of E, both Q_{REM} and Q_{NLD} become empty* **then**
    Skip the remaining operations of E;
    Set $S(X) = S_{dummy}$ and return $\emptyset$;
  end
  Set the obtained mesh after processing I_i as M_i;
  Append edge operations applied to M_{i-1} to the end of $S(X)$;
  if *M_i is a DM* **then**
    Apply E_c to M_i until it becomes a DM of m vertices;
    Append edge operations applied to M_i to the end of $S(X)$;
    Return M_i;
  end
end
Return $\emptyset$.

Let $f(X) : \Omega \subset \mathbb{R}^d \rightarrow \mathbb{R}$ be a real-valued objective function, $X = (x_1, x_2, \cdots, x_d)$ the variable, $x_i \in \mathbb{R}$, and the search domain Ω is non-empty and bounded in $\mathbb{R}^d$. With the aid of the mapping scheme presented in Section 5.1, we apply the canonical *DE/rand/1/bin* form of DE [Das and Suganthan 2011].

Consider a non-negative real-valued objective function

$$f(X) = D(M, M_{S(X)}),\ X \in \mathbb{R}^d_{\geq 0}, \tag{6}$$

where $D(,)$ is defined in Eq.(1). If $S(X)$ is dummy, $f(X) = \infty$.

Denote the k-th population by $\mathbf{X}_k = \{X_{k,j}\}_{j=1}^{N_p}$, where $X_{k,j} = \{x_{k,j,i}\}_{i=1}^{d}$ is the j-th vector (*agent*). We present the details of our algorithm as follows.

5.2.1 Initialization. We begin with a randomly initialized population $\mathbf{X}_0 = \{X_{0,j}\}_{j=1}^{N_p}$, where each initial agent $X_{0,j}$ is sampled in a bounded space Ω. We still use the definition $Y_{max} = \widetilde{n} - n$, and $X_{max} = \widetilde{n} - m$, which specify maximum continuous edge split number and edge collapse number, respectively. Then the bounded space Ω is represented as:

$$\Omega = \left\{ \{x_i\}_{i=1}^{d} \;\middle|\; \begin{array}{l} 0 \leq x_i \leq Y_{max},\ \text{if } i \text{ is odd} \\ 0 \leq x_i \leq X_{max},\ \text{otherwise} \end{array},\ x_i \in \mathbb{R} \right\}. \tag{7}$$

The initial population X_0 can be generated by producing random values uniformly distributed on closed interval $[0, Y_{max}]$ or $[0, X_{max}]$, denoted as

$$x_{0,j,i} = \begin{cases} rand_{j,i}[0,1] \cdot Y_{max},\ \text{if } i \text{ is odd} \\ rand_{j,i}[0,1] \cdot X_{max},\ \text{otherwise} \end{cases}, \quad i = 1, 2, ..., d,\ j = 1, 2, ..., N_p \tag{8}$$

where $rand_{j,i}[0,1]$ generates a uniformly distributed random number in $[0,1]$ and is instantiated independently for each (j,i).

5.2.2 Mutation. It generates new candidate solutions by adding an agent with scaled difference of two randomly selected agents. In the k-th iteration, for each agent $X_{k,j}$, three other agents $X_{k,rand1}$, $X_{k,rand2}$ and $X_{k,rand3}$ are randomly chosen from $\mathbf{X}_k$, where $rand1$, $rand2$, $rand3$ are distinct integers and not equal to j. A *mutative* agent $\bar{X}'_{k,j}$ against $X_{k,j}$ is generated as

$$\bar{X}'_{k,j} = X_{k,rand1} + F(X_{k,rand2} - X_{k,rand3}) \tag{9}$$

where F is a scalar factor in the range $(0,1)$. Since $\bar{X}'_{k,j}$ should belong to space Ω defined in Eq.(7), it is further processed by

$$\bar{x}_{k,j,i} = \max(\bar{x}'_{k,j,i}, 0)$$
$$x'_{k,j,i} = \begin{cases} \min(\bar{x}_{k,j,i}, Y_{max}),\ \text{if } i \text{ is odd} \\ \min(\bar{x}_{k,j,i}, X_{max}),\ \text{otherwise} \end{cases} \tag{10}$$

5.2.3 Crossover. A binomial crossover builds a *trial* agent $X''_{k,j}$ by copying elements from either *mutative* agent $X'_{k,j}$ or original agent $X_{k,j}$:

$$x''_{k,j,i} = \begin{cases} x'_{k,j,i},\ \text{if } (\text{rand}_{k,j,i}[0,1] \leq C_r \text{ or } i = i_{rand}) \\ x_{k,j,i},\ \text{otherwise} \end{cases} \tag{11}$$

where $C_r \in [0,1]$ is the *crossover rate*, with higher C_r indicating more elements in the original agent are replaced by new elements, and i_{rand} is a random index in $[1,d]$ to ensure that the *trial* agent get at least one element from the *mutative* agent.

5.2.4 Selection. Because our problem is a minimization problem, selection chooses the one with lower objective value from the *trial* agent $X''_{k,j}$ and original agent $X_{k,j}$, and sets it to be the j-th agent of next generation:

$$X_{k+1,j} = \begin{cases} X''_{k,j},\ \text{if } f(X''_{k,j}) \leq f(X_{k,j}) \\ X_{k,j},\ \text{otherwise} \end{cases} \tag{12}$$

where $f(X_{k,j})$ and $f(X''_{k,j}))$ is computed as $f(X) = D(M, M_{S(X)})$.

5.2.5 Termination conditions. The algorithm terminates when one of the following conditions is met: (1) the iteration number exceeds a user-specified iteration number n_{iter}; (2) the relative change $\frac{f_k - f_{k+1}}{f_k}$ does not exceed a threshold τ in consecutive n_c iterations, where f_k is the minimum objective function value in the kth population.In all our experiments, we use fixed parameters $n_{iter} = 100$, $\tau = 1.0e^{-4}$ and $n_c = 5$.

6 EXPERIMENTAL RESULTS & DISCUSSIONS

We implemented our algorithm in C++ and compared its performance with state of the art in terms of running time and accuracy. Timing was measured on an Intel Xeon E5-2698 v3 CPU 2.30 GHz. The accuracy was measured by the *approximation error*, which is

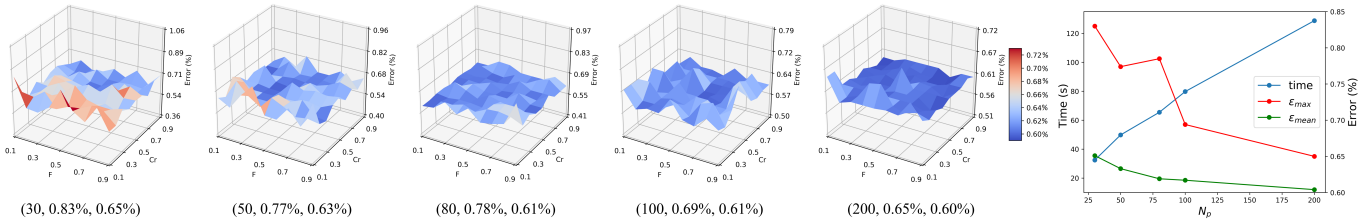


Fig. 8. Both approximation error ε and running time closely depend on the population size. We use the Octaflower model for an example. The 3-tuple below each 3D chart is $(N_p, \varepsilon_{max}, \varepsilon_{mean})$, where N_p is the population size, ε_{max} and ε_{mean} are the maximal and the mean approximation errors. In each 3D chart, the vertical axis is the approximation error ε and the two horizontal axes are the crossover rate C_r and the mutation parameter F. The curve plot shows that (1) the approximation error decreases when N_p increases and (2) the average time is linearly proportional to N_p.

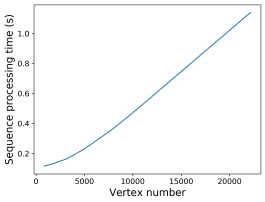
Table 2. Comparison of Liu, Dyer GA, Dyer GP and our method on models of three categories. Time is measured in seconds, n and m are the vertex numbers of input mesh and output simplified DM, respectively. Err and Err* are approximation errors and Err/Err* is the error ratio compared to our method. Itr no is the iteration number.

Category	Model	n	m	Liu			Dyer GA			Dyer GP			Ours		
				Err (%)	Err/Err*	Time (s)	Err (%)	Err/Err*	Time (s)	Err (%)	Err/Err*	Time (s)	Err* (%)	Time (s)	Itr no
CAD	Joint	221	200	4.54	14.18	0.5	5.47	17.08	51.2	2.49	7.76	54.8	0.32	6.4	6
	Headst	571	500	2.01	2.41	10.9	12.2	14.62	636.1	1.64	1.97	1288.7	0.83	252.6	10
	Fandisk	850	800	0.21	1.88	0.4	4.55	41.48	1.8	1.23	11.21	87.9	0.11	22.2	8
	Casting	5,096	4,000	0.29	3.89	0.8	0.28	3.80	29.3	0.19	2.53	170.2	0.07	261.0	17
	Octaflower	7,919	500	1.48	2.49	0.8	1.57	2.64	121.6	1.37	2.30	164.5	0.59	70.1	9
	Sharpsphere	10,443	500	3.57	2.74	3.4	2.54	1.95	184.2	2.28	1.75	416.4	1.30	190.2	14
Man-made	Chair	324	200	3.60	2.61	0.1	5.63	4.08	8.1	4.62	3.35	36.7	1.38	4.2	6
	Bench	2,707	1,500	1.28	2.21	7.6	0.90	1.56	296.1	0.95	1.63	1094.5	0.58	363.4	12
	Desk	3,472	3,000	0.32	2.34	6.6	0.61	4.53	112.7	0.27	2.00	644.0	0.14	127.1	6
Graphics	Teapot	22,162	10,000	0.06	1.33	5.5	0.31	7.10	244.2	0.26	5.99	1554.8	0.04	358.0	7
	Kitten	70,442	25000	0.04	1.35	16.2	0.05	1.76	625.8	0.26	9.89	6209.1	0.03	1385.9	11
	Bunny	98,996	50,000	0.03	1.28	25.4	0.09	4.18	1017.3	0.06	2.89	9144.8	0.02	1228.4	6
	Gargoyle	111,137	50,000	0.07	1.17	27.8	0.14	2.30	1149.9	0.20	3.45	8686.4	0.06	2501.5	15
	Lucy	155,814	50,000	0.07	1.09	42.7	0.11	1.75	2025.5	0.22	3.54	12491.7	0.06	1161.9	6

defined as the two-sided Hausdorff distance between the input mesh M and the output simplified DM, normalized by the diagonal length of model's bounding box. The Metro tool [Cignoni et al. 1998b] was chosen to quickly compute the approximate two-sided Hausdorff distance. We evaluated and compared different methods on a wide range of 3D models, including 1,000 CAD and man-made models collected from the Thingi10K[9] 3D dataset [Zhou and Jacobson 2016].

6.1 Time complexity

The running time of our DE-based method is linearly proportional to the product of QEM time complexity ($n \log n$), population size and iteration count. To generate a simplified DM in each agent, we check the result of every valid path-mapped vector X by executing its corresponding edge operations $S(X)$. Since we only maintain two priority queues for adding or deleting vertices, this operation takes $O(n \log n)$ time. The right inset shows the timings for examining a vector with respect to the vertex number. Our algorithm terminates, when either the energy does not decrease or the iteration number reaches a prescribed value. We observe for almost all experiments, our algorithm terminates due to convergence of energy function (see the last column in Table 2).



6.2 Parameter Setting

The DE algorithm (Agorithm 1) has three main parameters: the population size N_p, the crossover rate C_r in Eq. (11) and the mutation parameter F in Eq. (9). We observe that the approximation error closely depends on the population size N_p: the larger the population size, the more samples in the search space and the higher probability to reach the global optimum. Meanwhile, the running time is also linearly proportional to N_p and is very time-consuming when N_p becomes large. In our experiment, we observe that our algorithm is robust and highly consistent when $N_p \in [100, 200]$; see Figure 8 for some summarized results. Then we set $N_p = 100$ for a tradeoff between approximation errors and computational cost.

When the population size is fixed, we observe that low cross rate C_r leads to large approximation error. The reason is that low C_r makes the crossover process pick most elements from original agent, and hereby lowers population diversity. We also observe that both too high and too low mutation rate F lead to large approximation error (see $N_p = 100$ in Figure 8). The reason is that (1) too high F results in too much disturbance and low search efficiency, and (2) too low F lowers population diversity. In our experiments, we set $C_r = 0.9$ and $F = 0.5$ to increase population diversity while avoiding too much disturbance.

[9]https://ten-thousand-models.appspot.com/

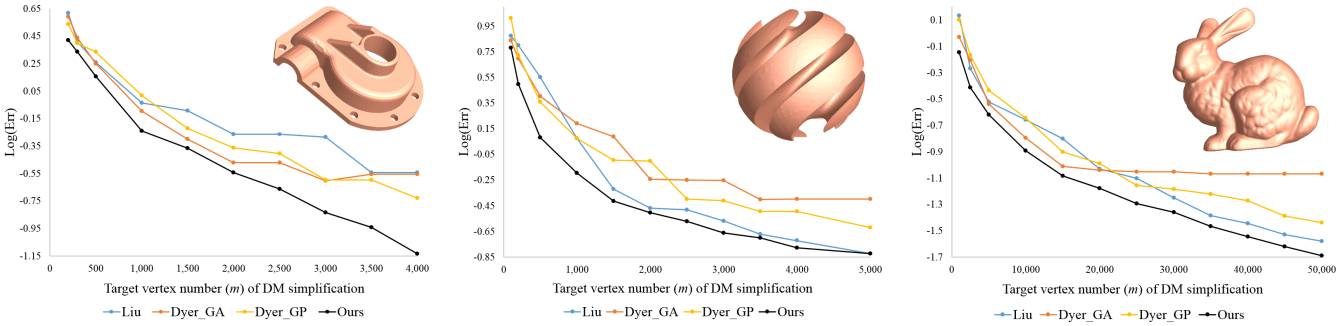


Fig. 10. The approximation error plots of three representative models with respect to different target vertex numbers. Our method (black curves) consistently outperforms all other methods.

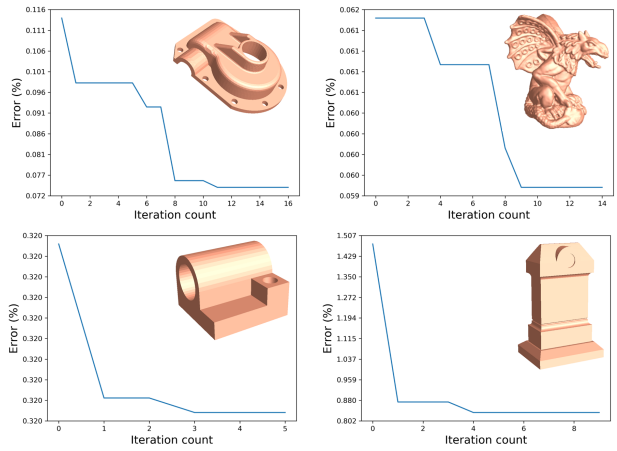


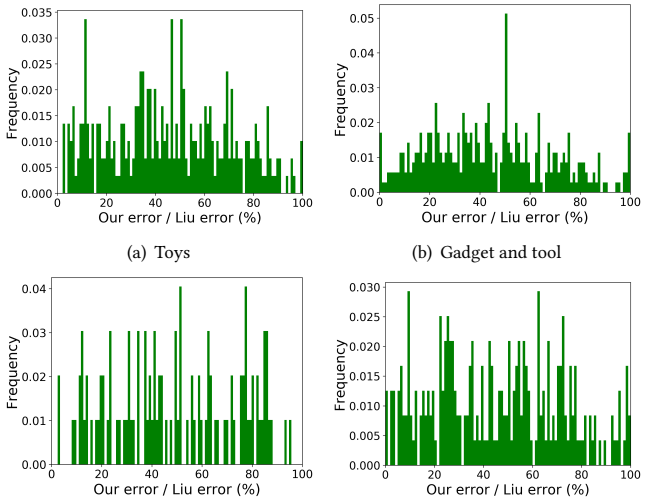
Fig. 9. Convergence plots of four representative models. See also the accompanying video.

Given the above parameter settings, our method converges quickly. Four representative examples are shown in Figure 9.

6.3 Performance and Comparisons.

We compare our DE solution with three state-of-the-art methods: Liu et al.'s method (Liu) [Liu et al. 2015], *Dyer GA* and *Dyer GP* in [Dyer et al. 2007a]. We experiment on CAD, man-made and graphics models with representative target vertex number, and Table 2 shows the statics. We further conduct experiments on models with varying target vertex numbers, and show three representative results in Figure 10. We observe that our method consistently outperforms other methods, and is particularly effective on CAD and man-made models. Figure 1 and 12 show the visual comparison of four methods (see also the accompanying video). Thanks to the two-sided Hausdorff distance that is used in error measure, our method can preserve sharp features well and deal with models with multiple connected components, and thus produces visually better results than the other three methods on models with sharp features.

We note that there are advanced algorithms to automatically detect sharp features and then preserve them in downstream applications. However, these methods require parameters and their results are sensitive to the parameter settings. For example, in [Lévy and Liu 2010] sharp features are detected by a normal anisotropy defined by a symmetric tensor field that penalizes the vertices far away from the tangent plane of the surface, and this metric contains



(a) Toys (b) Gadget and tool

(c) Household (d) Art and fashion

Fig. 11. Comparison between our method and Liu et al's method. The histograms show the error ratio $\varepsilon_{ours}/\varepsilon_{Liu}$ on the 1000 CAD and man-made objects, selected from Thingi10K dataset with labels in four categories: *Toys*, *Gadget and tool*, *Household* and *Art and fashion*.

an importance parameter. For complex models with feature sizes spanning different scales, it is tedious (and sometimes not even possible) to tune the optimal parameters to fit all scales. In sharp contrast, our method adopts the Hausdorff distance to measure the shape distortion, which intrinsically encodes and preserves the sharp features. As a result, it is completely parameter-free (and thus very robust) and fully automatic.

We further compare our method with Liu et al.'s method in terms of error ratio on the 1,000 CAD and man-made objects, organized in four categories. Figure 11 shows the results summarized in histograms. Again, our method effectively reduces the approximation errors and consistently outperforms Liu's method for all categories.

6.4 Dyer QEM vs. Liu QEM

Both Dyer GP method and Liu et al.' method upsample the input mesh to make it a DM in a geometry-preserving way. These two methods differ in the succedent edge collapse operations, i.e., Dyer QEM and Liu QEM. Our method also adopts Liu QEM, which only collapse a removable edge into one of its vertices. As a comparison, for every edge collapse, Dyer QEM allows to optimize the position

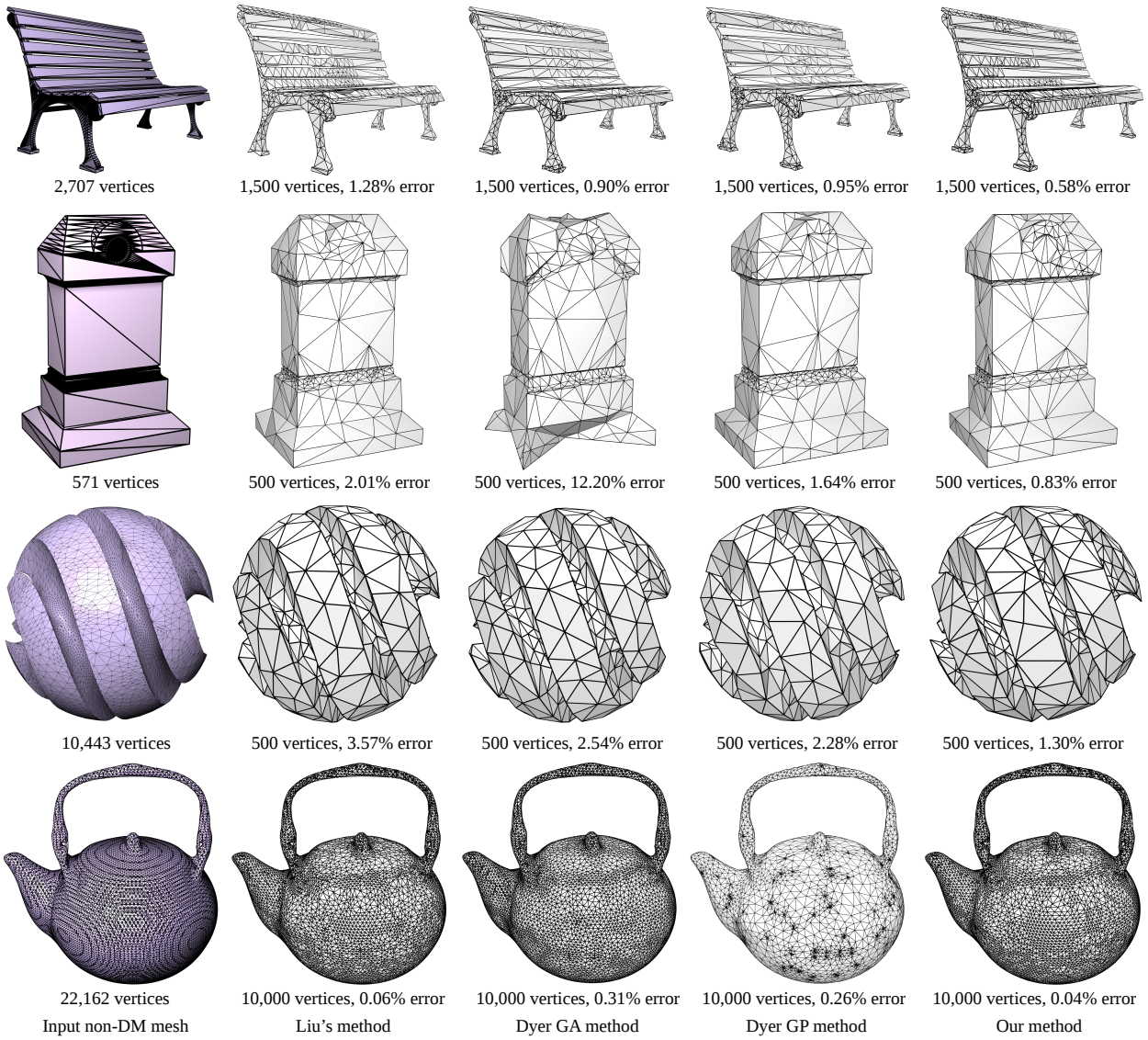
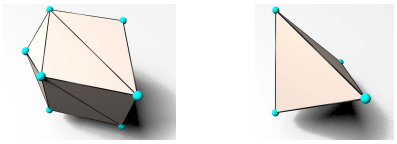


Fig. 12. Results and visual comparison. The bench model has multiple connected components and all models (except for the last one) have sharp features. Our method can deal with sharp features better than other methods. The last model is smooth without sharp feature. Although our result still has smaller Hausdorff distance than that of Liu et al's method, the visual difference is not significant. Images are rendered in high resolution, allowing zoom-in examination.

of target vertex in an allowable region, and thus, is more flexible than Liu QEM. However, the statistics in Table 2 and visual results in Figure 1 and 12 all show that our method is better than Dyer GP method, confirming that the global stochastic searching strategy plays a more important role than the optimal local edge operation.

6.5 Comparison with globally optimal solution

Our DE-based method can effectively find a global solution. To verify this property, one possible way is to compare the DE solution with the global optimization solution, which can be obtained by enumerating all possible monotonic paths in solution space. We design a 9-vertex irregular mesh by adding a vertex into a parallelepipedon and simplify it into a tetrahedron (Figure 13). For this toy model, brute force searching takes 3157.21 seconds to compute the globally optimal solution, while our DE-based method computes exactly the same solution in only 0.284 seconds. We note that, as discussed in Section 5, for meshes with even dozens of vertices, the number of all possible paths is exponential and thus intractable. A naïve approximation to this enumeration-based global solution is bread-first search (BFS) and at each grid node, the best mesh with smallest approximation error is selected and propagated. Unfortunately, this naïve BFS solution is still very time consuming and is only available to meshes with small vertex number. In Table 3, we compare our DE solution and BFS solution. The results show that (1) the error of our method is close to and still slightly better than the error of BFS, and (2) the time of our method is far less than BFS. Therefore, our DE solution is an effective global solution in search space.



(a) 9-vertex input mesh (b) 4-vertex output DM

Fig. 13. Given the 9-vertex mesh (a) as input, the globally optimal simplification of 4-vertex DM (b) can be found by exhaustive enumeration which takes 3157.21 seconds. Our DE-based method computes exactly the same solution in only 0.284 seconds.

Table 3. Comparison of our method and BFS method. n and m are the vertex numbers of input mesh and output simplified DM, respectively. Err is approximation errors and time is measured in seconds.

Model	n	m	BFS method		Our method	
			Err (%)	Time (s)	Err (%)	Time (s)
Kitten	1,370	1,000	0.048	13,624	0.038	52.4
Armadillo	1,500	1,000	0.871	64,146	0.777	18.6
Fandisk	850	800	0.148	221,753	0.110	22.2

6.6 Limitations

Our method can only work with 2-manifold meshes. For smooth models without sharp features, our results still have smaller Hausdorff distances than those of Liu et al's method; however, we observe that the visual differences are not significant (see the last model in Figure 12). Due to its global nature, our method is $10 - 100\times$ slower than Liu et al.'s method. Nevertheless, due to computational intractability to obtain ground truth of global optimization, our method can serve as a practical baseline for comparing the accuracy of other methods. For extremely large graphics models, we can pre-simplify these models using QEM, Liu et al's method or Dyer et al.'s method, and then our method can be tailored to fit different time and quality requirement.

7 CONCLUSION & FUTURE WORK

This paper presents the optimal DM simplification problem and formulates it by a 2D Cartesian grid model. We develop a novel DE-based method to compute an effective global solution. Extensive evaluation shows that our method consistently outperforms the existing methods in terms of approximation error. In particular, our method is highly effective for CAD models and man-made objects with sharp features but less details. Moreover, our method is fully automatic and can preserve sharp features and deal with models with multiple connected components.

From the application point of view, it is desired to develop a part-aware optimal DM where the parts are given different weights so that parts with higher weights are processed with higher priority. We also believe the DE-based path finding algorithm is a general computational framework that can be extended to solve other global optimal path problem, which is worth further investigation.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation of China (61725204, 61432003, 61521002 and 61661130156) and the Royal Society-Newton Advanced Fellowship (NA150431).

REFERENCES

Alexander I. Bobenko and Boris A. Springborn. 2007. A discrete Laplace-Beltrami operator for simplicial surfaces. *Discrete & Computational Geometry* 38, 4 (2007), 740–756.

Jean-Daniel Boissonnat, Ramsay Dyer, and Arijit Ghosh. 2013. Constructing Intrinsic Delaunay Triangulations of Submanifolds. *CoRR* abs/1303.6493 (2013).

Paolo Cignoni, Claudio Montani, and Roberto Scopigno. 1998a. A comparison of mesh simplification algorithms. *Computers & Graphics* 22, 1 (1998), 37–54.

P Cignoni, C Rocchini, and R Scopigno. 1998b. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum* 17, 2 (1998), 167–174.

Swagatam Das, Sankha Subhra Mullick, and Ponnuthurai Nagartnam Suganthan. 2016. Recent advances in differential evolution - An updated survey. *Swarm and Evolutionary Computation* 27 (2016), 1–30.

Swagatam Das and Ponnuthurai Nagartnam Suganthan. 2011. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation* 15, 1 (2011), 4–31.

Ramsay Dyer, Hao Zhang, and Torsten Möller. 2007a. Delaunay Mesh Construction. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing (SGP '07)*. 273–282.

Ramsay Dyer, Hao Zhang, and Torsten Möller. 2007b. Voronoi-Delaunay Duality and Delaunay Meshes. In *Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling (SPM '07)*. 415–420.

Ramsay Dyer, Hao Zhang, and Torsten Möller. 2008. Surface Sampling and the Intrinsic Voronoi Diagram. In *Proceedings of the Symposium on Geometry Processing (SGP '08)*. 1393–1402.

Herbert Edelsbrunner and Nimish R. Shah. 1997. Triangulating topological spaces. *International Journal of Computational Geometry and Applications* 7, 4 (1997), 365–378.

Matthew Fisher, Boris Springborn, Peter Schröder, and Alexander I. Bobenko. 2007. An Algorithm for the Construction of Intrinsic Delaunay Triangulations with Applications to Digital Geometry Processing. *Computing* 81, 2-3 (Nov. 2007), 199–213.

Michael Garland and Paul S. Heckbert. 1997. Surface Simplification Using Quadric Error Metrics. In *ACM SIGGRAPH '97*. 209–216.

Sayan Ghosh, Swagatam Das, Athanasios V. Vasilakos, and Kaushik Suresh. 2012. On Convergence of Differential Evolution Over a Class of Continuous Functions With Unique Global Optimum. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 42, 1 (2012), 107–124.

Paul S. Heckbert and Michael Garland. 1999. Optimal triangulation and quadric-based surface simplification. *Computational Geometry: Theory and Applications* 14, 1-3 (1999), 49–65.

Hugues Hoppe. 1996. Progressive Meshes. In *Proceedings of SIGGRAPH '96*. 99–108.

Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. 1993. Mesh Optimization. In *SIGGRAPH '93*. 19–26.

Reiner Horst, Panos M. Pardalos, and Nguyen Van Thoai. 2000. *Introduction to Global Optimization*. 2nd edition, Kluwer Academic Publishers.

Kaimo Hu, Dong-Ming Yan, David Bommes, Pierre Alliez, and Bedrich Benes. 2017. Error-Bounded and Feature Preserving Surface Remeshing with Minimal Angle Improvement. *IEEE Trans. Vis. Comput. Graph.* 23, 12 (2017), 2560–2573.

Jouni Lampinen and Ivan Zelinka. 1999. Mixed integer-discrete-continuous optimization with differential evolution. In *Proc. 5th Int. Mendel Conf. Soft Comput.* 71–76.

Bruno Lévy and Yang Liu. 2010. Lp Centroidal Voronoi Tessellation and Its Applications. *ACM Trans. Graph.* 29, 4, Article 119 (2010), 119:1–119:11 pages.

Yong-Jin Liu, Dian Fan, Chun-Xu Xu, and Ying He. 2017. Constructing Intrinsic Delaunay Triangulations from the Dual of Geodesic Voronoi Diagrams. *ACM Trans. Graph.* 36, 2, Article 15 (2017), 15:1–15:15 pages.

Yong-Jin Liu, Chun-Xu Xu, Dian Fan, and Ying He. 2015. Efficient Construction and Simplification of Delaunay Meshes. *ACM Trans. Graph.* 34, 6, Article 174 (2015), 174:1–174:13 pages.

Yong-Jin Liu, Chun-Xu Xu, Ran Yi, Dian Fan, and Ying He. 2016. Manifold Differential Evolution (MDE): A Global Optimization Method for Geodesic Centroidal Voronoi Tessellations on Meshes. *ACM Trans. Graph.* 35, 6, Article 243 (2016), 243:1–243:10 pages.

David P. Luebke. 2001. A Developer's Survey of Polygonal Simplification Algorithms. *IEEE Comput. Graph. Appl.* 21, 3 (2001), 24–35.

Manish Mandad, David Cohen-Steiner, and Pierre Alliez. 2015. Isotopic Approximation Within a Tolerance Volume. *ACM Trans. Graph.* 34, 4, Article 64 (2015), 64:1–64:12 pages.

Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. 1987. The Discrete Geodesic Problem. *SIAM J. Comput.* 16, 4 (1987), 647–668.

Igor Rivin. 1994. Euclidean Structures on Simplicial Surfaces and Hyperbolic Volume. *Annals of Mathematics* 139, 3 (1994), 553–580.

James C. Spall. 2003. *Introduction to Stochastic Search and Optimization*. 3rd edition, Wiley.

Jakob Vesterstrom and Rene Thomsen. 2004. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Proc. 6th Congress on Evolutionary Computation*. 1980–1987.

Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. *arXiv preprint* arXiv:1605.04797 (2016).