Efficient Mesh Optimization Schemes based on Optimal Delaunay Triangulations

Long Chen[a,*], Michael Holst[b,1]

[a]*Department of Mathematics, University of California at Irvine, Irvine, CA 92697*
Email: chenlong@math.uci.edu
[b]*Department of Mathematics, University of California at San Diego, La Jolla, CA 92093.*
Email: mholst@math.ucsd.edu

Abstract

In this paper, several mesh optimization schemes based on optimal Delaunay triangulations are developed. High-quality meshes are obtained by minimizing the interpolation error in the weighted L^1 norm. Our schemes are divided into classes of local and global schemes. For local schemes, several old and new schemes, known as mesh smoothing, are derived from our approach. For global schemes, a graph Laplacian is used in a modified Newton iteration to speed up the local approach. Our work provides a mathematical foundation for a number of mesh smoothing schemes often used in practice, and leads to a new global mesh optimization scheme. Numerical experiments indicate that our methods can produce well-shaped triangulations in a robust and efficient way.

Keywords: mesh smoothing, mesh optimization, mesh generation, Delaunay triangulation, Optimal Delaunay triangulation

1. Introduction

We shall develop fast and efficient mesh optimization schemes based on Optimal Delaunay Triangulations (ODTs) [1, 2, 3]. Let ρ be a given density function defined on a convex domain $\Omega \subset \mathbb{R}^n$, i.e. $\rho > 0$, $\int_\Omega \rho dx < \infty$. Let $\mathcal{T}$ a simplicial triangulation of Ω, and let $u(\boldsymbol{x}) = \|\boldsymbol{x}\|^2$ and u_I the piecewise linear nodal interpolation of u based on $\mathcal{T}$. We associate the following weighted L^1 norm of the interpolation error as an energy to the mesh $\mathcal{T}$

$$E(\mathcal{T}) = \int_\Omega |(u_I - u)(\boldsymbol{x})|\rho(\boldsymbol{x})d\mathbf{x}.$$

Let $\mathscr{T}_N$ denote the set of all triangulations with at most N vertices. Our mesh optimization schemes will be derived as iterative methods for solving the following optimization

*Corresponding Author. Telephone: (949)8246595; Fax: (949)8247993. The first author is supported by NSF Grant DMS-0811272, and in part by NIH Grant P50GM76516 and R01GM75309.
[1]The second author was supported in part by NSF Awards 0715146 and 0915220, and DTRA Award HDTRA-09-1-0036.

problem:

$$\inf_{\mathcal{T}\in\mathscr{T}_N} E(\mathcal{T}). \tag{1.1}$$

Minimizer of (1.1) will be called Optimal Delaunay Triangulations.

Mesh optimization by minimizing some energy, also known as the variational meshing method, has been studied by many authors; see, e.g. [4, 5, 6, 7] and references therein. There are many energies proposed in the literature for this purpose, including the widely used harmonic energy in moving mesh methods [8, 9, 10], summation of weighted edge lengths [11, 12], and the distortion energy used in the approach of Centroid Vornoni Tessellation (CVT) [13, 14]. The advantages of our approach are:

1. Mathematical analysis is provided to show minimizers of (1.1) will try to equidistribute the mesh size according to the density function as well as preserve the shape regularity.
2. Optimization of the connectivity of vertices is naturally included in our optimization problem.
3. Efficient algorithms, including local and global mesh optimization schemes, are developed for the optimization problem (1.1).

To solve the optimization problem (1.1), we decompose it into two sub-problems. Let us denote a triangulation $\mathcal{T}_N$ by a pair (p,t), where $p\in\Omega^N$ represents the set of N vertices and t represents the connectivity of vertices, i.e. how vertices are connected to form simplices, and rewrite the energy as $E(p,t)$. We solve the following two sub-problems iteratively:

1. Fix the location of vertices and solve $\min_t E(p,t)$;
2. Fix the connectivity of vertices and solve $\min_p E(p,t)$.

We stress from the outset that both problems $\min_t E(p,t)$ and $\min_p E(p,t)$ do not need to be solved exactly. We are not interested in the *optimal* mesh but rather meshes with good geometric quality (including the density of vertices and shape regularity of simplices). We shall show that the mesh quality will be considerably improved by performing just a few steps of the iteration methods developed in this paper.

Let us first consider the optimization problem $\min_t E(p,t)$. That is, for a fixed vertex set p, find the optimal connectivity of the vertices (in the sense of minimizing the weighted interpolation error $E(\mathcal{T})$.) In [2, 3], we proved that when Ω is convex, the minimizer is a Delaunay triangulation of the point set p. Thus, the problem $\min_t E(p,t)$ is simplified to:

$$\text{Given a set of vertices } p, \text{construct a Delaunay triangulation of } p. \tag{1.2}$$

The problem (1.2) is well studied in the literature [15, 16]. We can classify methods proposed for (1.2) as one of

- Local method: edge or face flipping;
- Global method: lifting method (QHULL).

The focus of this paper is on the optimization problem $\min_p E(p,t)$, namely optimizing the placement of vertices when the connectivity is fixed. We shall also discuss two types of methods:

- Local mesh smoothing;
- Global mesh optimization.

Local relaxation methods are commonly used methods for mesh improvement. For example, Gauss-Seidel-type relaxation methods consider a local optimization problem by moving only one vertex at a time. The vertex is moved inside the domain bounded by its surrounding simplicies while keeping the same connectivity to improve geometric mesh quality such as angles or aspect ratios. This is known as mesh smoothing in the meshing community [17, 18, 19, 1, 20, 21, 22, 23, 13]. With several formulas for the interpolation error, we shall derive a suitable set of local mesh smoothing schemes among which the most popular scheme, Laplacian smoothing, will be derived as a special case.

Local methods, however, can only capture the high frequency in the associated energy, and thus results in slow convergence when the number of grid points becomes larger; see [12, 24] for related discussions and numerical examples. To overcome slow convergence of local mesh smoothing schemes, some sophisticated multigrid-like methods, notably full approximation scheme (FAS), have been recently proposed [25, 26, 27, 28, 29]. To use multigrid-type methods, one has to generate and maintain a nested mesh hierarchy which leads to complex implementations with large memory requirements. The interpolation of point locations from the coarse grid to the fine grid can fold triangulations, and addressing this carefully leads to additional implementation complexity. See [27, 25] for related discussions.

We shall derive a global mesh optimization method by using another technique of multilevel methods: multilevel preconditioners. One iteration step of our method reads as

$$p^{k+1} = p^k - A^{-1}\nabla E(p^k, t), \tag{1.3}$$

where A is a graph Laplacian matrix with nice properties: it is symmetric and positive definite (SPD) and also a diagonally dominant M-matrix. Note that if we replace A by $\nabla^2 E(p^n)$ in (1.3), it becomes Newton's method. Our choice of A can be thought as a preconditioner of the Hessian matrix. Comparing with Newton's method, our choice of A has several advantages

- A is easy to compute, while $\nabla^2 E$ is relatively complicated;
- A^{-1} can be computed efficiently using algebraic multigrid methods (AMG) since A is an SPD and M-matrix, while $\nabla^2 E$ may not be;
- A is a good approximation of $\nabla^2 E$.

We should clarify that our methods are designed for mesh optimization, not mesh generation. Therefore, we only move interior nodes and assume all boundary nodes are well placed to capture the geometry of the domain. We note that many mesh generators become slow when the number of vertices becomes large. Therefore, we call mesh

generators only to generate a very coarse mesh, and then apply our mesh optimization methods to the subsequently refined meshes. By doing so, we can generate high-quality meshes with large numbers of elements in an efficient way.

The concept of Optimal Delaunay Triangulation (ODT) was introduced in [2] and some local mesh smoothing schemes were reported in a conference paper [1] and summarized in the first author's Ph. D thesis [3]. Application of ODT to other problems can be found in [30, 31, 32]. In this paper, we include some results from [1, 3] for the completeness and more importantly, present several new improvements listed below:

- several improved smoothing schemes for non-uniform density functions;
- a neat remedy for possible degeneration of elements near the boundary;
- a global mesh optimization scheme;
- some 3D numerical examples.

The rest of this paper is organized as follows. In section 2, we review the theory on Delaunay and Optimal Delaunay Triangulations. In section 3, we go over algorithms for the construction of Delaunay triangulation. In section 4, we give several formulae on the energy and its derivatives. Based on these formulae, we present several optimization schemes including local mesh smoothing and a global modified Newton method. In section 5, we provide numerical examples to show the efficiency of our methods. In the last section, we conclude and discuss future work.

2. Delaunay and Optimal Delaunay Triangulations

Delaunay triangulation (DT) is the most commonly used unstructured triangulation in many applications. It is often defined as the dual of the Voronoi diagram [33]. In this section we use an equivalent definition [34, 35] which involves only the triangulation itself.

Let V be a finite set of points in $\mathbb{R}^n$. The convex hull of V, denoted by $CH(V)$, is the smallest convex set which contains these points.

Definition 2.1. *A* Delaunay triangulation *of V is a triangulation of $CH(V)$ so that it satisfies* empty sphere condition*: there are no points of V inside the circumsphere of any simplex in the triangulation.*

There are many characterizations of Delaunay triangulations. In two dimensions. Sibson [36] observed that Delaunay triangulations maximize the minimum angle of any triangle. Lambert [37] showed that Delaunay triangulations maximize the arithmetic mean of the radius of inscribed circles of the triangles. Rippa [38] showed that Delaunay triangulations minimize the Dirichlet energy, i.e. the integral of the squared gradients. D'Azevedo and Simpson [39] showed that in two dimensions, Delaunay triangulations minimizes the maximum containing radius (the radius of the smallest sphere containing the simplex). Rajan [40] generalized this characterization to higher dimensions. Chen and Xu [2] characterize Delaunay triangulations from a function approximation point of view. We shall briefly survey the approach by Chen and Xu [2] in the following.

Definition 2.2. *Let $\Omega \subset \mathbb{R}^n$ be a bounded domain, $\mathcal{T}$ a triangulation of Ω, and $u_{I,\mathcal{T}}$ be the piecewise linear and globally continuous nodal interpolation of a given function $u \in \mathcal{C}(\bar{\Omega})$ based on the triangulation $\mathcal{T}$. Let $1 \leq q \leq \infty$. We define an* error-based mesh quality $Q(\mathcal{T}, u, q)$ *as*

$$Q(\mathcal{T}, u, q) = \|u - u_{I,\mathcal{T}}\|_{L^p(\Omega)} = \left(\int_\Omega |u(\boldsymbol{x}) - u_{I,\mathcal{T}}(\boldsymbol{x})|^q \, d\mathbf{x} \right)^{1/q}.$$

By choosing a special function $u(\boldsymbol{x}) = \|\boldsymbol{x}\|^2$, we can characterize the Delaunay triangulation as an optimal triangulation which achieves the best error-based mesh quality. The proof of the following result can be found in [2].

Theorem 2.3. *For a finite point set V, we let $\Omega = CH(V)$ and denote $\mathscr{T}_V$ all possible triangulations of Ω by using the points in V. Then*

$$Q(DT, \|\boldsymbol{x}\|^2, q) = \min_{\mathcal{T} \in \mathscr{T}_V} Q(\mathcal{T}, \|\boldsymbol{x}\|^2, q), \quad \forall\, 1 \leq q \leq \infty.$$

This type of result was first proved in $\mathbb{R}^2$ by D'Azevedo and Simpson [39] for $q = \infty$ and Rippa [41] for $1 \leq q < \infty$. Rajan proved the case $q = \infty$ in multiple dimensions [40]. Theorem 2.3 is a generalization of their work to general L^q norms in multiple dimensions. A similar result on the optimality of Delaunay triangulation in higher dimensions was also obtained by Melissaratos in a technical report [42].

Remark 2.4. Indeed we proved in [2] that $u_{I,DT}(\boldsymbol{x}) \leq u_{I,\mathcal{T}}(\boldsymbol{x})$ for all $\boldsymbol{x} \in \Omega$. Since u is convex (downwards), we have point-wise optimality

$$|u_{I,DT}(\boldsymbol{x}) - u(\boldsymbol{x})| \leq |u_{I,\mathcal{T}}(\boldsymbol{x}) - u(\boldsymbol{x})|.$$

Therefore Theorem 2.3 is trivially generalized to the general density ρ since $\rho d\mathbf{x}$ defines a measure.

We have shown that Delaunay triangulations optimize the connectivity when the vertices of triangulations are fixed. Now we free the locations of vertices to further optimize the triangulation.

Definition 2.5. *Let $\mathscr{T}_N$ denote the set of all triangulations with at most N vertices. Given a continuous function u on $\bar{\Omega}$ and $1 \leq q \leq \infty$, a triangulation $ODT \in \mathscr{T}_N$ is* optimal *if*

$$Q(ODT, u, q) = \inf_{\mathcal{T} \in \mathscr{T}_N} Q(\mathcal{T}, u, q).$$

We call it an Optimal Delaunay Triangulation (ODT) *with respect to u and q.*

The following theorem concerns the existence of optimal Delaunay triangulations and can be found in [2, 3]. In general, ODTs are not unique.

Theorem 2.6. *Given $1 \leq q \leq \infty$, an integer N, and a convex function u, there exists an optimal Delaunay triangulation $ODT \in \mathscr{T}_N$ with respect to u and q.*

Graded and anisotropic meshes are important to keep the geometry features and achieve better approximation in numerical solutions of partial differential equations. In [2], we generalize the concept of DT and ODT to general convex functions u. The density and the shape of the mesh will be controlled by the metric defined by the Hessian of u. In this paper, we keep the simplest quadratic form $u(\boldsymbol{x}) = \|\boldsymbol{x}\|^2$ and use the density function ρ to control the gradient of the mesh density.

In the following, we give an error analysis for the interpolation error $u - u_I$ to show that ODT will aim to produce shape regular and uniform meshes. For simplicity, we only present the case $q = 1$, i.e. the L^1 norm of the interpolation error. Similar results hold for general $q \in [1, \infty]$; see [3, 31].

Suppose τ is a d-simplex with vertices $\boldsymbol{x}_i, i = 1, \cdots, d+1$. Let $\tau_i(\boldsymbol{x})$ denote the simplex formed by vertices $\boldsymbol{x}_i$, $i = 1, \cdots, d+1$ with $\boldsymbol{x}_i$ being replaced by $\boldsymbol{x}$. We define $\lambda_i(\boldsymbol{x}) = |\tau_i(\boldsymbol{x})|/|\tau|$ as the barycentric coordinates of $\boldsymbol{x}$ with respect to $\boldsymbol{x}_i$ for $i = 1, \cdots, d+1$. By the definition of barycentric coordinates, it is easy to verify that

$$u_I(\boldsymbol{x}) = \sum_{i=1}^{d+1} \lambda_i(\boldsymbol{x}) u(\boldsymbol{x}_i), \ \boldsymbol{x} = \sum_{i=1}^{d+1} \lambda_i(\boldsymbol{x}) \boldsymbol{x}_i, \ \text{ and } \ \sum_{i=1}^{d+1} \lambda_i(\boldsymbol{x}) = 1. \tag{2.1}$$

Lemma 2.7. *For a simplex τ with vertices $(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_{d+1})$, let $t_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2$ be the squared edge length. For $u(\boldsymbol{x}) = \|\boldsymbol{x}\|^2$, we have*

$$\int_\tau (u_I - u)(\boldsymbol{x}) d\mathbf{x} = \frac{|\tau|}{(d+1)(d+2)} \sum_{i,j=1, j>i}^{d+1} t_{ij}. \tag{2.2}$$

Proof. By Taylor expansion,

$$u(\boldsymbol{x}_i) = u(\boldsymbol{x}) + \nabla u(\boldsymbol{x})(\boldsymbol{x}_i - \boldsymbol{x}) + (\boldsymbol{x} - \boldsymbol{x}_i)^t (\boldsymbol{x} - \boldsymbol{x}_i). \tag{2.3}$$

Here we use the fact that $\nabla^2 u = 2\boldsymbol{I}$. Multiplying both sides of (2.3) by λ_i and summing together, we obtain

$$\sum_{i=1}^{d+1} \lambda_i u(\boldsymbol{x}_i) = u(\boldsymbol{x}) \sum_{i=1}^{d+1} \lambda_i + \nabla u(\boldsymbol{x}) \sum_{i=1}^{d+1} \lambda_i (\boldsymbol{x}_i - \boldsymbol{x}) + \sum_{i=1}^{d+1} \lambda_i (\boldsymbol{x} - \boldsymbol{x}_i)^t (\boldsymbol{x} - \boldsymbol{x}_i).$$

Using the property (2.1), we simplify it as

$$u_I(\boldsymbol{x}) - u(\boldsymbol{x}) = \sum_{i,j=1}^{d+1} \lambda_i \lambda_j (\boldsymbol{x}_j - \boldsymbol{x}_i)^t (\boldsymbol{x} - \boldsymbol{x}_i). \tag{2.4}$$

We switch the indices i, j to get

$$u_I(\boldsymbol{x}) - u(\boldsymbol{x}) = \sum_{i,j=1}^{d+1} \lambda_i \lambda_j (\boldsymbol{x}_i - \boldsymbol{x}_j)^t (\boldsymbol{x} - \boldsymbol{x}_j). \tag{2.5}$$

Summing (2.4) and (2.5), we obtain

$$u_I(\boldsymbol{x}) - u(\boldsymbol{x}) = \frac{1}{2} \sum_{i,j=1}^{d+1} \lambda_i(\boldsymbol{x}) \lambda_j(\boldsymbol{x}) t_{ij}^2 = \sum_{i,j=1, j>i}^{d+1} \lambda_i(\boldsymbol{x}) \lambda_j(\boldsymbol{x}) t_{ij}^2.$$

Using the integral formula

$$\int_\tau \lambda_i(\boldsymbol{x})\lambda_j(\boldsymbol{x})d\mathbf{x} = \frac{1}{(d+1)(d+2)}|\tau|,$$

we get the desired result. □

Theorem 2.8. *For $u(\boldsymbol{x}) = \|\boldsymbol{x}\|^2$, there exists a constant C_d depending only on the dimension of space such that*

$$Q(\mathcal{T}_N, u, 1) \geq C_d \sum_{\tau\in\mathcal{T}} |\tau|^{1+2/d} \geq C_d N^{-2/d}|\Omega|^{\frac{d+2}{d}}. \tag{2.6}$$

Furthermore

1. *the first inequality becomes equality if and only if each simplex is equilateral;*
2. *the second inequality becomes equality if and only if all simplex are equal volumes.*

Therefore, minimizing $Q(\mathcal{T}, u, 1)$ will attempt to equidistribute volumes and edge lengths of all simplices in the triangulation.

Proof. First, by the error formula (2.2), we have, for $u(\boldsymbol{x}) = \|\boldsymbol{x}\|^2$,

$$Q(\mathcal{T}, u, 1) = \sum_{\tau\in\mathcal{T}} Q(\tau, u, 1) = \frac{1}{(d+1)(d+2)} \sum_{\tau\in\mathcal{T}} |\tau| \sum_{i,j=1,j>i}^{d+1} t_{ij} \geq C_d \sum_{\tau\in\mathcal{T}} |\tau|^{1+2/d}.$$

Here we use a geometric inequality [43]

$$\sum_{i,j=1,j>i}^{d+1} t_{ij} \geq C_d|\tau|^{2/d}. \tag{2.7}$$

Equality holds if and only if τ is an equilateral simplex.

We then use the Hölder inequality

$$|\Omega| = \sum_{\tau\in\mathcal{T}} |\tau| \leq \left(\sum_{\tau\in\mathcal{T}} |\tau|^{1+2/d}\right)^{\frac{d}{d+2}} \left(\sum_{\tau\in\mathcal{T}} 1\right)^{\frac{2}{d+2}} = \left(\sum_{\tau\in\mathcal{T}} |\tau|^{1+2/d}\right)^{\frac{d}{d+2}} N^{\frac{2}{d+2}},$$

to obtain

$$\sum_{\tau\in\mathcal{T}} |\tau|^{1+2/d} \geq N^{-2/d}|\Omega|^{\frac{d+2}{d}}.$$

The equality holds if and only if τ = constant $= N^{-1}|\Omega|$. □

For non-uniform density, we give a simple (not rigorous) analysis by assuming ρ is approximated by a piecewise constant function. We refer readers to [32] and [3] for the proof of general cases. For a given function $\rho \in L^r(\Omega), r > 0$, we define the weighted volume

$$|\tau|_{\rho^r} = \int_\tau \rho^r \, d\mathbf{x}.$$

When ρ is constant on the simplex τ, as in Lemma 2.7, we obtain the formula

$$\int_\tau (u_I - u)\rho \, d\mathbf{x} = C_d \sum_{i,j=1, j>i}^{d+1} t_{ij} |\tau|_\rho.$$

Again use (2.7), we then get

$$\int_\tau (u_I - u)\rho \, d\mathbf{x} \geq C_d |\tau|^{1+2/d} \rho = |\tau|_{\rho^r}^{1+2/d} \quad \text{with } r = \frac{d}{d+2}. \tag{2.8}$$

The inequality holds if and only if all edge lengths are equal. Summing over all elements and applying Hölder inequality, we have

$$\int_\Omega (u_I - u)\rho \, d\mathbf{x} = \sum_{\tau \in \mathcal{T}} \int_\tau (u_I - u)\rho \, d\mathbf{x} \geq \sum_{\tau \in \mathcal{T}} |\tau|_{\rho^r}^{1+2/d} \geq N^{-2/d} \|\rho\|_{L^r}.$$

For general density, we can approximate it by a piecewise constant function and the remainder is of high order $o(N^{-2/d})$. We conclude that for general density, an optimal triangulation will be shape regular and equidistribute the weighted volume $|\tau|_{\rho^r}$, which means when ρ is big, the volume $|\tau|$ should be small. Therefore, non-uniform density consequently leads to a shape-regular and graded mesh.

3. Algorithms for Delaunay Triangulations

In this section, we survey two popular algorithms, one local and another global, for the construction of a Delaunay triangulation for a given set of points.

3.1. Local method: edge flipping

One local method to construct a Delaunay triangulation is known as edge flipping [36, 33] in two dimensions or edge/face flipping [21, 33] in three dimensions. Here we describe a two dimensional edge flipping algorithm following [33] and discuss its convergence behavior from the function approximation point of view.

Given a 2D triangulation $\mathcal{T}$, we denote by $\mathcal{E}$ the edge set of $\mathcal{T}$ and ne the number of edges in $\mathcal{E}$. We say an edge $ab \in \mathcal{E}$ is *locally Delaunay* if (i) it belongs to only one triangle or (ii) it belongs to two triangles, abc and abd, and d lies outside the circumcircle of abc and c lies outside the circumcircle of abd. Notice that if ab is not locally Delaunay then the union of the two triangles sharing ab, i.e., the quadrilateral $acbd$ is convex. It is fairly easy to see, then, the other diagonal cd will be locally Delaunay. The flipping algorithm replaces ab by cd, through updating the set of triangles and edges.

We prove the flipping algorithm will terminate from the function approximation point of view. Different choices of diagonals will lead to different linear interpolations. The interpolation error will be the difference between the graph of the linear interpolation and the paraboloid. From Fig. 1, we see that $Q(\mathcal{T}_1, u, p) > Q(\mathcal{T}_2, u, p)$ where $\mathcal{T}_2$ is obtained via the flipping algorithm from $\mathcal{T}_1$. Thus local edge flipping algorithm will result a sequence of triangulations $\mathcal{T}_k, k = 1, 2, \ldots m$ in $\mathscr{T}_V$ with

$$Q(\mathcal{T}_1, u, p) > Q(\mathcal{T}_2, u, p) > \ldots > Q(\mathcal{T}_m, u, p) > 0.$$

Algorithm: Edge flipping for 2D triangulations

while *exist non locally Delaunay edge* **do**
find a non locally Delaunay edge ab;
find the two triangles sharing ab;
flip diagonals of the convex quadrilateral formed by these two triangles.
end

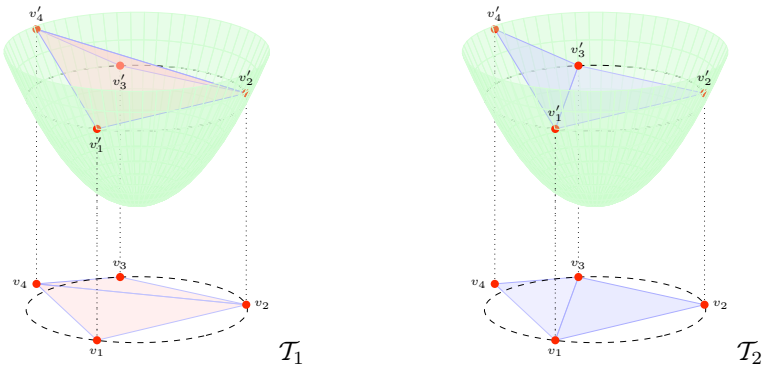


Figure 1: Different triangulations using different diagonals and the graph of corresponding linear interpolations. The triangulation $\mathcal{T}_2$ is obtained via the flipping algorithm from $\mathcal{T}_1$, and the interpolation error is decreased after the flipping.

Since $\#\mathscr{T}_V$ is fixed, the algorithm will stop. In 2D, it will end with a Delaunay triangulation.

Similar algorithms of swapping faces, can be defined in 3D. However in 3D it could get stuck in cases where we would like to flip but we cannot. Extra effort is needed to resolve this non-transformable case, for example, by changing their local neighborhood [21, 33].

The running time of the edge flipping algorithm is $\mathcal{O}(N^2)$ and the worst case is possible (that is, flipping all edges in the initial triangulation to get a Delaunay triangulation). To speed up the algorithm, one can interleave flipping edges with adding points randomly. With a directed acyclic graph data structure, the running time of the randomized incremental algorithm can be reduced to $\mathcal{O}(N \log N)$ [44, 33].

3.2. Global method: the lifting trick

We now give a geometric explanation of Theorem 2.3 and present a global method for constructing Delaunay triangulations.

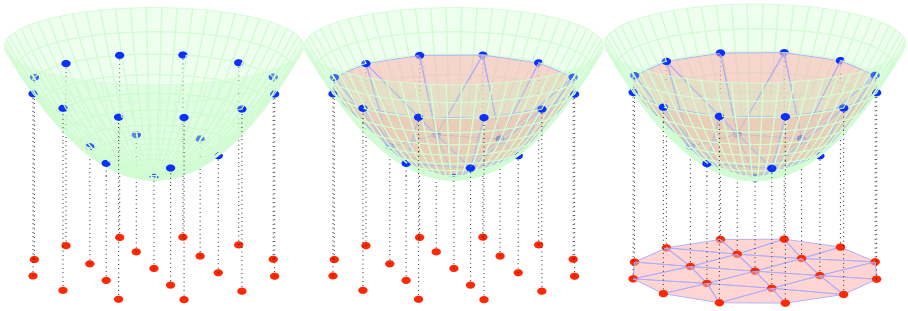
For a given point set V in $\mathbb{R}^d$, we have a set of points V' in $\mathbb{R}^{d+1}$ by lifting point in V to the paraboloid $x_{n+1} = \|\boldsymbol{x}\|^2$. The convex hull $CH(V')$ can be divided into *lower* and *upper* parts; a facet belongs to the lower convex hull if it is supported by a hyperplane that separates V' from $(\mathbf{0}, -\infty)$. We may assume the facets of the lower convex hull are simplices since if $n+2$ more vertices forms a facet, we can choose any

triangulation of this facet. The projection of a lower convex hull of V' in $\mathbb{R}^{d+1}$ is a DT of V in $\mathbb{R}^n$.

This is known as the *lifting trick* in the mesh generation community [45], and a widely used algorithm to construct Delaunay triangulations based on this approach is by using QHULL [16] to construct the convex hull. We classify it as a global method by assuming there is a fast algorithm to construct the convex hull of a points set. We present the algorithm below and refer to Fig. 2 for an illustration.

Algorithm: The Lifting Method

1. lift points in $\mathbb{R}^d$ to the graph of the paraboloid in $\mathbb{R}^{d+1}$;
2. construct convex hull of lifted points in $\mathbb{R}^{d+1}$;
3. project the lower convex hull back to $\mathbb{R}^d$.



(a) lift points to the paraboloid (b) construct a convex hull in $\mathbb{R}^3$ (c) project back to $\mathbb{R}^2$

Figure 2: Three steps of the lifting method to construct a Delaunay triangulation

As was the case earlier, the interpretation of the lifting trick from the function approximation point of view is more transparent. We construct a linear interpolant based on the lifted points on the paraboloid. The interpolation error in L^1 norm will be the volume of the region bounded between the graph of the paraboloid and the linear interpolation. Since the paraboloid is convex, the graph of the best linear interpolation will be the lower convex hull of these lifted points.

The global method is expensive when the goal is to improve the mesh quality, and not to construct Delaunay triangulation. For example, in 2D, the computational cost of Qhull is like $\mathcal{O}(N \log N)$ and in 3D it could be $\mathcal{O}(N^2)$, where N is the number of vertices. We can call it when the location of mesh points are changed dramatically. After the location of points is stabilized, we use the local flipping method to improve the connectivity.

4. Algorithms for Optimal Delaunay Triangulation

In this section, we present iterative methods for the construction of optimal Delaunay triangulation. We also classify our methods into local and global classes. The

local methods are relatively easy to implement but slow to converge for large number of vertices. The global method can significantly speed up the convergence rate and may save computation cost (provided one can invert a symmetric and positive definite matrix efficiently.)

Since now the connectivity of vertices is fixed, we simplify our energy notation as $E(p), p = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N), \boldsymbol{x}_i \in \mathbb{R}^d$, where $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N$ are all *interior nodes*. Here we assume the boundary nodes $b = (\boldsymbol{x}_{N+1}, \cdots, \boldsymbol{x}_{N+Nb}), \boldsymbol{x}_i \in \mathbb{R}^d$ are fixed and give a good discretization of $\partial\Omega$. When we refer to each vertex, we use $\boldsymbol{x}_i$ and when we treat all vertices as a set of points, we use a single letter p. We denote

$$\partial_i E := \frac{\partial E}{\partial \boldsymbol{x}_i} := (\frac{\partial E}{\partial x_i^1}, \cdots, \frac{\partial E}{\partial x_i^d}),$$

and $\nabla E = (\partial_1 E, \cdots, \partial_N E)^T$. Note that $\partial_i E$ is a vector in $\mathbb{R}^d$ and thus ∇E is a vector field defined on each vertex. The Hessian $\nabla^2 E$ in general should be a $dN \times dN$ matrix. But we will always approximate it by one $N \times N$ matrix for all d components.

4.1. Overview of Iterative Methods

Let p^k be the position of interior vertices in the kth-step. A general iteration method for solving $\min_p E(p)$ can be formulated as

$$p^{k+1} = p^k - B^{-1}\nabla E(p^k), \tag{4.1}$$

where $-\nabla E(p^k)$ is the direction in which the energy decreased in the most rapid rate, and B is an invertible matrix. We unify different methods as different choices of B.

4.1.1. Richardson-type method

A simple choice $B^{-1} = \alpha$ leads to the steepest descent method. The parameter α is called step size and can be found by the line search. This method can be also read as the forward Euler method with time step α for solving the gradient flow

$$\frac{\partial p}{\partial t} = -\nabla E(p).$$

4.1.2. Jacobi-type method

We use the diagonal information of the Hessian matrix, i.e. $B^{-1} = \alpha\,\mathrm{diag}(\nabla^2 E(p^k))^{-1}$. We write out the component-wise iteration in the following subroutine.

```
function p^{k+1} = JacobiMethod(p^k)
Compute ∇E(p^k) and ∂_{ii}E(p^k).
for i=1:N
    x_i^{k+1} = x_i^k − α∂_{ii}E(p^k)^{−1}∂_i E(p^k);
end
```

The step size α can be determined by the line search. For the simplicity and efficiency, a fixed step size $\alpha \in (0, 1]$ is usually used.

4.1.3. Gauss-Seidel type method

Another local relaxation scheme is to move one point $\boldsymbol{x}_i$ at a time. Most mesh smoothing schemes are in this form.

```
function p^{k+1} = GaussSeidelMethod(p^k)
for i=1:N
    Compute ∂_i E(p^k) and ∂_ii E(p^k);
    x_i^{k+1} = x_i^k − ∂_ii E(p^k)^{-1} ∂_i E(p^k);
end
```

The difference of `JacobiMethod` and `GaussSeidelMethod` (G-S) is that in the G-S method, ∇E and $\nabla^2 E$ are updated once a vertex is moved. When the energy is locally convex considering as a function of one vertex only, which is the case for our energy E, then the energy will be strictly decrease in the G-S method and thus ensure the converges of the iteration. In this sense, G-S is superior than Jacobi method. On the other hand, Jacobi method is computationally efficient since no update of $\nabla^2 E$ and E inside the `for` loop and the method is embarrassingly parallel.

4.1.4. Newton's method

We choose $B = \nabla^2 E(p^k)$ as the Hessian matrix of the energy to get the well known Newton's method. If $\nabla^2 E(p^k)$ is symmetric positive definite (SPD), it is well known that Newton's method will converge with a quadratic rate provided the initial guess is sufficiently close to the minimizer.

Richardson, Jacobi, and Gauss-Seidel type methods are classified as local methods since when updating $\boldsymbol{x}_i$ only information of the neighboring vertices of $\boldsymbol{x}_i$ is used (through the computation of $\partial_i E$). Newton's method is global since the inversion of the Hessian matrix will bring the local effect $\partial_i E$ for $\boldsymbol{x}_i$ to all vertices.

Local methods are frequently used in variational mesh adaptation for several reasons. First, it is easy to implement. Second, convergence can be easily ensured. For Gauss-Seidel method, the energy is decreasing if it is locally convex. For Richardson- and Jacobi-type methods, if we choose the step size α small enough, one can easily prove the corresponding mapping $T : p^k \mapsto p^{k+1}$ is a contraction and thus the scheme converges.

The drawback of local methods is the slow convergence. From a multi-scale point of view, local smoothing cannot carry the information in the coarse level to the fine level efficiently. The global method, such as Newton's method, is very desirable when the energy is globally convex (as a function of all vertices) and the initial guess is sufficiently close to a minimizer, due to the quadratic convergent rate. However, when the energy is non-convex and the initial guess is not close, the convergence conditions are very restrictive. Another complexity of Newton's method is the computation of the Hessian matrix and its inverse. In many cases, both the computation of $\nabla^2 E$ and $(\nabla^2 E)^{-1}$ are not easy or efficient. Later in the paper we propose a global method which strikes a good balance between the local and global methods.

4.2. Local mesh smoothing schemes

We first give formulas for $E, \nabla E$ and the diagonal of $\nabla^2 E$. We then modify the energy slightly to get approximated formula for ∇E. We derive formulas for uniform density and adapt to the non-uniform and piecewise constant density through numerical quadrature. Based on these formulas, we derive several local mesh smoothing schemes and develop techniques to address issues near the boundary.

4.2.1. ODT smoothing

Recall that p is the set of all interior nodes of a triangulation. For a vertex $\boldsymbol{x}_i \in p$, let ω_i be the star of $\boldsymbol{x}_i$, i.e., the set of all simplices containing $\boldsymbol{x}_i$, and φ_i the hat function of $\boldsymbol{x}_i$, i.e. a piecewise linear function with value one at $\boldsymbol{x}_i$ and zero at other vertices. Let $|\cdot|$ denote the Lebesgue measure in $\mathbb{R}^d$.

Lemma 4.1. *For uniform density $\rho = 1$,*

$$E(p) = \frac{1}{d+1} \sum_{i=1}^{N} \boldsymbol{x}_i^2 |\omega_i| - \int_{\Omega} \|\boldsymbol{x}\|^2 d\mathbf{x}. \tag{4.2}$$

Proof. Since $u(\boldsymbol{x}) = \boldsymbol{x}^2$ is convex, $|u_I - u| = u_I - u = \sum_{i=1}^{N} u(\boldsymbol{x}_i)\varphi_i - u$. Then

$$\int_{\Omega} |u - u_I| \, d\mathbf{x} = \int_{\Omega} u_I \, d\mathbf{x} - \int_{\Omega} \|\boldsymbol{x}\|^2 \, d\mathbf{x} = \sum_{i=1}^{N} u(\boldsymbol{x}_i) \int_{\Omega} \varphi_i \, d\mathbf{x} - \int_{\Omega} \|\boldsymbol{x}\|^2 d\mathbf{x}.$$

The desired results then follows from the integral formula for φ_i. □

We then compute the gradient ∇E using this formula.

Lemma 4.2. *Let $\boldsymbol{x}_i$ be an interior node. For uniform density $\rho = 1$, one has*

$$\partial_i E(\boldsymbol{x}_i) = \frac{1}{d+1} \Big[2\boldsymbol{x}_i |\omega_i| + \sum_{\tau_j \in \omega_i} \sum_{\boldsymbol{x}_k \in \tau_j, \boldsymbol{x}_k \neq \boldsymbol{x}_i} \|\boldsymbol{x}_k\|^2 \nabla_{\boldsymbol{x}_i} |\tau_j| \Big], \tag{4.3}$$

$$\partial_i E(\boldsymbol{x}_i) = \frac{1}{d+1} \sum_{\tau_j \in \omega_i} \sum_{\boldsymbol{x}_k \in \tau_j, \boldsymbol{x}_k \neq \boldsymbol{x}_i} \|\boldsymbol{x}_k - \boldsymbol{x}_i\|^2 \nabla_{\boldsymbol{x}_i} |\tau_j|, \tag{4.4}$$

$$\partial_i E(\boldsymbol{x}_i) = \frac{2}{d+1} \sum_{\tau_j \in \omega_i} (\boldsymbol{x}_i - \boldsymbol{c}_j) |\tau_j|, \tag{4.5}$$

where $\mathbf{c}_j$ is the center of the circum-sphere of τ_j.

Proof. When $\boldsymbol{x}_i$ is moved inside ω_i, the domain formed by ω_i does not change. But it will change the star ω_j for other vertices $\boldsymbol{x}_j$ connected with $\boldsymbol{x}_i$. We thus need to include the constant $\nabla_{\boldsymbol{x}_i} |\tau_j|$ in an appropriate way. Formula (4.3) immediately follows from the differentiation of formula (4.2) and the fact $|\omega_i|$ does not depend on $\boldsymbol{x}_i$.

To prove (4.4), we note that the interpolation error depends only on the quadratic part of the approximated function. More specifically, if we change the function $u(\boldsymbol{x}) = \boldsymbol{x}^2$ to $v(\boldsymbol{x}) := (\boldsymbol{x} - \boldsymbol{x}_i)^2$, we have $u - u_I = v - v_I$. The second formula is then obtained by using v in the first formula.

To prove (4.5), we need some preparation. First we give more explanation on the constant $\nabla_{\boldsymbol{x}_i}|\tau_j|$. Let us use $\boldsymbol{x}$ to denote the free node $\boldsymbol{x}_i$. When $\boldsymbol{x}$ is moving around in ω_i, the volume $|\tau_j(\boldsymbol{x})|$ is a linear function of $\boldsymbol{x}$. Let F^j be the face opposite to $\boldsymbol{x}_i$ in τ_j. Then obviously $|\tau_j(\boldsymbol{x})| = 0$ for $\boldsymbol{x} \in F^j$, i.e. F^j is part of the zero level set of linear function $|\tau_j(\boldsymbol{x})|$. Therefore the vector

$$f_{F^j} := \sum_{\boldsymbol{x}_k \in \tau_j, \boldsymbol{x}_k \neq \boldsymbol{x}_i} \|\boldsymbol{x}_k\|^2 \nabla_{\boldsymbol{x}_i}|\tau_j|$$

is a normal vector of F^j pointing to $\boldsymbol{x}_i$ with magnitude depending on F^j only. Physically f_{F^j} can be interpreted as a force acting on $\boldsymbol{x}_i$ from the "wall" F^j. When F^j is an interior face sharing by two simplices, the vector $-f_{F^j}$ will be the force acting on the other vertex opposite to F^j.

Then we derive formulae for the circumcenter of a simplex. Since $|\omega_i| = \sum_{j\in\omega_i} |\tau_j|$ does not depend on $\boldsymbol{x}_i$, we have $\sum_{j\in\omega_i} \nabla_{\boldsymbol{x}_i}|\tau_j| = \nabla_{\boldsymbol{x}_i}|\omega_i| = 0$. By (4.4), if the neighboring vertices $\boldsymbol{x}_k$ lie on the same sphere with center $\boldsymbol{x}_i$ and radius R, then

$$\partial_i E(\boldsymbol{x}_i) = \frac{d}{d+1} R^2 \sum_{j\in\omega_i} \nabla_{\boldsymbol{x}_i}|\tau_j| = 0,$$

and consequently the optimal location is the circumcenter. In particular, when the star is a simplex, the minimal point is the circumcenter of this simplex. By (4.3), we thus obtain a formula for the circumcenter in terms of the force f_F:

$$2|\tau_j|c_j = -\sum_{F\in\tau_j} f_F. \tag{4.6}$$

We now add artificial forces f_F and $-f_F$ for each interior face F of the star ω_i. Then the second term in (4.3) can be written as

$$\sum_{F\in\partial\omega_i} f_F = \sum_{\tau_j\in\omega_i} \sum_{F\in\tau_j} f_F = -2 \sum_{\tau_j\in\omega_i} |\tau_j| c_j.$$

Formula (4.5) is then obtained by writing the first term $2\boldsymbol{x}_i|\omega_i| = 2\sum_{\tau_j\in\omega_i} |\tau_j|\boldsymbol{x}_i$. □

Formula (4.3) is derived in Chen [1] and (4.4) can be found in Chen [3]. The simplified version (4.5) is firstly derived in Alliez et. al. [24]. From (4.5), we conclude that for an ODT, $\nabla E = 0$, and therefore

$$\boldsymbol{x}_i = \sum_{j\in\omega_i} \frac{|\tau_j|}{|\omega_i|} \boldsymbol{c}_j. \tag{4.7}$$

Namely for an ODT, each interior vertex $\boldsymbol{x}_i$ is a weighted centroid of circumcenters of simplices in the star of $\boldsymbol{x}_i$.

Next we compute the second derivative of the energy.

Lemma 4.3. *For uniform density $\rho = 1$, one has*

$$\partial_{ii} E(\boldsymbol{x}_i) = \frac{2}{d+1} |\omega_i|. \tag{4.8}$$

Proof. Note that both $\nabla_{\boldsymbol{x}_i}|\tau_j|$ and $|\omega_i|$ do not depends on $\boldsymbol{x}_i$. Then (4.8) comes from the differentiation of (4.3) with respect to $\boldsymbol{x}_i$. □

The error formula (4.2) and the gradient formula (4.3) can be easily generalized to convex functions u by simply replacing $\|\boldsymbol{x}_k\|^2$ by $u(\boldsymbol{x}_k)$ and $2\boldsymbol{x}_i$ by $\nabla u(\boldsymbol{x}_i)$; see [2]. This generalization is useful when the convex function is available, which might be the case for the numerical solution of partial differential equations (PDEs). When applying to mesh optimization, it is hard to find such a convex function. Therefore we switch to a density-based generalization.

Lemma 4.4. *For general density ρ,*

$$E(p) = \sum_{i=1}^{N} \boldsymbol{x}_i^2 |\omega_i|_{\rho\varphi_i} - \int_{\Omega} \|\boldsymbol{x}\|^2 \rho(\boldsymbol{x}) d\mathbf{x}, \tag{4.9}$$

$$\partial_i E(\boldsymbol{x}_i) \approx 2\boldsymbol{x}_i |\omega_i|_{\rho\varphi_i} + \sum_{\tau_j \in \omega_i} \sum_{\boldsymbol{x}_k \in \tau_j, \boldsymbol{x}_k \neq \boldsymbol{x}_i} \|\boldsymbol{x}_k\|^2 \nabla_{\boldsymbol{x}_i} |\tau_j|_{\rho\varphi_i}, \tag{4.10}$$

$$\partial_i E(\boldsymbol{x}_i) \approx \frac{2}{d+2} \sum_{\tau_j \in \omega_i} (\boldsymbol{x}_i - \boldsymbol{c}_j) \rho_{\tau_j} |\tau_j|. \tag{4.11}$$

Proof. The formula (4.9) is proved as before. The formula (4.10) only gives an approximation to $\partial_i E$. The term $\int_{\omega_i} \rho \partial_i \varphi_i \, d\mathbf{x}$ is skipped. The practical formula (4.11) is obtained by one point quadrature $|\tau_j|_{\rho\varphi_i} \approx \rho_{\tau_j} |\tau_j|/(d+1)$ and the same argument as before. □

Formula (4.11) will be the one we used in this paper to compute ∇E for general density. It retains its nice geometric meaning by only changing of the weight of circumcenters. Meanwhile it is a good approximation of the true gradient when ρ varies smoothly (more precisely $|\nabla \rho| \leq C$). The formula (4.10) is presented for possibly high-order approximations of the gradient. For example, we could use ρ_I to replace ρ and compute the weighted volume using middle points of edges (which is exact in two dimensions). Suppose $\boldsymbol{x}_i$, $\boldsymbol{x}_j$, and $\boldsymbol{x}_k$ are three vertices of the triangle τ_j; we could then use quadrature

$$|\tau_j|_{\rho\varphi_i} \approx \frac{|\tau|}{6} (2\rho(\boldsymbol{x}_i) + \rho(\boldsymbol{x}_j) + \rho(\boldsymbol{x}_k))$$

and consequently a better approximation of the gradient

$$\nabla_{\boldsymbol{x}_i} |\tau_j|_{\rho\varphi_i} \approx \frac{1}{3} \left[\rho(\boldsymbol{x}_i) \nabla_{\boldsymbol{x}_i} |\tau_j| + \nabla \rho(\boldsymbol{x}_i) |\tau_j| \right].$$

Thanks to these formulas, we now derive a local mesh smoothing scheme by considering the minimization of the energy as a function of $\boldsymbol{x}_i$ only. Namely we fix all other vertices and consider the optimization problem

$$\min_{\boldsymbol{x}_i} E(\boldsymbol{x}_i). \tag{4.12}$$

We use (4.11) to get a ODT based scheme

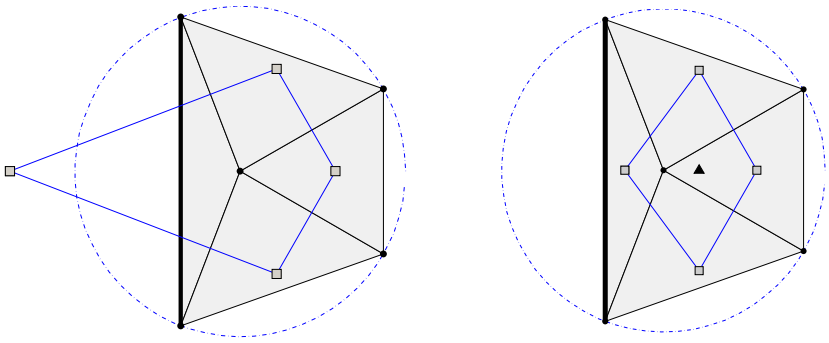
$$\boldsymbol{x}_i^{new} = (1-\alpha)\boldsymbol{x}_i + \alpha \sum_{\tau_j \in \omega_i} \frac{|\tau_j|_{\rho_\tau}}{|\omega_i|_{\rho_\tau}} \boldsymbol{c}_j. \tag{4.13}$$

When $\alpha = 1$, the geometric explanation is first computing circumcenters of simplices in the star, and then move the point to the weighted center of the polytope formed by these circumcenters. The step size α is incorporated to avoid the folding of simplices as explained below. The initial setting is $\alpha = 1$. If moving $\boldsymbol{x}_i$ to $\boldsymbol{x}_i^{new}$ results a non-valid triangulation (some simplices could have negative signed volume), we then reduce α by half. That is we use $\alpha = 1, 1/2, 1/4, \cdots$ as a simple line search strategy.

When near the boundary, the ODT smoothing could result squashed/stretched elements. For example, in Fig. 3 (a), since the neighboring vertices lie on a circle, the weighted average of circumcenters is the center of this circle, which results a stretched triangle near the boundary. If this edge is inside the domain, the stretched triangle will disappear after one or two edge flipping. But for a boundary edge, flipping is not possible.

The reason for this problem near boundary is that the object function in our minimization problem is the interpolation error which is not directly related to geometric mesh quality such as aspect ratio. It is well known that optimization of location of vertices may result well distributed points which is not necessarily lead to meshes with good aspect ratio [46]. Obtuse triangles are examples in 2D and similar examples in 3D are known as silvers . Optimization of topology, i.e., the connectivity of points is necessary to further improve the aspect ratio, which is possible for interior parts but may fail for boundary parts.

Ideally we should change the object function near the boundary. Commonly used object function include: minimum/maximum angle [22, 23, 47], aspect ratio [48, 17], or distortion metrics [19].



(a) circumenters (b) modified centers

Figure 3: For boundary elements, we use barycenters not circumcenters. The bold line is the boundary of the domain. The square dots are centers of triangles. In (a), since the neighboring vertices lie on a circle, the weighted average of centers is the center of this circle, which results a stretched triangle near the boundary. In (b), we change the circumcenter of the triangle near the boundary to the barycenter. Then the weighted average of modified centers, the triangle dot, is moved away from the boundary.

To keep the simple form and geometric interpretation of our mesh smoothing scheme, we simply modify the center for the squashed element close to the boundary following the rule:

For boundary elements, use the barycenter.

Here boundary elements are defined as simplexes containing at least one node on the boundary. For the implementation, one can record a logical array `isBdNode` to indicate which node is on the boundary and boundary elements can be easily found by

```
% 2D
isBdElem = isBdNode(t(:,1)) | isBdNode(t(:,2)) | isBdNode(t(:,3));
% 3D
isBdElem = isBdNode(t(:,1)) | isBdNode(t(:,2)) | ...
           isBdNode(t(:,3)) | isBdNode(t(:,4));
```

With such a modification, the polytope formed by the centers is still inside the domain. When stretched elements appear near the boundary, their volume is small. Therefore, the contribution of the corresponding center to the new location is not major, as the other centers will attract the points back; see Fig. 3 (b).

4.2.2. CPT smoothing

We present another formula for our energy. The proof is given in [1]. For completeness, we include it here.

Lemma 4.5. *For uniform density $\rho = 1$, one has*

$$E(p) = \frac{1}{d+1}\sum_{i=1}^{N}\int_{\omega_i}\|\boldsymbol{x}-\boldsymbol{x}_i\|^2 d\mathbf{x}. \tag{4.14}$$

Proof. Recall that $\{\lambda_i(\boldsymbol{x})\}_{i=1}^{d+1}$ is the barycenter coordinate of $\boldsymbol{x}$ in the simplex τ. Then $\boldsymbol{x} = \sum_{i=1}^{d+1}\lambda_i\boldsymbol{x}_i$ and

$$\begin{aligned}
\sum_{k=1}^{d+1}\int_{\tau}\|\boldsymbol{x}-\boldsymbol{x}_k\|^2 &= \sum_{i,j,k=1}^{d+1}\int_{\tau}\lambda_i\lambda_j(\boldsymbol{x}_i-\boldsymbol{x}_k)^t(\boldsymbol{x}_j-\boldsymbol{x}_k)\\
&= \frac{|\tau|}{(d+2)(d+1)}\sum_{i,j,k=1}^{d+1}(\boldsymbol{x}_i-\boldsymbol{x}_k)^t(\boldsymbol{x}_j-\boldsymbol{x}_k)\\
&= \frac{d+1}{2}\frac{|\tau|}{(d+2)(d+1)}\sum_{i,j=1}^{d+1}\|\boldsymbol{x}_i-\boldsymbol{x}_j\|^2\\
&= (d+1)\int_{\tau}(u_I-u)(\boldsymbol{x})d\mathbf{x}.
\end{aligned}$$

The last equality follows from Lemma 2.7. The third one is obtained by summing up the following basic identity for $i,j,k = 1,\cdots,d+1$:

$$\|\boldsymbol{x}_i-\boldsymbol{x}_j\|^2 = \|\boldsymbol{x}_i-\boldsymbol{x}_k\|^2 + \|\boldsymbol{x}_j-\boldsymbol{x}_k\|^2 - 2(\boldsymbol{x}_i-\boldsymbol{x}_k)^t(\boldsymbol{x}_j-\boldsymbol{x}_k).$$

By rearranging the summation from element-wise to vertex-wise, we obtain

$$\int_{\Omega}(u_I-u)(\boldsymbol{x})d\mathbf{x} = \frac{1}{d+1}\sum_{i=1}^{N_T}\sum_{k=1}^{d+1}\int_{\tau_i}\|\boldsymbol{x}-\boldsymbol{x}_{\tau,k}\|^2 = \frac{1}{d+1}\sum_{i=1}^{N}\int_{\omega_i}\|\boldsymbol{x}-\boldsymbol{x}_i\|^2.$$

Here N_T is the number of elements in the triangulation. □

The formula (4.14) motives a slightly different energy for general density function ρ defined as:

$$\tilde{E}(p) = \frac{1}{d+1}\sum_{i=1}^{N}\int_{\omega_i} \|\boldsymbol{x}-\boldsymbol{x}_i\|^2\rho(\boldsymbol{x})d\boldsymbol{x}. \tag{4.15}$$

When ρ is piecewise constant on each simplex, $E = \tilde{E}$ in views of Lemma 4.5. Otherwise, $\tilde{E}$ can be thought as a high-order perturbation of E.

We shall derive an approximation of the derivative $\partial_i\tilde{E}$ by considering the following 1D optimization problem:

$$\min_{\boldsymbol{x}_i} \tilde{E}_i, \quad \text{where } \tilde{E}_i = \frac{1}{d+1}\int_{\omega_i} \|\boldsymbol{x}-\boldsymbol{x}_i\|^2\rho(\boldsymbol{x})\,d\mathbf{x}. \tag{4.16}$$

Since $\cup_i\omega_i$ is an overlapping decomposition of Ω, the change of ω_i will affect other patches and thus the overall energy will not necessarily be reduced by solving this local optimization problem. Nevertheless it will result an efficient local mesh smoothing scheme.

Again since the domain formed by ω_i does not depends on $\boldsymbol{x}_i$, differentiation of the energy $\tilde{E}_i$ gives the following result.

Lemma 4.6. *For general density ρ, one has*

$$\partial_i\tilde{E}_i = \frac{2}{d+1}\Big[\boldsymbol{x}_i\int_{\omega_i}\rho(\boldsymbol{x})d\mathbf{x} - \int_{\omega_i}\boldsymbol{x}\rho(\boldsymbol{x})\,d\mathbf{x}\Big] \tag{4.17}$$

and therefore the minimizer of (4.16) is the centroid of ω_i with respect to the density $\rho(\boldsymbol{x})$, namely

$$\boldsymbol{x}_i^* = \frac{\int_{\omega_i}\boldsymbol{x}\rho(\boldsymbol{x})d\mathbf{x}}{\int_{\omega_i}\rho(\boldsymbol{x})d\mathbf{x}}. \tag{4.18}$$

Definition 4.7. *For a triangulation $\mathcal{T}$, if for any vertex $\boldsymbol{x}_i \in \mathcal{T}$, $\boldsymbol{x}_i$ is also the centroid of its patch ω_i with respect to the density ρ, we call it* Centroidal Patch Triangulation *(CPT) with respect to the density ρ.*

Remark 4.8. This mimics the definition of Centroidal Voronoi Tessellations in [14] for which the generator and centroid of each Voronoi region coincide. For various application of CVT to the mesh generation and numerical solution of PDEs, we refer to [14, 13, 49, 50, 51, 52, 53]. Mesh smoothing based on CVT in Du and Gunzburger [13] is the mostly closely related work.

We use one point numerical quadrature to evaluate the integral, i.e. $\int_\tau \boldsymbol{x}\rho_i(\boldsymbol{x})\,d\mathbf{x} \approx \boldsymbol{b}_\tau\rho_\tau|\tau|$ where $\boldsymbol{b}_\tau$ is the barycenter of τ and $\rho_\tau = \rho(\boldsymbol{b}_\tau)$. The quadrature is exact for uniform density $\rho = 1$. We then get an approximation of the gradient as

$$\partial_i E(\boldsymbol{x}_i) \approx \partial_i\tilde{E}_i = \frac{2}{d+1}\sum_{\tau_j\in\omega_i}(\boldsymbol{x}_i-\boldsymbol{b}_j)|\tau_j|_{\rho_{\tau_j}}, \tag{4.19}$$

where $\boldsymbol{b}_j$ is the barycenter of simplex τ_j. Note that $\partial_i E(\boldsymbol{x}_i) = \sum_{x_j \in \omega_i} \partial_i E_j$. We skip the contribution from other patches to get an approximation which is more computationally efficient.

We thus get a CPT-based mesh smoothing scheme

$$\boldsymbol{x}_i^{new} = (1-\alpha)\boldsymbol{x}_i + \alpha \sum_{\tau_j \in \omega_i} \frac{|\tau_j|_{\rho_\tau}}{|\omega_i|_{\rho_\tau}} \boldsymbol{b}_j. \tag{4.20}$$

What is the difference between ODT and CPT smoothing? In CPT (4.20) we use the barycenter while in ODT (4.13) the circumcenter. There are several advantages to using barycenters over circumcenters:

- The barycenter (an averaging of coordinates of vertices) is easy to compute while the computation of circumcenter is relatively costly; see the formula (4.6), for example.
- Barycenters will be always inside the simplex. Therefore moving to the average of the barycenters without changing the connectivity will still result a valid triangulation.
- No modification is needed near the boundary.

On the other hand, we should be aware that barycenter-based CPT smoothing only minimizes an approximation of the energy. Numerical examples in Section 5 show that ODT is slightly better than CPT.

What is the right choice of the density function ρ? It could be given *a priori*. Namely, the density is given by the user according to *a priori* information. For example, we choose $\rho = 1$ for uniform meshes and non-constant ρ for graded meshes. In practice, especially when solving partial differential equations, the density could be given by *a posteriori* error estimate, which of course is problem dependent. In application to mesh optimization, ρ can be chosen as the curvature of the boundary curves or as distance to the boundary or other feature, with high density near the feature of interest. The density ρ can be also estimated from the current triangulation. We will discuss one of such choice in the next.

4.2.3. Laplacian smoothing

Laplacian smoothing [54], in its simplest form, involves moving each vertex to the arithmetic average of the neighboring points.

$$\boldsymbol{x}_i^{new} = \frac{1}{k} \sum_{\boldsymbol{x}_j \in \omega_i, \boldsymbol{x}_j \neq \boldsymbol{x}_i} \boldsymbol{x}_j, \tag{4.21}$$

where k is the number of vertices of ω_i. Laplacian smoothing is easy to implement and requires very low computational cost, but it operates heuristically and does not guarantee an improvement in the geometric mesh quality. We now derive a variant of Laplacian smoothing from CPT smoothing by a special choice of the density.

When the mesh is obtained by local refinement driven by error indicators, it is reasonable to assume that local refinement will equidistribute the product $\rho|\tau|$. Namely

we could take $\rho_\tau = C/|\tau|$, where C is chosen to normalize the density such that $\sum_\tau \rho_\tau = 1$. If we chose $\rho_\tau = C/|\tau|$ and $\alpha = 3/2$ in formula (4.20) in 2D, we obtain Laplacian smoothing, i.e. Laplacian smoothing is an *over-relaxation* of CPT smoothing. We emphasis that in 3D, no such a relation exists. For a vertex $\boldsymbol{x}_j$ of ω_i and $\boldsymbol{x}_j \neq \boldsymbol{x}_i$, the number of simplices containing the edge formed by $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in ω_i is not fixed while in two dimensions, this number is always two since $\boldsymbol{x}_i$ is an interior vertex.

From the discussion above, we see Laplacian smoothing will preserve the current mesh density. Therefore if triangles in the current mesh are well shaped, Laplacian smoothing will be more efficient (without computing the volume). But even in this case, i.e., to keep the current mesh density, we recommend using CPT or ODT smoothing (with choices $\rho = 1/|\tau|, \alpha = 1$)

$$\boldsymbol{x}_i^{new} = \frac{1}{k} \sum_{\tau_j \in \omega_i} \boldsymbol{b}_j, \quad \text{or} \quad \boldsymbol{x}_i^{new} = \frac{1}{k} \sum_{\tau_j \in \omega_i} \boldsymbol{c}_j, \tag{4.22}$$

where, recall that, $\boldsymbol{b}_j$ is barycenter and $\boldsymbol{c}_j$ is circumcenter.

4.3. Global Mesh Optimization Scheme

Writing $\boldsymbol{b}_\tau$ as combination of vertices, i.e. $\boldsymbol{b}_\tau = \sum_{k=1}^{d+1} \boldsymbol{x}_k/(d+1)$ in (4.19), we obtain another form of the approximated gradient

$$\partial_i \tilde{E}_i = \sum_{\tau \in \omega_i} \sum_{\boldsymbol{x}_k \in \tau} \frac{2|\tau|_{\rho_\tau}}{(d+1)^2} (\boldsymbol{x}_i - \boldsymbol{x}_k). \tag{4.23}$$

Therefore we can approximate ∇E by a nonlinear diffusion system. More specifically, let $\bar{p} = p \cup b$ be the set of all vertices and p_τ the set of vertices of τ. We define a matrix $\bar{A}(\bar{p})$ such that

$$\bar{A}(\bar{p})\bar{p} = \sum_{\tau \in \mathcal{T}} A_\tau p_\tau, \quad \text{with } A_\tau = \frac{2|\tau|_{\rho_\tau}}{d(d+1)} \begin{pmatrix} d & -1 & \dots & -1 \\ -1 & d & \dots & -1 \\ \dots & \dots & \dots & \dots \\ -1 & -1 & \dots & d \end{pmatrix}. \tag{4.24}$$

We then define $A = \bar{A}(1:N, 1:N)$ as the submatrix of $\bar{A}$ restricted to interior vertices. From the element-wise relation, it is not hard to write out the matrix A explicitly. Let $\boldsymbol{x}_i, i = 1, \cdots, N + Nb$ be N interior points and Nb boundary points. We construct the $(N + Nb) \times (N + Nb)$ matrix $\bar{A}$ by

$$a_{i,j} = -\frac{2}{d(d+1)} \sum_{\tau_k \in \omega_i \cap \omega_j} |\tau_k|_\rho \quad \text{for } i \neq j, i, j = 1 : N + Nb \tag{4.25}$$

$$\text{and} \quad a_{i,i} = \sum_{j \neq i} -a_{i,j} \quad \text{for } i = 1 : N + Nb. \tag{4.26}$$

The scaling factor $2/(d(d+1))$ is chosen such that $a_{ii} = \partial_{ii} E$ when the density is uniform, i.e., $\rho = 1$.

We choose A as an approximation of $\nabla^2 E$ to obtain an ODT-based global mesh optimization scheme

$$p^{k+1} = p^k - \alpha A^{-1} \nabla E(p^k), \tag{4.27}$$

or CPT-based global mesh optimization scheme by using the approximated gradient

$$p^{k+1} = p^k - \alpha A^{-1} \tilde{\nabla} \tilde{E}(p^k). \tag{4.28}$$

It is easy to see the matrix A is an SPD and M-matrix. These nice properties enable us to solve the resulting algebraic system efficiently, e.g. using algebraic multigrid methods (AMG). See [55] on the nearly optimal complexity analysis for AMG for an SPD and M-matrix on general unstructured grids.

Why not solve Euler-Lagrange type equation $\nabla \tilde{E} = 0$ directly? Let us write $A_{pb} = \bar{A}(1:N,\ N+1:N+Nb)$. Then in view of (4.23), we have

$$\tilde{\nabla} \tilde{E}(p) \approx \frac{d}{d+1}(Ap + A_{pb}b).$$

Therefore $\nabla \tilde{E} = 0$ becomes a diffusion equation with Dirichlet boundary condition:

$$A(\bar{p})p = -A_{pb}(\bar{p})b. \tag{4.29}$$

Note that this is a non-linear equation since the matrix $A(\bar{p})$ depends on p also. The simplest fixed-point iteration is $p^{k+1} = -A^{-1}(\bar{p}^k)A_{pb}(p^k)b$ may not converge. The iteration (4.28) can be written as

$$p^{k+1} = p^k - \alpha A^{-1}\frac{d}{d+1}(Ap^k + A_{pb}b) = (1 - \frac{\alpha d}{d+1})p^k - \frac{\alpha d}{d+1}A^{-1}A_{pb}b,$$

which is a damped fixed-point iteration.

We can construct more efficient mesh optimization schemes by using advanced numerical algorithms for solving the nonlinear diffusion equation (4.29). One such algorithm is the two-grid method developed in [56]. Let us use subscript H and h to denote meshes with different scales and assume $h \ll H$. The mesh $\mathcal{T}_H$ is called the coarse mesh and $\mathcal{T}_h$ is the fine mesh. To solve the nonlinear equation on the fine mesh $\mathcal{T}_h$, we first solve it on the coarse mesh to get an accurate approximation p_H and then start the fixed-point iteration with p_H. The computational cost of the nonlinear problem on the coarse mesh is usually small.

In our setting, we will replace the coarse grid solver by any robust (not necessarily efficient) mesh generator, which also solves the deficiency of fixing boundary nodes in our mesh improvement schemes. We then uniformly refine the coarse mesh several times and apply our local and global methods to further improve the mesh; see `odtmesh2d` in Section §5.3.

5. Numerical Examples

In this section, we present several numerical examples to show the efficiency of our mesh smoothing and optimization schemes. We first present local mesh smoothing

schemes of Gauss-Seidel type for both uniform density and non-uniform density cases. We then give an example to show fast convergence of our new global method compared with local methods. Next we combine a robust mesh generator, distmesh [12], with our efficient mesh optimization methods to obtain a robust and efficient mesh generator in 2D. Last we present examples on local mesh smoothing schemes for 3D meshes.

To measure the mesh quality, we use the ratio of radius of inscribed and circumscribed circles. In 2D, we have the following formula

$$q(\tau) = 2\frac{r_{\text{in}}}{r_{\text{out}}} = \frac{(b+c-a)(c+a-b)(a+b-c)}{abc}, \tag{5.1}$$

where r_{in} and r_{out} are radius of inscribed and circumscribed circles of the triangle τ, and a, b, c are the side lengths. The idea equilateral triangle has $q = 1$, and the smaller the q is, the worse the aspect ratio of τ. As indicated in [12], if all triangles have $q > 0.5$, the results are good. We calculate the minimal and mean value of the mesh quality and sometimes show its histogram.

We also include a measure of uniformity [12]: the standard deviation of the ratio of actual sizes to desired sizes specified by a function h. That number is normalized to measure the relative sizes. The smaller the value of the uniformity, the more uniform (with respect to h) is the point distribution.

5.1. Local mesh smoothing

For completeness, we present a Gauss-Seidel type mesh smoothing and edge flipping algorithms for a 2D triangulation below.

```
for k = 1:step
    % mesh smoothing
    for i = 1:N
        if x* is interior
            x* = smoother(x_i, ω_i, ρ);
        end
        if x* is acceptable
            x_i = x*;
        end
    end
    % edges flipping
    for e = 1:ne
        if e is non locally Delaunay
            find the convex quadrilateral sharing e;
            flip the diagonals;
        end
    end
end
```

Recall that ω_i is the triangles attached to the vertex $\boldsymbol{x}_i$ and ρ is a density function specified by users. Here $\boldsymbol{x}^*$ is acceptable means by replacing $\boldsymbol{x}_i$ by $\boldsymbol{x}^*$, the resulting mesh is still valid, i.e. all signed area is positive.

We incorporate the edge flipping after one step of mesh smoothing since it can change the topological structure of the mesh to further minimize the energy – these are two sub-problems associated with the minimization of our energy. However, we do not

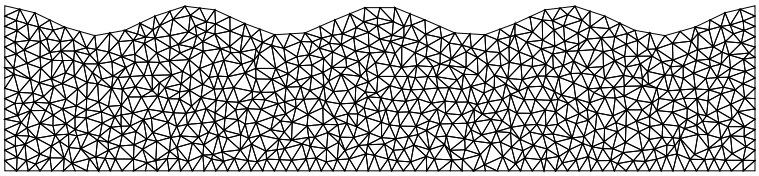
repeat edge flipping until the mesh is Delaunay. Instead we only perform one loop for all edges and flip non Delaunay edges.

There are several variants of the above subroutine. One can perform two or more iterations of smoothing loops and followed by one loop of edge flipping. To further save computational cost, inside each mesh smoothing or edge flipping loop, one can restrict the operation to triangles or edges with bad respect ratio.

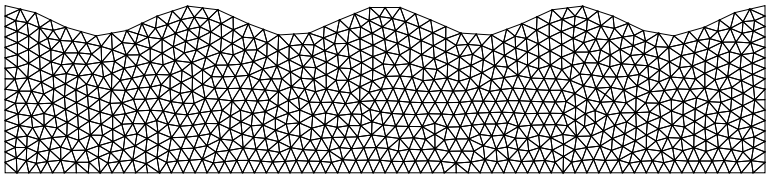
5.1.1. Uniform density

The goal of the mesh smoothing is to get a mesh consisting of equilateral triangles with equal areas. We use CPT (4.20) and ODT (4.13) smoothing with step size $\alpha = 1$ and uniform density $\rho = 1$, and show the reduction of the interpolation error.

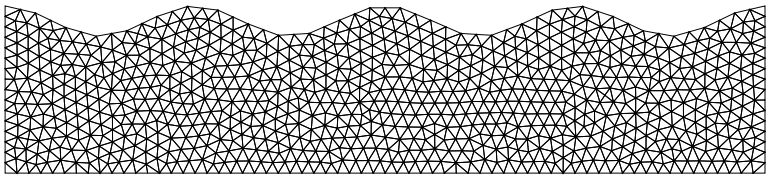
We use a mesh for a wavy channel displayed in Fig. 4(a). The nodes inside the domain are perturbed randomly while the nodes on the boundary is equally spaced since only interior nodes are allowed to move. We perform ten iterations and present



(a) Original mesh: min(q) = 0.141, mean(q) = 0.875, uniformity = 15.62%



(b) ODT smoothing: min(q) = 0.709, mean(q) = 0.964, uniformity = 5.16%



(c) CPT smoothing: min(q) = 0.708, mean(q) = 0.967, uniformity = 6.15%

Figure 4: Comparison of meshes obtained by ODT and CPT smoother for 10 iterations.

meshes obtained by different smoothers in Fig. 4. Both ODT and CPT result in much better meshes than the initial mesh; see Fig. 4(b) and 4(c). The mesh quality obtained by ODT and CPT smoothers are almost the same. However ODT results in a better uniformity than CPT.

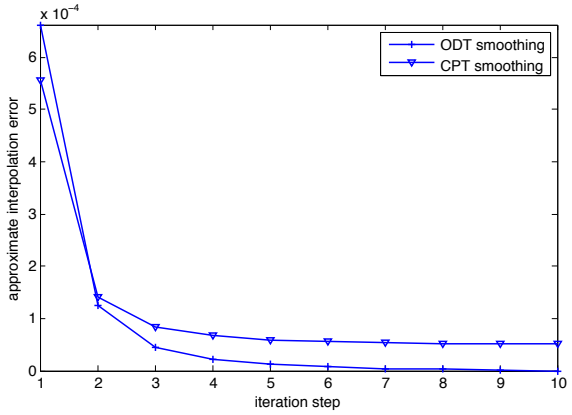


Figure 5: Error comparison of ODT and CPT smoother.

In Fig. 5, we plot the energy shifted by a constant. Let E_k^1, E_k^2 denote the integral $\int_\Omega u_I(\boldsymbol{x})d\mathbf{x}$ on the mesh obtained by k-th iteration of ODT or CPT smoothing, respectively. Ideally the energy will be obtained by subtracting $\int_\Omega \|\boldsymbol{x}\|^2 \, d\mathbf{x}$ from E_k; see the formula in Lemma 4.1. Instead we compute approximate energy by $E_k^1 - E_{10}^1$ and $E_k^2 - E_{10}^1$. Fig. 5 clearly shows the reduction of this approximated energy after each iteration. The ODT smoother is better in reducing the energy since it has a provable reduction property. The numerical convergence of the interpolation errors for those smoothers is very clear from this picture. A Gauss-Seidel type convergence history is also evident. Namely, the energy is decreased most rapidly in the first few steps and then the curve becomes flat.

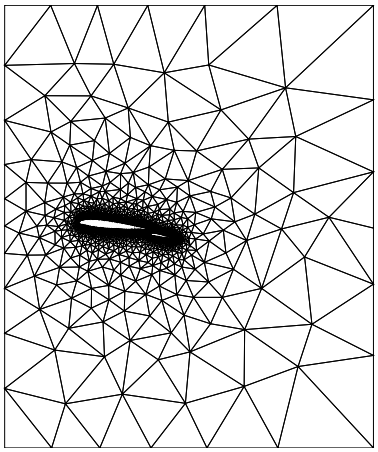
5.1.2. Non-uniform density

In practice, to resolve the complex geometry or to resolve the singularity of the solution to some PDEs, it is advantageous to have different sizes in different regions. The mesh size will be controlled by the density function. We shall show our mesh smoothing schemes are also effective for non-uniform density functions.

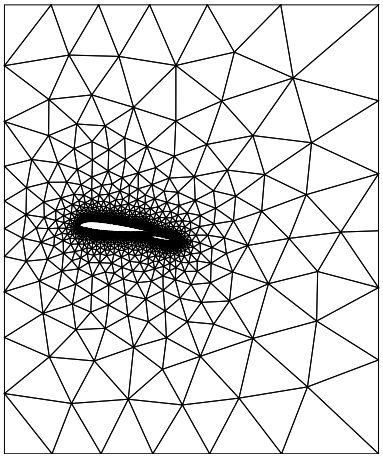
For highly curved boundaries, the curvature or the distance to the boundary may be used in the formulation of ρ. For functions with singularity, a posteriori or a priori error estimators can serve in the role of density function. Usually we assume the density ρ is specified by the user and we will present relevant examples later.

There are some cases for which the user may not know the density function a priori. In other words, the user provides a mesh as an input, and wants an output mesh with better mesh quality while keeping similar mesh density. In this case, we simply choose $\rho = C/|\tau|$ in ODT or CPT smoothers and thus use the formulae (4.22). The constant C is not important since it will cancel out in the normalization.

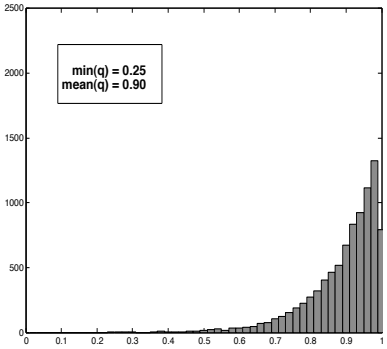
We choose an air foil mesh displayed in Fig 6(a). The average of the mesh quality is 0.9 and the minimal one is 0.25. In Fig 6(b), we display the mesh obtained by performing three iterations of ODT mesh smoothing. The average quality is improved to 0.95 and the minimal one is 0.57. Histograms of the mesh quality for the original



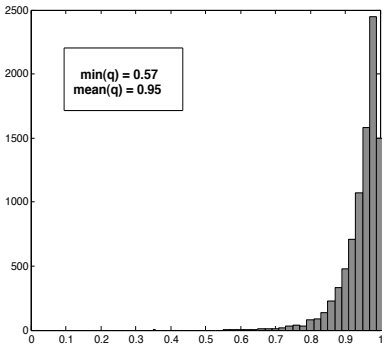
(a) The original mesh of an airfoil.



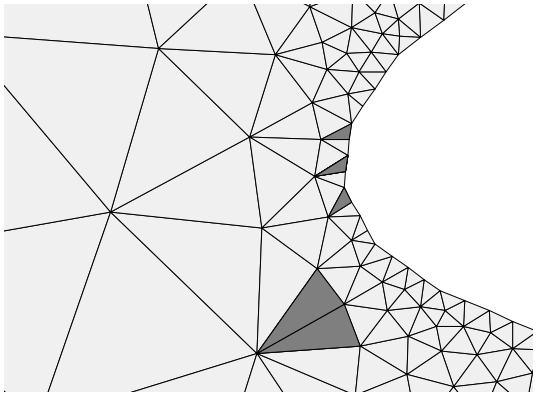
(b) A smoothed mesh after 3 steps of ODT smoothing.



(c) Quality of the original mesh.



(d) Quality of the smoothed mesh.



(e) Sample triangles with quality $q < 0.7$.

Figure 6: ODT smoothing applied to a non-uniform mesh.

mesh and smoothed mesh are presented in Fig 6(c) and 6(d). The mesh density in the original mesh is kept in the smoothed mesh.

We would expect higher quality for the $\min(q)$. By tracing the triangles with smaller q, we found it is constrained by the topology of the input mesh. We could improve the mesh quality by removing some grid points on the boundary. However in this example, the goal is to improve the mesh quality by using the same set of vertices, so we are not allowed to delete any vertex. This suggests incorporating our mesh smoothing/optimization into the mesh generator; see Section §5.3.

5.2. Global mesh optimization schemes

Again for completeness, we recall a general mesh optimization algorithm below.

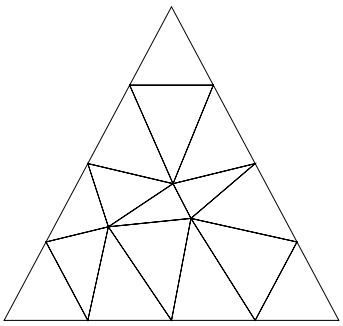
```
for k = 1:step
    p = p + αA^{-1}∇E(p);
    t = delaunay(p);
end
```

We choose uniform density $\rho = 1$. We compute an area based graph Laplacian A using the element-wise formula (4.24) and the standard assembling procedure. The inverse A^{-1} is computed using the backslash (or left matrix divide) function in MATLAB which is a very fast direct solver for SPD matrices. The step size α is included for generality and a common choice is $\alpha \in [0.75, 1]$. If we replace A by $\mathrm{diag}(A)$, we get a Jacobi-type local mesh smoothing method. We shall compare our global method with this local method.

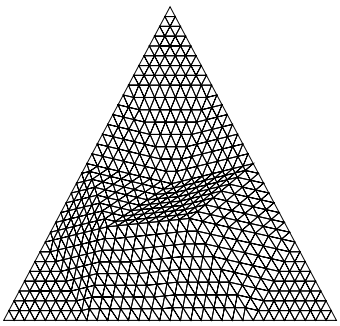
To illustrate that local method will fail to capture the error made in the coarse level, we construct a special input triangulation in two steps. First we perturb a uniform mesh of an equilateral triangle by moving 3 interior nodes; See Fig 7(a). Then we apply several global refinements; See Fig. 7(b). The quality of the finest mesh is determined by that of the coarse mesh.

We apply both local mesh smoothing and global mesh optimization to the mesh in Fig. 7(b). As we have shown in the previous step, the local method will improve the mesh quality and the resulting mesh is equilateral almost everywhere. But the improvement is gradually; see Fig. 7(c) for a mesh obtained by 42 iterations of the local method. The reason is, inside each coarse triangle, the star around a vertex is symmetric and ∇E vanishes. Therefore, the local method can only move vertices on the edge of the coarse mesh first and then conduct the movement to the interior nodes gradually. In contrast, the global method is extremely fast and it leads to the best triangulation, i.e., all triangles are equilateral, in four iterations; see Fig. 7(d). Again we plot approximated energies by shifting the integral $\int_\Omega u_I(\boldsymbol{x})dx$ in Fig. 8. The global method will converge in very few steps, and Newton-type convergence is observed.

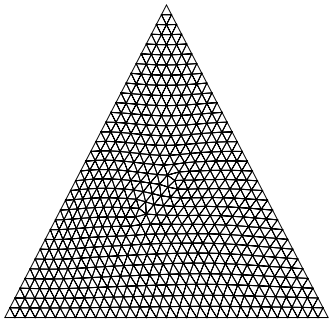
Remark 5.1. It is likely that the global method can bring the energy down to a lower level. In this example, the global method leads to the global minimizer. In general it may not be the case.



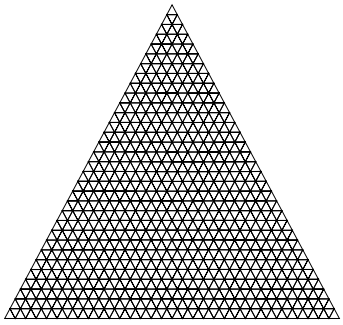
(a) A coarse triangulation $\mathcal{T}_0$



(b) Global refinement of $\mathcal{T}_0$



(c) Jacobi-type local mesh smoothing (42 iterations)



(d) Global mesh optimization (4 iterations)

Figure 7: Comparison of meshes obtained by local and global methods based on ODT.

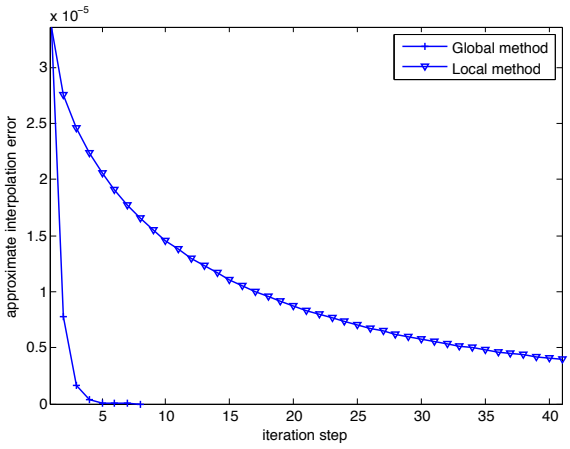


Figure 8: Error comparison of a local method and a global method

5.3. Combination of Distmesh and ODT mesh optimization

One constraint in our previous examples is the fixing of the boundary nodes. The distribution of boundary nodes is crucial for capturing the geometry of the domain, which is considered a function of mesh generators. It is natural to combine our mesh optimization schemes with any robust mesh generator to produce a better meshing algorithm.

Our previous example indicates that the global method can capture both fine and coarse level errors. This motivates a paradigm based on the combination: first, we apply any robust (not necessarily efficient) mesh generator at a very coarse level to capture the geometry of the domain; then, we apply several uniform refinement of the coarse mesh and use our global mesh optimization to obtain a fine and smoothed mesh.

This paradigm will be extremely useful for large size simulation of PDEs where the number of nodes is above several hundred of thousands. For such large size meshes, local mesh smoothing methods will not be efficient.

The mesh generator we choose is distmesh [12]. The geometry of the domain is represented by a signed distance function d and the nodes are distributed by solving for equilibrium in a truss structure. The combination of distance function representation and node movements from spring-like forces, which can be also treat as a local mesh smoothing method, enable distmesh to capture the boundary and improve mesh quality. It can produce non-uniform meshes by using an element size function h to weight the force associated to each edge. The size function h is small for a denser region i.e. where ρ is big. A relation between them is $\rho = 1/h^r$ with $r = 2$ or 3.

The main disadvantages of distmesh are slow execution and the possibility of non-termination when the number of grid points is large. The slow execution is partially due to the local feature of the mesh movement. The possibility of nontermination is from the lack of convergence theory. As noted in [12], the equilibrium position is hard to find due to the discontinuity in the force function when the topology of the mesh is changed. Furthermore, near the boundary, some stretched triangles are possible; see Fig. 3 (a) for such an example. At equilibrium, it is highly possible the vertices will be projected onto the boundary to eliminate such stretched triangles but without guarantee. However it may take a long time to reach the equilibrium, especially when the number of grid points is large.

We shall use distmesh in the coarse level. For small size meshes, say around 300 vertices, it is fast and reliable. We modify `distmesh2d` in [12] into a mesh smoothing subroutine `distmeshsmoothing`. Since we need only a initial guess for the fine mesh, we perform a fixed number of iterations rather than run the algorithm to reach an equilibrium configuration. Then we apply global uniform refinement several times. To better capture the boundary, we also apply `distmeshsmoothing` on the fine mesh a few times which will take care of the high frequency of the error. After that we use global method `odtmeshoptimization`. Lastly, to eliminate some bad triangles caused by some degree 2 vertices, which are vertices sharing by only two triangles, on the boundary (see Fig 6(e)), we code a `cleanup` subroutine. We summarize our algorithm as follows.

```
function [p,t] = odtmesh2d(fd,fh,h0,hbox,pfix,varargin)
```

```
h0 = h0*2^level;
[p,t] = initmesh(fd,fh,h0,hbox,pfix);
[p,t] = distmeshsmoothing(p,t,fd,fh,pfix,80);
[p,t] = uniformrefine(p,t,level);
[p,t] = distmeshsmoothing(p,t,fd,fh,pfix,10);
[p,t] = odtmeshopt(p,t,fd,fh,pfix,3);
[p,t] = cleanup(p,t,fd,pfix);
```

The interface of `odtmesh2d` is identical to `distmesh2d`. In the input arguments, `fd` is the distance function, `fh` is the size function, `h0` is the distance between points in the initial distribution, `hbox` is a rectangle containing the domain, `pfix` are fixed nodes, and `varargin` allows additional parameters; see [12] for details.

We include two examples to show the change of mesh quality and uniformity. Since we apply `distmeshsmoothing` only a fixed number of times, the quality after the smoothing is not necessarily high. Some triangles (near the boundary) may still have bad aspect ratio. `odtmeshoptimization` (with modification near the boundary) will improve the quality dramatically. The final step `cleanup` will further take care of those bad triangles caused by degree 2 boundary nodes.

```
(d) Square with hole (refined at hole) h = 0.02
1. distmeshsmoothing   - Min(q) 0.48 - Mean(q) 0.95 - Uniformity 5.1%
2. uniform refinement  - Min(q) 0.48 - Mean(q) 0.95 - Uniformity 6.9%
3. distmeshsmoothing   - Min(q) 0.25 - Mean(q) 0.96 - Uniformity 4.6%
4. odtmeshoptimization - Min(q) 0.63 - Mean(q) 0.97 - Uniformity 5.0%
5. clean up            - Min(q) 0.71 - Mean(q) 0.97 - Uniformity 5.4%

(e) Geometric Adaptivity h = 0.01
1. distmeshsmoothing   - Min(q) 0.50 - Mean(q) 0.96 - Uniformity 6.4%
2. uniform refinement  - Min(q) 0.50 - Mean(q) 0.96 - Uniformity 6.4%
3. distmeshsmoothing   - Min(q) 0.54 - Mean(q) 0.96 - Uniformity 6.1%
4. odtmeshoptimization - Min(q) 0.73 - Mean(q) 0.97 - Uniformity 6.1%
5. clean up            - Min(q) 0.73 - Mean(q) 0.97 - Uniformity 6.1%
```

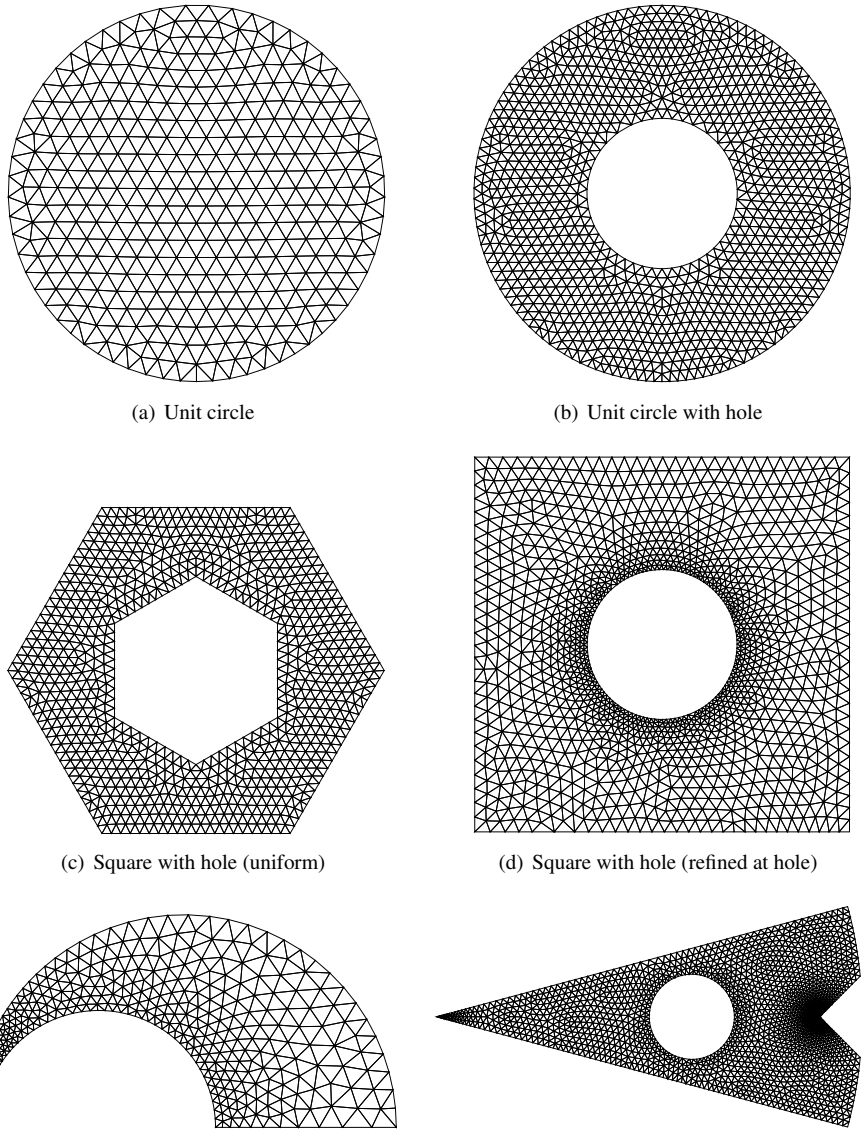
Table 1: Quality of all sample meshes in Fig. 9.

Examples	(a)	(b)	(c)	(d)	(e)	(f)
min(q)	0.74	0.74	0.76	0.71	0.73	0.57
mean(q)	0.99	0.98	0.98	0.97	0.97	0.97
uniformity	2.9%	4.5%	3.8%	5.4%	6.1%	7.6%

We present some sample meshes produced by `odtmesh2d` in Fig. 9 and the corresponding quality in Table 1. Almost all examples have minimal $q > 0.7$ and the average quality is greater than 0.97. The last example (f) has a sharp corner and the minimal quality is constrained by the triangle containing that corner.

5.4. Three dimensional mesh generation and optimization

The methods and formulae presented in this paper are for general dimensions. In this subsection, we shall provide two examples of mesh smoothing for three dimensional tetrahedron meshes.



(a) Unit circle

(b) Unit circle with hole

(c) Square with hole (uniform)

(d) Square with hole (refined at hole)

(e) Geometric Adaptivity

(f) Pie with hole

Figure 9: Sample meshes produced by `odtmesh2d`.

We first present an example of local mesh smoothing using CPT (4.20) and ODT (4.13) smoothing with step size $\alpha = 1$ and uniform density $\rho = 1$.

We choose a initial mesh of the unit ball produced by distmesh, and apply only few steps of the mesh smoothing schemes in distmesh. The mean value of the mesh quality for the initial mesh is 0.74, and quite a few bad elements ($q \leq 0.4$) exist; see Fig. 10(a) (b). We then apply CPT (4.20) and ODT (4.13) smoothing to this initial mesh. Note that, in ODT, we modify the centers of the boundary elements (defined as tetrahedrons containing at least one boundary vertex) to be barycenter, not circumcenter. We present the histogram of the mesh quality for two meshes obtained by applying 40 steps of ODT and CPT smoothing in Fig. 10(d) (f). We apply the `delaunayn` command in MATLAB after each 2 smoothing steps to optimize the connectivity.

We observe similar behavior as in 2D case. The mesh quality is improved significantly by ODT and CPT; see Fig 10(b), 10(d), and 10(f). The improvement of the mesh quality due to ODT and CPT are similar; concerning uniformity, ODT smoothing produces a better result when compared with CPT smoothing.

ODT smoothing has been included in the Computational Geometry Algorithms Library (CGAL) [57]. We use CGAL to demonstrate the effectiveness of ODT smoothing on 3D meshing. The initial grid of the standard bunny object is obtained by applying constrained Delaunay refinement; a description of this algorithm can be found in [58] or [59]. The average shape quality of 0.78 is reasonably good.

We then apply ODT and CPT smoothing to the initial grid. The density ρ in CGAL ODT smoothing is determined as follows: first we compute a size function at all vertices $h(\boldsymbol{x}) = \text{mean}(\|\boldsymbol{x} - \boldsymbol{c}\|)$, where $\boldsymbol{c}$ is the circumcenter, and the average is taken in the star of the vertex $\boldsymbol{x}$. This leads to a piecewise linear size function. The element-wise density function is chosen as $\rho_\tau = 1/h(\boldsymbol{b}_\tau)^3$, with $\boldsymbol{b}_\tau$ as the barycenter of the element τ. Lloyd mesh smoothing from CVT [13] is also included in CGAL. The difference between ODT and CVT is the average of circumcenters is in the Voronoi region, not the star of the vertex.

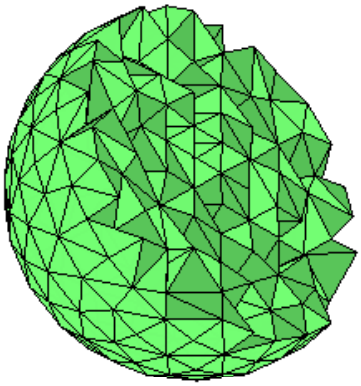
In each step of mesh smoothing, a surface smoothing step is also included for the boundary mesh. After several steps smoothing, constrained Delaunay algorithms are applied to ensure the current triangulation is still Delaunay (constrained to the boundary).

We apply these three mesh smoothing schemes to the same initial grid in Fig. 5.4, and compare the mesh quality. From Fig. 11, we can conclude:

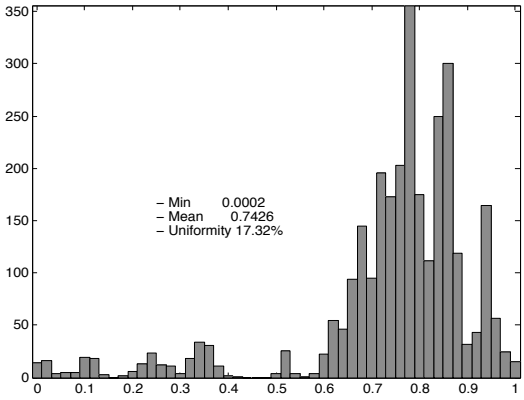
1. The mesh quality is improved by these three mesh smoothing schemes.
2. ODT mesh smoothing is the best among these three smoothers.
3. In the first 10 steps, the ODT and CPT smoothing improved the quality a lot and then the improvement is slow down. While CVT smoother in the 10 steps is not as good as other two methods.

Although our mesh smoothing scheme is not sliver-free (there are still some tetrahedron with tiny aspect ratios), it appears that mesh smoothing based ODT will produce fewer sliver tetrahedra; see [24, 60] for more supporting numerical examples.

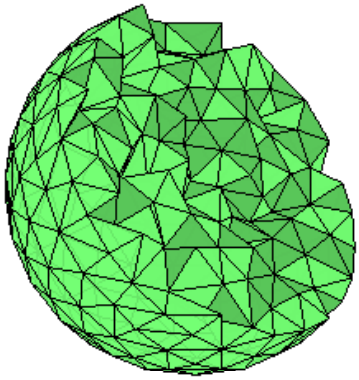
In [24, 60], the authors use and improve some of the local mesh smoothing schemes developed in our previous work [1]. We expect the more efficient global method developed in this paper will further improve the performance of 3D mesh generation.



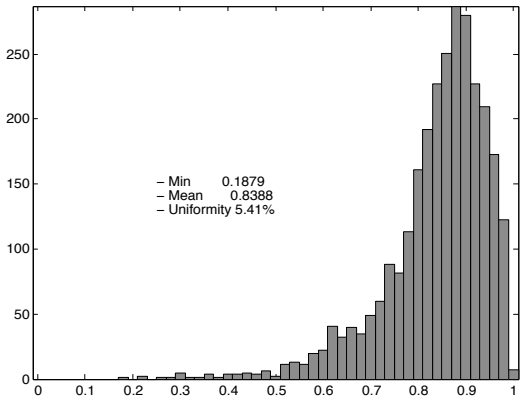
(a) The original mesh of the unit ball.



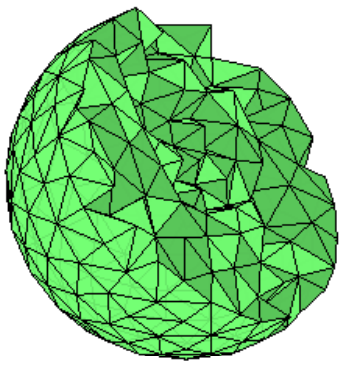
(b) Quality of the original mesh.



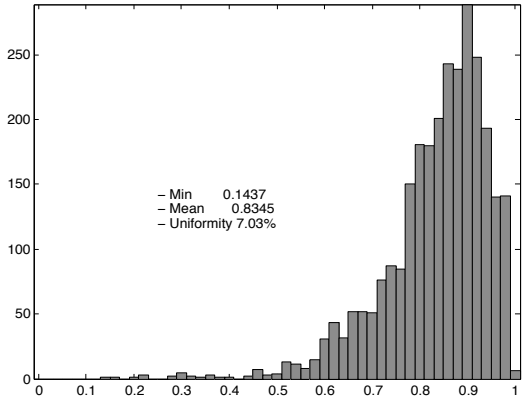
(c) A smoothed mesh after 40 steps of ODT smoothing.



(d) Quality of the smoothed mesh in (c).

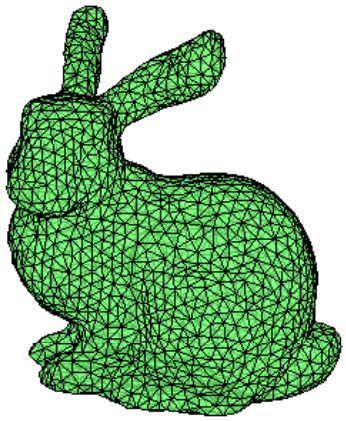


(e) A smoothed mesh after 40 steps of CPT smoothing.

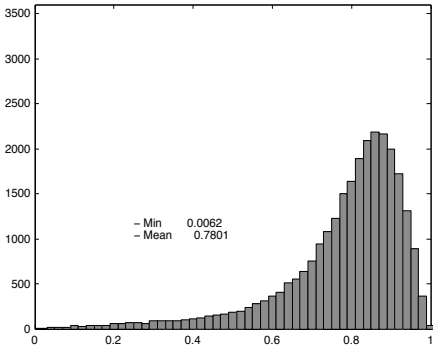


(f) Quality of the smoothed mesh in (e).

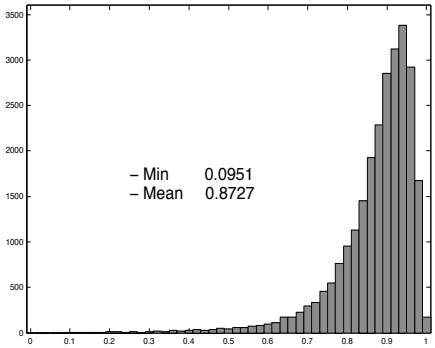
Figure 10: ODT and CPT mesh smoothing applied to a mesh of the unit ball.



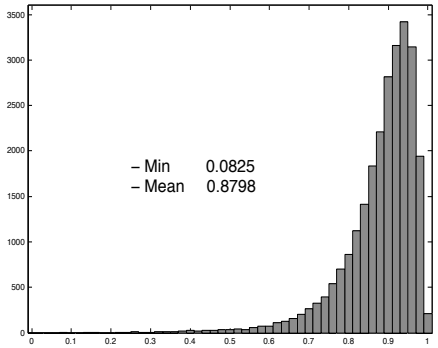
(a) An initial grid of a bunny.



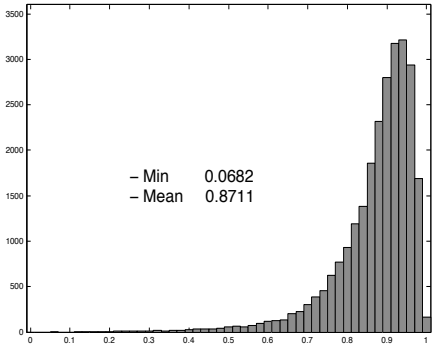
(b) Quality of the initial mesh.



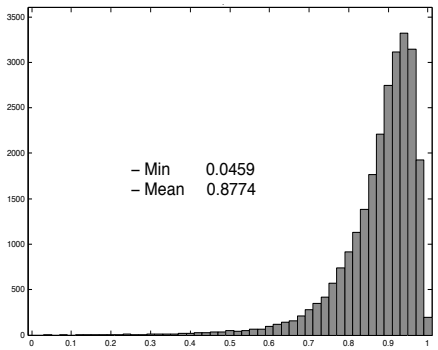
(c) Quality of a mesh after 10 steps ODT smoothing.



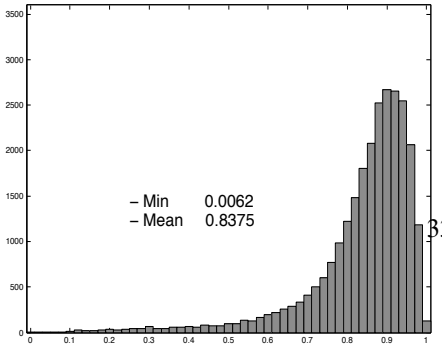
(d) Quality of a mesh after 50 steps ODT smoothing.



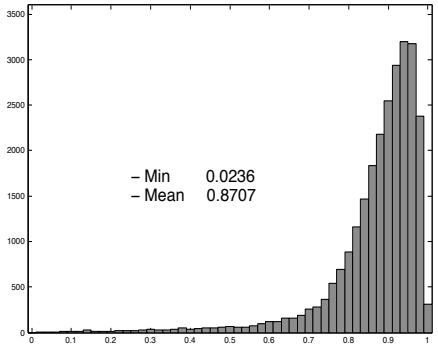
(e) Quality of a mesh after 10 steps CPT smoothing.



(f) Quality of a mesh after 50 steps ODT smoothing.



(g) Quality of a mesh after 10 steps CVT smoothing.



(h) Quality of a mesh after 50 steps CVT smoothing.

Figure 11: ODT, CPT and CVT mesh smoothing applied to a mesh of the bunny.

However, the implementation of such a method requires critical and non-trivial work on the boundary surface mesh; see [61, 62, 63].

6. Conclusion and Further Work

In this paper we have introduced several local mesh smoothing and global mesh optimization schemes based on minimizing energies related to a weighted interpolation error. We combined our global mesh optimization of a fine mesh with a robust mesh generator for a coarse mesh, to get an efficient mesh generation algorithm in two dimensions. The splitting of scales in the mesh generation and optimization procedures is critical for the numerical solution of partial differential equations. In future work, we will apply such a methodology to adaptive finite element methods, in the hope to develop more efficient adaptive mesh refinement procedures. We shall also investigate the combination of surface mesh smoothing with 3D mesh smoothing or optimization.

Acknowledgment

The authors would like to thank Huayi Wei from Xiangtan University on the preparation of the last 3D example: mesh smoothing for a mesh of the bunny using CGAL.

[1] L. Chen, Mesh smoothing schemes based on optimal Delaunay triangulations, in: 13th International Meshing Roundtable, Sandia National Laboratories, Williamsburg, VA, pp. 109–120.

[2] L. Chen, J. Xu, Optimal Delaunay triangulations, J. Comput. Math. 22 (2004) 299–308.

[3] L. Chen, Robust and Accurate Algorithms for Solving Anisotropic Singularities, Ph.D. thesis, Department of Mathematics, The Pennsylvania State University, 2005.

[4] P. Knupp, S. Steinberg, Fundamentals of grid generateion, CRC Press (1994).

[5] G. Carey, Computational grids: generation, adaptation, and solution strategies, CRC, 1997.

[6] V. D. Liseikin, Grid Generation Methods, Springer Verlag, Berlin, 1999.

[7] W. Huang, Variational mesh adaptation: isotropy and equidistribution, Journal of Computational Physics 174 (2001) 903–924.

[8] A. S. Dvinsky, Adaptive grid generation from harmonic maps on Riemannian manifolds, Journal of Computational Physics 95 (1991) 450–476.

[9] W. Huang, R. D. Russell, Moving mesh strategy based on a gradient flow equation for two-dimensional problems, SIAM J. Sci. Comput. 20 (1999) 998–1015.

[10] R. Li, T. Tang, P. Zhang, Moving mesh methods in multiple dimensions based on harmonic maps, Journal of Computational Physics 170 (2001) 562–588.

[11] A. Anderson, X. Zheng, V. Cristini, Adaptive unstructured volume remeshing - i: The method, Journal of Computational Physics 208 (2005) 616–625.

[12] P.-O. Persson, G. Strang, A simple mesh generator in matlab, SIAM Rev. 46 (2004) 329–345.

[13] Q. Du, M. Gunzburger, Grid generation and optimization based on centroidal Voronoi tessellations, Appl. Math. Comp. 133 (2002) 591–607.

[14] Q. Du, V. Faber, M. Gunzburger, Centroidal voronoi tessellations: Applications and algorithms, SIAM Rev. 41(4) (1999) 637–676.

[15] K. Q. Brown, Voronoi diagrams from convex hulls, Inform. Process. Lett. 9 (1979) 223–228.

[16] C. Barber, D. Dobkin, H. Huhdanpaa, The quickhull algorithm for convex hulls, ACM Transactions on Mathematical Software (TOMS) 22 (1996) 469–483.

[17] M. Aiffa, J. E. Flaherty, A geometrical approach to mesh smoothing, Comput. Methods Appl. Mech. Engrg. 192 (2003) 4497–4514.

[18] N. Amenta, M. Bern, D. Eppstein, Optimal point placement for mesh smoothing, Journal of Algorithms (1999) 302–322.

[19] R. E. Bank, R. K. Smith, Mesh smoothing using a posteriori error estimates, SIAM J. Numer. Anal. 34 (1997) 979–997.

[20] L. A. Freitag, M. T. Jones, P. E. Plassmann, A parallel algorithm for mesh smoothing, SIAM J. Sci. Comput. 20 (1999) 2023–2040.

[21] L. Freitag, C. Ollivier-Gooch, Tetrahedral mesh improvement using swapping and smoothing, International Journal of Numerical Methods in Engineering 40 (1997) 3979–4002.

[22] L. Freitag, On combining laplacian and optimization-based mesh smoothing techniques, AMD Trends in Unstructured Mesh Generation, ASME 220 (July 1997) 37–43.

[23] T. Zhou, K. Shimada, An angle-based approach to two-dimensional mesh smoothing, in: 9th International Meshing Roundtable, Sandia National Laboratories, pp. 373–384.

[24] P. Alliez, D. Cohen-Steiner, M. Yvinec, M. Desbrun, Variational tetrahedral meshing, ACM Trans. Graph. 24 (2005) 617–625.

[25] M. Berndt, M. Shashkov, Multilevel accelerated optimization for problems in grid generation, in: Proc. 12th Int. Meshing Roundtable, Sandia Nat. Lab., 2003, pp. 351–359.

[26] Y. Koren, I. Yavneh, A. Spira, A multigrid approach to the scalar quantization problem, IEEE Transactions on Information Theory 51 (2005) 2993–2998.

[27] Y. Koren, I. Yavneh, Adaptive multiscale redistribution for vector quantization, SIAM J. Sci. Comput. 27 (2006) 1573–1593.

[28] R. M. Spitaleri, Full-fas multigrid grid generation algorithms, Appl. Numer. Math. 32 (2000) 483 – 494.

[29] Q. Du, M. Emelianenko, Uniform convergence of a nonlinear energy-based multilevel quantization scheme, SIAM J. Numer. Anal. 46 (2008) 1483–1502.

[30] L. Chen, New analysis of the sphere covering problems and optimal polytope approximation of convex bodies, Journal of Approximation Theory 133 (2005) 134–145.

[31] L. Chen, On minimizing the linear interpolation error of convex quadratic functions, East Journal of Approximation 14 (2008) 271–284.

[32] L. Chen, P. Sun, J. Xu, Optimal anisotropic simplicial meshes for minimizing interpolation errors in L^p-norm, Math. Comp. 76 (2007) 179–204.

[33] H. Edelsbrunner, Triangulations and meshes in computational geometry, Acta Numer. (2000) 1–81.

[34] M. Bern, D. Eppstein, Mesh generation and optimal triangulation, in: D.-Z. Du, F. Hwang (Eds.), Computing in Euclidean Geometry, volume 1, World Scientific, Lecture Notes Series on Computing – Vol. 1, 1992, pp. 23–90.

[35] S. Fortune, Voronoi diagrams and Delaunay triangulations, in: Computing in Euclidean Geometry, Edited by Ding-Zhu Du and Frank Hwang, World Scientific, Lecture Notes Series on Computing - Vol. 1, 1992.

[36] R. Sibson, Locally equiangular triangulations, Computer Journal 21 (1978) 243–245.

[37] T. Lamber, The Delaunay triangulation maximize the mean inradius, in: Proc. 6th Canad. Conf. Comput. Geom, pp. 201–206.

[38] S. Rippa, Minimal roughness property of the Delaunay triangulation, Comput. Aided Geom. Design 7 (1990) 489–497.

[39] E. F. D'Azevedo, R. B. Simpson, On optimal interpolation triangle incidences, SIAM J. Sci. Statist. Comput. 6 (1989) 1063–1075.

[40] V. T. Rajan, Optimality of the Delaunay triangulation in R^d, Proc. of the Seventh Annual Symp. on Comp. Geom (1991) 357–363.

[41] S. Rippa, Long and thin triangles can be good for linear interpolation, SIAM J. Numer. Anal. 29 (1992) 257–270.

[42] E. A. Melissaratos, Lp Optimal d Dimensional Triangulations for Piecewise Linear Interpolation: A New Result on data Dependent Triangulations, Technical Report RUU-CS-93-13, Department of Information and Computing Sciences, Utrecht University, 1993.

[43] D. S. Mitrinovic, J. E. Pecaric, V. Volenec, Recent Advances in Geometric Inequalities, Mathematics and its applications: East European Series 28, 1989.

[44] L. J. Guibas, D. E. Knuth, M. Sharir, Randomized incremental construction of delaunay and voronoi diagrams, in: Proceedings of the seventeenth international colloquium on Automata, languages and programming, Springer-Verlag New York, Inc., New York, NY, USA, 1990, pp. 414–431.

[45] H. Edelsbrunner, R. Seidel, Voronoi diagrams and arrangements, Disc. and Comp. Geom. 8 (1986) 25–44.

[46] H. Edelsbrunner, X.-Y. Li, G. Miller, A. Stathopoulos, D. Talmor, S.-H. Teng, A. Üngör, N. Walkington, Smoothing and cleaning up slivers, in: STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing, ACM, New York, NY, USA, 2000, pp. 273–277.

[47] H. Xu, T. S. Newman, An angle-based optimization approach for 2d finite element mesh smoothing, Finite Elem. Anal. Des. 42 (2006) 1150–1164.

[48] V. Parthasarathy, S. Kodiyalam, A constrained optimization approach to finite element mesh smoothing, Finite Elements in Analysis and Design 9 (1991) 309–320.

[49] Q. Du, D. Wang, Tetrahedral mesh generation and optimization based on centroidal voronoi tessellations, International Journal for Numerical Methods in Engineering 56 (2003) 1355–1373.

[50] Q. Du, D. Wang, Recent progress in robust and quality Delaunay mesh generation, J. Comput. Math. 195 (2006) 8–23.

[51] Q. Du, M. Gunzburger, L. Ju, Voronoi-based finite volume methods, optimal Voronoi meshes, and PDEs on the sphere, Comput. Methods Appl. Mech. Engrg. 192 (2003) 3993–3957.

[52] L. Ju, Conforming centroidal voronoi delaunay triangulation for quality mesh generation, International Journal of Numerical Analysis and Modeling To appear (2007).

[53] L. Ju, M. Gunzburger, W. Zhao, Adaptive finite element methods for elliptic PDEs based on conforming centroidal Voronoi Delaunay triangulations, SIAM J. Sci. Comput. To appear (2007).

[54] D. A. Field, Laplacian smoothing and Delaunay triangulation, Communications in Applied Numerical Methods 4 (1988) 709–712.

[55] D. A. Spielman, S.-H. Teng, Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems, Preliminary draft (2006).

[56] J. Xu, Two-grid discretization techniques for linear and nonlinear PDEs, SIAM J. Numer. Anal. 33 (1996) 1759–1777.

[57] CGAL, Computational Geometry Algorithms Library, ???? Http://www.cgal.org.

[58] J. R. Shewchuk, Tetrahedral mesh generation by Delaunay refinement, in: 14th Annual ACM Symposium on Computational Geometry, pp. 86–95.

[59] T. K. Dey, Delaunay mesh generation of three dimensional domains, Technical Report OSU-CISRC-9/07-TR64 (2007).

[60] J. Tournois, C. Wormser, P. Alliez, M. Desbrun, Interleaving delaunay refinement and optimization for practical isotropic tetrahedron mesh generation, ACM Trans. Graph. 28 (2009) 1–9.

[61] Q. Du, D. Wang, Constrained boundary recovery for three dimensional Delaunay triangulations, International Journal for Numerical Methods in Engineering 61 (2004) 1471–1500.

[62] Q. Du, M. Gunzburger, L. Ju, Constrained centroidal Voronoi tessellations for surfaces, SIAM J. Sci. Comput. 24 (2003) 1488–1506.

[63] Z. Yu, M. J. Holst, J. A. McCammon, High-fidelity geometric modeling for biomedical applications, Finite Elements in Analysis and Design 44 (2008) 715 – 723.