DOI: 10.1111/j.1467-8659.2010.01812.x

Pacific Graphics 2010
P. Alliez, K. Bala, and K. Zhou
(Guest Editors)

Volume 29 (2010), Number 7

Fast Updating of Delaunay Triangulation of Moving Points by Bi-cell Filtering

Yuanfeng Zhou[1], Feng Sun[1], Wenping Wang[1], Jiaye Wang[2], and Caiming Zhang[2]

[1]Department of Computer Science, The University of Hong Kong, Hong Kong, China
[2]School of Computer Science and Technology, Shandong University, Jinan, China

Abstract

Updating a Delaunay triangulation when data points are slightly moved is the bottleneck of computation time in variational methods for mesh generation and remeshing. Utilizing the connectivity coherence between two consecutive Delaunay triangulations for computation speedup is the key to solving this problem. Our contribution is an effective filtering technique that confirms most bi-cells whose Delaunay connectivities remain unchanged after the points are perturbed. Based on bi-cell flipping, we present an efficient algorithm for updating two-dimensional and three-dimensional Delaunay triangulations of dynamic point sets. Experimental results show that our algorithm outperforms previous methods.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

1. Introduction

Delaunay triangulation is a fundamental data structure in computational geometry. It is versatile in a spectrum of applications including mesh generation for finite element method [EAD*06], surface reconstruction from scanned data in computer graphics [CG04] and intensity field reconstruction in astrophysics [Sch07]. The computation of Delaunay triangulations has extensively been studied in recent decades [BDP*02]. Efficient and robust methods in two dimensions [She96] and three dimensions [CGA] are available.

Some applications, such as mesh generation [DBC07, ACSYD05, TWAD09, YWLL10] and remeshing [AdVDI05, LWL*09, YLL*09], require to iteratively construct the Delaunay triangulation of a point set which changes slightly in each iteration, and computing the Delaunay triangulation in each iteration is the most time-consuming step. Figure 1 shows the initialization and connectivity changes in two iterations of a typical Delaunay meshing algorithm in two dimensions, based on the CVT approach using the Lloyd iterations. Note that 30 edges are flipped at the 10th iteration and only 2 edges are flipped at the 100th iteration. It is therefore natural to ask if the update can be done efficiently in such

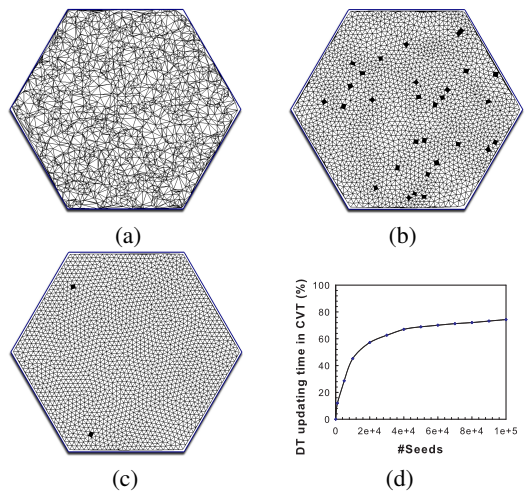


Figure 1: *Delaunay triangulations of 2,000 points in a typical meshing algorithm. Edge flips (different connectivities) from previous iteration are highlighted in dark. (a) Delaunay triangulation of initialization; (b) The 10th iteration; (c) The 100th iteration; (d) The percentage of computation time taken for DT updating with different number of seeds.*

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.
Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

cases without computing the Delaunay triangulation from scratch.

Although frequently used, it is inefficient to always build the Delaunay triangulation from scratch in each iteration, which overlooks the spatial coherence between consecutive iterations. Figure 1(d) shows the percentage of time used to build Delaunay triangulation with different number of seeds. Compared with existing mature methods for computing Delaunay triangulations of fixed points, algorithms for computing Delaunay triangulations of moving points need further investigation. In this paper, we focus on the latter and propose an efficient approach.

The problem is formulated as follows: given a point set S and its Delaunay triangulation $\mathcal{DT}(S)$, we need to compute the Delaunay triangulation $\mathcal{DT}(S')$ of a point set S', which is somewhat perturbed from S. The method that computes the Delaunay triangulation $\mathcal{DT}(S')$ from scratch will be referred to as the ***rebuilding*** method, in which the spatial coherence between S and S' is not utilized at all. Currently, no existing algorithms outperform rebuilding consistently on typical three-dimensional data sets, albeit the computation can be accelerated by one order of magnitude in two dimensions by the method in [dCTAD09], which is deemed the fastest method for updating Delaunay triangulation of moving points so far.

Contribution: Our contribution is a simple and effective check to quickly confirm most bi-cells in $\mathcal{DT}(S)$ which remain Delaunay for S' after perturbation of data points. We use bi-cell flipping to update a bi-cell in $\mathcal{DT}(S)$ if it fails the check and is therefore potentially non-Delaunay, supposing that it is flippable. If the local connectivity becomes non-Delaunay and not flippable, which is a relatively rare case, we then rebuild the Delaunay triangulation of S' from scratch. Experiments show our algorithm outperforms other existing methods and runs 2 to 3 times faster than rebuilding in three dimensions.

2. Review

Several methods have been proposed to update Delaunay triangulations of points evolving in iterations. We will give a brief review of each approach. We first breifly introduce Delaunay triangulation.

2.1. Delaunay Triangulation

The *Delaunay triangulation* of a set S of n points in $\mathbb{R}^d$, $\mathcal{DT}(S)$, is a simplicial complex [Mun93, dBvKOS97] which satisfies the *empty sphere* condition. By *empty sphere*, we mean that no points in S lie inside the circumscribing sphere of any simplex in $\mathcal{DT}(S)$. Figure 2 (a) illustrates the empty sphere property of a two-dimensional Delaunay triangulation of four points.

Different algorithms have been proposed to construct Delaunay triangulations [PS93, For86] when the set S is given and fixed. There are also some algorithms and implementations that build Delaunay triangulations when the set S is given on the fly [Dev02, LJ05, CGA].

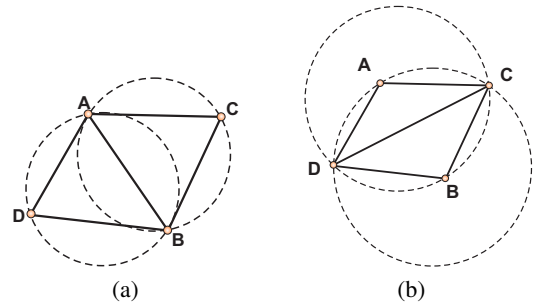


Figure 2: *Two typical two-dimensional bi-cells. (a) satisfies the empty sphere condition; (b) does not satisfy the empty sphere condition, since* **B** *is inside the circumcircle of the triangle* **ADC**.

Since our algorithm is based on bi-cell flipping, we briefly introduce the concepts in the bi-cell flipping algorithm.

Definition 1 (Bi-cell) : A bi-cell in two-dimensional triangulation is a pair of triangles sharing an edge. A bi-cell in three-dimensional triangulation is a pair of tetrahedra sharing a face.

Remark: In two dimensions, there is a one-to-one mapping between bi-cells and edges. So we may also use an edge to denote a bi-cell.

Figure 2 shows two typical two-dimensional bi-cells. (a) shows a bi-cell satisfying the empty sphere condition and (b) shows a bi-cell violating the condition. Flipping the edge **CD** to **AB** turns (b) to (a) to generate the Delaunay triangulation. In two dimensions, the Delaunay triangulation can be constructed by edge (i.e., bi-cells) flipping until all edges (i.e., bi-cells) are Delaunay [Wag36, dBvKOS97].

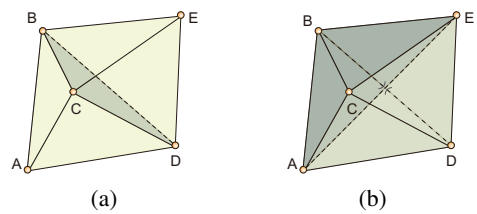


Figure 3: *A three-dimensional bi-cell and its flipping. (a) A three-dimensional bi-cell. (b) Three bi-cells result from flipping the face* **BCD** *to the edge* **AE**.

Figure 3 (a) shows a three-dimensional bi-cell. Unlike the two-dimensional case, it is still open in three dimensions whether one triangulation can always reach another triangulation of the same point set by flipping [Joe89]. However, such cases are not encountered in our experiments. The overall performance of our bi-cell flipping algorithm is better than rebuilding, as we will see later in this paper.

Now we shall review existing methods.

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

2.2. Kinetic Framework

Given a set of points S, its Delaunay triangulation $\mathcal{DT}(S)$ and S' perturbed from S, we build $\mathcal{DT}(S')$. This problem can be solved in a continuous manner using the framework of the kinetic data structure (*kinetic framework*) [Gui98].

[Rus07] proposes to morph $\mathcal{DT}(S)$ to $\mathcal{DT}(S')$ by keeping track of all connectivity changes when all points in S move to points in S', respectively. Most time is consumed on the computation of the connectivity changes which are not relevant to $\mathcal{DT}(S')$ and thus unnecessary. Due to the complexity of maintaining all connectivity changes, this method is considerably slower than rebuilding $\mathcal{DT}(S')$ from scratch in all experiments in [Rus07]. Here we conclude that kinetic framework is not suitable for efficiently building Delaunay triangulations of evolving point sets when we do not want to keep track of the continuous motion between S and S'.

2.3. Flipping Update Algorithm

Guibas and Russell consider using flipping as the main operation to update $\mathcal{DT}(S)$ to $\mathcal{DT}(S')$ [GR04]. Conceptually, the procedure is to first check all cells in $\mathcal{DT}(S)$. If the orientation of one cell is reversed, then remove some vertices in $\mathcal{DT}(S)$ to make it an embedded triangulation. Then, flipping bi-cells to make it Delaunay. Finally insert the removed vertices with updated geometric positions in S' to get $\mathcal{DT}(S')$. In three dimensions, flipping may get stuck and rebuilding is triggered to compute $\mathcal{DT}(S')$. Their experiments show this approach is often faster than rebuilding. Shewchuk proposes *star splaying* to build Delaunay triangulations from a "nearly Delaunay" triangulation by local correction [She05]. It can resolve the flipping stuck in algorithm of [GR04] without calling rebuilding.

2.4. Vertex Filtering Algorithm

De Castro *et al.* propose a new approach that associates a safe zone to each vertex in $\mathcal{DT}(S)$ [dCTAD09]. One vertex is said to be filtered if it stays in its safe zone and therefore does not trigger a change of its connectivity to its neighboring vertices. If a vertex does not pass the filtering:

(i) In two dimensions, it is relocated if the triangulation is not embedded anymore, otherwise it is repaired by edge flipping.

(ii) In three dimensions, it is relocated if the local star of the vertex is not Delaunay anymore.

To relocate an updated vertex, the algorithm first inserts the point with updated geometric position and removes the point with previous outdated position. After this relocation, safe zones of some nearby vertices are affected and a series of updates of these safe zones are performed to keep all safe zones valid. This algorithm is suitable for mesh generation and remeshing applications based on variational approaches in which the displacement of vertices decreases progressively along iterations. The algorithm is the fastest one known so far for updating Delaunay triangulation of a point set with small perturbation and is the one to be compared with our algorithm.

A more detailed explanation of the safe zone defined in [dCTAD09] is needed for later discussion. The figure on the right shows a two-dimensional bi-cell and two concentric circles it defines. The center of the two circles is the intersection point of the two perpendicular bisectors of AB and CD. Then the median circle, shown in solid line, defines a tolerance zone for the bi-cell in that the connectivity will remain unchanged if after perturbation the two outer-vertices stay outside the median circle and the two inner vertices stay inside the median circle. Hence, a circle is assigned as a safe zone to each vertex – the circle is centered at the vertex and its radius is the distance between the median circle to the outer circle. It follows that the original edge connectivity will remain unchanged if all the vertices after perturbation stay inside their safe zones. The idea of the vertex safe zone in three dimension is similar but its construction is more complex. Please refer to the details in [dCTAD09].

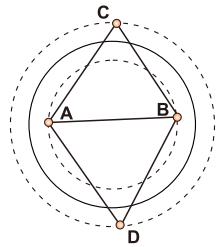


Figure 4: *Tolerance defined for a two-dimensional bi-cell in [dCTAD09].*

Note that the above filtering condition (or safe zone) is conservative, since that the vertices stay in their safe zones is only a sufficient condition for keeping the same edge connectivity, not a necessary one.

No existing algorithms outperform rebuilding consistently for typical three-dimensional triangulations, though in two dimensions the computation can be accelerated by one order of magnitude [dCTAD09]. Therefore, we wish to develop more efficient filtering conditions so that more perturbed vertices whose original connectivity still satisfying Delaunay condition can be filtered, thus reducing the instances of unnecessarily invoking the procedure to update the edge connectivity.

3. A New Delaunay Certificate Safe Zone for Bi-cell Filtering

In this section, we shall give a fundamentally new, and efficient filtering condition for checking the Delaunay condition after perturbation. For simplicity of discussion, the basic idea will be explained in two dimensions but it is easy to extend to three dimensions.

3.1. Delaunay Condition of a Bi-cell

Figure 5 (a) illustrates a two-dimensional Delaunay bi-cell composed of two cells $\triangle \mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$ and $\triangle \mathbf{p}_1\mathbf{p}_3\mathbf{p}_4$. The *empty*

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

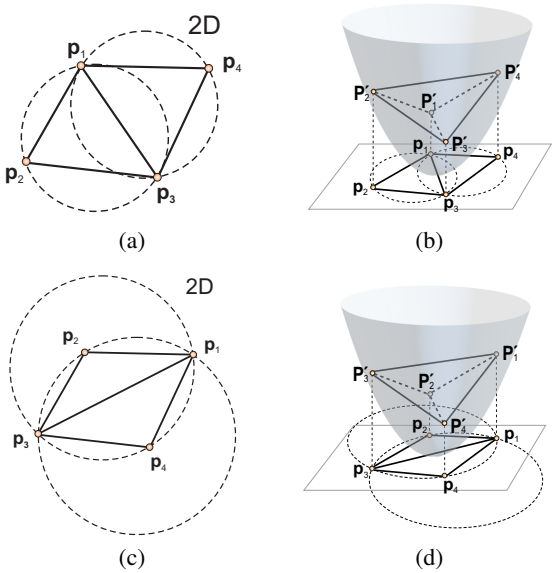


Figure 5: *Delaunay condition of a two-dimensional bi-cell. (a) a bi-cell satisfying empty sphere condition; (b) positive orientation of the lifted tetrahedron; (c) a bi-cell violating empty sphere condition; (d) negative orientation of the lifted tetrahedron.*

sphere condition is equivalent to the positive orientation of the tetrahedron lifted from the four points to the paraboloid $z = x^2 + y^2$, as illustrated in Figure 5 (b). Here we denote $\mathbf{P}_i = (x_i, y_i, x_i^2 + y_i^2)$ as the lifted point of $\mathbf{p}_i = (x_i, y_i)$ to the paraboloid $z = x^2 + y^2, i = 1,2,3,4$. The bi-cell composed of $\triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$ and $\triangle\mathbf{p}_1\mathbf{p}_3\mathbf{p}_4$ is Delaunay if and only if the volume of the tetrahedron is positive, or equivalently, the determinant $|M|$ of the matrix M is positive, where M is

$$\begin{pmatrix} x_1 & y_1 & x_1^2+y_1^2 & 1 \\ x_2 & y_2 & x_2^2+y_2^2 & 1 \\ x_3 & y_3 & x_3^2+y_3^2 & 1 \\ x_4 & y_4 & x_4^2+y_4^2 & 1 \end{pmatrix}.$$

Figure 5 (c) and (d) show a non-Delaunay bi-cell violating the empty sphere condition and its lifted tetrahedron with negative orientation, respectively.

To summarize, we have the following theorem [GS85, dBvKOS97].

Theorem 1 Given a two-dimensional bi-cell Bc, the following four statements are equivalent:

1. Bc is Delaunay;
2. Bc satisfies the empty sphere condition;
3. The lifted tetrahedron of Bc has positive orientation;
4. The determinant of the corresponding matrix M of Bc is positive.

3.2. Perturbation Theorem

Theorem 1 connects the empty sphere condition of a Delaunay bi-cell with the positive determinant of its corresponding matrix. This suggests that one may update a Delaunay triangulation with existing tools in matrix computation. Here we recall a well-known result in matrix theory and show how to apply it to updating Delaunay triangulations.

Theorem 2 (Perturbation Theorem) Suppose $A \in \mathbb{R}^{n\times n}$ and $\|\Delta A\| \le \varepsilon\|A\|$. If $\varepsilon \cdot \kappa(A) < 1$, then $A + \Delta A$ is nonsingular, where $\|.\|$ is a matrix norm, $\kappa(A) = \|A\| \cdot \|A^{-1}\|$ is the condition number of the matrix A.

The theorem is part of Lemma 2.7.1 in [GVL96].

Given the input point set S, its triangulation $\mathcal{DT}(S)$ and S' perturbed from S, we use Theorem 2 to filter bi-cells in $\mathcal{DT}(S)$. Let Bc be a bi-cell in $\mathcal{DT}(S)$ composed of $\triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$ and $\triangle\mathbf{p}_1\mathbf{p}_3\mathbf{p}_4$. With a perturbation, we denote $\mathbf{p}_i'$ in S' as $\mathbf{p}_i + \delta\mathbf{p}_i$, that is $x_i' = x_i + \delta x_i$ and $y_i' = y_i + \delta y_i$. Then, the corresponding matrix M' of the perturbed Bc is $M + \Delta M$, where ΔM is

$$\begin{pmatrix} \delta x_1 & \delta y_1 & \Delta_1 & 0 \\ \delta x_2 & \delta y_2 & \Delta_2 & 0 \\ \delta x_3 & \delta y_3 & \Delta_3 & 0 \\ \delta x_4 & \delta y_4 & \Delta_4 & 0 \end{pmatrix},$$

where $\Delta_i = 2x_i\delta x_i + \delta x_i^2 + 2y_i\delta y_i + \delta y_i^2, i = 1,2,3,4$.

In three dimensions, the i-th row in M is $(x_i, y_i, z_i, x_i^2 + y_i^2 + z_i^2, 1)^T$ and the i-th row in ΔM is $(\delta x_i, \delta y_i, \delta z_i, \Delta_i, 0)^T$, where $\Delta_i = 2x_i\delta x_i + \delta x_i^2 + 2y_i\delta y_i + \delta y_i^2 + 2z_i\delta z_i + \delta z_i^2$, for $i = 1,2,3,4,5$.

By Theorems 1 and 2, we may check the perturbed corresponding matrix ΔM to filter bi-cells which are still Delaunay after perturbation, as stated in the next theorem.

Theorem 3 Given a Delaunay bi-cell Bc, let M be its corresponding matrix. Let the corresponding matrix be $M + \Delta M$ after perturbation. If $\|\Delta M\| < \frac{\|M\|}{\kappa(M)}$ and none of the two cells of Bc is reversed in orientation, Bc after perturbation is still Delaunay.

Proof: To prove that Bc after perturbation is still Delaunay, we need to show that the orientation of the lifted bi-cell, as a tetrahedron, is preserved, that is, $|M + \Delta M|$ is positive, where $|\cdot|$ stands the determinant of a matrix. The proof is by contradiction. Construct a function $f(t) = |M + t \cdot \Delta M| : \mathbb{R} \to \mathbb{R}$. Since $f(t)$ is a polynomial, it is continuous over t.

Suppose $|M + \Delta M|$ is negative. Since $f(0)$ is positive and $f(1)$ is negative, there exists $0 < t_0 < 1$ such that $f(t_0) = 0$. However, since $\|\Delta M\| < \frac{\|M\|}{\kappa(M)}$ and $0 < t_0 < 1$, we have $\|t_0 \cdot \Delta M\| = t_0 \cdot \|\Delta M\| < \frac{\|M\|}{\kappa(M)}$; hence $M + t_0 \cdot \Delta M$ is nonsingular. So, $f(t_0) = |M + t_0 \cdot \Delta M| \neq 0$, a contradiction.

We conclude that Bc after perturbation is still Delaunay. □

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

Note that $\|\Delta M\| < \frac{\|M\|}{\kappa(M)}$ is a sufficient condition to ensure that the bi-cell Bc is Delaunay after perturbation, given the triangulation is embedded.

3.3. Safe Zone of a Bi-cell – Computation and Properties

To compute the safe zone efficiently, we should choose a suitable matrix norm. Although the 2-norm is often used in theoretical analysis, our tests show that the 1-norm and ∞-norm lead to more efficient implementation. We choose to use the ∞-norm in all experiments presented in this paper.

Given an interior bi-cell Bc in $\mathcal{DT}(S)$ composed of $\triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$ and $\triangle\mathbf{p}_1\mathbf{p}_3\mathbf{p}_4$, since $\kappa(M) = \|M\| \cdot \|M^{-1}\|$, the sufficient condition is simplified as $\|\Delta M\| < \frac{1}{\|M^{-1}\|}$, where the precomputed $\|M^{-1}\|$ is updated only when it becomes invalid. Here, we call $\frac{1}{\|M^{-1}\|}$ the **safe tolerance** of Bc. The ∞-norm of ΔM is the maximum absolute row sum, that is, $\max\{|\delta x_i| + |\delta y_i| + |\Delta_i|, i = 1,2,3,4\}$.

3.3.1. Translation Invariance

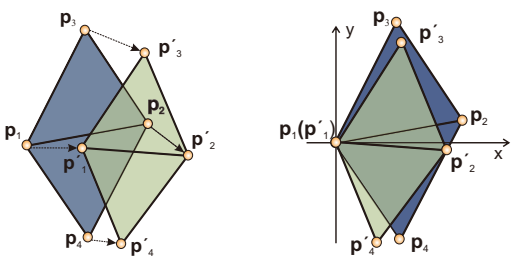


Figure 6: *Bi-cell translation after perturbation. (a) The bi-cell* $\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3\mathbf{p}_4$ *and after perturbation, it becomes* $\mathbf{p}'_1\mathbf{p}'_2\mathbf{p}'_3\mathbf{p}'_4$ *(b) To make the computation more efficient, we translate both bi-cells by moving their first points (*$\mathbf{p}_1$ *and* $\mathbf{p}'_1$*, resp.) to the origin.*

To make the computation of safe zones and checking more efficient, we perform a preprocessing step to translate both the original bi-cell and the perturbed bi-cell by moving their first points ($\mathbf{p}_1$ and $\mathbf{p}'_1$, resp.) to the origin. After this translation, three leftmost entries in the first row of M and ΔM are zero.

Our bi-cell filtering operation possesses the following property called *translation invariance*.

Property 1 (Translation Invariance) Suppose that the perturbation applied to the bi-cell Bc is a translation, that is, $\mathbf{p}'_i = \mathbf{p}_i + \delta\mathbf{p}, i = 1,2,3,4$. Then, with the translation processing, $\Delta M = \mathbf{0}$.

It is not hard to see that scale invariance holds also. First apply a scaling to make the longest (or shortest) edges of the bi-cells before and after perturbation to have equal length. If the perturbation is a scaling, we have $\Delta M = \mathbf{0}$ since the bi-cells before and after perturbation are congruent. However, we choose not to apply the scaling preprocessing since it involves square root computation.

3.3.2. Boundary Handling

Following [dCTAD09], we add *a point at infinity*, also known as *infinite point*, to simplify the handling of boundary. All points on the convex hull of S have *infinite edges* connected with infinite point, and then, each edge on the convex hull is adjacent to two triangles including one *infinite triangle* and each *infinite edge* is adjacent to two *infinite triangles*.

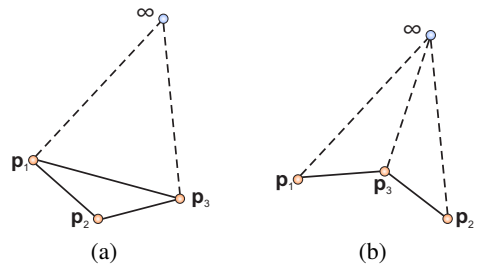


Figure 7: *Safe zone computation for infinite bi-cells. (a) A two-dimensional boundary bi-cell composed of one finite triangle and one infinite triangle; (b) Another two-dimensional bi-cell composed of two infinite triangles.*

A boundary bi-cell has at least one cell adjacent to infinite point. There are two types of boundary bi-cells in two dimensions. Figure 7 (a) shows a boundary bi-cell composed of one finite triangle $\triangle\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$ and one infinite triangle $\triangle\mathbf{p}_1\mathbf{p}_3\infty$. Figure 7 (b) shows another type boundary bi-cell composed of two infinite triangles $\triangle\mathbf{p}_1\mathbf{p}_2\infty$ and $\triangle\mathbf{p}_2\mathbf{p}_3\infty$. Delaunay condition of the boundary bi-cell shown in Figure 7 (a) (resp. (b)) is that the determinant of the matrix below is positive (resp. negative):

$$\begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix}.$$

The matrix above is called the corresponding matrix M of a boundary bi-cell. ΔM of a boundary bi-cell is

$$\begin{pmatrix} \delta x_1 & \delta y_1 & 0 \\ \delta x_2 & \delta y_2 & 0 \\ \delta x_3 & \delta y_3 & 0 \end{pmatrix}.$$

4. Bi-cell Filtering Algorithm

The input of the bi-cell filtering algorithm is a point set S, its Delaunay triangulation $\mathcal{DT}(S)$ in which the safe zone of each bi-cell is precomputed and the perturbed point set S'. The output of the algorithm is the Delaunay triangulation of the perturbed set S', $\mathcal{DT}(S')$. Our algorithm extends the potential of flipping algorithm by using a new method to first filter most bi-cells which are Delaunay. We construct

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

and maintain a bi-cell list for $\mathcal{DT}(S)$ denoted as **bc_list**. For each bi-cell $\Diamond\mathcal{B}$, we store a number for bi-cell's tolerance: ω, and reference positions for the vertices. In two dimensions, nine *double* variables (one for ω and eight for coordinates of four vertices) and four *int* variables (indices of four vertices) are stored for each bi-cell. Our algorithm is shown in Algorithm 1.

Remark: In the bi-cell filtering algorithm, if the triangulation is not embedded, that is, some cells are reversed (the related vertices are also called reversed vertices), Delaunay triangulation cannot be obtained by flipping. So we check orientations of all the cells of the triangulation before bi-cell filtering. If one cell fails the checking, we call rebuilding to construct Delaunay triangulation.

Algorithm 1 Updating Delaunay triangulation by bi-cell filtering

```
update all vertices of DT(S) with perturbed points.
check the orientations of all cells in DT(S).
if the triangulation is embedded then
  for all bi-cells {◊B} ∈ bc_list do
    if ◊B is out of its safe zone ω then
      insert ◊B into a list Q;
      repeat
        popup ◊B_t from the top of Q;
        if ◊B_t is not Delaunay then
          if ◊B_t is flippable then
            flip ◊B_t;
            update the affected bi-cells Σ◊B_n of ◊B_h
            due to the flipping;
            insert Σ◊B_n into Q;
            ω_t ← new tolerance of ◊B_t;
          end if
        else
          ω_t ← new tolerance of ◊B_t;
        end if
      until Q is EMPTY
    end if
  end for
else
  call rebuilding algorithm to update DT(S);
  update bc_list;
  exit Algorithm 1;
end if
```

5. Implementation and Experiments

In this section, we give the implementation details of the bi-cell filtering algorithm and show experiments to compare it with the vertex filtering method [dCTAD09] and the rebuilding method [CGA].

5.1. Implementation Details

Our implementation uses *Triangulation_3* in *CGAL version 3.4* [CGA] and *exact predicates inexact constructions* kernel [BFG*08] to compute Delaunay triangulation in *rebuilding* method. In *vertex filtering* and *bi-cell filtering* methods, we use *double* precision in safe zone computation and use *exact predicate* supplied by *CGAL* to check the Delaunay condition in our algorithm. All experiments are run on a workstation with an Intel Xeon 3.16 GHz CPU, 8GB memory. The operating system is 64-bit Windows 7, Visual Studio 2008 (in release mode with "-O2" option). Note that we only collect timing data on computing Delaunay triangulations, without including the time spent updating data points, which is application specific, such as using Lloyd's iteration in CVT computation.

5.2. Experiments

5.2.1. Data Sets

To compare the performance of the chosen methods, we generate several data sets from typical applications of Delaunay triangulation, that is, variational mesh generation and remeshing [YLL*09, LWL*09, YWLL10]. The energy function of centroidal Voronoi tessellation (CVT) [DFG99] is defined as:

$$\mathbf{F}(\mathbf{X}) = \sum_{i=1}^{n} \int_{V_i} \rho(\mathbf{x}) \|\mathbf{x} - \mathbf{x}_i\|^2 d\sigma,$$

where $\rho(\mathbf{x})$ is a density function over the domain. In these applications, minimizing the CVT function distributes points evenly in a two- or three-dimensional domain and restricted Voronoi diagram [ES97] is computed to find connectivities of these points. As reported in [YLL*09, LWL*09], the step to build Delaunay triangulation is often the bottleneck in computation. The Lloyd method [Llo82] is a common approach to computing CVT. We use several sets of points generated by the Lloyd method to set up experiments for comparison.

5.2.2. Comparison on Two-dimensional Meshing

We choose a regular 32-sided polygon as the domain with two different density functions, $\rho(\mathbf{x}) = 1$ and $\rho(\mathbf{x}) = \mathbf{x}^2$. The comparison is shown in Figure 8. With uniform density, vertex filtering is the fastest. With nonuniform density, bi-cell filtering is better than vertex filtering. Rebuilding is the slowest one. We observe that the performance of both vertex filtering and bi-cell filtering is approximately inversely proportional to the percentage of filtering failure. Overall, the bi-cell filtering method and the vertex filtering method have comparable performances. When the density function is nonuniform and the initialization is bad, large displacement of points slows down the vertex filtering algorithm due to the high percentage of filtering failure. However, in this case, benefiting from the translation invariance property 1, bi-cell filtering performs better.

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

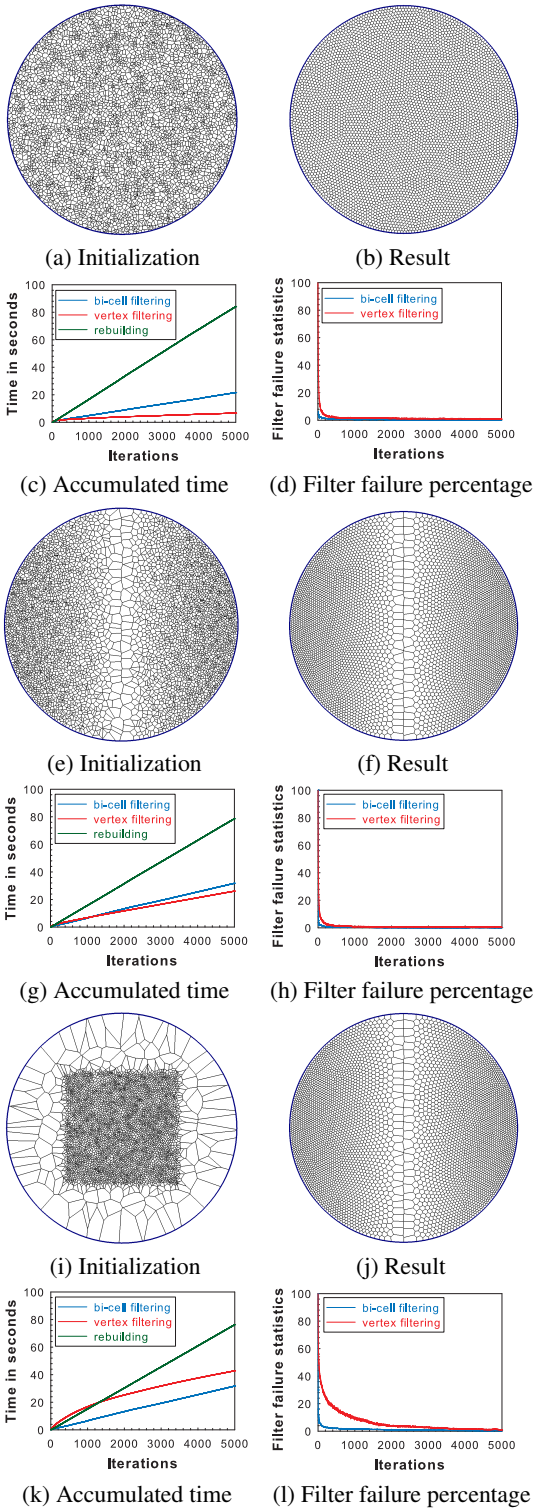


Figure 8: *5,000 points in a two-dimensional regular 32-gon domain with 5,000 Lloyd iterations. (a)∼(d) are of uniform density. (e)∼(l) are of density* $\rho = x^2$.

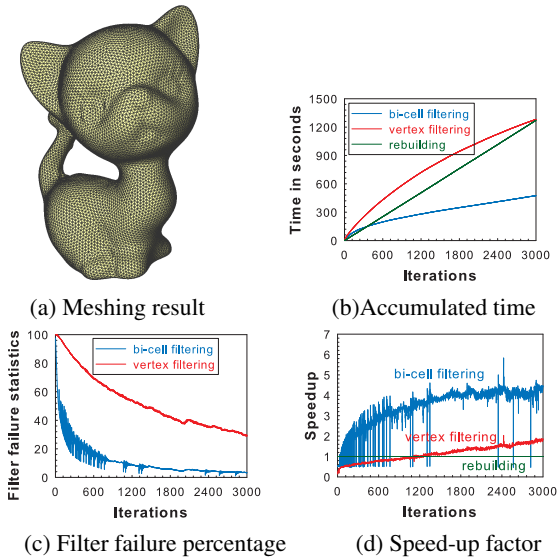


Figure 9: *Comparison on the kitten model with 15,000 points and 3,000 Lloyd iterations. Rebuilding is triggered 146 times in 3,000 iterations by bi-cell filtering.*

5.2.3. Comparison on Surface Remeshing

The data is generated with the remeshing algorithm in [YLL*09]. Note that, our algorithm returns the three-dimensional Delaunay triangulation of the data points, the restricted Delaunay triangulation (RDT) [ES97] is extracted for surface remeshing. The result is shown in Figure 9. Bi-cell filtering is the fastest. *Speed up factor* records the speed-up achieved over the rebuilding algorithm. When reversed cells occur, we call rebuilding to update Delaunay triangulation, then, the time cost is higher than rebuilding in that iteration since the bi-cell list is rebuilt.

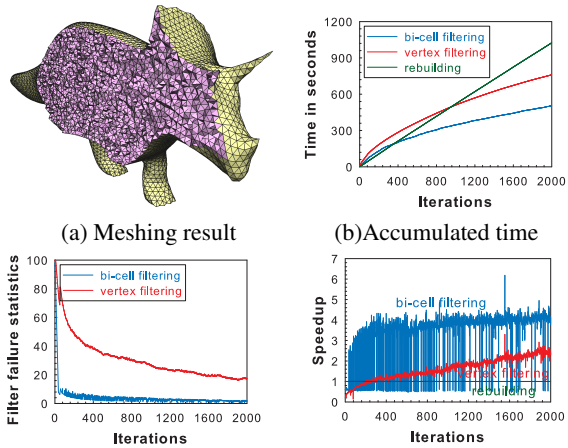
5.2.4. Comparison on Three-dimensional Meshing

All data of three-dimensional meshing are generated by the framework in [YWLL10]. Figure 10 and 11 compare three algorithms on two models. Bi-cell filtering is the fastest on both models. In the converging stage, the percentage of bi-cells failing the filtering test is typically 1% to 5%.

We also compare these approaches with different initialization on the same model. See Figure 12. One initialization is uniform and the other is nonuniform. With nonuniform initialization, bi-cell filtering outperforms rebuilding after 300 iterations. Vertex filtering outperforms rebuilding after several thousand iterations.

All these three methods are scalable with different number of points, as shown in Figure 13. On the bunny model, the comparison of vertex filtering and bi-cell filtering is similar with different number of points, from 10,000 to 30,000, increased by 5,000 each time. The statistics of the average time of each iteration can be found in Table 1.

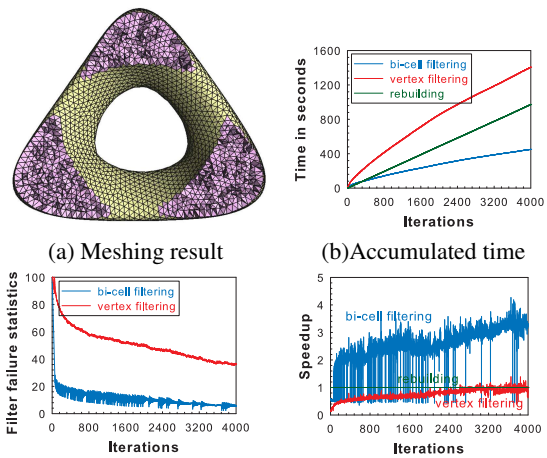
© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.



(a) Meshing result (b)Accumulated time

(c) Filter failure percentage (d) Speed-up factor

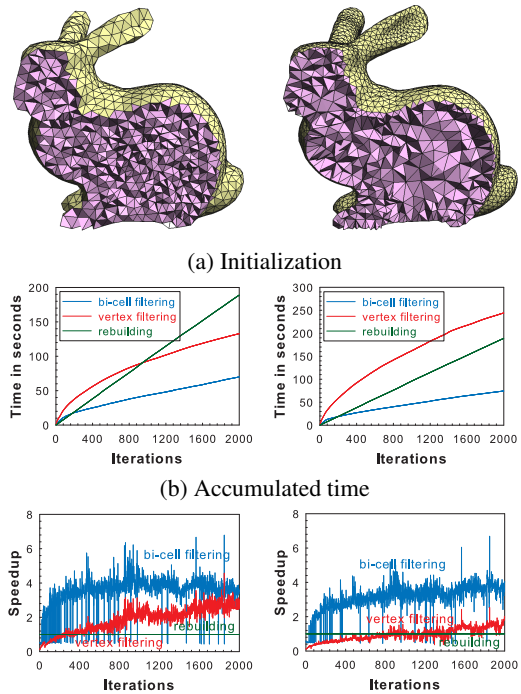
Figure 10: *Comparison on the triceratops model with 25,000 points and 2,000 Lloyd iterations. Rebuilding is triggered 298 times in 2,000 iterations by bi-cell filtering.*



(a) Meshing result (b)Accumulated time

(c) Filter failure percentage (d) Speed-up factor

Figure 11: *Comparison on the genus3 model with 13,000 points and 4,000 Lloyd iterations. Rebuilding is triggered 187 times in 4,000 iterations by bi-cell filtering.*



(a) Initialization

(b) Accumulated time

(c) Speed-up factor

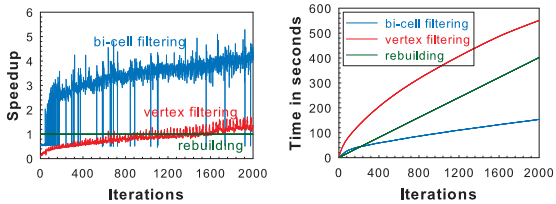
Figure 12: *Comparison on the bunny model with uniform and non-uniform sampling with 5,000 points and 2,000 Lloyd iterations. Left: Uniform; Right: Nonuniform.*

#seeds	*rebuilding*	*vertex filtering*	*bi-cell filtering*
10k	0.2009s	0.2755s	0.0767s
15k	0.2817s	0.3445s	0.1272s
20k	0.4308s	0.5167s	0.1731s
25k	0.5380s	0.6863s	0.2160s
30k	0.6592s	0.9293s	0.2695s

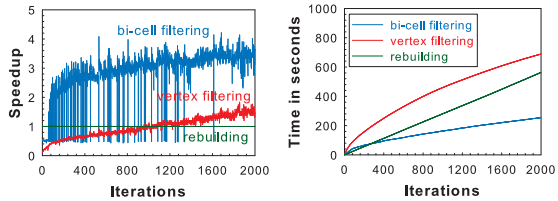
Table 1: *Average speed comparison: bunny model, 2,000 Lloyd iterations.*

Remark: In two dimensions, the vertex filtering approach [dCTAD09] is accelerated by flipping if the movement of one vertex does not cause its adjacent cell reversed. The bi-cell filtering method takes the advantage of the efficiency of flipping in both two and three dimensions. Experiments show that bi-cell filtering method always outperforms vertex filtering method in three dimensions. Compared with vertex filtering, the bi-cell filtering is recommended especially for practical applications involving three-dimensional Delaunay triangulations. In some applications in which most of the points are fixed but only a few ones, vertex filtering is still a good choice because its safe zone is defined on the vertex directly. Overall, bi-cell filtering will trigger rebuilding frequently in the beginning iterations. So we can run rebuilding directly for the first few iterations then switch to bi-cell filtering algorithm. Experimentally, one can count the number of reversed cells, when the percentage of reversed cells drops below 10%, one can switch to bi-cell filtering algorithm for updating. The time of our algorithm comprises *reverse checking* of all cells, *bi-cell filtering checking*, *bi-cell Delaunay checking*, *bi-cell flipping* and *bi-cell updating*. We show in Figure 14 the time breakdown of our algorithm and compare it with the time of checking the Delaunay conditions of all bi-cells. From this comparison, one can see the efficiency and effectiveness of the bi-cell filtering.

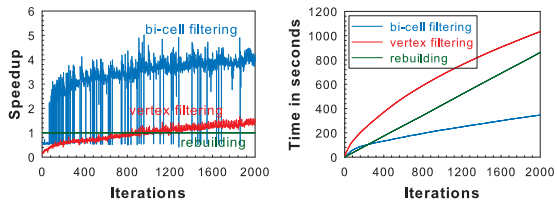
© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.



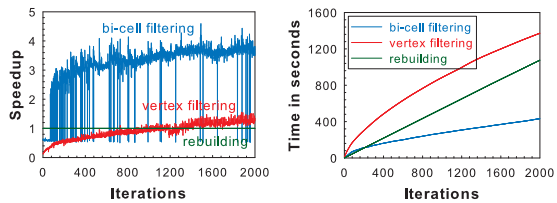
(a) 10,000 points



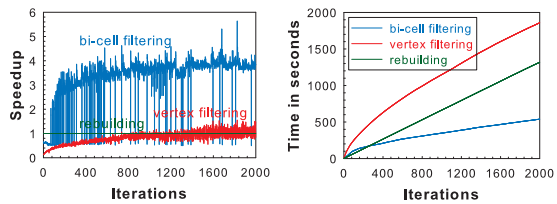
(b) 15,000 points



(c) 20,000 points



(d) 25,000 points



(e) 30,000 points

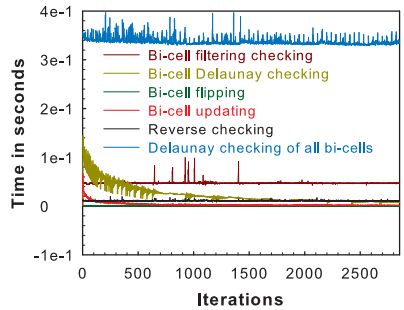
Figure 13: *Comparison on bunny model with different numbers of seeds with 2,000 Lloyd iterations. Left: Speed-up factor; Right: Accumulated time.*

6. Conclusion and Future Work

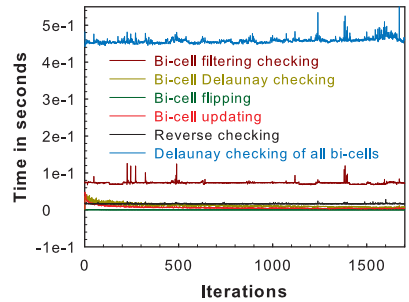
We address the problem of updating Delaunay triangulations of moving points from an algebraic point of view and present an efficient algorithm. An effective tolerance is defined on bi-cell with theoretical foundation in matric theory. The new algorithm filters most Delaunay bi-cells and uses flipping to amend non-Delaunay bi-cells, and enjoys a nice property: *translation invariance*.

When a triangulation is not embedded, rebuilding is triggered to construct Delaunay triangulation. In the future, we will investigate this problem further by trying to use more efficient local repairs to modify the triangulation to make it embedded.

Whether flipping works for constructing Delaunay triangulations in three and four dimensions is still open [San00]. We are interested in studying this open problem and extending our bi-cell filtering algorithm to update four-dimensional Delaunay triangulation.



(a) Breakdown of running time (kitten model)



(b) Breakdown of running time (triceratops model)

Figure 14: *Comparison on running time of bi-cell filtering algorithm and Delaunay checking time without filtering.*

Acknowledgements

We thank AIM@SHAPE Project [Aim] for kindly providing three-dimensional models used in this paper. We would like to thank Pedro Machado for providing the code of vertex filtering algorithm. We thank Dong-ming Yan for the mesh generation and remeshing programs and many helpful discussions during this work. We also thank the reviewers for constructive and detailed comments. The work of W. Wang is partially supported by the Research Grant Council of Hong Kong (project no.: 718209 and 718010). The work of C. Zhang is partially supported by the National Basic Research Program of China(973 Program) (2006CB303102), the State Key Program of NSFC project (60933008), and the National 863 High-Tech Program of China (2009AA01Z304).

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

References

[ACSYD05] ALLIEZ P., COHEN-STEINER D., YVINEC M., DESBRUN M.: Variational tetrahedral meshing. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses* (New York, NY, USA, 2005), ACM, p. 10. 1

[AdVDI05] ALLIEZ P., DE VERDIÈRE E. C., DEVILLERS O., ISENBURG M.: Centroidal Voronoi diagrams for isotropic surface remeshing. *Graph. Models 67*, 3 (2005), 204–231. 1

[Aim] Shape repository of AIM@SHAPE project. http://shapes.aimatshape.net. 9

[BDP*02] BOISSONNAT J.-D., DEVILLERS O., PION S., TEILLAUD M., YVINEC M.: Triangulations in CGAL. *Computational Geometry 22*, 1-3 (2002), 5 – 19. 1

[BFG*08] BRONNIMANN H., FABRI A., GIEZEMAN G.-J., HERT S., HOFFMMANN M., HETTNER L., SCHIRRA S., PION S.: 2D and 3D geometry kernel, 2008. In CGAL User and Reference Manual. 6

[CG04] CAZALS F., GIESEN J.: *Delaunay Triangulation Based Surface Reconstruction: Ideas and Algorithms*. Tech. Rep. RR-5393, INRIA, 11 2004. 1

[CGA] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org. 1, 2, 6

[DBC07] DEBARD J.-B., BALP R., CHAINE R.: Dynamic Delaunay tetrahedralisation of a deforming surface. *Vis. Comput. 23*, 12 (2007), 975–986. 1

[dBvKOS97] DE BERG M., VAN KREVELD M., OVERMARS M., SCHWARZKOPF O.: *Computational Geometry: Algorithms and Applications*, 1 ed. Springer, July 1997. 2, 4

[dCTAD09] DE CASTRO P. M. M., TOURNOIS J., ALLIEZ P., DEVILLERS O.: Filtering relocations on a Delaunay triangulation. *Computer Graphics Forum 28*, 5 (July 2009), 1465–1474. 2, 3, 5, 6, 8

[Dev02] DEVILLERS O.: The Delaunay hierarchy. *International Journal of Foundations of Computer Science 13* (2002), 163–180. 2

[DFG99] DU Q., FABER V., GUNZBURGER M.: Centroidal Voronoi tessellations: applications and algorithms. *SIAM Review 41* (1999), 637–676. 6

[EAD*06] EDELSBRUNNER H., ABLOWITZ M. J., DAVIS S. H., HINCH E. J., ISERLES A., OCKENDON J., OLVER P. J.: *Geometry and Topology for Mesh Generation (Cambridge Monographs on Applied and Computational Mathematics)*. Cambridge University Press, New York, NY, USA, 2006. 1

[ES97] EDELSBRUNNER H., SHAH N.: Triangulating topological spaces. *International Journal of Computational Geometry Applications 7*, 4 (1997), 365–378. 6, 7

[For86] FORTUNE S.: A sweepline algorithm for Voronoi diagrams. In *SCG '86: Proceedings of the second annual symposium on Computational geometry* (New York, NY, USA, 1986), ACM, pp. 313–322. 2

[GR04] GUIBAS L., RUSSEL D.: An empirical comparison of techniques for updating Delaunay triangulations. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry* (New York, NY, USA, 2004), ACM, pp. 170–179. 3

[GS85] GUIBAS L., STOLFI J.: Primitives for the manipulation of general subdivisions and the computation of Voronoi. *ACM Trans. Graph. 4*, 2 (1985), 74–123. 4

[Gui98] GUIBAS L. J.: Kinetic data structures: a state of the art report. In *WAFR '98: Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics : the algorithmic perspective* (Natick, MA, USA, 1998), A. K. Peters, Ltd., pp. 191–209. 3

[GVL96] GOLUB G. H., VAN LOAN C. F.: *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996. 4

[Joe89] JOE B.: Three-dimensional triangulations from local transformations. *SIAM J. Sci. Stat. Comput. 10*, 4 (1989), 718–741. 2

[LJ05] LIU Y., JACK S.: A comparison of five implementations of 3D Delaunay tessellation. In *Combinatorial and Computational Geometry*, Jacob E. G., Janos P., Emo W., (Eds.). Cambridge University Press, 2005, pp. 439–458. 2

[Llo82] LLOYD S. P.: Least squares quantization in PCM. *IEEE Transactions on Information Theory 28*, 2 (1982), 129–137. 6

[LWL*09] LIU Y., WANG W., LÉVY B., SUN F., YAN D.-M., LU L., YANG C.: On centroidal Voronoi tessellation—energy smoothness and fast computation. *ACM Trans. Graph. 28*, 4 (2009), 1–17. 1, 6

[Mun93] MUNKRES J. R.: *Elements of Algebraic Topology*. Westview Press, December 1993. 2

[PS93] PREPARATA F. P., SHAMOS M. I.: *Computational Geometry: An Introduction (Monographs in Computer Science)*. Springer, August 1993. 2

[Rus07] RUSSELL D.: *Kinetic data structures in practice*. PhD thesis, Stanford University, Stanford, CA, USA, 2007. Adviser-Guibas, Leonidas. 3

[San00] SANTOS F.: A point set whose space of triangulations is disconnected. *J. Amer. Math. Soc 13*, 3 (2000), 611–637. 9

[Sch07] SCHAAP W. E.: *DTFE: the Delaunay Tessellation Field Estimator*. PhD thesis, University of Groningen, 2007. 1

[She96] SHEWCHUK J. R.: Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, Lin M. C., Manocha D., (Eds.), vol. 1148 of *Lecture Notes in Computer Science*. Springer-Verlag, May 1996, pp. 203–222. From the First ACM Workshop on Applied Computational Geometry. 1

[She05] SHEWCHUK R.: Star splaying: an algorithm for repairing delaunay triangulations and convex hulls. In *SCG '05: Proceedings of the twenty-first annual symposium on Computational geometry* (New York, NY, USA, 2005), ACM, pp. 237–246. 3

[TWAD09] TOURNOIS J., WORMSER C., ALLIEZ P., DESBRUN M.: Interleaving Delaunay refinement and optimization for practical isotropic tetrahedron mesh generation. *ACM Trans. Graph. 28*, 3 (2009), 1–9. 1

[Wag36] WAGNER K.: Bemerkungen zum vierfarbenproblem. *Jber. Deutsch. Math.-Verein. 46* (1936), 26–32. 2

[YLL*09] YAN D.-M., LÉVY B., LIU Y., SUN F., WANG W.: Isotropic remeshing with fast and exact computation of restricted Voronoi diagram. *Comput. Graph. Forum 28*, 5 (2009), 1445–1454. 1, 6, 7

[YWLL10] YAN D.-M., WANG W., LÉVY B., LIU Y.: Efficient computation of 3d clipped Voronoi diagram. In *Geometric Modeling and Processing (GMP 2010)* (2010), pp. 269–282. 1, 6, 7

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.