Report Number: WUCSE-2012-45

2012

Delaunay-restricted Optimal Triangulation of 3D Polygons

Ming Zou, Tao Ju, and Nathan Carr

Triangulation of 3D polygons is a well studied topic of research. Existing methods for finding triangulations that minimize given metrics (e.g., sum of triangle areas or dihedral angles) run in a costly O(n4) time [BS95,BDE96], while the triangulations are not guaranteed to be free of intersections. To address these limitations, we restrict our search to the space of triangles in the Delaunay tetrahedralization of the polygon. The restriction allows us to reduce the running time down to O(n2) in practice (O(n3) worst case) while guaranteeing that the solutions are intersection free. We demonstrate experimentally that the reduced search space... Read complete abstract on page 2.

Delaunay-restricted Optimal Triangulation of 3D Polygons

Ming Zou, Tao Ju, and Nathan Carr

Complete Abstract:

Triangulation of 3D polygons is a well studied topic of research. Existing methods for finding triangulations that minimize given metrics (e.g., sum of triangle areas or dihedral angles) run in a costly O(n4) time [BS95,BDE96], while the triangulations are not guaranteed to be free of intersections. To address these limitations, we restrict our search to the space of triangles in the Delaunay tetrahedralization of the polygon. The restriction allows us to reduce the running time down to O(n2) in practice (O(n3) worst case) while guaranteeing that the solutions are intersection free. We demonstrate experimentally that the reduced search space is not overly restricted. In particular, triangulations restricted to this space usually exist for practical inputs, and the optimal triangulation in this space approximates well the optimal triangulation of the polygon. This makes our algorithms a practical solution when working with real world data.

2012-45

Delaunay-restricted Optimal Triangulation of 3D Polygons

Authors: Ming Zou, Tao Ju, and Nathan Carr

Corresponding Author: taoju@cse.wustl.edu

Abstract: Triangulation of 3D polygons is a well studied topic of research. Existing methods for finding triangulations that minimize given metrics (e.g., sum of triangle areas or dihedral angles) run in a costly $O(n^4)$ time \cite{Barequet95,Barequet96}, while the triangulations are not guaranteed to be free of intersections. To address these limitations, we restrict our search to the space of triangles in the Delaunay tetrahedralization of the polygon. The restriction allows us to reduce the running time down to $O(n^2)$ in practice ($O(n^3)$ worst case) while guaranteeing that the solutions are intersection free. We demonstrate experimentally that the reduced search space is not overly restricted. In particular, triangulations restricted to this space usually exist for practical inputs, and the optimal triangulation in this space approximates well the optimal triangulation of the polygon. This makes our algorithms a practical solution when working with real world data.

Notes:
Work submitted to but not accepted by SGP'12.

Type of Report: Other

Volume xx (200y), Number z, pp. 1–11

Delaunay-restricted Optimal Triangulation of 3D Polygons

Ming Zou[1], Tao Ju[1], Nathan Carr[2]

[1]WUSTL [2]Adobe Systems

Abstract

Triangulation of 3D polygons is a well studied topic of research. Existing methods for finding triangulations that minimize given metrics (e.g., sum of triangle areas or dihedral angles) run in a costly $O(n^4)$ time [BS95, BDE96], while the triangulations are not guaranteed to be free of intersections. To address these limitations, we restrict our search to the space of triangles in the Delaunay tetrahedralization of the polygon. The restriction allows us to reduce the running time down to $O(n^2)$ in practice ($O(n^3)$ worst case) while guaranteeing that the solutions are intersection free. We demonstrate experimentally that the reduced search space is not overly restricted. In particular, triangulations restricted to this space usually exist for practical inputs, and the optimal triangulation in this space approximates well the optimal triangulation of the polygon. This makes our algorithms a practical solution when working with real world data.

1. Introduction

Given a 3D polygon (a closed loop of non-intersecting line segments), a *triangulation* of the polygon is a set of non-intersecting triangles that connect the vertices on the polygon and form a disk-like surface bounded by the polygon [BDE96]. Computing the triangulation is an important step in many geometry processing tasks, such as computing discrete minimal surfaces [PP93], lofting surfaces from curve sketches [NISA07, JC08], and filling holes on scanned data [BS95, Lie03]. The triangulation usually serves as the base domain that allows for further refinement and smoothing.

If many triangulations are available, often times the one that minimizes some metric is sought. A triangulation with minimal sum of triangle areas would be desirable for computing discrete minimal surfaces. On the other hand, minimizing the squared sum of dihedral angles between adjacent triangles produces fair surfaces that are ideal for sketch-based modeling [RSW*07]. For hole-filling, a triangulation whose normals match with those in the surrounding triangles is preferred [Lie03]. We consider an *optimal triangulation* as one that minimizes the sum of some weighted combination of per-triangle metrics (e.g., area, perimeter, and normal derivation from pre-existing triangles) and bi-triangle metrics (e.g., square sum of dihedrals).

Finding triangulations is a difficult problem. Barequet et al. showed that deciding if a 3D polygon is triangulable is an NP-hard problem [BDE96], not to mention computing the optimal triangulation. To make the problem tractable, previous works typically ignore the non-intersecting requirement of the triangulation [BS95, Lie03]. In this case, the optimal triangulation can be computed in polynomial time by a simple extension of the classical dynamic programming algorithm for triangulating a 2D polygon [Gil79, Kli80]. Besides creating possibly intersecting surfaces, an important practical limitation of this approach is its prohibitive cost. If bi-triangle metrics are used in the objective function, the dynamic programming algorithm would take $O(n^4)$ time and $O(n^3)$ space for n polygon vertices. In our implementation, we found that the algorithm would take minutes for just a few hundred vertices and easily exceed memory limits on desktop computers for slightly larger inputs.

In this paper, we present a significantly more efficient way to compute non-intersecting, close-to-optimal triangulations. The key observation behind our method is that the more likely set of surface connections occurs between parts of the polygon that are close in space. We use this observation to restrict our search for triangulations over the space of triangles that arise from the Delaunay tetrahedralization of the polygon vertices. The restriction brings several benefits. First, any triangulation in this space is guaranteed to be free of self-intersections. Second, the optimal triangulation in this space, which we call the *Delaunay-restricted optimal triangulation* (DOT), can be computed in $O(n^3)$ time in the worse case and $O(n^2)$ space using dynamic programming.

In practice, we notice that time complexity is $O(n^2)$ or better for most inputs, and our implementation can triangulate polygons with thousands of vertices in seconds while consuming little memory. Last but not least, we have observed in extensive experiments that the DOTs exist for a broad range of input polygons and are reasonable approximations of the optimal triangulations.

Our use of the Delaunay complex is akin in spirit to the work of Shamos and Hoey [SH75], who used the Voronoi diagram (dual of the Delaunay complex) to speed up various proximity-based tasks in 2D. While the idea is simple, we believe that our method is the first practical and robust (albeit approximate) solution to the problem of optimal triangulation of 3D polygons. Our technical contributions are:

- We give detailed algorithms for computing DOT by extending a previous algorithm for triangulating a 2D polygon (Section 3).
- We perform theoretical analysis and experimental validations of the properties of the algorithms, including the complexity, the existence of the solutions, and the quality of the triangulations (Section 4,5).

2. Previous Work

The triangulation of a 2D polygonal domain is a well-studied problem in computational geometry [ES96]. A triangulation exists for every simple polygon. Gilbert [Gil79] and Klincsek [Kli80] independently developed an $O(n^3)$-time and $O(n^2)$-space algorithm for computing the triangulation that minimizes that sum of total edge lengths (also known as the *minimum weight triangulation*). The algorithm, based on dynamic programming, constructs the optimal triangulation of a larger domain from the optimal triangulations of smaller sub-domains. The algorithm has been extended to handle hole vertices within the polygon [GBL05] but the time complexity scales exponentially with the number of holes. Note that the general problem of minimum weight triangulation of a set of 2D points is *NP*-hard [MR06].

The triangulability of a 3D polygon was studied by Barequet et al. [BDE96], who proved its *NP*-hardness by reduction from 3-*SAT*. The paper studied the triangulability of several special classes of polygons. In particular, a knotted polygon has no triangulation, and any polygon with a non-intersecting projection on a plane or sphere can be triangulated. However, it is not clear from this work whether triangulations exist for polygons that arise in practical applications, such as curve sketches or hole boundaries. One of the contributions of this current work is giving empirical evidences that DOT (which is a triangulation) exists in most practical cases.

If self-intersections are allowed in a triangulation, one can easily extend the method of Gilbert and Klincsek to find an optimal triangulation in $\mathbb{R}^3$ that minimizes the sum of per-triangle metrics, such as area [BS95, Lie03]. The extension has the same time and space complexity as the 2D algorithm. To yield smoother surfaces with minimal twisting, Liepa attempted to minimize the sum of dihedral angles between neighboring triangles [Lie03]. His algorithm, however, uses a greedy heuristic that does not guarantee to find the optimal triangulation. Barequet et al. [BDE96] mentioned an $O(n^4)$-time and $O(n^3)$-space extension of the method of Gilbert and Klincsek that can find the triangulation that minimizes bi-triangle metrics. However, we are not aware of any implementation of this extension for practical applications, which is probably due to the high computational cost. In the literature of hole-filling, greedy heuristics guided by distances [MD93] or angles [WLG03, VPK05] are often used for triangulating a hole boundary. However, these heuristics typically do not have guarantees of optimality.

Several methods are capable of creating intersection-free triangulations, but they are either applicable to a narrow range of input curves or limited in the range of output triangulations. When the polygon is sufficiently planar, one can first project the polygon to a best-fitting plane, triangulate the planar projection, and finally lift the triangles to 3D [RW97]. However, the planarity condition is overly strict for practical spatial polygons, whether they are curve sketches or hole boundaries. On the other hand, the method of Rose et al. [RSW*07] can produce a surface from any curve network by iteratively extracting the surface on the convex hull and computing the convex hull of the remaining curve parts. While the method generates appealing surfaces for sketch inputs, the output surfaces are limited to those that are close to being developable. It is not suited for finding surfaces that minimize other objective functions (e.g., minimum area). In contrast, our algorithms produces surfaces minimizing any choice of per-triangle and bi-triangle metrics, and it can be applied to complex, highly non-planar 3D polygons.

While the paper is only concerned with the problem of triangulation, there are various techniques for directly creating a smooth surface that spans a spatial curve, without the need of an initial triangulated domain, using techniques such as Radial Basis Functions [BPB06, BMS*10], Moving-Least Squares [WO07] and B-spline fitting [KSI*07]. In the literature of hole-filling, the surrounding geometry of a hole or prior knowledge of the shape can be utilized to further improve the geometry of the filling patch; see the survey in [Ju09]. We omit detailed review of these methods as they are beyond the scope of this paper.

Our work builds on the large body of work and knowledge on Delaunay tetrahedralization. It is well-known that there are $O(n^2)$ simplices in the Delaunay tetrahedralization of n points in $\mathbb{R}^3$. This is a tight upper bound when the points are sampled along a spatial curve [AAD07], which is the type of inputs in our setting. The incremental flipping algorithm of Edelsbrunner and Shah [ES96] can produce the Delaunay tetrahedralization in expected time $O(n^2)$. In our work,

we use the implementation of this algorithm in the publicly available package Tetgen [Si09].

3. Algorithms

The input of our algorithms consist of any closed, non-intersecting 3D polygon C and the set of all triangle facets S in the Delaunay tetrahedralization of the vertices in C (which we call the *Delaunay triangles*). In addition, the user can specify any choice of per-triangle and/or bi-triangle metrics, as well as the weights to combine them. The output is the DOT, or none if there is no triangulation restricted to the Delaunay triangles (see more discussion in Section 4).

We will present two algorithms, one that handles only per-triangle metrics, and another that is less efficient but capable of handling both per-triangle and bi-triangle metrics. The first algorithm, *DOT*1, is a direct application of the 2D triangulation method of Gilbert and Klincsek [Gil79, Kli80] (GK) with minor modifications to use any per-triangle metric and to restrict the search to S. Compared with previous extensions of GK to 3D polygons [BS95, Lie03], *DOT*1 uses a restricted search space while these other works consider all possible triangles connecting any triplets of polygon vertices. The second algorithm, *DOT*2, extends *DOT*1 to handle bi-triangle metrics. The extension closely follows the strategy given in [BDE96] while restricting the search to S.

3.1. *DOT*1: handling per-triangle metrics

Since our objective function is an additive sum of metrics, the optimal triangulation of the whole curve (or a curve segment) consists of optimal triangulations of its segments (or sub-segments). This is the key idea behind the GK algorithm.

Let us represent a "segment" of the curve by a pair $\{e,\delta\}$, where e is some edge (which may or may not lie on the curve) present in the triangle set S and δ is a binary flag (0 or 1) denoting the "side" of the edge e where the curve segment lies (see Figure 1 (a)). We use the convention that the 0 side of an edge e on the input curve is the side where the rest of the curve lies. We can relate the "cost" (i.e., sum of metrics) of the optimal triangulation of this segment, $cost(e,\delta)$, to the cost of its sub-segments as:

$$cost(e,\delta) = \min(\infty, \min_{t\in\Pi[e,\delta]}(w(t) + \sum_{i=1,2} cost(e_i, \Delta(t,e_i)))) \quad (1)$$

Here, $\Pi[e,\delta]$ denotes the list of triangles in S incident to e and lying on the side δ, w is a per-triangle metric function, e_1, e_2 are the other two edges of the triangle t that are not e, and $\Delta(t,e)$ returns the side of e that is *opposite* to t (see Figure 1(a)). Note if $\Pi[e,\delta] = \emptyset$, the cost is infinity since there are no triangulations of the segment $\{e,\delta\}$.

The recursive property above leads naturally to a recursive algorithm for computing the optimal triangulation of C restricted to triangles in S. However, a naive implementation would easily have exponentially large complexity, as the depth of recursion can be as large as $O(\|C\|)$. The remedy, which leads to a polynomial time algorithm, is to avoid recomputing for the same curve segment $\{e,\delta\}$ multiple times, but rather to save the computed triangulation (and its cost) in memory for look-up by future calls.

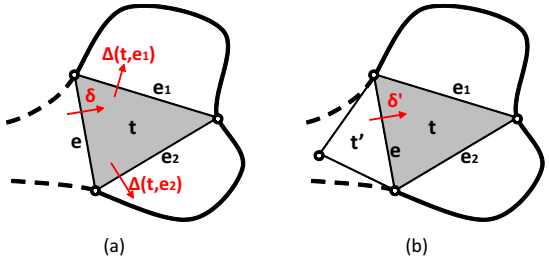

Figure 1: *Notations used in the algorithms.*

A pseudo-code of the algorithm is shown on the left of Figure 2. During initialization, the adjacency table Π is first built from the input triangles (which can be done in a single pass of S), and two tables $cost[], tri[]$ are set up. The entry $cost[e,\delta]$ will store the cost of the optimal triangulation of the curve segment $\{e,\delta\}$, and $tri[e,\delta]$ will store the triangle incident to e in the optimal triangulation. The core of the algorithm involves recursive calls to $COST()$ which follows Equation 1. Finally, the optimal triangulation is built based on information stored in the table $tri[]$ using another recursive function $TILE()$.

3.2. *DOT*2: handling per- and bi-triangle metrics

When bi-triangle metrics (e.g., dihedral angle) are present, the cost of a triangulation within a curve segment $\{e,\delta\}$ needs to be evaluated in the context of the triangle on the other side of e. Accordingly, we augment our representation of a curve segment to be an edge-triangle pair $\{e,t'\}$ where t' is an triangle incident to e that lies on the other side of the curve segment (see Figure 1 (b)). If the edge e lies on the curve and $t' = \emptyset$, the augmented segment is the entire input curve. A similar recursive relation to that in Equation 1 holds for the cost of the optimal triangulation of the augmented segment, $cost(e,t')$:

$$cost(e,t') = \min(\infty, \min_{t\in\Pi[e,\delta']}(w(t) + w(t,t') + \sum_{i=1,2} cost(e_i,t))) \quad (2)$$

where $\delta' = \Delta(t',e)$, and $w(t,t')$ is the bi-triangle metric function. If the edge e lies on the curve and $t' = \emptyset$, $\delta' = 0$.

The pseudo-code of the algorithm is shown on the right of Figure 2. It shares a similar structure to the previous algorithm *DOT*1. The primary differences are that the tables $cost[], tri[]$ are larger now and that the recursive functions

```
// Triangulating curve C restricted to triangles S
// with minimal one-triangle metrics w
DOT1(C,S)
    build edge-triangle adjacency table Π    // Initializing
    for each edge e in Π and δ = 0,1
        cost[e,δ] ← −∞,   tri[e,δ] ← null
    let e be the any edge on C    // Computing cost
    if COST(e,0) ≠ ∞
        return TILE(e,0)    // Building surface
    else return null

COST(e,δ)    // Get cost of tiling the segment {e,δ}
    if cost[e,δ] ≠ −∞    // Computed before
        return cost[e,δ]
    else if e is on C and δ ≠ 0    // Reaching the boundary
        return 0
    else    // Divide the segment
        cost[e,δ] ← ∞
        for each t ∈ Π[e,δ]
            let the edges of t be e,e1,e2
            c ← w(t) + COST(e1,Δ(t,e1)) + COST(e2,Δ(t,e2))
            if c < cost[e,δ]
                cost[e,δ] ← c,   tri[e,δ] ← t
        return cost[e,δ]

TILE(e,δ)    // Get triangles tiling the segment {e,δ}
    t ← tri[e,δ]
    let the edges of t be e,e1,e2
    return {t} ∪ TILE(e1,Δ(t,e1)) ∪ TILE(e2,Δ(t,e2))
```

```
// Triangulating curve C restricted to triangles S
// with minimal one-triangle and bi-triangle metrics w
DOT2(C,S)
    build edge-triangle adjacency table Π    // Initializing
    for each edge e in Π and t' ∈ Π[e,{0,1}]
        cost[e,t'] ← −∞,   tri[e,t'] ← null
    let e be the any edge on C    // Computing cost
    if COST(e,null) ≠ ∞
        return TILE(e,null)    // Building surface
    else return null

COST(e,t')    // Get cost of tiling the augmented segment {e,t'}
    if cost[e,t'] ≠ −∞    // Computed before
        return cost[e,t']
    else if e is on C and t' ≠ null    // Reaching the boundary
        return 0
    else    // Divide the segment
        cost[e,t'] ← ∞
        for each t ∈ Π[e,Δ(t',e)]
            let the edges of t be e,e1,e2
            c ← w(t) + w(t,t') + COST(e1,t) + COST(e2,t)
            if c < cost[e,t']
                cost[e,t'] ← c,   tri[e,t'] ← t
        return cost[e,t']

TILE(e,t')    // Get triangles tiling the augmented segment {e,t'}
    t ← tri[e,t']
    let the edges of t be e,e1,e2
    return {t} ∪ TILE(e1,t) ∪ TILE(e2,t)
```

Figure 2: *Pseudo-code of algorithms DOT1 and DOT2. The main differences are highlighted.*

$COST(), TILE()$ are called with different parameters that are edge-triangle pairs, instead of pairs of a single edge and a binary flag as in $DOT1$. The latter difference is the main reason of increased complexity of $DOT2$ over $DOT1$ (see next section).

4. Analysis

In this section, we give theoretical bounds on the complexity of the algorithms and discuss conditions of the input under which the algorithms will produce a triangulation.

4.1. Complexity

Our complexity bounds are closely tied to the size of S, which, as mentioned before, has complexity $O(n^2)$ for an input polygon with n vertices. We consider the complexity of space and time separately below.

Lemma 4.1 Both algorithms $DOT1, DOT2$ use $O(n^2)$ space.

Proof: The space usage in both programs is dominated by three tables, the adjacency table Π, the cost table $cost[]$ and the triangle table $tri[]$. The size of Π is a constant factor of the number of triangles in S, and has $O(n^2)$ complexity. In $DOT1$, the size of $cost[]$ and $tri[]$ is proportional to the number of edges incident to triangles in S. In $DOT2$, their size is same as that of Π. In both cases, the size of $cost[], tri[]$ is bounded by $O(n^2)$.□

Lemma 4.2 The algorithms $DOT1, DOT2$ have worst case time complexity of $O(n^2)$ and $O(n^3)$, respectively.

Proof: We first consider algorithm $DOT1$. The initialization stage has time complexity of $O(n^2)$, as it involves a single pass over all triangles and the tables $cost[], tri[]$, all of which are bounded in size by $O(n^2)$. In the main part, each $COST(e,\delta)$ uses constant time except when it is first called, in which case it spawns $2\|\Pi[e,\delta]\|$ number of other recursive calls. So the total time complexity of this part is $O(\sum_{e\in\Pi,\delta\in\{0,1\}} \|\Pi[e,\delta]\|)$. The sum is the size of the adjacency table Π, which we have shown above to be $O(n^2)$. Finally, the tiling stage invokes the same number of calls to $TILE()$ as the number of triangles on the resulting triangulation, which has complexity $O(n)$. So the overall complexity of $DOT1$ is $O(n^2)$.

Algorithm $DOT2$ has the same complexity as $DOT1$ in their initialization and tiling stages. In the main part, each $COST(e,t')$ uses constant time except when it is first called, in which case it spawns $2\|\Pi[e,\Delta(t',e)]\|$ number of other re-

cursive calls. Since there are $O(n^2)$ number of edge-triangle pairs $\{e,t'\}$ in S and $\|\Pi[e,\delta]\| < n$ for any e,δ, the time complexity of this part is bounded by $O(n^3)$. Hence the overall complexity of *DOT*2 is $O(n^3)$. □

Since the Delaunay tetrahedralization can be computed in expected $O(n^2)$ time [ES96], the end-to-end time complexity for computing DOT the two algorithms are still $O(n^2)$ and $O(n^3)$. For practical curve data, however, we have observed that both algorithms run in $O(n^2)$ time or better (see next section). In contrast, without restricting the search space, previous extensions of the GK algorithm to 3D polygons [BS95, BDE96, Lie03] need to use $O(n^3)$ and $O(n^4)$ time respectively for minimizing per-triangle and bi-triangle metrics.

4.2. Existence of solutions

Since not all 3D polygons are triangulable, a polygon may not have any triangulation that is restricted to the Delaunay triangles. We say that such polygons are not *Delaunay-triangulable*. For these polygons our algorithm will fail to return any solution. Figure 3 show two such examples. The hexagon in (a) was found by Barequet et. al. [BDE96] and is a 3D polygon with the fewest vertices that cannot be triangulated (the authors proved that any polygon with vertices fewer than 6 is triangulable). The polygon in (b) is a spiral square coil where the square turns have non-equal diameters. The polygon is triangulable, according to the sufficient conditions proven in [BDE96], because it has a non-intersecting projection the view plane in (c). However, our algorithms return no solution for this input.

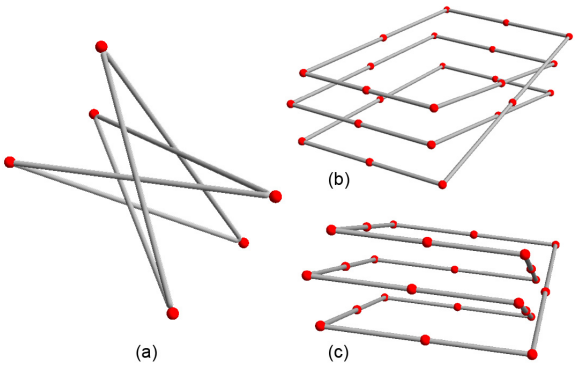

Figure 3: *A hexagon that is not triangulable (a), and two views of a triangulable 24-vertex polygon that is not Delaunay-triangulable (b,c).*

To evaluate whether our algorithms are suited for a particular application, it would be useful to have some geometrically characterizations of the kind of inputs that are Delaunay-triangulable. We next offer some preliminary results in this direction by giving a few necessary and sufficient conditions. Unfortunately, just like the general triangulability problem [BDE96], we do not have conditions that are both necessary and sufficient. In the next section, we will resort to extensive experimental results to demonstrate that most practical polygons are indeed Delaunay-triangulable.

Lemma 4.3 A polygon is not Delaunay-triangulable if

1. it is knotted, or
2. some polygon edges are not Delaunay (i.e., not incident to any Delaunay triangles).

Proof: It is obvious that each edge of the input polygon needs to be incident to some Delaunay triangles for a triangulation to exist. Also, it was shown that any knotted polygon is not triangulatable [BDE96], which also implies that it is not Delaunay-triangulable. □

It is worth pointing out that the two necessary conditions are fairly mild in practice. If the input polygon is intended to bound some disk-like surface, then the polygon cannot be knotted. Also, any 3D polygon can be turned into one whose edges are all Delaunay by subdividing the polygon edges. In particular, the edge subdivision algorithm of Shewchuk [She02] (which he calls "edge protection") returns a provably good output in which the new vertices are placed no denser than a constant factor of the local feature size of the original polygon. Alternatively, if the polygon is sampled from some smooth space curve (which is typical for sketch inputs), the polygon edges are always Delaunay as long as the samples are placed denser than the local feature size of the curve (Theorem 12 in [ABE98]–although the theorem was stated in 2D, the statement and the proof can be easily extended to higher dimensions).

The two necessary conditions are not sufficient for a polygon to be Delaunay-triangulatable. For example, both polygons in Figure 3 meet the two criteria. On the other hand, it is easy to show that a special class of polygons is always Delaunay-triangulable:

Lemma 4.4 A 3D polygon whose edges are Delaunay and all lying on the convex hull of the polygon is Delaunay-triangulatable.

Proof: If the polygon is planar, we simply take the Delaunay triangles lying in the interior of the polygon. In the non-planar case, the convex hull has a spherical topology and the polygon is a simple closed curve on the convex hull. By Schoenflies theorem, the polygon divides the convex hull into two patches with disk topology. Since the convex hull consists of Delaunay triangles and that all polygon edges are Delaunay, each patch is a triangulation restricted to the Delaunay triangles. □

Note that this last condition is stricter than the sufficient condition for triangulability given in [BDE96], which states that every polygon with a non-intersecting projection on some sphere is triangulable.

5. Results

We now present experimental results of our implementation of the algorithms. The implementation was done in C++ and closely follows the pseudo-code in Figure 2. The tests were done on an 8-core workstation with 12GB of memory (with 2GB available to our program). The test suite includes both piecewise smooth curves, which are typical for sketching inputs, and synthetically generated hole boundaries on triangular meshes. These results validate the complexity bounds given in the previous section and demonstrate the existence and optimality of DOT for these practical data.

We also compared our results with previous extensions of the GK algorithm to triangulating 3D polygons [BS95, Lie03], which search over all possible triangles for the optimal triangulation while allowing self-intersections. To implement these extensions, we simply re-use our code for $DOT1, DOT2$ but feeding it with the set of all triangles instead of only the Delaunay triangles. We also got rid of the adjacency table Π, since the triangles on one side of an edge can now be inferred directly from the indices of the two vertices of the edge. We thus created two new programs $GK1, GK2$, the former handling only per-triangle metrics, and the latter capable of handling both per-triangle and bi-triangle metrics. Note that our implementation, which uses recursion, is more costly than the implementations in [BS95, Lie03], which uses the more standard iterative dynamic programming approach. However, both ours and theirs share the same asymptotic complexity.

5.1. Test data

To evaluate our algorithm on the kind of curves from sketched-based interfaces, we first consider polygons that are sampled from unknotted piece-wise smooth curves with various shapes, as shown on the left of Figure 4:

- *Monkey*: Uniform samples on the boundary curve of a monkey saddle.
- *Mobius*: Uniform samples on the boundary of a Mobius strip.
- *Spiral*: Uniform samples on two spirals, one shifted upwards from another, and on the straight line segments that connect the ends of the spirals.
- *Moment*: The "moment curve" whose ith vertex has coordinates $p_i = \{i, i^2, i^3\}$.

Note that *Moment* was originally used to demonstrate the $O(n^2)$ upper bound of the number of simplicies in a Delaunay tetrahedralization. This is because there is a tetrahedron connecting every four distinct points of the form $\{p_i, p_{i+1}, p_j, p_{j+1}\}$ for any i, j. To demonstrate the $O(n^3)$ upper bound of the time complexity of our algorithm $DOT2$, we create an additional example, which we call *Twist*, that connects vertices of *Moment* in a crisscross manner (see bottom left of Figure 4). Specifically, assuming there are an odd number n of vertices on *Moment*, they are connected as

$$p_1, p_{n-1}, p_3, p_{n-3}, \ldots, p_4, p_{n-2}, p_2, p_n.$$

That is, take the list of vertices on *Moment* with odd, ascending indices and interleave with the list of vertices with even, descending indices. Note that the edges of *Twist* are still Delaunay, by the very property of *Moment*. In our tests, we range the number of vertices in each example from 50 to 5000.

Our second test data is a suite of synthetically generated hole boundaries on meshes. Each hole starts from a randomly selected triangle on the mesh and grows to a specified number of triangles. At each step of growth, we randomly select a none-hole triangle that is incident to the hole boundary and whose inclusion in the hole results in a boundary homeomorphic to the original boundary (in this way, the boundary curve remains unknotted). Finally, the edge subdivision algorithm of Shewchuk [She02] is applied to the hole boundary to obtain a polygon with Delaunay edges.

To generate curves with varying shapes and complexity, we used the Happy Buddha and the head portion of David from the Stanford Scanning Repository. Both meshes have non-trivial shapes and Buddha has a complex topology. For each mesh, we generated 1000 holes with size ranging from 100 to 10K triangles, which cover up to 20% of the surface area. Several of these holes are shown on the top of Figure 5. Note that the hole boundaries can be highly contorted and far from being planar, particularly when the holes cover more surface areas. Although real-world holes on scanned data seldom cover this much of the surface, we created these large holes mainly to investigate Delaunay triangulability in more complex and non-typical scenarios.

5.2. Performance

The performance of our algorithms on the polygons sampled from smooth curves is plotted in the 3rd and 4th columns of Figure 4. We used triangle area metric for $DOT1$ and the sum of squared dihedral metric for $DOT2$. Observe that, as the number of samples increases, the growth rates of time and space of $DOT1, DOT2$, as well as of the time taken by Tetgen to generate Delaunay triangles, are at worst quadratic in all examples (except for the time of $DOT2$ on *Twist*, which will be discussed below). This agrees with our theoretical analysis earlier. The actual amount of time and space consumed for each example is highly correlated with the number of Delaunay triangles (plotted in the 2nd column). Also note that $DOT1$ is more efficient than $DOT2$, even though they share the same growth rate in performance.

The only outlier in our results is *Twist*, which causes algorithm $DOT2$ to exhibit cubic growth in running time - the theoretical upper bound. This can be explained as follows. Note each polygon edge $e = \{p_i, p_{i+1}\}$ on *Moment* becomes an interior edge on *Twist*, where there are $Mod(2i-1, n)$

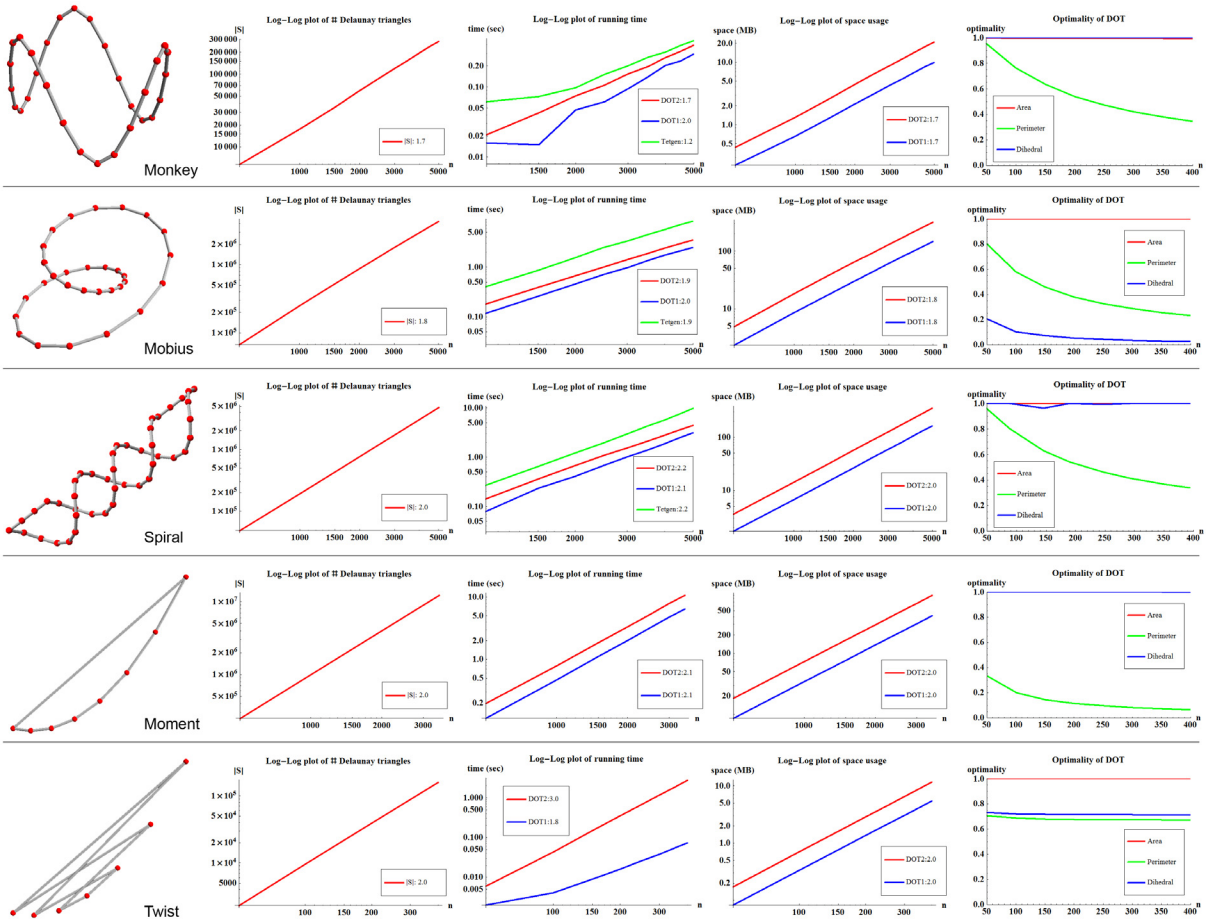

Figure 4: *Piecewise smooth curves used in our tests (1st column) and plots of the number of Delaunay triangles (2nd column), the running time of Tetgen, DOT*1 *and DOT*2 *(3rd column), the space usage of DOT*1 *and DOT*2 *(4th column), and the optimality under area, perimeter, and dihedral metrics (last column). The growth rate of time and space (as the slope of the best-fitting line in the log-log space) is noted in the legends. The Delaunay triangles in* Moment *and* Twist *were generated by combinatorial enumeration, since Tetgen gave numerical errors due to the large difference in the scale of coordinates.*

and $Mod(n-2i-1,n)$ Delaunay triangles respectively on the two sides of the edge. One can find $O(n)$ such edges, which we call *hinge edges*, that are incident to $O(n)$ Delaunay triangles on both sides. For each hinge edge e, the $DOT2$ algorithm has to invoke $O(n)$ calls $COST(e,t')$ for some t' on one side of e, and each call spawns $O(n)$ new calls to compute the cost of the triangulation on the other side of e. The complexity of these calls for each hinge edge e is $O(n^2)$, and the total complexity over all $O(n)$ hinge edges is $O(n^3)$.

We should point out that *Twist* is an extremely pathological case that rarely arises in practice, and the complexity of $DOT2$ was observed to be quadratic or lower for all other test data in our experiments. A possible reason of the practical complexity of $DOT2$ could be that there are usually very few hinge edges in a 3D polygon. However, deriving conditions of a polygon under which $DOT2$ uses quadratic time awaits further study.

The performance of our algorithms on the hole data set is shown on the right of Figure 5 (top-right and bottom-left plots). Each dot in the plots represents an individual hole boundary in our data set. Observe that the growth rates of time and space for both $DOT1, DOT2$ are now close to being linear. Note that the number of Delaunay triangles also grows in a close-to-linear fashion. Triangulation is highly efficient for this practical data set, taking only a fraction of a second and hardly any memory for hole boundaries containing thousands of points.

We now compare the performance of our algorithms with that of $GK1, GK2$. Since their time and space consumptions depend only on the number of vertices in the polygon and

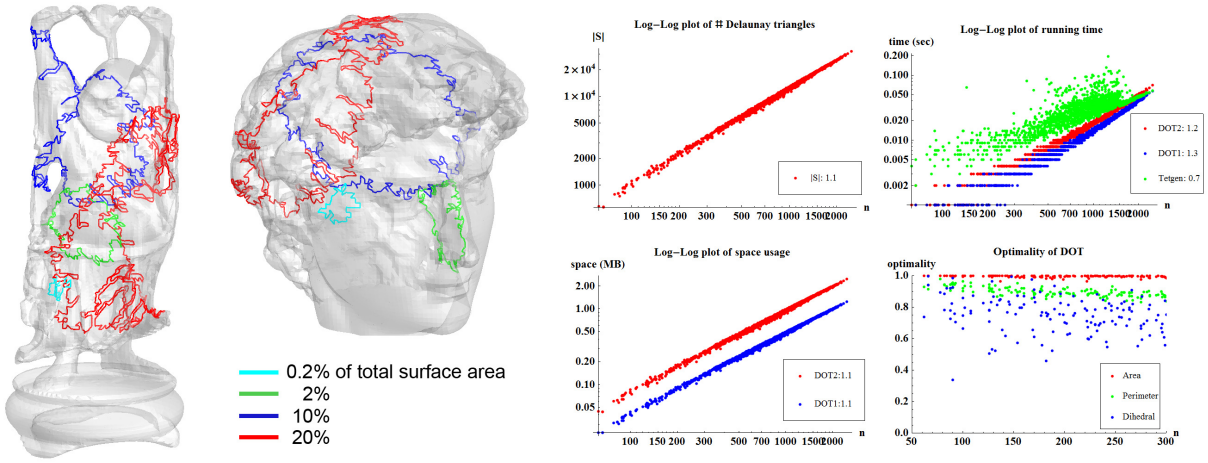

Figure 5: *Examples of synthetic hole boundaries on meshes used in our tests (left) and plots (right) of the number of Delaunay triangles, the running time of Tetgen, DOT*1 *and DOT*2*, the space usage of DOT*1 *and DOT*2*, and the optimality under area, perimeter, and dihedral metrics. The growth rate of time and space (as the slope of the best-fitting line in the log-log space) is noted in the legends.*

not on the actual coordinates, we only need to test them on one data set (e.g., *Monkey*). The results are plotted in Figure 6, with $GK1$ using the triangle area metric and $GK2$ using the sum of squared dihedral metric. In agreement with the analysis in [BDE96], $GK1, GK2$ take respectively $O(n^3)$ and $O(n^4)$ time (with $O(n^2)$ and $O(n^3)$ space). Note that $GK2$ becomes too slow (taking a couple of minutes) even for a few hundred points, and quickly exhausts system memory as the size of the input continues to increase.

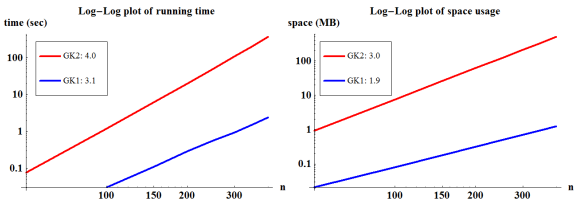

Figure 6: *Running time and space usage of algorithms $GK1, GK2$, which search for the optimal triangulation over the set of all possible triangles.*

5.3. Delaunay triangulability

Our algorithms returned solutions on all of our test data (smooth curves plus 2000 hole boundaries) except for 32 hole boundaries on the Happy Buddha. The hole in the vast majority of these failure cases (28 out of 32) covers over 10% of the surface area of the model, and the smallest hole covers 5% of the surface area. The hole boundaries all have very convoluted shapes (see Figure 7). Given that polygons like these typically do not arise in practice, we believe the algorithms will return solutions for the majority of the practical data, whether they are piecewise smooth curve sketches or hole boundaries on real-world meshes. However, the chance of success may decrease with increasing complexity of the polygon shape. Since Delaunay triangulability implies triangulability, our results also suggest that most practical 3D polygons are triangulable.

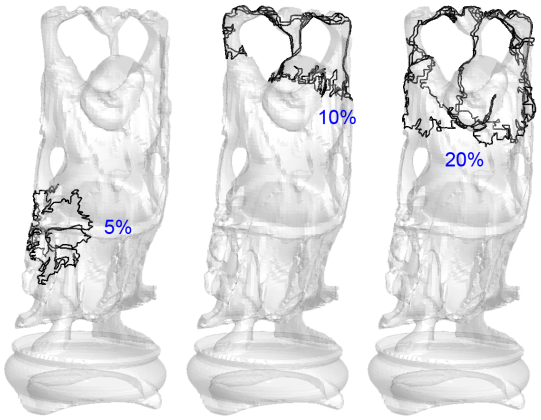

Figure 7: *Hole boundaries covering various ratios of the surface area that are not Delaunay triangulable.*

5.4. Optimality

Our algorithms only compute approximate solutions to the much harder problem of optimal triangulation considering all possible triangles. In $\mathbb{R}^2$, researchers have shown that the Delaunay triangulation of a point set does not in general minimize the sum of edge lengths. In fact, the sum of edge

lengths in a 2D Delaunay triangulation may be longer than the minimum by a factor of $\Omega(n)$ for n points [MZ79, Kir80]. Here we will explore the optimality of DOT for triangulating a 3D polygon. We will independently consider three commonly used metrics to minimize, the sum of per-triangle areas, the sum of per-triangle perimeters (which is similar to the sum of edge lengths), and the sum of bi-triangle squared dihedrals. For each metric, the optimality of DOT is measured as the ratio of the cost of the optimal, but possibly self-intersecting triangulation produced by *GK*2 (which is a lower bound of the cost of the true optimal triangulation) over the cost of DOT.

We plot the optimality for the smooth curves and hole boundaries respectively in Figure 4 (last column) and 5 (bottom right plot). Due to the high computational cost of *GK*2, we only considered polygons containing up to a few hundred vertices. Observe that the DOTs minimizing the area metric are consistently close to optimal triangulations in both data sets. This agrees with the fact that the Delaunay triangles tend to connect nearby polygon parts. The DOTs minimizing the perimeter metric are fairly close to optimal on hole boundaries, but are less optimal on smooth curves. In both types of data, the optimality of DOT drops with the increasing size of input, which is reminiscent of 2D Delaunay triangulations [MZ79, Kir80]. On the other hand, DOTs minimizing the dihedral metric are consistently close to optimal on smooth curves (except for *Mobius*) but much less so on hole boundaries.

In sum, the optimality of DOT seems to have a great dependence on the specific shape of the polygon when using perimeter or dihedral metrics, while the area-minimizing DOT seems to be close-to-optimal in most cases. In Figure 8 we take a closer look at the *Mobius* example where the optimality of DOT is low under both perimeter and dihedral metrics. In this case, the results of *GK*2 contains no self-intersections, and hence they are the optimal triangulations. Note that the perimeter-minimizing optimal triangulation uses many "ear-cutting" triangles, while the dihedral-minimizing optimal triangulation uses a fan of triangles originating from a single vertex. Many of the ear-cutting or fan triangles are not Delaunay, resulting in higher cost of the DOTs.

The flexibility of handling different metrics and their combinations allows our algorithm to be able to generate a wide range of triangulations. In Figure 9 we show several variations of triangulations created for two input polygons, using objective functions such as minimal triangle areas, minimal squared dihedrals, maximal squared dihedrals (which maximize twisting), minimal deviation with boundary normals, and their combinations.

6. An application

Sketch based modeling systems have been developed that allow users to rapidly create networks of space curves that define 3D shapes [BBS08]. As a final example, we apply our algorithms to compute triangulations of some real-world sketch inputs. Given a network of sketched spatial curves, we first apply the method of Abbasinjad et al. [AJA11] to extract individual loops of curves that form the patch boundaries. Then each patch was triangulated individually by minimizing the sum of squared dihedral metric using our algorithm. Starting from the initial triangulation, we can obtain a final surface using refinement [Lie03] and smoothing [AJC11] operators. The results for two sketch inputs, Roadster and Spider, are shown in Figure 10.

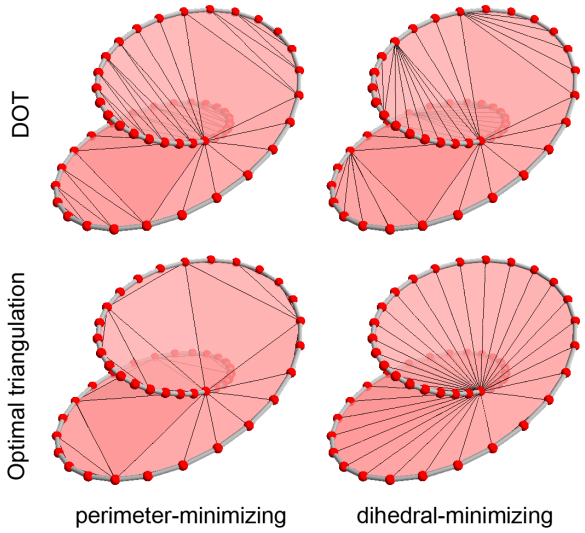

Figure 8: *Comparing DOTs of* Mobius *(top) with optimal triangulations (bottom) under two different metrics.*

7. Conclusions and Discussion

We introduced efficient algorithms for computing approximately optimal triangulations of 3D polygons by restricting the search space to the Delaunay complex. Besides avoiding self-intersections, the algorithms have significantly better efficiency than previous attempts at this problem. We give complexity bounds of the algorithms and provide empirical evidence that the complexity is often much lower (at worst quadratic) for polygons in practical applications such as curve sketches and hole boundaries. We also showed through experiments that most practical polygons are Delaunay triangulable, and with the right choice of the minimizing metric, the Delaunay-restricted optimal triangulation (DOT) can well approximate the optimal triangulation.

The work opens up several interesting questions concerning Delaunay-restricted triangulations of a 3D polygon. How to characterize polygons that are Delaunay triangulable? Can we derive theoretical bounds on the approximation error of DOTs, particularly when minimizing the area metric? And

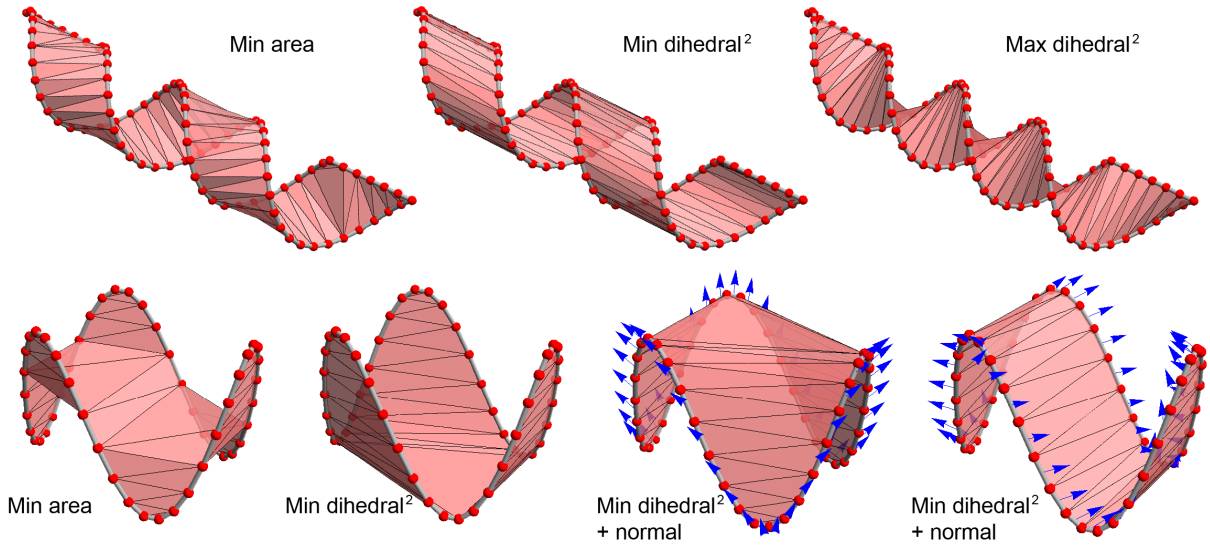

Figure 9: *Variations of triangulations generated by our algorithm for* Spiral *(top) and* Monkey *(middle). Prescribed boundary normals, if present, are shown as arrows.*

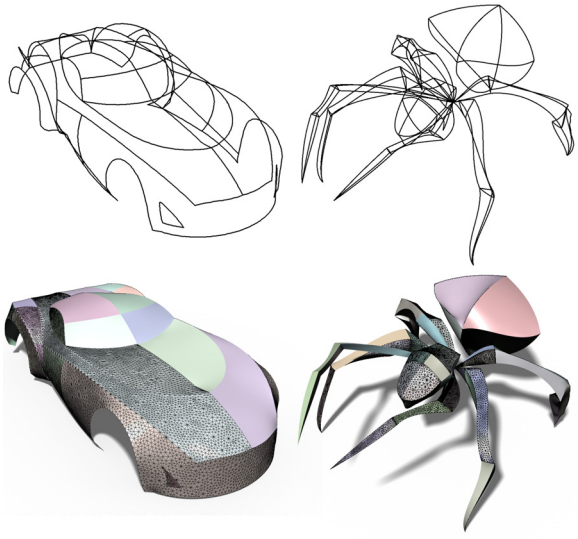

Figure 10: *ILove Sketch curve data [BBS08] with resolved patches [AJA11] (top) is first triangulated minimizing the dihedral angle bending using our method. The surface is then refined using the method of [Lie03], and a final boundary normal conforming bi-Laplacian surface is produced(shown bottom) using the method of Andrews et. al [AJC11].*

can we prove the quadratic complexity of our dynamic programming algorithms for practical polygons?

It is conceivable that the dynamic programming approach can be used to triangulate more general inputs, such as multiple 3D polygons that bound a single surface. It would be interesting to see if the restriction to the Delaunay triangles would result in efficient algorithms as it did for a single polygon. Another direction of extension is the problem of tiling a 3D polygon by a surface with non-zero genus. Note that many un-triangulable polygons, such as knots, can have tilings with non-trivial topology. Again, the questions listed above apply to these extended scenarios as well.

References

[AAD07] AMENTA N., ATTALI D., DEVILLERS O.: Complexity of delaunay triangulation for points on lower-dimensional polyhedra. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (2007).

[ABE98] AMENTA N., BERN M., EPPSTEIN D.: The crust and the β-skeleton: combinatorial curve reconstruction. *Graph. Models Image Process. 60*, 2 (Mar. 1998), 125–135.

[AJA11] ABBASINEJAD F., JOSHI P., AMENTA N.: Surface patches from unorganized space curves. *Comput. Graph. Forum 30*, 5 (2011), 1379–1387.

[AJC11] ANDREWS J., JOSHI P., CARR N. A.: A linear variational system for modeling from curves. *Comput. Graph. Forum 30*, 6 (2011), 1850–1861.

[BBS08] BAE S.-H., BALAKRISHNAN R., SINGH K.: Ilovesketch: as-natural-as-possible sketching system for creating 3d curve models. In *Proceedings of the 21st annual ACM symposium on User interface software and technology* (New York, NY, USA, 2008), UIST '08, ACM, pp. 151–160.

[BDE96] BAREQUET G., DICKERSON M., EPPSTEIN D.: On triangulating three-dimensional polygons. In *Proceedings of the twelfth annual symposium on Computational geometry* (1996), SCG '96, pp. 38–47.

[BMS*10] BRAZIL E. V., MACEDO I., SOUSA M. C.,

DE FIGUEIREDO L. H., VELHO L.: Sketching variational hermite-rbf implicits. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium* (2010), SBIM '10, pp. 1–8.

[BPB06] BRANCH J., PRIETO F., BOULANGER P.: Automatic hole-filling of triangular meshes using local radial basis function. In *3DPVT '06: Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)* (2006), pp. 727–734.

[BS95] BAREQUET G., SHARIR M.: Filling gaps in the boundary of a polyhedron. *Comput. Aided Geom. Des. 12*, 2 (Mar. 1995), 207–229.

[ES96] EDELSBRUNNER H., SHAH N. R.: Incremental topological flipping works for regular triangulations. *Algorithmica 15*, 3 (1996), 223–241.

[GBL05] GRANTSON M., BORGELT C., LEVCOPOULOS C.: Minimum weight triangulation by cutting out triangles. In *Proceedings of the 16th international conference on Algorithms and Computation* (2005), ISAAC'05, pp. 984–994.

[Gil79] GILBERT P. D.: New results in planar triangulations.

[JC08] JOSHI P., CARR N. A.: Repoussé: Automatic inflation of 2d artwork. In *SBIM '08: Proceedings of the sixth Eurographics workshop on Sketch-Based Interfaces and Modeling* (Aire-la-Ville, Switzerland, Switzerland, 2008), Eurographics, Eurographics Association, pp. 49–56.

[Ju09] JU T.: Fixing geometric errors on polygonal models: a survey. *J. Comput. Sci. Technol. 24*, 1 (Jan. 2009), 19–29.

[Kir80] KIRKPATRICK D. G.: A note on delaunay and optimal triangulations. *Inf. Process. Lett. 10*, 3 (1980), 127–128.

[Kli80] KLINCSEK G.: Minimal triangulations of polygonal domains. In *Combinatorics 79*, Hammer P. L., (Ed.), vol. 9 of *Annals of Discrete Mathematics*. Elsevier, 1980, pp. 121 – 123.

[KSI*07] KUMAR A., SHIH A. M., ITO Y., ROSS D. H., SONI B. K.: A hole-filling algorithm using non-uniform rational b-splines. In *Proceedings of 16th International Meshing Roundtable* (2007), pp. 169–182.

[Lie03] LIEPA P.: Filling holes in meshes. In *Symposium on Geometry Processing* (2003), pp. 200–206.

[MD93] MAKELA I., DOLENC A.: Asome efficient procedures for correcting triangulated models. In *Proc. Solid Free form Fabrication Symposium* (Austin, Texas, USA, 1993), pp. 126–134.

[MR06] MULZER W., ROTE G.: Minimum weight triangulation is np-hard. In *Proceedings of the twenty-second annual symposium on Computational geometry* (2006), SCG '06.

[MZ79] MANACHER G. K., ZOBRIST A. L.: Neither the greedy nor the the delaunay triangulation approximates the optimum. *Inf. Process. Lett. 9* (1979), 31–34.

[NISA07] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: Fibermesh: designing freeform surfaces with 3d curves. *ACM Trans. Graph. 26*, 3 (2007), 41.

[PP93] PINKALL U., POLTHIER K.: Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics 2* (1993), 15–36.

[RSW*07] ROSE K., SHEFFER A., WITHER J., CANI M.-P., THIBERT B.: Developable surfaces from arbitrary sketched boundaries. In *Proc. Eurographics Symposium on Geometry Processing* (2007).

[RW97] ROTH G., WIBOWOO E.: An efficient volumetric method for building closed triangular meshes from 3-d image and point data. In *Proceedings of the conference on Graphics interface '97* (Toronto, Ont., Canada, Canada, 1997), Canadian Information Processing Society, pp. 173–180.

[SH75] SHAMOS M. I., HOEY D.: Closest-point problems. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science* (1975), pp. 151–162.

[She02] SHEWCHUK J. R.: Constrained delaunay tetrahedralizations and provably good boundary recovery. In *In Eleventh International Meshing Roundtable* (2002), pp. 193–204.

[Si09] SI H.: Tetgen:a quality tetrahedral mesh generator and a 3d delaunay triangulator, 2009. URL: `http://tetgen.berlios.de/`.

[VPK05] VARNUSKA M., PARUS J., KOLINGEROVA I.: Simple holes triangulation in surface reconstruction. In *Proceedings of Algoritmy* (2005), pp. 280–289.

[WLG03] WAGNER M., LABSIK U., GREINER G.: Repairing non-manifold triangle meshes using simulated annealing. *International Journal of Shape Modeling 9*, 2 (2003), 137–153.

[WO07] WANG J., OLIVEIRA M. M.: Filling holes on locally smooth surfaces reconstructed from point clouds. *Image Vision Comput. 25*, 1 (2007), 103–113.