2007

Connecting Polygonizations via Stretches and Twangs

Mirela Damian
Villanova University

Robin Flatland
Siena College

Joseph O'Rourke
Smith College, jorourke@smith.edu

Suneeta Ramswami
Rutgers University - Camden

Connecting Polygonizations via Stretches and Twangs

Mirela Damian* Robin Flatland† Joseph O'Rourke‡ Suneeta Ramaswami§

Abstract

We show that the space of polygonizations of a fixed planar point set S of n points is connected by $O(n^2)$ "moves" between simple polygons. Each move is composed of a sequence of atomic moves called "stretches" and "twangs". These atomic moves walk between weakly simple "polygonal wraps" of S. These moves show promise to serve as a basis for generating random polygons.

1 Introduction

This paper studies polygonizations of a fixed planar point set S of n points. Let the n points be labeled p_i, $i = 0, 1, \ldots, n-1$. A *polygonization* of S is a permutation σ of $\{0, 1, \ldots, n-1\}$ that determines a polygon: $P = P_\sigma = (p_{\sigma(0)}, \ldots, p_{\sigma(n-1)})$ is a simple (non-self-intersecting) polygon. We will abbreviate "simple polygon" to *polygon* throughout. As long as S does not lie in one line, which we will henceforth assume, there is at least one polygon whose vertex set is S. A point set S may have as few as 1 polygonization, if S is in convex position,[1] and as many as $2^{\Theta(n)}$ polygonizations. For the latter, see Fig. 1a.

Our goal in this work is to develop a computationally natural and efficient method to explore all polygonizations of a fixed set S. One motivation is the generation of "random polygons" by first generating a random S and then selecting a random polygonization of S. Generating random polygons efficiently is a long unsolved problem; only heuristics [AH96] or algorithms for special cases [ZSSM96] are known. Our work can be viewed as following a suggestion in the latter paper:

> "start with a ... simple polygon and apply some simplicity-preserving, reversible operations ... with the property that any simple polygon is reachable by a sequence of operations"

Our two operations are called *stretch* and *twang* (defined in Section 2.2). Neither is simplicity preserving, but they are nearly so in that they produce polygonal wraps defined as follows.

Definition 1 A *polygonal wrap* $\mathcal{P}_\sigma$ is determined by a sequence σ of point indices drawn from $\{0, 1, \ldots, n-1\}$ with the following properties:

1. Every index in $\{0, 1, \ldots, n-1\}$ occurs in σ.
2. Indices may be repeated. If index i appears at least twice in σ, we call p_i a *point of double contact*.
3. For sufficiently small $\varepsilon > 0$, there exists a perturbation within an ε-disk of the points in double contact, separating each such point into two or more points, so that there is a simple closed curve C that passes through the perturbed points in σ order. Sometimes such a P is called "weakly simple" because its violations of simplicity (i.e., its self-touchings) avoid proper crossings.[2]

Fig. 1b shows a polygonal wrap with five double-contacts (p_1, p_4, p_5, p_8 and p_9). Note that a polygon is a polygonal wrap without double-contact points.

*Dept. of Computer Science, Villanova Univ., Villanova, PA 19085, USA. mirela.damian@villanova.edu.
†Dept. of Computer Science, Siena College, Loudonville, NY 12211, USA. flatland@siena.edu.
‡Dept. of Computer Science, Smith College, Northampton, MA 01063, USA. orourke@cs.smith.edu.
§Dept. of Computer Science, Rutgers University, Camden, NJ 08102, USA. rsuneeta@camden.rutgers.edu.

[1] S is in convex position if every point in S is on the hull of S.

[2] Two segments properly cross if they share a point x in the relative interior of both, and cross transversely at x.

Stretches and twangs take one polygonal wrap to another. A stretch followed by a natural sequence of twangs, which we call a *cascade*, constitutes a *forward move*. Forward moves (described in further detail in Section 2.3) take a polygon to a polygon, i.e., they are simplicity preserving. Reverse moves will be introduced in Section 6. A *move* is either a forward or a reverse move. We call a stretch or twang an *atomic move* to distinguish it from the more complex forward and reverse moves.

Our main result is that the configuration space of polygonizations for a fixed S is connected by forward/reverse moves, each of which is composed of a number of stretches and twangs, and that the diameter of the space is $O(n^2)$ moves. We can bound the worst-case number of atomic moves constituting a particular forward/reverse move by the geometry of the point set. Experimental results on random point sets show that, in the practical situation that is one of our motivations, the bound is small, perhaps even constant. We have also established loose bounds on the worst-case number of atomic operations as a function of n: an exponential upper bound and a quadratic lower bound. Tightening these bounds has so far proven elusive and is an open problem.

One can view our work as in the tradition of connecting discrete structures (e.g., triangulations, matchings) via local moves (e.g., edge flips, edge swaps). Our result is comparable to that in [vLS82], which shows connectivity of polygonizations in $O(n^3)$ edge-edge swap moves through intermediate self-crossing polygons. The main novelty of our work is that the moves, and even the stretches and twangs, never lead to proper crossings, for polygonal wraps have no such crossings. We explore the possible application to random polygons briefly in Section 8. For the majority of this paper, we concentrate on defining the moves and establishing connectivity.

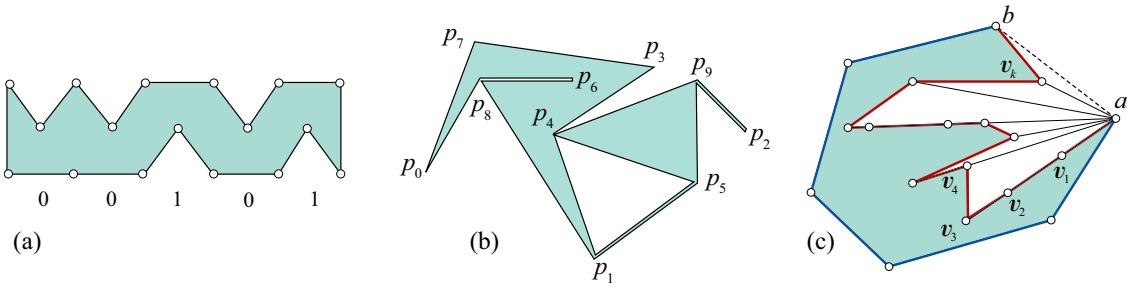

Figure 1: Examples. (a) A set of $n = 3k + 2$ points that admits 2^k polygonizations. (b) Polygonal wrap $\mathcal{P}_\sigma$ with $\sigma = (0, 8, 6, 8, 1, 5, 9, 2, 9, 4, 5, 1, 4, 3, 7)$ (c) A polygonization with one pocket with lid ab.

We begin by defining pockets, which play a central role in our algorithms for polygonal transformations. Then in Section 2.1 we describe two natural operations that transform one polygon into another but fail to achieve connectivity of the configuration space of polygonizations, which motivates our definitions of stretches and twangs in Section 2.2. Following these preliminaries, we establish connectivity and compute the diameter in Sections 3–7. We conclude with open problems in Section 9.

1.1 Pockets and Canonical Polygonization

Let P be a polygonization of S. A hull edge ab that is not on ∂P is called a *pocket lid*. The polygon external to P bounded by P and ab is a *pocket* of P.

Lemma 2 *Any point set S not in convex position has a polygonization with one pocket only* [CHUZ92].

For a fixed hull edge ab, we define the canonical polygonization of S to be a polygon with a single pocket with lid ab (cf. Lemma 2) in which the pocket vertices are ordered by angle about vertex a, and from closest to farthest from a if along the same line through a. We call this ordering the *canonical order* of the pocket vertices; see Fig. 1c.

2 Polygonal Transformations

Let P be a polygon defined by a circular index sequence σ. We examine operations that permute this sequence, transforming P into a new polygon with the same set of vertices linked in a different order.

Throughout the paper we use $\triangle abc$ to denote the closed triangle with corners a, b and c.

2.1 Swaps and Hops

We begin by defining two natural transformation operations, a *swap* and a *hop*. A swap operation is a transposition of two consecutive vertices of P that results in a new (non-self-intersecting) polygon. Fig. 2a illustrates the swap operation. It is well known that transpositions connect all permutations, but this is not the case for swaps. Because we require that the resulting polygon be simple, a vertex pair cannot be swapped if the operation results in a self-intersecting polygon. Fig. 2b shows an example of a polygon for which no vertex pair can be swapped without creating an edge crossing. Thus, swaps do not suffice to connect all polygonizations, which motivates our definition of a more powerful move, which we call a *hop*.

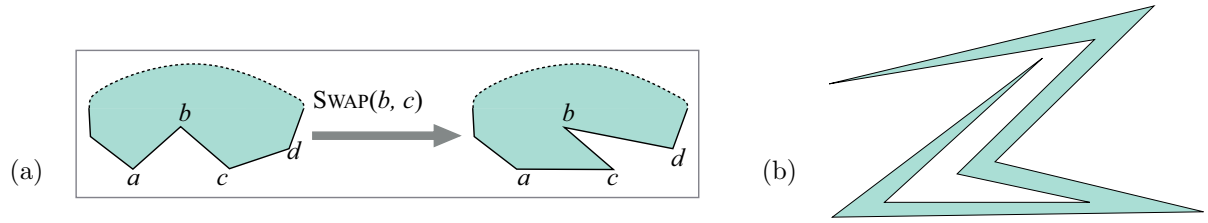

Figure 2: (a) $\textsc{Swap}(b, c)$ illustrated (b) Polygon admitting no swaps.

The hop operation generalizes the swap by allowing a vertex to hop to any position in the permutation, as long as the resulting polygon is simple. Fig. 3a shows the stretching of the edge ab down to vertex v, effectively "hopping" v between a and b in the permutation. We denote this operation by $\textsc{Hop}(e, v)$, where $e = ab$ (note the first argument is *from* and the second *to*).

To specify the conditions under which a hop operation is valid, we introduce some definitions, which will be used subsequently as well. A polygon P has two sides, the interior of P and the exterior of P. Let $abc = (a, b, c)$ be three vertices consecutive in the polygonization P. For noncollinear vertices, we distinguish between the *convex side* of b, that side of P with angle $\angle abc$ smaller than π, and the *reflex side* of b, the side of P with angle $\angle abc$ larger than π. Note that this definition ignores which side is the interior and which side is the exterior of P, and so is unrelated to whether b is a convex or a reflex vertex in P. Every true vertex has a convex and a reflex side (collinear vertices will be discussed in Section 2.2). To ensure that the resulting polygon is simple, $\textsc{Hop}(e, v)$ is valid iff the following two conditions hold: (1) the triangle induced by the two edges incident to v is empty of other polygon vertices and (2) the triangle induced by e and v lies on the reflex side of v and is empty of other polygon vertices.

Although more powerful than a swap, there also exist polygons that do not admit any hops. Fig. 3b shows an example (the smallest we could find) in which each edge-vertex pair violates one or both of the two conditions above. This example shows that hops do not suffice to connect all polygonizations.

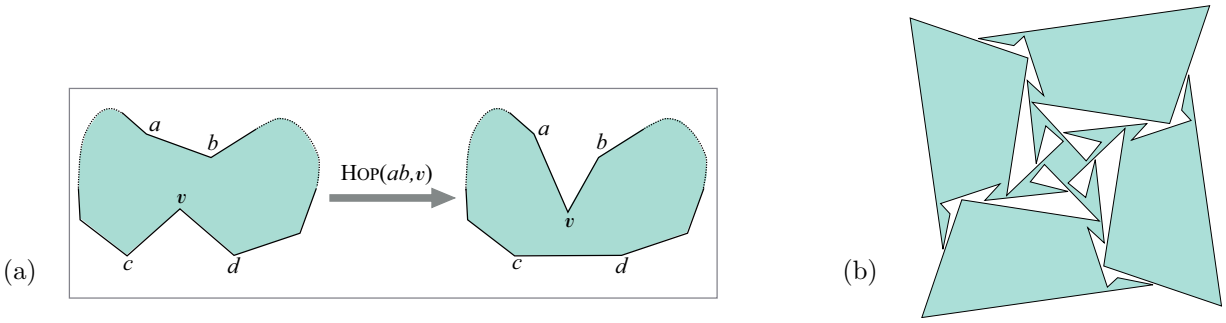

Figure 3: (a) $\textsc{Hop}(ab, v)$ illustrated (b) Polygon admitting no HOPs.

The limited transformation capabilities of the swap and hop operations motivate our introduction of two new operations, *stretch* and *twang*. The former operation relaxes the two hop conditions and allows the creation of a polygonal wrap. The latter operation restores the polygonal wrap to a polygon. We show that

together they are capable of transforming any polygon into a canonical form (Sections 3-5), and from there to any other polygon (Sections 6-7).

2.2 Stretches and Twangs

Unlike the $\textsc{Hop}(e, v)$ operation, which requires v to fully see the edge e into which it is hopping, the $\textsc{Stretch}(e, v)$ operation only requires that v see a point x in the interior[3] of e. The stretch is accomplished in two stages: (i) temporarily introduce two new "pseudovertices" on e in a small neighborhood of x (this is what we call $\textsc{Stretch}_0$ below), and (ii) remove the pseudovertices immediately using twangs.

$\textsc{Stretch}_0$. Let v see a point x in the interior of an edge e of P. By *see* we mean "clear visibility", i.e., the segment vx shares no points with ∂P other than v and x (see Fig. 4a). Note that every vertex v of P sees such an x (in fact, infinitely many x) on some e. Let x^- and x^+ be two points to either side of x on e, both in the interior of e, such that v can clearly see both x^- and x^+. Two such points always exist in a neighborhood of x. We call these points *pseudovertices*. Let $e = ab$, with x^- closer to the endpoint a of e. Then $\textsc{Stretch}_0(e, v)$ alters the polygon to replace e with (a, x^-, v, x^+, b), effectively "stretching" e out to reach v by inserting a narrow triangle $\triangle x^- v x^+$ that sits on e (see Fig. 4b).

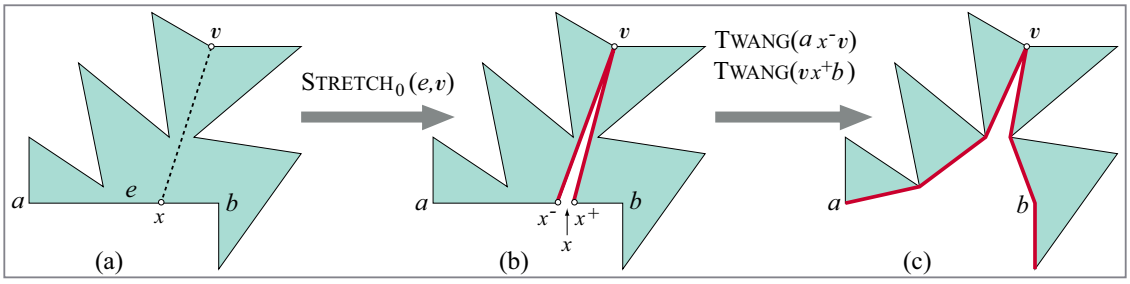

Figure 4: $\textsc{Stretch}(e, v)$ illustrated (a) v sees $x \in e$ (b) $\textsc{Stretch}_0(e, v)$ (c) $\textsc{Stretch}(e, v)$.

To complete the definition of $\textsc{Stretch}(e, v)$, which removes the pseudovertices x^+ and x^-, we first define the twang operation.

Twang. Informally, if one views the polygon boundary as an elastic band, a twang operation detaches the boundary from a vertex v and snaps it to v's convex side.

Definition 3 The operation $\textsc{Twang}(abc)$ is defined for any three consecutive vertices $abc \in \sigma$ such that

1. $\{a, b, c\}$ are not collinear.
2. b is either a pseudovertex, or a vertex in double contact. If b is a vertex in double contact, then $\triangle abc$ does not contain a *nested* double contact at b. By this we mean the following: Slightly perturb the vertices of P to separate each double-contact into two or more points, so that P becomes simple. Then $\triangle abc$ does not contain any other occurrence of b in σ. (E.g., in Fig. 5a, $\triangle a'bc'$ contains a second occurrence of b.)

Under these conditions, the operation $\textsc{Twang}(abc)$ replaces the sequence abc in $\mathcal{P}$ by $\text{sp}(abc)$, where $\text{sp}(abc)$ indicates the shortest path from a to c that stays inside $\triangle abc$ and does not cross $\partial\mathcal{P}$. We call b the *twang vertex*. Whenever a and c are irrelevant to the discussion, we denote the twang operation by $\textsc{Twang}(b)$.

Informally, $\textsc{Twang}(abc)$ "snaps" the boundary to wrap around the hull of the points in $\triangle abc$, excluding b (see Fig. 5a). A twang operation can be viewed as taking a step toward simplicity by removing either a pseudovertex or a point of double contact. We should note that $\text{sp}(abc)$ includes every vertex along this path, even collinear vertices. If there are no points inside $\triangle abc$, then $\text{sp}(abc) = ac$, and $\textsc{Twang}(abc)$ can

[3] By "interior" we mean "relative interior," i.e., not an endpoint.

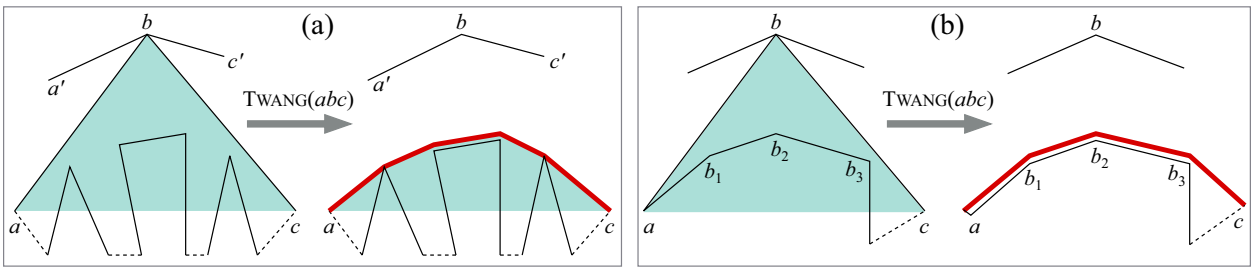

Figure 5: TWANG(abc) illustrated (a) TWANG(abc) replaces abc by sp(abc) (b) TWANG(abc) creates the hairpin vertex a and three doubled edges ab_1, b_1b_2 and b_2b_3.

be viewed as the reverse of HOP(ac, b). If a=c (i.e., ab and bc overlap in $\mathcal{P}$), we call b a *hairpin* vertex of $\mathcal{P}$; in this case, TWANG(aba) replaces aba in $\mathcal{P}$ by a. Hairpin vertices and "doubled edges" arise naturally from twangs. In Fig. 5b for instance, TWANG(abc) produces a hairpin vertex at a and doubled edges ab_1, b_1b_2, b_2b_3. So we must countenance such degeneracies. In general, there are points interior to the triangle, and the twang creates new points of double contact. Below, we will apply twangs repeatedly to remove all double contacts.

STRETCH. We can now complete the definition of STRETCH(e, v), with $e = ab$. First execute STRETCH$_0$(e, v), which picks the two pseudovertices x^+ and x^-. Then execute TWANG(ax^-v) and TWANG(vx^+b), which detach the boundary from x^+ and x^- and return to a polygonal wrap of S (see Fig. 4c). We refer to e (v) as the *stretch edge* (*vertex*).

2.3 Twang Cascades

A twang in general removes one double contact and creates perhaps several others. A TWANGCASCADE applied on a polygonal wrap $\mathcal{P}$ removes all points of double contact from $\mathcal{P}$:

TWANGCASCADE($\mathcal{P}$)

Loop for as long as $\mathcal{P}$ has a point of double contact b:

1. Find a vertex sequence abc in $\mathcal{P}$ that satisfies the twang conditions (cf. Def. 3).
2. TWANG(abc).

Note that for any point of double-contact b, there always exists a vertex sequence abc that satisfies the twang conditions and therefore the twang cascade loop never gets stuck. That a twang cascade eventually terminates is not immediate. The lemma below, whose proof we omit, shows that TWANG(abc) shortens the perimeter of the polygonal wrap (because it replaces abc by sp(abc)) by at least a constant depending on the geometry of the point set. Therefore, any twang cascade must terminate in a finite number of steps.

Lemma 4 *A single twang* TWANG*(abc) decreases the perimeter of the polygonal wrap by at least* $2d_{\min}(1 - \sin(\alpha_{\max}/2))$*, where* $d_{\min}$ *is the smallest pairwise point distance and* $\alpha_{\max}$ *is the maximum convex angle formed by any triple of non-collinear points.*

Supplementing this geometric bound, Corollary 13 in Appendix 3 establishes a combinatorial upper bound of $O(n^n)$ on the number of twangs in any twang cascade. An impediment to establishing a better bound is that a point can twang more than once in a cascade. Indeed we present in Appendix 2 an example in which $\Omega(n)$ points each twang $\Omega(n)$ times in one cascade, providing an $\Omega(n^2)$ lower bound.

2.3.1 Forward Move

We define a *forward move* on a polygonization P of a set S as a stretch (with the additional requirement that the pseudovertices on the stretch edge lie on the reflex side of the stretch vertex), followed by a twang and then a twang cascade, as described below:

ForwardMove(P, e, v)

Preconditions: (i) P is a simple polygon, (ii) e and v satisfy the conditions of Stretch(e, v), and (iii) v is a noncollinear vertex such that pseudovertices x^+ and x^- on e lie on the reflex side of v. {Let u, v, w be the vertex sequence containing v in P (necessarily unique, since P is simple).}

1. $\mathcal{P} \leftarrow$ Stretch(e, v).
2. $\mathcal{P} \leftarrow$ Twang(uvw).
3. $P' \leftarrow$ TwangCascade$(\mathcal{P})$.

A ForwardMove takes one polygonization P to another P' (see Fig. 6), as follows from Lemma 4. Next we discuss two important phenomena that can occur during a forward move.

Stretch Vertex Placement. We note that the initial stretch that starts a move might be "undone" by cycling of the cascade. This phenomenon is illustrated in Fig. 6, where the initial Stretch(ab, v) inserts v between a and b in the polygonal wrap (Fig. 6b), but v ends up between c and b in the final polygonization (Fig. 6f). Thus any attempt to specifically place v in the polygonization sequence between two particular vertices might be canceled by the subsequent cascade. This phenomenon presents a challenge to reducing a polygon to canonical form (discussed in Section 5).

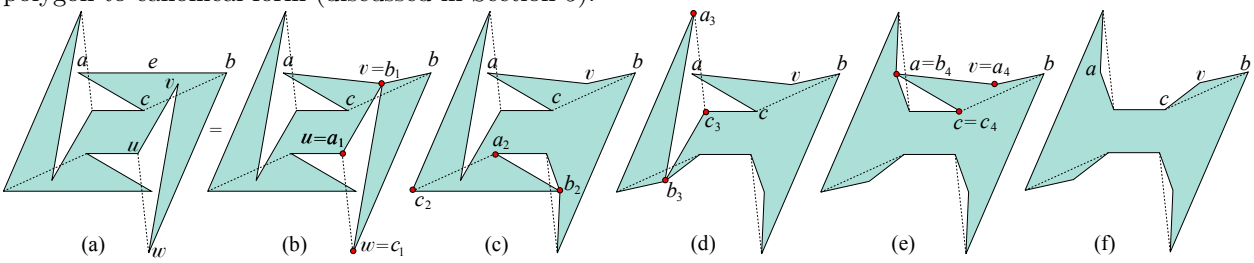

Figure 6: Forward move illustrated. (a) Initial polygon P (b) After Stretch(ab, v) (c) After Twang$(a_1b_1c_1)$ (d) After Twang$(a_2b_2c_2)$ (e) After Twang$(a_3b_3c_3)$ (f) After Twang$(a_4b_4c_4)$.

3 Single Pocket Reduction Algorithm

Now that the basic properties of the moves are established, we aim to show that our moves suffice to connect any two polygonizations of a point set S. The plan is to reduce an arbitrary polygonization to the canonical polygonization. En route to explaining this reduction algorithm, we show how to remove any particular pocket by redistributing its vertices to other pockets. This method will be applied repeatedly in Section 4 to move all pockets to one particular pocket.

In this section we assume that P has two or more pockets. We use hull(P) to refer to the closed region defined by the convex hull of P. For a fixed hull edge e that is the lid of a pocket A, the goal is to reduce A to e by redistributing the vertices of A among the other pockets, using forward moves only. This is accomplished by the Single Pocket Reduction algorithm (described below), which repeatedly picks a hull vertex v of A and attaches v to a pocket other than A; see Fig. 7 for an example run. Call a vertex v of P a *true* corner if the two polygon edges incident to v are non-collinear.

Single Pocket Reduction(P, e) Algorithm

Loop for as long as the pocket A of P with lid e contains three or more vertices:

1. Pick an edge-vertex pair (e, v) such that
   - e is an edge of P on ∂B for some pocket $B \neq A$
   - $v \in A$ is a non-lid true corner vertex on hull(A) that sees e
2. $P \leftarrow$ ForwardMove(P, e, v).

We now establish that the Single Pocket Reduction algorithm terminates in a finite number of iterations. First we prove a more general lemma showing that a twang operation can potentially reduce, but never expand, the hull of a pocket.

Lemma 5 (Hull Nesting under Twangs) *Let A be a pocket of a polygonal wrap $\mathcal{P}$ and let vertex $b \notin \text{hull}(\mathcal{P})$ satisfy the twang conditions. Let A' be the pocket with the same lid as A after* Twang(b). *Then $A' \subseteq \text{hull}(A)$.*

Proof: Let abc be the vertex sequence involved in the twang operation. Then Twang(abc) replaces the path abc by $\text{sp}(abc)$. If abc does not belong to ∂A, then Twang(abc) does not affect A and therefore $A' \equiv A$. So assume that abc belongs to ∂A. This implies that b is a vertex of A. Note that b is a non-lid vertex, since $b \notin \text{hull}(\mathcal{P})$. Then $\triangle abc \subset \text{hull}(A)$, and the claim follows from the fact that $\text{sp}(abc) \subset \triangle abc$. □

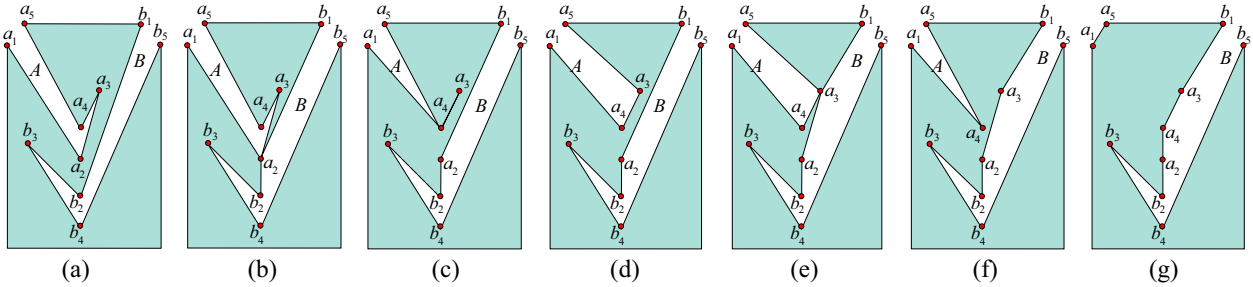

Figure 7: Single Pocket Reduction(P, a_1a_5) illustrated: (a) Initial P; (b) After Stretch(b_1b_2, a_2); (c) After Twang$(a_1a_2a_3)$; (d) After Twang$(a_3a_4a_5)$; (e) After Stretch(a_2b_1, a_3); (f) After Twang$(a_4a_3a_5)$; (g) After Stretch(a_2a_3, a_4)+Twang$(a_1a_4a_5)$.

Lemma 6 *The* Single Pocket Reduction *algorithm terminates in $O(n)$ forward moves.*

Proof: Let S denote the set of vertices of P in hull(A). Thus $|S| = O(n)$. We show that $|S|$ decreases by at least 1 in each loop iteration, thus establishing the claim of the lemma.

First observe that the existence of an edge-vertex pair (e, v) selected in Step 1 is guaranteed by the fact that P has two or more pockets. Step 2 of the Single Pocket Reduction algorithm, which performs a forward move to a different polygonization, attempts to reduce A by vertex v, thus decrementing $|S|$. We now show that this step is successful in that it does not reattach v back to A. Furthermore, we show that S acquires no new vertices during this step. These together show that $|S|$ decreases by at least 1 in each loop iteration.

The first step of the forward move, Stretch(e, v), does not affect S. The second step, Twang(avb), replaces the path avb by $\text{sp}(avb)$, thus eliminating v from A. Since v is a true corner vertex of A, hull(A) does not contain v at the end of this step. Let A' be the pocket of P with the same lid as A at the end of TwangCascade(P). Since a hull vertex never twangs, Lemma 5 implies that hull(A') is a subset of hull(A) and therefore $|S|$ does not increase during the twang cascade. Furthermore, since hull(A) does not contain v after the first twang operation, v must lie outside of hull(A') at the end of the twang cascade. □

4 Multiple Pocket Reduction Algorithm

For a given hull edge e, the goal is to transform P to a polygon with a single pocket with lid e, using forward moves only. If e is an edge of the polygon, for the purpose of the algorithm discussed here we treat e as a (degenerate) target pocket T. We assume that, in addition to T, P has one or more other pockets, otherwise there is nothing to do. Then we can use the Single Pocket Reduction algorithm to eliminate all pockets of P but T, as described in the Pocket Reduction algorithm below.

Pocket Reduction (P, e) Algorithm
If e is an edge of P, set $T \leftarrow e$, otherwise set $T \leftarrow$ the pocket with lid e (in either case, we treat T as a pocket). For each pocket lid $e' \neq e$ Call Single Pocket Reduction(P, e')

Observe that the POCKET REDUCTION algorithm terminates in $O(n^2)$ forward moves: there are $O(n)$ pockets each of which gets reduced to its lid edge in $O(n)$ forward moves (cf. Lemma 6).

Fig. 8 illustrates the POCKET REDUCTION algorithm on a 17-vertex polygon with three pockets A, B and C, each of which has 3 non-lid vertices, and target pocket T with lid edge $e = t_1t_2$. The algorithm first calls SINGLE POCKET REDUCTION(P, a_1a_5), which transfers to B all non-lid vertices of A, so B ends up with 6 non-lid vertices (this reduction is illustrated in detail in Fig. 7). Similarly, SINGLE POCKET REDUCTION(P, b_1b_5) transfers to C all non-lid vertices of B, so C ends up with 9 non-lid vertices, and finally SINGLE POCKET REDUCTION(P, c_1c_5) transfers all these vertices to T.

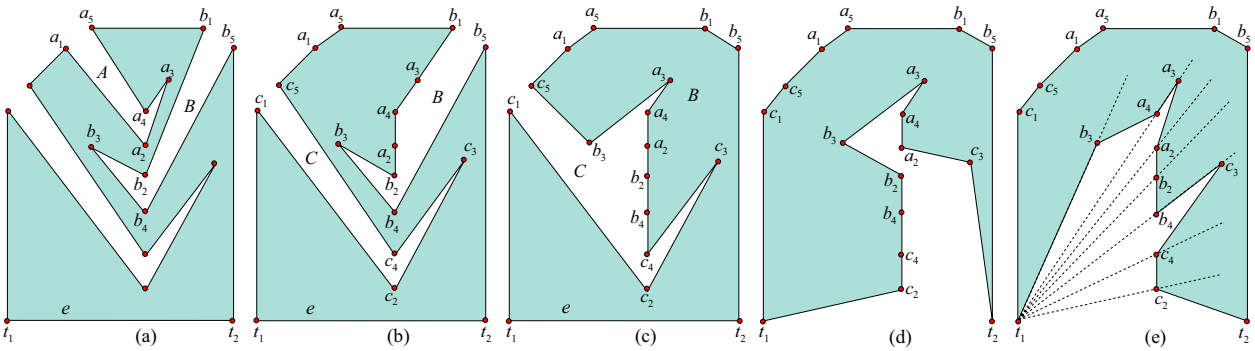

Figure 8: (a-e) POCKET REDUCTION(P, t_1t_2): (a) Initial P; (b) After SINGLE POCKET REDUCTION(P, a_1a_5); (c) After SINGLE POCKET REDUCTION(P, b_1b_5); (d) After SINGLE POCKET REDUCTION(P, c_1c_5); (e) After CANONICAL POLYGONIZATION(P, t_1t_2).

This example shows that the $O(n^2)$ bound on the number of forward moves is tight: an n-vertex polygon with a structure similar to the one in Fig. 8a has $O(n)$ pockets. The number of forward moves performed by the POCKET REDUCTION algorithm is therefore $3 + 6 + 9 + \ldots \frac{3n}{5} = \Theta(n^2)$, so we have the following lemma:

Lemma 7 *The* POCKET REDUCTION *algorithm employs* $\Theta(n^2)$ *forward moves.*

5 Single Pocket to Canonical Polygonization

Let $P(e)$ denote an arbitrary one-pocket polygonization of S with pocket lid $e = ab$. Here we give an algorithm to transform $P(e)$ into the canonical polygonization $P_c(e)$. This, along with the algorithms discussed in Sections 3 and 4, gives us a method to transform any polygonization of S into the canonical form $P_c(e)$. Our canonical polygonization algorithm incrementally arranges pocket vertices in canonical order (cf. Section 1.1) along the pocket boundary by applying a series of forward moves to $P(e)$.

CANONICAL POLYGONIZATION(P, e) ALGORITHM

Let $e = ab$. Let $a = v_0, v_1, v_2, \ldots, v_k, v_{k+1} = b$ be the canonical order of the vertices of pocket $P(e)$.
For each $i = 1, 2, \ldots, k$

1. Set $\ell_i \leftarrow$ line passing through a and v_i
2. Set $e_{i-1} \leftarrow$ pocket edge $v_{i-1}v_j$, with $j > i - 1$
3. If e_{i-1} is not identical to $v_{i-1}v_i$, apply FORWARDMOVE(e_{i-1}, v_i).

We now show that the one-pocket polygonization resulting after the i-th iteration of the loop above has the points $v_0, \ldots, v_i$ in canonical order along the pocket boundary. This, in turn, is established by showing that the FORWARDMOVE in the i-th iteration involves only points in the set $\{v_i, v_{i+1}, \ldots, v_k\}$. These observations are formalized in the following lemma, whose proof appears in Appendix 1:

Lemma 8 *The* i*-th iteration of the* CANONICAL POLYGONIZATION *loop produces a polygonization of* S *with one pocket with lid* e *and with vertices* $v_0, \ldots, v_i$ *consecutive along the pocket boundary.*

Lemma 9 *The* CANONICAL POLYGONIZATION *algorithm constructs* $P_c(e)$ *in* $O(n)$ *forward moves.*

6 Reverse Moves

Connectivity of the space of polygonizations will follow by reducing two given polygonizations P_1 and P_2 to a common canonical form P_c, and then reversing the moves from P_c to P_2. Although we could just define a reverse move as a time-reversal of a forward move, it must be admitted that such reverse moves are less natural than their forward counterparts. So we concentrate on establishing that reverse moves can be achieved by a sequence of atomic stretches and twangs.

Reverse Stretch. The reverse of STRETCH(e, v) may be achieved by a sequence of one or more twangs, as illustrated in Fig. 9a. This result follows from the fact that the "funnel" created by the stretch is empty, and so the twangs reversing the stretch do not cascade.

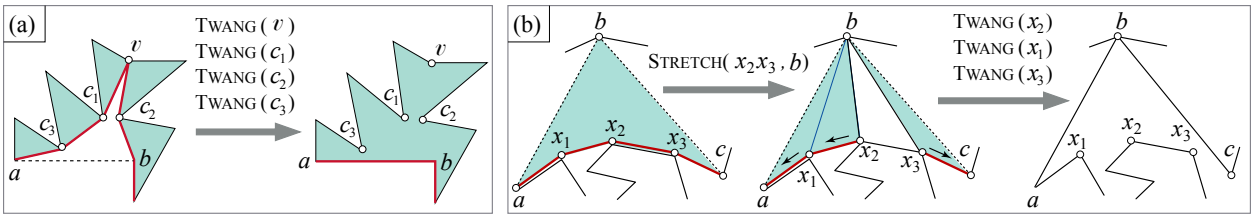

Figure 9: Reverse atomic moves: (a) STRETCH(ab, v) is reversed by TWANG(v), TWANG(c_1), TWANG(c_2), TWANG(c_3). (b) TWANG(b) is reversed by STRETCH(x_2x_3, b), TWANG(x_2), TWANG(x_1) and TWANG(x_3).

Reverse Twang. An "untwang" can be accomplished by one stretch followed by a series of twangs. Fig. 9b illustrates how TWANG(abc) may be reversed by one STRETCH(e, b), for any edge e of sp(abc), followed by zero or more twangs. Observe that the initial stretch in the reverse twang operation is not restricted to the reflex side of the stretch vertex, as it is in a FORWARDMOVE. If b is a hairpin vertex (i.e., a and c coincide), we view ac as an edge of length zero and the reverse of TWANG(b) is simply STRETCH(e, b).

Consequence. We have shown that the total effect of any forward move, consisting of one stretch and a twang cascade, can be reversed by a sequence of stretches and twangs. We call this sequence a *reverse move*. One way to view the consequence of the above two results can be expressed via regular expressions. Let the symbols s and t represent a STRETCH and TWANG respectively. Then a forward move can be represented by the expression st^+: a stretch followed by one or more twangs. A reverse stretch, s^{-1} can be achieved by one or more twangs: t^+. And a reverse twang t^{-1} can be achieved by st^*. Thus the reverse of the forward move st^+ is $(t^{-1})^+s^{-1} = (st^*)^+t^+$, a sequence of stretches and twangs, at least one of each.

7 Connectivity and Diameter of Polygonization Space

We begin with a summary the algorithm which, given two polygonizations P_1 and P_2 of a fixed point set, transforms P_1 into P_2 using stretches and twangs only.

POLYGON TRANSFORMATION(P_1, P_2) Algorithm

1. Select an arbitrary edge e of hull(P_1).
2. $P_1 \leftarrow$ POCKET REDUCTION(P_1, e); $M_1 \leftarrow$ atomic moves of [$P_2 \leftarrow$ POCKET REDUCTION(P_2, e)].
3. $P_c \leftarrow$ CANONICAL POLYGONIZATION(P_1, e); $M_2 \leftarrow$ atomic moves of [CANONICAL POLYGONIZATION(P_2, e).]
4. Reverse the order of the moves in $M_1 \oplus M_2$ ($\oplus$ represents concatenation).
5. For each stretch s (twang t) in $M_1 \oplus M_2$ in order, execute reverse stretch s^{-1}(reverse twang t^{-1}) on P_c.

This algorithm, along with Lemmas 7 and 9, establishes our main theorem:

Theorem 10 *The space of polygonizations of a fixed set of n points is connected via a sequence of forward and reverse moves. The diameter of the polygonization space is $O(n^2)$ moves.*

Computational Complexity. With appropriate preprocessing, each twang operation can be carried out in $O(n)$ time (since sp() might hit $O(n)$ vertices). So the running time of a single forward/reverse move is $T \cdot O(n)$, where T is an upper bound on the number of twangs in a move.

8 Random Polygons

Let G be the graph whose nodes are polygonizations and whose arcs are the moves defined in this paper. We know that G can have an exponential number N of vertices. We have established that it is connected, and that it has diameter $O(n^2)$. One way to generate "random polygons," as mentioned in Section 1, is to start with some polygonization P of a random set of n points S, and repeatedly select moves randomly. An immediate question here is: How many moves are needed to achieve adequate mixing, i.e., ergodicity? If we view the random moves as a random walk in G, and ask for the expected time for a random walk to visit all N vertices (the *expected cover time*) of a connected graph G, the answer is known: $\Theta(N^3)$ [Fei95]. Unfortunately, the maximum number of polygonizations of a set of n points has been shown to be $\Omega(4.6^n)$ [GNT00], and it seems likely that the expected number of polygonizations of a random set of points is also exponential (although we have not found this established in the literature). Thus, exponentially many moves are needed to cover the polygonization graph in the worst case, and perhaps in the expected case as well. Although disappointing, this is inevitable given the size of G, and mitigated somewhat by the relatively small diameter of G.

We have implemented a version of random polygon generation. After creating an initial polygonization, we move from polygonization to polygonization via a sequence of forward moves, where additional stretches are permitted in the cascade to simulate reverse moves. Trials on random polygons suggest that the average length of a cascade for polygons of up to n=100 vertices is about 1.3, with 7 the maximum cascade length observed in trials of thousands of forward moves. Cascade length seems to be independent of n. Thus, even though we only have loose bounds on the length of a twang cascade, for random point sets the mean length appears to be a constant less than 2.

9 Open Problems

Our work leaves many interesting problems open. The main unresolved question is establishing a tighter combinatorial bound on the number of twangs T in a twang cascade and thereby resolving the computational complexity of the polygon transformation algorithm. We have shown (in Appendices) that T is $\Omega(n^2)$ and $O(n^n)$, leaving a large gap to be closed. Another related question asks to improve the efficiency of the polygon transformation algorithm in terms of forward moves (the lower bound in Lemma 7 is for our particular algorithm, not all algorithms).

In Section 7 we established connectivity with forward moves and their reverse, and although both moves are composed of atomic stretches and twangs, the forward moves seem more naturally determined. This suggests the question of whether forward moves suffice to ensure connectivity. It remains to be seen if the polygonization moves explored in this paper will be effective tools for generating random polygons. One possibility is to start from a doubled random noncrossing spanning tree, which is a polygonal wrap. Finally, we are extending our work to 3D polyhedralizations of a fixed 3D point set.

References

[AH96] T. Auer and M. Held. Heuristics for the generation of random polygons. In *Proc. 8th Canad. Conf. Comput. Geom.*, pages 38–43, 1996.

[CHUZ92] J. Czyzowicz, F. Hurtado, J. Urrutia, and N. Zaguia. On polygons enclosing point sets. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 132–136, 1992.

[Fei95] U. Feige. A tight upper bound on the cover time for random walks on graphs. *RSA: Random Structures & Algorithms*, 6, 1995.

[GNT00] A. García, M. Noy, and J. Tejel. Lower bounds on the number of crossing-free subgraphs of k_n. *Comput. Geom. Theory Appl.*, 16(4):211–221, 2000.

[vLS82] J. van Leeuwen and A. A. Schoone. Untangling a travelling salesman tour in the plane. In J. R. Mühlbacher, editor, *Proc. 7th Internat. Workshop Graph-Theoret. Concepts Comput. Sci.*, pages 87–98, München, 1982. Hanser.

[ZSSM96] C. Zhu, G. Sundaram, J. Snoeyink, and J. S. B. Mitchell. Generating random polygons with given vertices. *Comput. Geom. Theory Appl.*, 6:277–290, 1996.

Appendix 1: Canonical Polygonization

Lemma 11 *The i-th iteration of the* Canonical Polygonization *loop produces a polygonization of S with one pocket with lid e and with vertices $v_0, \ldots, v_i$ consecutive along the pocket boundary.*

Proof: The proof is by induction. The base case corresponds to $i = 1$ and is trivially true for the case when $e_0 = v_0v_1$. Otherwise, v_1 sees e_0 (since no edge can block visibility from v_0 to v_1) and therefore Stretch(e_0, v_1) is possible. See Fig. 10a. Furthermore, v_1 may not twang a second time during the twang cascade of the forward move. This is because a second Twang(v_1) may only be triggered by the twang of a hull vertex, which can never occur (hull vertices never twang). This implies that the ForwardMove in Step 3 of the first iteration creates a one-pocket polygonization in which v_0 and v_1 are consecutive along the pocket boundary (see Fig. 10a,b). This completes the base case.

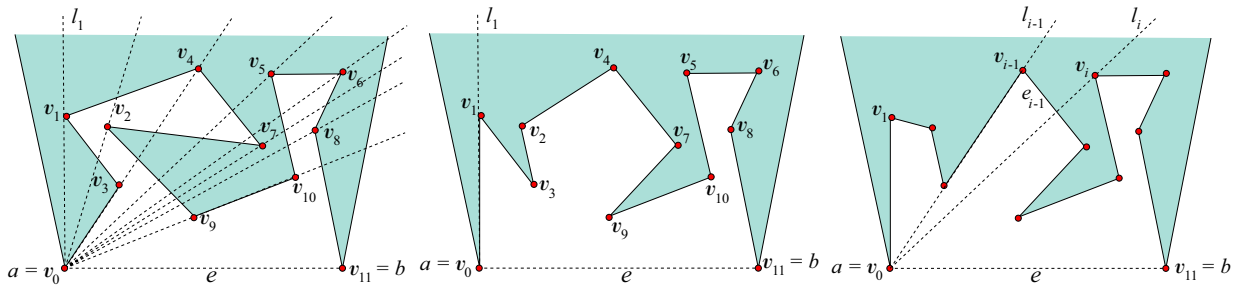

Figure 10: Canonical Polygonization: Stretch(v_i, x) is always possible. (a) Base case ($i = 1$): v_1 sees v_0 (b) After iteration 1, v_0 and v_1 are consecutive along the pocket boundary (c) v_i sees x.

To prove the inductive step, suppose that the lemma holds for iterations $1, \ldots, i-1$. Note that the existence of the edge e_{i-1} selected in Step 2 of the algorithm follows immediately from the fact that $v_0, v_1, \ldots v_{i-1}$ are consecutive along the pocket boundary (cf. inductive hypothesis). If e_{i-1} is identical to $v_{i-1}v_i$, there is nothing to prove. So assume that e_{i-1} and $v_{i-1}v_i$ are distinct. We now show that v_i sees e_{i-1}, so that Stretch(e_{i-1}, v_i) is possible.

First observe that the wedge bounded by ℓ_{i-1} and ℓ_i is either degenerate (if v_0, v_{i-1}, v_i are collinear), or is empty of any pocket points (since v_i follows v_{i-1} in the cw sorted order). In the former case, v_i sees v_{i-1} and e_{i-1}. In the latter case, e_{i-1} must intersect ℓ_i (cf. Fig. 10c). In either case, v_i sees e_{i-1} and hence Stretch(e_{i-1}, v_i) is possible. This along with the induction hypothesis implies that at the end of stretch operation, vertices $v_0, v_1, \ldots v_i$ are consecutive along the pocket boundary.

Next we show by contradiction that the twang cascade of the forward move involves only vertices $v_{i+1}, \ldots, v_k$, so that $v_0, v_1, \ldots v_i$ remain consecutive along the pocket boundary. Suppose the claim is false. For ease of presentation, define rank$(v_i) = i$. Let y be the first vertex with rank$(y) \leq i$ to get into double contact. Clearly y cannot coincide with a, since a is a hull vertex and cannot get into double contact. Let Twang(qrs) be the twang that created the double contact at y. Note that at the time of Twang(qrs), vertices $v_0, v_1, \ldots v_i$ are consecutive along the pocket boundary, since none of these vertices was in double contact prior to y (by choice of y) and therefore could not have twanged. Since Twang(qrs) creates the double contact at y, $y \in \triangle qrs$ and lies on sp(qrs). We also have that rank$(r) > i$, by our choice of y.

Two cases are possible: (i) y lies strictly to the left of ℓ_i, and (ii) y lies on ℓ_i In either case, since $y \in \triangle qrs$ and r lies on or to the right of ℓ_i, it must be that min{rank(q), rank(s)} $<$ rank(y). Suppose w.l.o.g that rank$(q) <$ rank(y). Thus we have that rank$(q) <$ rank$(y) \leq$ rank$(v_i) <$ rank(r). In other words rank$(r) -$ rank$(q) \geq 2$, but since $q \in \{v_0, v_1, \ldots v_{i-1}\}$ and qr is an edge of the pocket, it must be that rank$(r) -$ rank(q) $= 1$ (since $v_0, v_1, \ldots v_i$ are consecutive along the pocket boundary). Thus we have reached a contradiction. This completes the induction step. □

Appendix 2: Twang Cascade Lower Bound

Fig. 11 shows an example in which one point v twangs twice in a cascade, which gives a hint at the complex changes that can occur during a cascade. We will return to this example in Appendix 3 below.

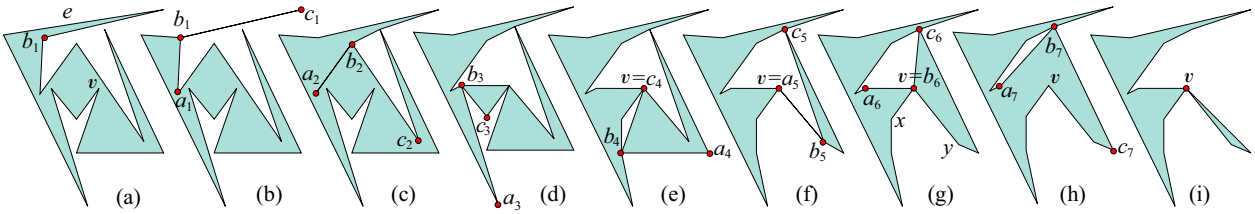

Figure 11: Point v twangs twice: (a) Initial P (b) After $\textsc{Stretch}(e, b_1)$ (c–i) After $\textsc{Twang}(a_i b_i c_i)$, $i = 1 \ldots 7$ (i) v in double contact a second time, and can twang a second time.

Figure 12 displays an example in which $\Omega(n)$ points each twang $\Omega(n)$ times in one cascade, providing an $\Omega(n^2)$ lower bound on the length of a cascade. The cascade is initiated by $\textsc{Stretch}(e, v)$ followed by $\textsc{Twang}(v)$. From then on $\textsc{Twang}(a_i, b_i, c_i)$ twangs in a cycle. Each such twang alters the path to $\mathrm{sp}(a_i, c_i)$, which wraps around b_{i+1}. In the next pass through the cycle, $\textsc{Twang}(a_i, b_{i+1}, c_i)$ occurs. This continues just twice in this figure, but in general the number of cycles is the number of b_j vertices inside each $\triangle a_i b_i c_i$. Figure 13 shows that this number can be $\Omega(n)$. Here the b_j vertices are evenly spaced on a circle, and

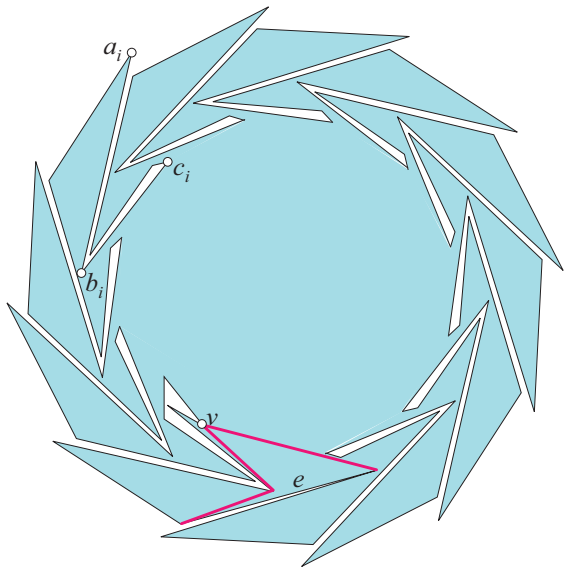

Figure 12: $\textsc{Stretch}(e, v)$ followed by $\textsc{Twang}(v)$ initiates a quadratic-length twang cascade.

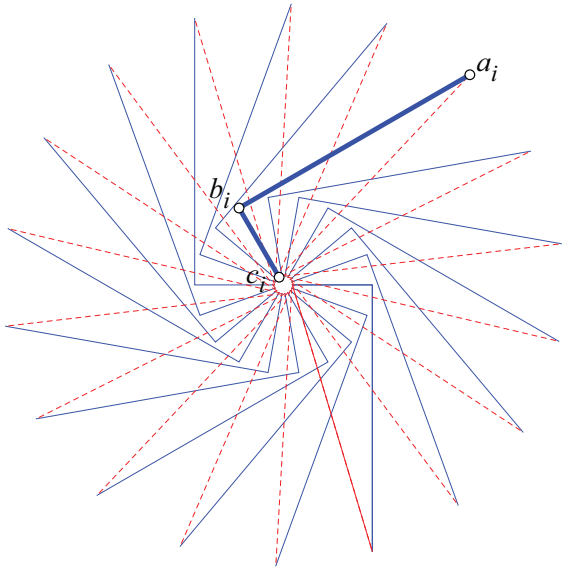

Figure 13: It is possible for each triangle $\triangle a_i b_i c_i$ to enclose $\Omega(n)$ vertices b_j.

$\angle(a_i, b_i, c_i) = 90°$. As the length $|a_i b_i|$ grows longer, the fraction of points inside $\triangle a_i b_i c_i$ approaches $n/4$.

Appendix 3: Twang Cascade Upper Bound

Figs. 6, 11, and 12 support an intuition that a twang cascade simplifies pocket entanglements in some sense. We capture this notion combinatorially in the *pocket hierarchy tree* T_P for a polygonization P, a unique representation of the nesting of pockets and subpockets of P in a tree. Each pocket has a *level* k, with P itself level 0, the pockets described in Section 1.1 as level-1 pockets, and its subpockets at level 2, and so on. We will call all of these *pockets*, and use a superscript to distinguish the level k and subscripts to distinguish among the pockets at the same level: P_i^k.

Each node of T_P is the list of vertices in hull(P_i^k). If P_i^k is convex, it has no children. Otherwise, each edge ab of hull(P_i^k) which is not an edge of the polygonization P is a pocket lid for the vertices of P from a to b. Fig. 14 illustrates the hierarchy. Note that every vertex $v \in P$ is on the hull of one or more pockets of T_P.

For each pocket P_i^k, we define a *pocket count* $S(P_i^k)$ to be the number of points on or in hull(P_i^k). For example, in Fig. 14c, the pocket with lid $(7, 0)$ contains points $\{7, 0, 8, 10, 11, 3, 5, 9\}$ and so has pocket count 8. Note that we count points 8 and 10 only once even though they are in double contact. Finally, we define the *pocket vector* V for P to list the sum of all pocket counts for all pockets at each level: $V = <V_1, V_2, V_3, \ldots>$, $V_k = \sum_i S(P_i^k)$. The pocket vectors in Fig. 14 are:

(c)	$<13, 18, 14, 3>$
(d)	$<13, 18, 13, 5, 4>$
(e)	$<13, 17, 10, 5, 4>$
(f)	$<13, 17, 9, 5, 4>$
(g)	$<13, 17, 9, 4, 3>$
(h)	$<13, 17, 9, 3>$
(i)	$<13, 17, 8>$

Let $V = <V_1, V_2, V_3, \ldots>$ and $W = <W_1, W_2, W_3, \ldots>$ be two pocket vectors. We define a lexicographic ordering relation on them: $V < W$ iff $V_1 = W_1, V_2 = W_2, \ldots, V_k = W_k, V_{k+1} < W_{k+1}$. The steady odometer-like decrementing of the pocket vector evident in the above example is the sense in which a twang cascade simplifies pocket structure:

Theorem 12 *A twang* TWANG(abc) *always strictly reduces the pocket vector.*

Proof: Assume P_i^k is the highest pocket in the hierarchy for which vertex b is on hull(P_i^k). We know at least one such pocket exists.

Case 1: b is a lid vertex of pocket P_i^k.

We show by contradiction that this case is impossible: Since b is a lid vertex of pocket P_i^k, it must be a vertex on the hull of the parent of P_i^k. But this contradicts our assumption that P_i^k is the highest pocket having b on its hull. (If b has no parent because it is a level-1 pocket, then it must be on the hull of the entire polygon, which is a contradiction since such vertices are never twanged.)

For example, in Fig. 14c, let $b = 10$. Then b is a lid vertex of pocket $\{10, 8, 9\}$ at level 3, but it is also a non-lid hull vertex of pocket $\{11, 7, 8, 10\}$ at level 2.

Case 2: b is not a lid vertex of pocket P_i^k.

This case implies that a, b, and c are all part of pocket P_i^k's boundary, as are edges ab and bc. Because a twang is performed at b, we know b is a corner point of hull(P_i^k). Thus after the twang, hull(P_i^k) loses this corner point and $S(P_i^k)$ goes down by one.

Now we show that the pocket counts for pockets at levels $< k$ do not change as a result of TWANG(abc), and in addition no other pocket besides P_i^k at level k changes. First, observe that in addition to P_i^k, the only pockets that may include edges ab and bc are P_i^k's ancestors and descendents in the hierarchy. Therefore, TWANG(abc) affects only these pockets and no others. Of these pockets, only changes in P_i^k's ancestors could increase the pocket vector. We show now that the counts for the ancestors do not changed. First observe that only the lid vertices of a pocket are hull vertices of its parent. Since b is not a lid vertex, TWANG(abc) cannot change the hull of its parent, and similarly cannot change the hull of any of its ancestors. If the hulls do not change, then the point counts for the pockets do not change.

□

Corollary 13 *A twang cascade can have at most $O(n^n)$ steps.*

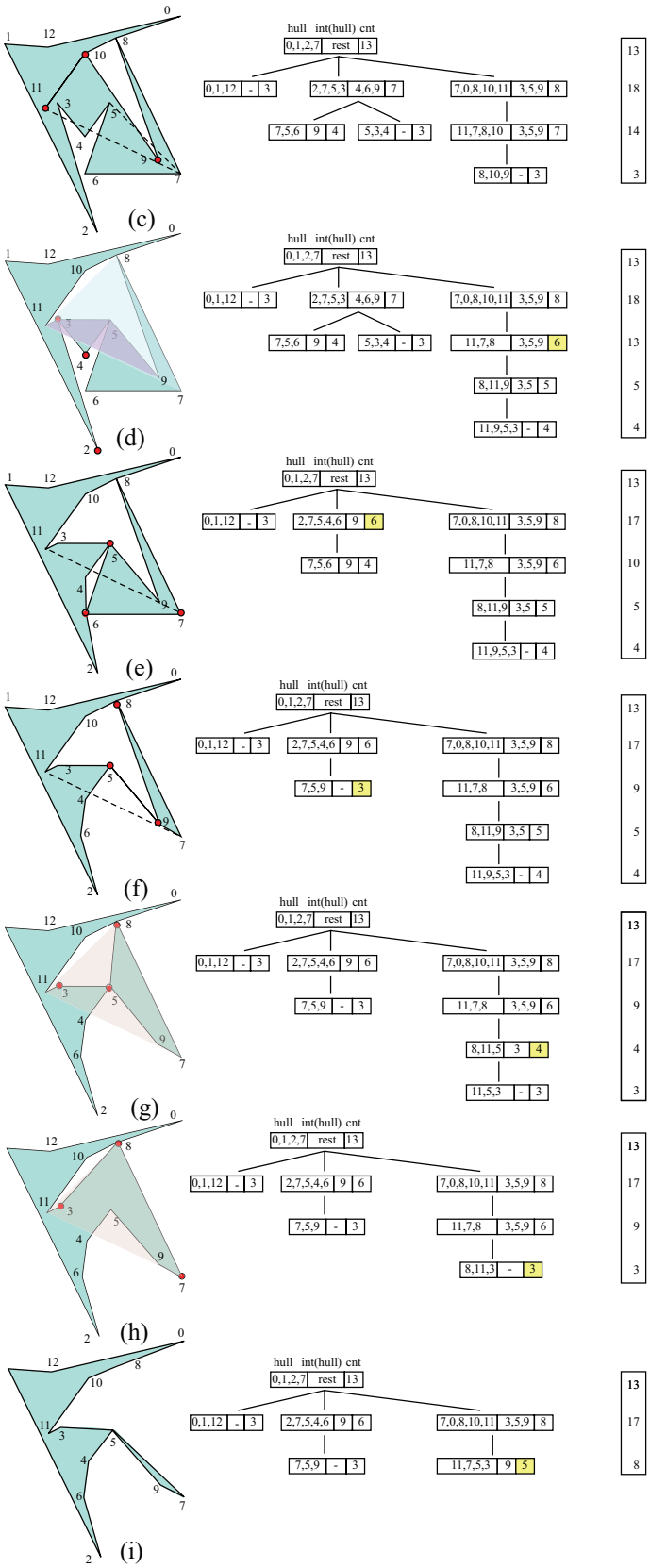

Figure 14: Pocket hierarchy for cascade illustrated in Fig. 11. Middle column displays the pocket tree (each node showing: points on hull, points interior, $S()$ pocket count), right column the pocket vector transposed.

Proof: Let m be the maximum length of a pocket vector: $m = n/3 = O(n)$. In the worst case, a decrement of V_k by 1 is followed by $V_k, \ldots, V_m$ each being (somehow) reset to n and counting down through their full range. If this happens for every decrement, then the "odometer" counts through all m^n distinct possible pocket vectors. □

Note particularly the change in the pocket vector from Fig. 14c to d: although V_2 decrements from 14 to 13, V_3 increases from 3 to 5 (and, in addition, T_P grows in depth). It is this phenomenon that makes establishing a better bound via the pocket hierarchy problematical.