On finding a minimal enclosing parallelogram

Christian Schwarz* Jürgen Teich† Emo Welzl‡

Brian Evans†

TR-94-036

August 1994

Abstract

Given a convex polygon C with n vertices, we show how a parallelogram with minimal area enclosing C can be computed in linear time $O(n)$. The problem is of interest in digital signal processing.

*International Computer Science Institute (ICSI), Berkeley, USA.

†Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, USA.

‡FU Berlin, Germany. Work was done while this author was visiting ICSI Berkeley.

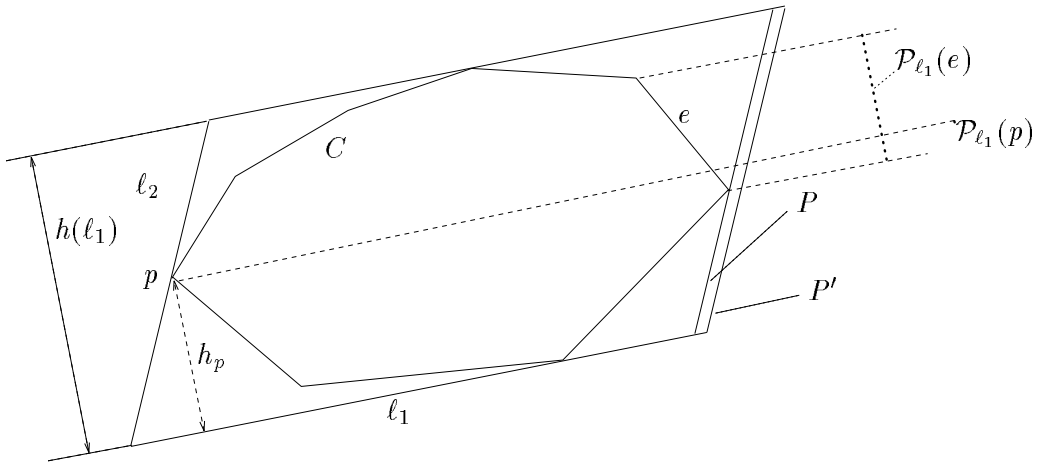

Figure 1: Illustration of terminology associated with MEP's. Each side of a MEP of C must touch C; P' cannot be a MEP of C.

---

1 Introduction

Let C be a convex polygon in the plane. We denote by P_C a parallelogram that encloses C and that has minimal area among all such parallelograms. In this paper, we show how to compute P_C in linear time if the ordered list of edges or vertices of C is given.

Related work has been done on finding a minimal enclosing triangle, see e.g. [KL85, OAMB86], a minimal enclosing square [FS75], a minimal enclosing k-gon [CY84, ACY85], and a minimal enclosing equiangular or regular k-gon [DA84]. The purpose of the latter paper was to identify a restricted class of k-gons that yielded algorithms that were faster than those for the general enclosing k-gon problem. In a regular polygon, all edges have equal length. In equiangular polygons, all internal angles are the same, and the algorithm in [DA84] was in fact extended to the more general class of k-gons whose internal angles form a fixed sequence $\gamma_1, \ldots, \gamma_k$, where $\gamma_i < \pi, 1 \leq i \leq k$ and $\sum_{i=1}^{k} \gamma_i = (k-2)\pi$.

In this paper we show how to compute, for a given convex polygon with n vertices, its enclosing parallelogram in $O(n)$ time. Note that whereas e.g. a rectangle would be contained in the above described class of polygons with a fixed angle sequence, our problem is different since the angles of the desired enclosing parallelogram are not given in advance. In Section 2 we prove some important properties of the problem. In Section 3, we give the algorithm, and in Section 4 we describe an application of the algorithm to a problem in digital signal processing.

2 Properties of a minimal enclosing parallelogram

The parallel line pairs of a parallelogram P are called the *axes* ℓ_1, ℓ_2 of P. Each axis of P has an upper and a lower side. For a given convex polygon C, a *minimal enclosing parallelogram (MEP)* P_C is a parallelogram which contains C and has minimal *area* among all such parallelograms.

As shown in Figure 1, each side of P_C must intersect (i.e. touch) C in at least one point. Let p be an intersection of C and P_C on the axis ℓ_2. The *height* h_p is defined as the distance of p to its orthogonal projection onto the lower side of ℓ_1. The distance between the two sides of ℓ_1 is denoted

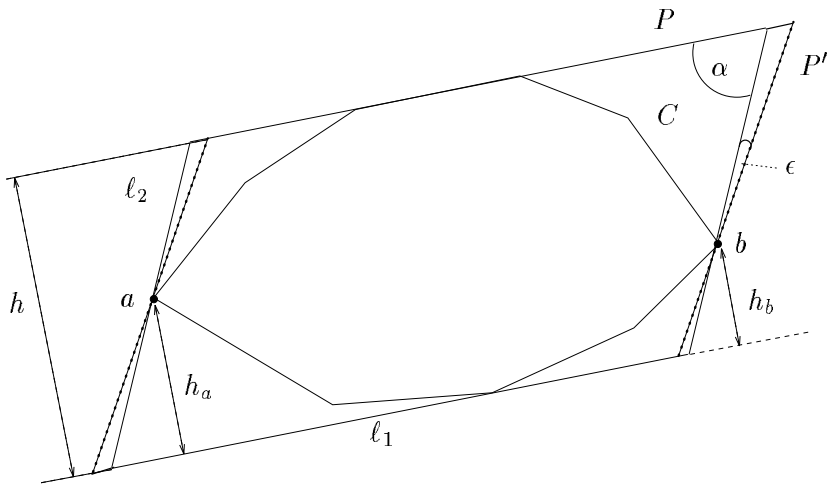

Figure 2: Illustration of the proof of Lemma 1.

---

by $h(\ell_1)$. Let f be a vertex or an edge of C. The *projection of f w.r.t. ℓ_1*, denoted by $\mathcal{P}_{\ell_1}(f)$, is the projection of f onto the plane orthogonal to ℓ_1. Analogous definitions hold for ℓ_2 replacing ℓ_1.

Lemma 1 *For any convex polygon C, there is a MEP P_C such that for each axis of P_C, there is one side that contains an edge of C.*

Proof: Let P_C be a MEP of C. Assume that an axis, w.l.o.g. ℓ_2, does not contain a polygon edge on both sides. Let a and b be the intersection points of C with P_C, as illustrated in Figure 2. We will show that if $h_a \neq h_b$, then we are able to rotate axis ℓ_2 around a and b with a suitably small angle ϵ such that the resulting parallelogram has smaller area than P_C, which is a contradiction to the minimality of P_C.

Refer to Figure 2. Let α be the sharp angle between ℓ_1 and ℓ_2, $0 < \alpha \leq \pi/2$, and let $h := h(\ell_1)$. Assume we rotate axis ℓ_2 around a and b clockwise with angle ϵ. Then the area of the parallelogram changes by $\delta \cdot h \cdot (h_a - h_b)$, where $\delta = \tan(\epsilon + \pi/2 - \alpha) - \tan(\pi/2 - \alpha)$, which, for suitably small ϵ, has the same sign as ϵ, since $0 < \alpha \leq \pi/2$. Since $h \neq 0$, we need $h_a = h_b$, because otherwise a suitable choice of ϵ results in a parallelogram with smaller area than P_C.

However, when $h_a = h_b$, rotating ℓ_2 around a and b yields a parallelogram with the same area. Consider the outer angles of C with ℓ_2 at a and at b. Rotating by the smallest of these four angles results in a MEP P'_C with one of the sides of ℓ_2 containing a polygon edge. ∎

As illustrated in Figure 3, Lemma 1 states that each axis of P_C contains an edge of C.

Lemma 2 *Let C be a convex polygon and let P_C be a MEP of C. Let e be the edge of C contained in P_C by Lemma 1. W.l.o.g. let e be on axis ℓ_2. Let f be the intersection of C with the other side of ℓ_2 in P_C. (f can be a vertex or an edge of C.) Then the projections $\mathcal{P}_{\ell_2}(e)$ and $\mathcal{P}_{\ell_2}(f)$ have a non-empty intersection.*

Proof: We distinguish two cases. Refer to Figure 4.

In the first case, assume that f is a vertex p of C. Then the lemma claims $h_{e_1} \leq h_p \leq h_{e_2}$. Assume w.l.o.g. that $h_p > h_{e_2}$. Rotate ℓ_2 counterclockwise by a suitably small angle ϵ. Similar to

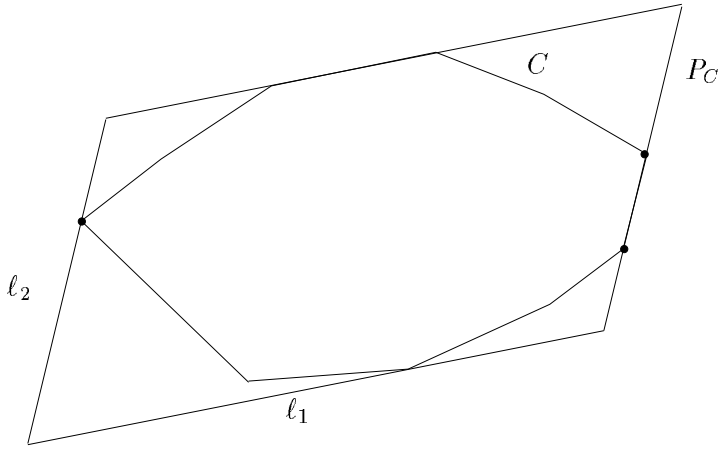

Figure 3: Each axis of P_C contains an edge of C.

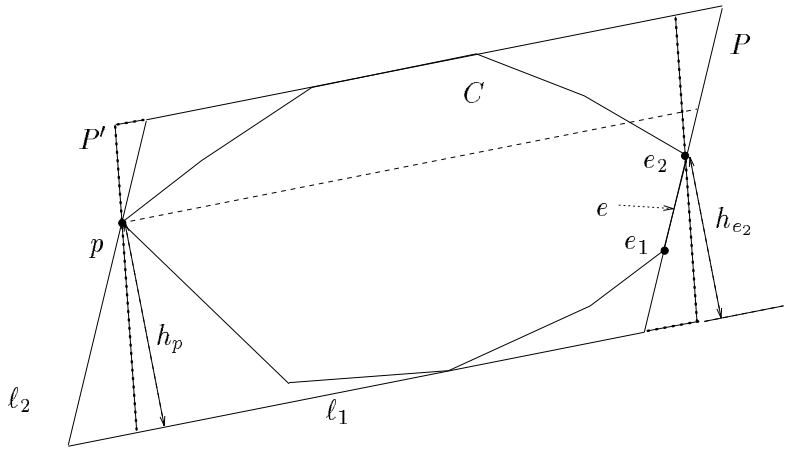

Figure 4: Illustration of the proof of Lemma 2. The projections $\mathcal{P}_{\ell_1}(p)$ and $\mathcal{P}_{\ell_1}(e)$ do not intersect, and the parallelogram P' obtained from P by rotation of ℓ_2 has smaller area than P.

the proof of Lemma 1, the area change is proportional to $h(\ell_1) \cdot (h_{e_2} - h_p) < 0$, a contradiction to the minimality of P_C.

In the second case, f is an edge of C rather than a vertex. Denoting the endpoints of this edge by f_1 and f_2, we can show that if $h_{f_2} < h_{e_1}$ (or $h_{e_2} < h_{f_1}$), then rotating axis ℓ_2 counterclockwise around e_1 and f_2 (or clockwise around e_2 and f_1) yields a parallelogram P' with smaller area. ∎

3 The Algorithm

Let e and v be an edge and a vertex of a convex polygon C, respectively. Furthermore, let ℓ_e be the supporting line of e and ℓ_v the line through v that is parallel to ℓ_e. The pair (e, v) is called an *antipodal pair* of C if ℓ_e and ℓ_v support C. (The supporting line of a line segment e is the line containing e, and the supporting line of a convex polygon C is a tangent of C.) Using this terminology, it follows from Lemma 1 that there is a parallelogram P_C such that for both its axes ℓ_1, ℓ_2, the intersection with C consists of an antipodal pair. Therefore, we can compute P_C as follows.

The simple algorithm

1. compute a list L of all antipodal pairs of C
2. for each pair of antipodal pairs in L, compute the area of the induced enclosing parallelogram, and report the one with minimum area.

Let n denote the number of vertices in C. Then there are $O(n)$ antipodal pairs, and they can be computed in $O(n)$ time, using the "rotating calipers" algorithm, see [HT85]. Step 2, which checks all pairs in the list computed in step 1, consequently takes time $O(n^2)$. We have

Lemma 3 *The simple algorithm given above computes P_C in $O(n^2)$ time.*

Note that the above algorithm only uses the basic property given in Lemma 1. Using Lemma 2 we shall obtain a linear time algorithm.

The refined algorithm

1. compute a list $L = L[1], \ldots, L[n]$ of all antipodal pairs of C, such that they are stored in clockwise ordering of their edges, each of which has a pointer to its antipodal vertex.[1]
2. We move two edge pointers z_1, z_2 clockwise around C. Each configuration z_1, z_2 represents a pair of antipodal pairs, and the corresponding enclosing parallelogram is computed as in the simple algorithm. Briefly, during the clockwise movement of z_1 and z_2, z_1 is moved once around the polygon, while z_2 is always between one and $n-1$ edges ahead of z_1. (That is, intuitively, both z_1 and z_2 only move forward and z_2 can never "pass" z_1.)

   We now describe precisely the scheme to move the pointers around C. For convenience, we identify edges by their number in L, i.e. we use $z_1 = i$ to denote that z_1 points to the i-th element in L. Consider the rays emanating clockwise from z_1 and z_2. We say (z_1, z_2) is a *candidate pair* if the clockwise rotation α of the ray from z_2 to that of z_1 satisfies $0 < \alpha < \pi$. See Figure 5.

[1] There are two (adjacent) vertices when there are parallel edges in C, and we canonically refer to the first in clockwise order in the following.

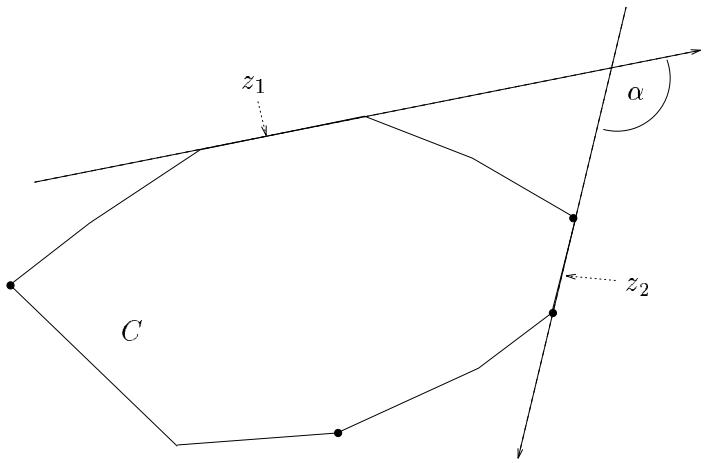

Figure 5: (z_1, z_2) is a candidate pair of the convex polygon C.

It is clear that for any two edges $z_1 \neq z_2$, either (z_1, z_2) or (z_2, z_1) is a candidate pair. Since the parallelogram corresponding to these pairs is the same, it suffices to consider all candidate pairs. We denote by $CP(z_1)$ the sequence of edges that form a candidate pair with z_1 as its first component. These edges form a continous subsequence of the polygon, so $CP(z_1) = [z_1 + 1, z_1 + 2, \ldots, z_1 + m_{z_1}]$, where edge numbers are taken modulo n.

We start by setting $z_1 := z_2 := 1$. Then we repeat the following steps:

(a) If $z_1 = z_2$, advance z_2 by one. Let e be the edge in C pointed to by z_2, let v be its antipodal vertex, and let ℓ be the supporting line of the edge pointed to by z_1. We define $\mathcal{P}_\ell$, the projection w.r.t. ℓ, as we defined projection w.r.t. an axis in Section 2. Then $\mathcal{P}_\ell(v)$ is a real number and $\mathcal{P}_\ell(e)$ is an interval in $\mathbb{R}$. As long as $\mathcal{P}_\ell(v) < \mathcal{P}_\ell(e)$, i.e. $\mathcal{P}_\ell(v)$ is smaller than the lower boundary of $\mathcal{P}_\ell(e)$, advance the pointer z_2.

(b) If $\mathcal{P}_\ell(v) \in \mathcal{P}_\ell(e)$, compute the corresponding parallelogram and keep track of the minimum computed so far, otherwise do nothing. In the special case where $\mathcal{P}_\ell(v)$ intersects the lower boundary of $\mathcal{P}_\ell(e)$, advance z_2 to the successor of e, say e', and do the same computation for the pair (e', v).

If z_1 is the last element of L, **stop**, otherwise advance z_1 by one and go to step a).

Lemma 4 *The refined algorithm correctly computes a MEP and runs in time $O(n)$.*

Proof: Let z_1 and z_2 be defined as above. We have seen that out of the $n(n-1)$ pairs of edges $z_1 \neq z_2$, we only need to check, i.e. compute the resulting parallelogram, for edge pairs defined as candidate pairs in the description of the algorithm. We shall prove that all candidate pairs which are not checked cannot form a MEP by Lemma 2.

For a candidate pair (z_1, z_2), i.e. $z_2 \in CP(z_1)$, we define a *signature*

$$\sigma_{z_1}(z_2) = \begin{cases} 1, & \mathcal{P}_{z_1}(v) > \mathcal{P}_{z_1}(e) \\ 0, & \mathcal{P}_{z_1}(v) \in \mathcal{P}_{z_1}(e) \\ -1, & \mathcal{P}_{z_1}(v) < \mathcal{P}_{z_1}(e) \end{cases}$$

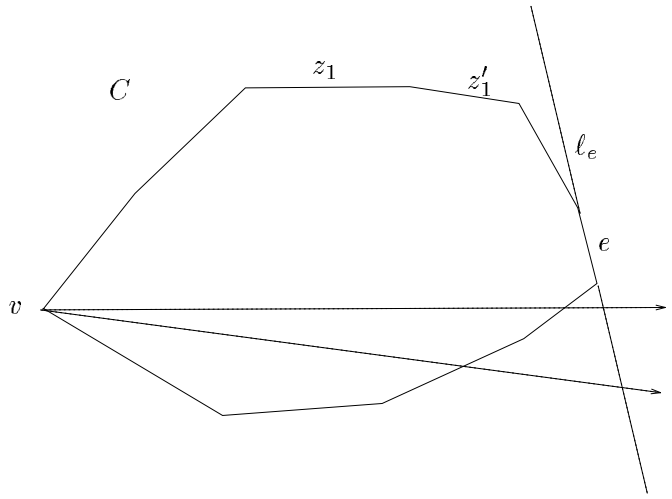

Figure 6: Illustration of the proof of Claim 2.

---

where e is the edge with number z_2 and v is its antipodal vertex (the first in clockwise order if there are two antipodal vertices.)

Claim 1: The sequence of z_2's that are checked for one value of z_1 is a subsequence of $CP(z_1)$, and the signature function σ_{z_1} is monotonic on $CP(z_1)$. Furthermore, if C is in general position, there is at most one z_2 in $CP(z_1)$ such that $\sigma_{z_1}(z_2) = 0$. Otherwise, in case there is an edge z_1 such that for two vertices v, v', we have $\mathcal{P}_{z_1}(v) = \mathcal{P}_{z_1}(v')$, then there are at most two z_2 for which $\sigma_{z_1}(z_2) = 0$, and these must point to adjacent edges.

Proof: First observe that if $\sigma_{z_1}(z_2) = -1$, then the angle of the rays emanating from z_1 and the clockwise successor of z_2 as depicted in Figure 5 must be less than π. Therefore, the sequence of z_2's checked in one iteration of z_1 is actually a subsequence of $CP(z_1)$.

Now let z_2 and z_2' be in $CP(z_1)$ such that z_2 is closer to z_1 in clockwise ordering, i.e. comes before z_2' in $CP(z_1)$. Let e and e' be the edges numbered z_2 and z_2', respectively, and let v and v' their corresponding antipodal vertices as defined above. Apart from intersection at common extreme points of adjacent edges, the projections of the edges in $CP(z_1)$ form a partition, and $\mathcal{P}_{z_1}(e) > \mathcal{P}_{z_1}(e')$. Also, $\mathcal{P}_{z_1}(v) \leq \mathcal{P}_{z_1}(v')$ by the geometry of antipodal vertex-edge pairs. This shows that $\sigma_{z_1}(z_2) \leq \sigma_{z_1}(z_2)$, i.e. σ_{z_1} is monotonic on $CP(z_1)$.

Finally, if C is in general position, there can be at most one antipodal pair (e, v) such that $\mathcal{P}_{z_1}(v) \in \mathcal{P}_{z_1}(e)$. In the case where there are two edges e and e' that are adjacent and v is projected to the common endpoint of e and e', we get two antipodal pairs (e, v), (e', v) with this property, proving the claim on the uniqueness of z_2 for which $\sigma_{z_1}(z_2) = 0$. ∎

Claim 2: Let $z_2 \in CP(z_1)$ and $z_2 \in CP(z_1 + 1)$. If $\sigma_{z_1}(z_2) = -1$, then $\sigma_{z_1+1}(z_2) = -1$. If $\sigma_{z_1}(z_2) = 0$ and $\sigma_{z_1}(z_2 + 1) = 0$, then $\sigma_{z_1+1}(z_2) = -1$ also holds.

Proof: Let e be the edge with number z_2 and v its antipodal vertex. The condition $\mathcal{P}_{z_1}(v)$ op $\mathcal{P}_{z_1}(e)$, op $\in \{<, \in, >\}$, can be checked as follows. Refer to Figure 6. Draw a line parallel to z_1 through v, and let p the intersection of this line with the supporting line ℓ_e of e. Then $\sigma_{z_1}(z_2) = -1, 0$ or 1, if

p is below, on or above e. Now assume that p is below e on ℓ_e. If we move from z_1 to z_1+1, the parallel line through v moves clockwise by the angle between z_1 and z_1+1. If the intersection p with ℓ_e was below the edge e, it is even further down now, which proves the first part of the claim.

If $\sigma_{z_1}(z_2) = 0$ and $\sigma_{z_1}(z_2+1) = 0$, then we have the special case that there are two adjacent edges e, e' and a vertex v such that $\mathcal{P}_{z_1}(v)$ intersects the projections of e and e' at their lower and upper boundary, respectively. Using an argument similar to the general case, see Figure 6, it follows that the projection w.r.t. to z_1' of the upper edge e must be completely above $\mathcal{P}_{z_1'}(v)$, and so the signature $\sigma_{z_1'}$ is -1 for this antipodal pair. ∎

We now return to the proof of the lemma. Consider the beginning of a new iteration, when we move z_1 to z_1+1. Then the algorithm does not check the candidate pairs $(z_1+1, z_1+2), \ldots, (z_1+1, z_2-1)$. From the description of the algorithm, if the polygon is in general position, then the pointer z_2 has moved to the first position z_2 for which $\sigma_{z_1}(z_2) \geq 0$. Therefore $\sigma_{z_1}(z_2-1) = -1$ and by Claim 2, we have $\sigma_{z_1+1}(z_2-1) = -1$. In the special case where $\sigma_{z_1}(z_2-1) = 0$ and $\sigma_{z_1}(z_2) = 0$, Claim 2 also shows that $\sigma_{z_1+1}(z_2-1) = -1$. Using Claim 1, we get that $\sigma_{z_1+1}(z) = -1$ for $z \in \{z_1+2, z_1+3, \ldots, z_2-1\}$.

It follows that the algorithm checks all candidate pairs (z_1, z_2) for which $\sigma_{z_1}(z_2) = 0$. Lemma 2 implies that these are the only pairs which can give a MEP. This establishes the correctness of the algorithm.

Since z_2 is at most n edges ahead of z_1, and z_1 moves once around the polygon, the running time of the algorithm is $O(n)$. ∎

We summarize in

Theorem 1 *Let C be a convex polygon with n vertices, and assume the edges of C are available in sorted order. Then a minimal enclosing parallelogram P_C can be computed in time $O(n)$.*

4 Application

The application that motivated this research is compressing two-dimensional signals based on their frequency content. We give a brief sketch and refer the reader to [ETS94] for details. A common two-dimensional signal is an image, which has two spatial axes [Lim90]. Another two-dimensional signal arises in measuring seismic data which requires a position axis and a time axis.

The frequency content of certain image, seismic, and other two-dimensional signals may cluster in one region. For example, when an image is digitized from a photograph, the digitized data is often *oversampled*. That is, the amount of data can be reduced with no loss of information. In seismic processing, only one connected region in the frequency spectrum is preserved. After the processing, the resulting signal can also be compressed without loss of information.

Viewing the frequency content of a signal as a density plot will reveal if the frequency content clusters in one region. If so, a designer using a pointing device can outline the region to be preserved. The preserved region can then be extracted and compressed.

We use the cascade of linear operations shown in Figure 7 to extract and compress the preserved regions. Figure 7 shows the frequency content of the signal at the input of the cascade which is parallelogram-shaped. The cascade of linear operations consists of an upsampler, filter, and downsampler. In the "time" (spatial) domain, the upsampler changes the basis vectors of its input signal and increases the number of samples by a factor equal to $|\det L|$. In the frequency domain, the increase in the sampling rate causes $|\det L| - 1$ shifted copies of the frequency content to be created. The filter extracts one of the copies of the region to be preserved (in Figure 7, the region

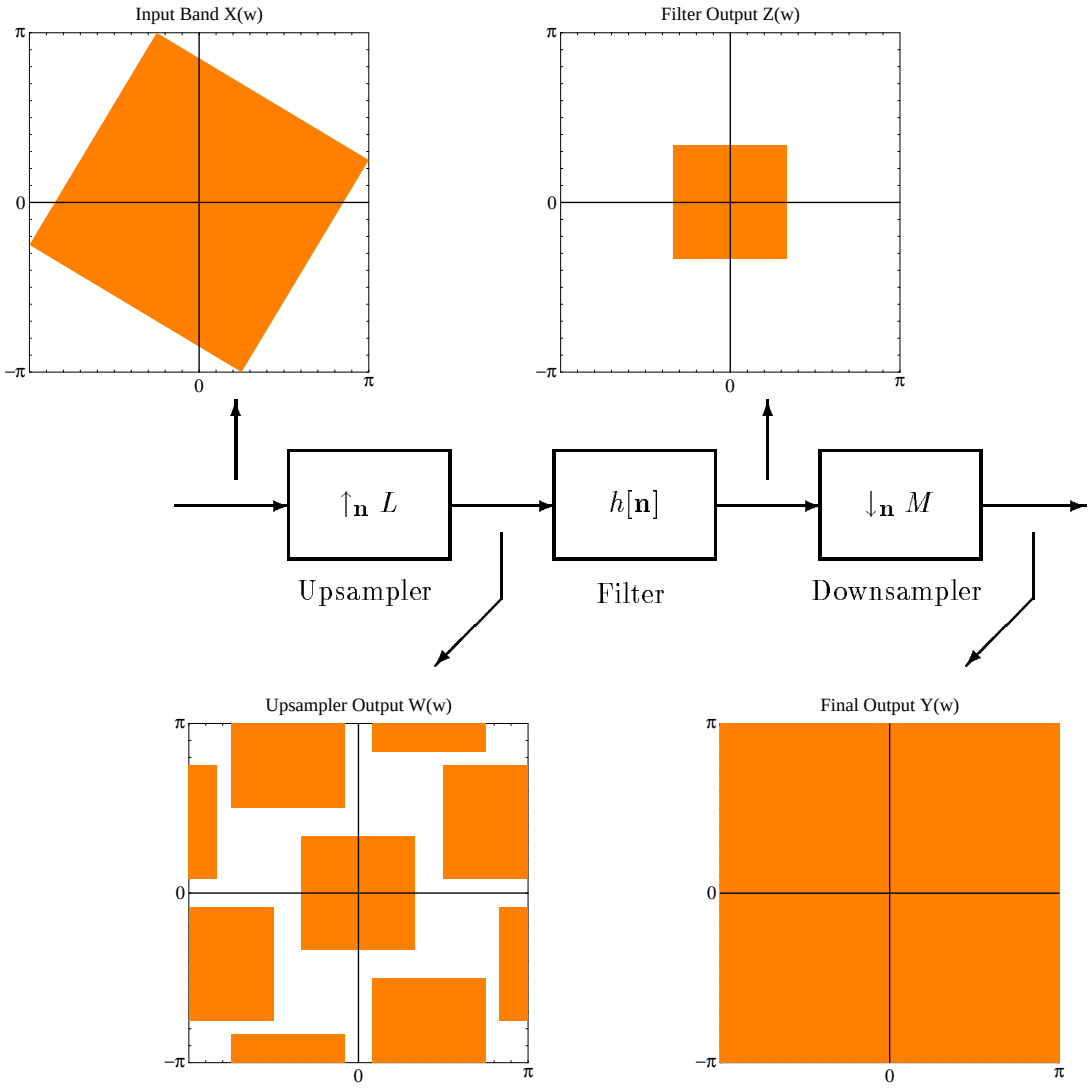

Figure 7: Flow Graph of a Linear Two-Dimensional Compression System

---

with the lowest frequency is extracted). In the "time" domain, the downsampler changes the basis vectors of its input signal and decreases the number of samples by a factor of $|\det M|$. With L and M computed correctly, the output of the downsampler will be the input signal resampled at the lowest rate possible without losing any information.

The overall cascade essentially maps one preserved connected region onto one period of the frequency domain. (The frequency domain is periodic with period of 2π in each of the two frequency variables.) Since the system is linear, the mapping is performed by a rational matrix (i.e., a linear operator). The rational matrix can always be factored into a product $L^{-1}M$, where L and M are integer matrices, see [CV93, ETS94]. The overall compression ratio is the area of one period of the frequency domain ($4\pi^2$) divided by the area of the parallelogram, which is equivalent to $|\det M|/|\det L|$. The compression ratio thus measures what we save in sampling: the ratio of the total number of samples in the original signal divided by the number of samples in the output signal.

The frequency content of the input signal can be any shape that when periodically replicated

with the same orientation fills the entire two-dimensional plane. For convex polygon shapes, the only two candidates are parallelograms and elongated hexagons. In the case of the parallelogram, the procedure to compute the rational matrix mapping is already known [CV93]. (The vertices of the parallelogram, however, must be rational multiples of π to compute the rational matrix.) Resampling elongated hexagons is an area of future work.

5 Conclusion

In this paper, we considered the problem of computing a minimum area enclosing parallelogram for a given convex polygon. For a polygon with n vertices, the algorithm runs in time $O(n)$. As a preliminary step, we also described a simpler algorithm that runs in time $O(n^2)$. Previously known methods discretized the problem using a grid. While a fine resolution is desirable for appropriate modeling, the complexity of such a method increases sharply with the resolution of that grid.

We implemented and animated both the simple quadratic and the linear time algorithm in `C++`, using the algorithms library LEDA [NM90]. As a result, we found that the implementation of the linear-time algorithm is not substantially more difficult than that of the simple quadratic algorithm. Consequently, the code sizes for both algorithms are almost equal, and the same holds for the constant factors in the running times. This is strongly reflected by the fact that the linear-time algorithm is already faster than the quadratic algorithm for the smallest reasonable problem size $n = 5$. We give more details of the implementation and our experiments in Appendix A. As further work, we plan to include the linear-time algorithm in the digital signal processing system Ptolemy [BHLM94].

Finally, there are more related geometric problems that arise in design of digital processing methods as described in Section 4. First, elongated hexagons are another desirable shape to enclose filters that occupy a single region of the frequency domain. Also, one would like to be able to process filters that occupy more than one region of the frequency domain.

References

[ACY85] A. Aggarwal, J.S. Chang, and C.K. Yap. Minimum area circumscribing polygons. *The Visual Computer: International Journal of Graphics*, 1:112–117, 1985.

[BHLM94] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A framework for simultating and prototyping hetereogenous systems. *Int. Journal of Computer Simulation, special issue on Simulation Software Development*, 4:155–182, April 1994.

[CV93] T. Chen and P. P. Vaidyanathan. The role of integer matrices in multidimensional multirate systems. *IEEE Transactions on Signal Processing*, 41(3):1035–1047, March 1993.

[CY84] J.S. Chang and C.K. Yap. A polynomial solution for potato peeling and other polygon inclusion and enclosure problems. In *Proc. 25th Annual IEEE Sympos. Found. Comput. Sci.*, pages 408–419, 1984.

[DA84] A. DePano and A. Aggarwal. Finding restricted k-envelopes for convex polygons. In *Proc. 22nd Allerton Conf. on Comm. Control and Computing*, pages 81–90, 1984.

n	5	10	20	40	100	1000
antipodal_pairs	0.00	0.01	0.02	0.03	0.07	0.63
simple_algorithm	0.04	0.17	0.71	2.63	17.55	1577.38
linear_algorithm	0.02	0.03	0.07	0.11	0.30	2.57

Table 1: Running times (in seconds) of the main parts of the algorithms on a SPARC-1 workstation. n is the number of vertices of the given convex polygon.

[ETS94] B.L. Evans, J. Teich, and C. Schwarz. Automated design of two-dimensional rational decimation systems. In *IEEE Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, Nov 1994.

[FS75] H. Freeman and R. Shapira. Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Communications of the ACM*, 18:409–413, 1975.

[HT85] M. E. Houle and G. T. Toussaint. Computing the width of a set. In *Proc. 1st Annual ACM Sympos. Comput. Geom.*, pages 1–7, 1985.

[KL85] V. Klee and M.C. Laskowski. Finding smallest triangles containing the given polygon. *J. Algorithms*, 6:359–375, 1985.

[Lim90] J. Lim. *Two-Dimensional Signal and Image Processing*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1990.

[NM90] S. Näher and K. Mehlhorn. LEDA — a library of efficient data types and algorithms. In *Proc. 17th Internat. Colloq. Autom. Lang. Prog.*, volume 443 of *Lecture Notes in Computer Science*, pages 1–5. Springer-Verlag, 1990.

[OAMB86] J. O'Rourke, A. Aggarwal, S. Maddial, and M. Baldwin. An optimal algorithm for finding enclosing triangles. *J. Algorithms*, 7:258–269, 1986.

A Implementation and experimental results

Both the simple qudratic algorithm and the linear-time algorithm share a first stage, in which the antipodal pairs of the given convex polygon are computed. Accordingly, our implementation essentially consists of three procedures *antipodal_pairs, simple_algorithm*, and *linear_algorithm*, where the two latter procedures both receive the result of the first procedure as their input.

Table 1 shows our experimental results. We show the running times in seconds for each of the above described procedures on a SPARC-1 workstation. For manual input of the polygon by a designer, problem sizes from 5 to about 40 are reasonable. We ran the algorithms for several values in this range. For further confirmation of the running times, we also examined larger input sizes. These results may become practically relevant if the algorithm is used with automatically generated inputs. However, the running times of the algorithms scale as described by their asymptotic complexity already for the fairly moderate input sizes which are of interest for manual input. Particularly, the running time of the linear algorithm for these problem sizes always remains in a range that can be considered as real-time.

We now give the essential parts of the code implementing the above described procedures, omitting code for type conversion, animation and I/O to avoid clutter. We make use of the data types and algorithms for planar geometry and list processing provided by LEDA [NM90]. In addition to that, we define types for parallelograms and antipodal pairs. We explain their meaning in cases where it is not self-explanatory. It is interesting to see that all the nontrivial work that needs to be done can be reduced to *left_turn* and *right_turn* primitives.

The data types for antipodal pairs and parallelogram are as follows.

```
class evtype {
 segment e;
 point v;

 friend class pargram;
public:
 segment Edge() { return e; }
 point Vertex() { return v; }
 double angle(evtype& p) { return(e.angle(p.e)); }
 double width() const { return(line(e).perpendicular(v).length()); }
};
```

Function *angle* returns the angle by the edges of two antipodal edge-vertex pairs, and the *width* of an antipodal pair (e, v) is the distance of p to its orthogonal projection on the supporting line of e.

```
class pargram {
  point a, b, c;

  point fourth() const {
    point d(a);
    return(d + vector(c.xcoord()-b.xcoord(),c.ycoord()-b.ycoord()));
  }
  double angle() { return(fabs(segment(a,b).angle(segment(b,c)))); }
 public:
  pargram(evtype& z1, evtype& z2) {
    double z1dir = z1.e.direction();
    double z2dir = z2.e.direction();
    line(z1.e).intersection(line(z2.e),a);
    line(z1.e).intersection(line(z2.v,z2dir),b);
    line(z2.v,z2dir).intersection(line(z1.v,z1dir),c);
  }
  double area() {
    return(segment(a,b).length()*segment(b,c).length()*sin(angle()));
  }
};
```

A parallelogram is defined by three vertices. We have functions to compute the fourth point, the angle between the two axes, and the area of the parallelogram, as well as a constructor to canonically form a parallelogram from two antipodal pairs as described in Section 2.

We arrange the vertices of the given convex polygon in clockwise order in array A. This array is used by the procedure *antipodal_pair* as a circular list. The result of the procedure is the array

lap which contains the antipdal edge-vertex pairs of the polygon, in clockwise ordering of the edges in these pairs.

```
void antipodal_pairs(array<evtype>& lap, array<point>& A) {
  int i; int j = A.low()+1;
  for (i = A.low(); i <= A.high(); i++) {
    vector v( - A[i].xcoord() + A[(i+1)%(A.high()+1)].xcoord(),
              - A[i].ycoord() + A[(i+1)%(A.high()+1)].ycoord());
    while (left_turn(A[j],A[(j+1)%(A.high()+1)],A[(j+1)%(A.high()+1)]+v))
      j = (j+1)%(A.high()+1);
    evtype init(A[i],A[(i+1)%(A.high()+1)],A[j]);
    lap[i] = init;
  }
}
```

The procedure *simple_algorithm* computes the corresponding parallelogram for all $\binom{n}{2}$ pairs of antipodal pairs in the array *lap* and keeps track of the one with minimal area.

```
void simple_algorithm(array<evtype>& lap) {
  double opt = 1e25;
  pargram optpar;
  int i, j;
  for (i = lap.low(); i < lap.high(); i++) {
    for (j = i+1; j <= lap.high(); j++) {
      evtype z1 = lap[i];
      evtype z2 = lap[j];
      pargram P(z1,z2);
      double a = P.area();
      if (a < opt) {
         opt = a;
         optpar = P;
      }
    }
  }
}
```

In the linear-time algorithm, we compute the area of far less configurations (i.e. pairs of antipodal pairs). Only for those configurations that are feasible by Lemma 2 we need to compute the corresponding parallelogram.

```
void linear_algorithm(array<evtype>& lap) {
  double opt = 1e25;
  pargram optpar, P;
  int i; int j = lap.low();
  evtype z1;
  evtype z2 = lap[j];
  for (i = lap.low(); i <= lap.high(); i++) {
    z1 = lap[i];
```

```
    if (i == j) {
        j = (j+1)%(lap.high()+1);
        z2 = lap[j];
    }
    vector v( - z1.Edge().start().xcoord() + z1.Edge().end().xcoord(),
              - z1.Edge().start().ycoord() + z1.Edge().end().ycoord());
    while (right_turn(z2.Vertex(),z2.Edge().end(),z2.Edge().end()+v))
      j = (j+1)%(lap.high()+1); z2 = lap[j];
    if (!left_turn(z2.Vertex(),z2.Edge().start(),z2.Edge().start()+v)) {
      pargram P(z1,z2);
      double a = P.area();
      if (a < opt) {
        opt = a;
        optpar = P;
      }
    }
    if (!left_turn(z2.Vertex(),z2.Edge().end(),z2.Edge().end()+v)) {
      j = (j+1)%(lap.high()+1); z2 = lap[j];
      double a = P.area();
      pargram P(z1,z2);
      if (a < opt) {
         opt = a;
         optpar = P;
      }
    }
  }
}
```

Note that the pointers z_1 and z_2 only move forward around the polygon. For a fixed z_1 and a given start position of z_2, the algorithm advances z_2 as long as the projection (w.r.t. the direction of z_1) of the antipodal vertex that belongs to z_2 is *below* the corresponding projection of the edge of z_2. This is implemented by the primitive *right_turn*. Having done that, we only need to compute a parallelogram for the configuration if the projection of the vertex of z_2 is not *above* the projection of the edge of z_2. This is implemented by the primitive *left_turn*. The rest of the code takes care of the special case that the projection of z_2's vertex coincides with the projection of the common extreme point of the edge of z_2 and the following edge on the polygon.