98, 1991, pp. 31-35.

[To3] Toussaint, G. T., “New results in computational geometry relevant to pattern recognition in practice,” in *Pattern Recognition in Practice II*, E. S. Gelsema and L. N. Kanal, Editors, North-Holland, 1986, pp.135-146.

[To4] Toussaint, G. T., Editor, *Computational Morphology*, North-Holland, 1988.

[TV] Tarjan, R. E. and Van Wyk, C. J., “An O(n log log n)-time algorithm for triangulating simple polygons,” *SIAM Journal on Computing*, vol. 17, 1988, pp. 143-178.

[WS] Woo, T. C. and Shin, S. Y., “A linear time algorithm for triangulating a point-visible polygon,” *ACM Transactions on Graphics*, vol. 4, January 1985, pp.60-70.

[YTT] Yashiro, H., Takahashi, T. and Takikawa, K., “An $O(n(1+t_o))$ algorithm for a simple polygon triangulation and its evaluation,” IEICE Tech. Rept., PRU-89-41, September 1989.

with simple data structures," *ACM Symposium on Computational Geometry*, Berkeley, California, June 6-8, 1990, vol. 6, pp. 34-43.

[Kn1] Knopp, K., *Theory of Functions,* Part I, translated by F. Bagemihl from the fifth German Edition, Dover, New York, 1945.

[Kn2] Knopp, K., *Funktionentheorie* I., Sammlung Göschen Band 668, Walter de Gruyter, 1970.

[LC] Lee, S. H. and Chwa, K. Y., "A new triangulation linear class of simple polygons," *International Journal of Computer Mathematics,* vol. 22, 1987, pp.135-147.

[Le] Lennes, N. J., "Theorems on the simple finite polygon and polyhedron," *American Journal of Mathematics*, vol. 33, 1911, pp.37-62.

[Le1] Levy, L. S., *Geometry: Modern Mathematics via the Euclidean Plane,* Prindle, Weber & Schmidt, Inc., Boston, Mass., 1970.]

[Ma] Mandelbrot, B. B., *Fractals: Form, Chance, and Dimension*, W. H. Freeman & Co., San Francisco, 1977.

[Me] Meisters, G. H., "Polygons have ears," *American Mathematical Monthly,* vol. 82, June/ July 1975, pp.648-651.

[RS] Rupert, J. and Seidel, R., "On the difficulty of tetrahedralizing 3-dimensional non-convex polyhedra," *ACM Symposium on Computational Geometry*, vol. 5, June 5-7 1989, Saarbrucken, West Germany, pp. 380-392.

[SV] Schoone, A. A. and van Leeuwen, J., "Triangulating a star-shaped polygon," Tech. Report, RUV-CS-80-3, University of Utrecht, April 1980.

[Sh1] Shermer, T., "Computing bushy and thin triangulations," in *Snapshots of Computational and Discrete Geometry*, G. T. Toussaint, Ed., Tech. Rept. SOCS-88.11, June 1988, pp. 119-133.

[Sh2] Shermer, T., "Generating anthropomorphic k-spirals," in *Snapshots of Computational and Discrete Geometry*, G. T. Toussaint, Ed., Tech. Rept. SOCS-88.11, June 1988, pp. 233-244.

[TA] Toussaint, G. T. and Avis, D., "On a convex hull algorithm for polygons and its application to triangulation problems," *Pattern Recognition*, vol. 15, No. 1, 1982, pp.23-29.

[To1] Toussaint, G. T., "A new linear algorithm for triangulating monotone polygons," *Pattern Recognition Letters*, vol. 2, March 1984, pp. 155-158.

[To2] Toussaint, G. T., "Anthropomorphic polygons," *American Mathematical Monthly*, vol.

[CI] Chazelle, B. and Incerpi, J., “Triangulation and shape complexity,” *ACM Transactions on Graphics*, vol. 3, 1984, pp.135-152.

[El] ElGindy, H. A., “A linear algorithm for triangulating weakly externally visible polygons,” Tech. Report MS-CIS-86-75, University of Pennsylvania, September 1985.

[ET1] ElGindy, H. and Toussaint, G. T., “On triangulating palm polygons in linear time,” *Proc. Computer Graphics International’88*, Geneva, May 24-27, 1988, pp. 308-317.

[ET2] ElGindy, H. and Toussaint, G. T., “On geodesic properties of polygons relevant to linear-time triangulation,” *The Visual Computer*, vol. 5, no. 1/2, March 1989, pp. 68-74.

[EAT] ElGindy, H., Avis, D. and Toussaint, G. T., “Applications of a two-dimensional hidden-line algorithm to other geometric problems,” *Computing*, vol. 31, 1983, pp.191-202.

[EET] ElGindy, H., Everett, H. and Toussaint, G. T., “Slicing an ear in linear time,” to appear in *Pattern Recognition Letters.*

[FM] Fournier, A. and Montuno, D. Y., “Triangulating simple polygons and equivalent problems,” *ACM Transactions on Graphics*, vol. 3, April 1984, pp.153-174.

[FP] Feng, H-Y. F. and Pavlidis, T., “Decomposition of polygons into simpler components: feature generation for syntactic pattern recognition,” *IEEE Transactions on Computers*, vol. C-24, June 1975, pp.636-650.

[Fo] Forder, H. G., *The Foundations of Euclidean Geometry*, Cambridge University Press, 1927.

[GJPT] Garey, M. R., Johnson, D. S., Preparata, F. P. and Tarjan, R. E., “Triangulating a simple polygon,” *Information Processing Letters*, vol. 7, 1978, pp.175-179.

[Gr] Graham, R. L., “An efficient algorithm for determining the convex hull of a finite planar set,” *Info. Proc. Lett.* **1** (1972), 132-133.

[HM] Hertel, S. and Mehlhorn, K., “Fast triangulation of simple polygons,” *Proc. FCT, LNCS* 158, 1983, pp.207-215.

[Ho] Ho, W.-C., “Decomposition of a polygon into triangles,” *The Mathematical Gazette*, vol. 59, 1975, pp.132-134.

[Hon] Honsberger, R., *Ingenuity in Mathematics*, Random House, Inc., 1970.

KET] Kong, X., Everett, H. and Toussaint, G. T., “The Graham scan triangulates simple polygons,” *Pattern Recognition Letters*, in press.

[KKT] Kirkpatrick, D. G., Klawe, M. M., & Tarjan, R. E., “O(n log log n) polygon triangulation

Type of Polygon	**Sleeve Searching Algorithm** $O(n(1+t_0))$	**Sinuosity Algorithm** $O(n \log s)$
Convex	$O(n)$	$O(n)$
Anthropomorphic & Two-Ear polygons	$O(n)$	$O(n \log n)$
Edge-visible	$O(n^2)$ †	$O(n \log n)$

Table 1: A comparison of the sleeve-searching algorithm with the sinuosity algorithm of Chazelle & Incerpi in terms of worst-case complexity for some classes of polygons.
† Edge-visible polygons can be triangulated in O(n) time with the algorithm in [TA].

In this appendix we describe the prune and search algorithm for finding an ear in linear time. The recursive function FindAnEar takes as input a good subpolygon and a vertex. Initially we call FindAnEar with the simple polygon P and any vertex of P.

ALGORITHM *FindAnEar*(GSP,x_i). Given a good subpolygon GSP of a polygon P and a vertex x_i of GSP this algorithm reports a proper ear.

```
1. If $x_i$ is an ear report it and exit.
2. Find a vertex $x_j$ such that $(x_i,x_j)$ is a diagonal of GSP. Let GSP′ be the good subpolygon
   of GSP formed by $(x_i,x_j)$. Relabel the vertices of GSP′ so that $x_i=x_0$ and $x_j=x_{k-1}$ (or $x_j$
   $= x_0$ and $x_i = x_{k-1}$ as appropriate) where k is the number of vertices of GSP′.
3. FindAnEar(GSP′, $\lfloor k/2 \rfloor$)).
```

END *FindAnEar*

8. References

[Ca] Cairns, S. S., “An elementary proof of the Jordan-Schoenflies theorem, *Proc. Amer. Math. Soc.,* vol. 2, 1951, pp.860-867.

[Ch1] Chazelle, B., “A theorem on polygon cutting with applications,” *Proc. 23rd IEEE Symposium on Foundations of Computer Science, Chicago,* November 1982, pp. 339-349.

[Ch1] Chazelle, B., “Triangulating a simple polygon in linear time,” Technical Report CS-TR-264-90, Dept. of Computer Science, Princeton University, May 1990.

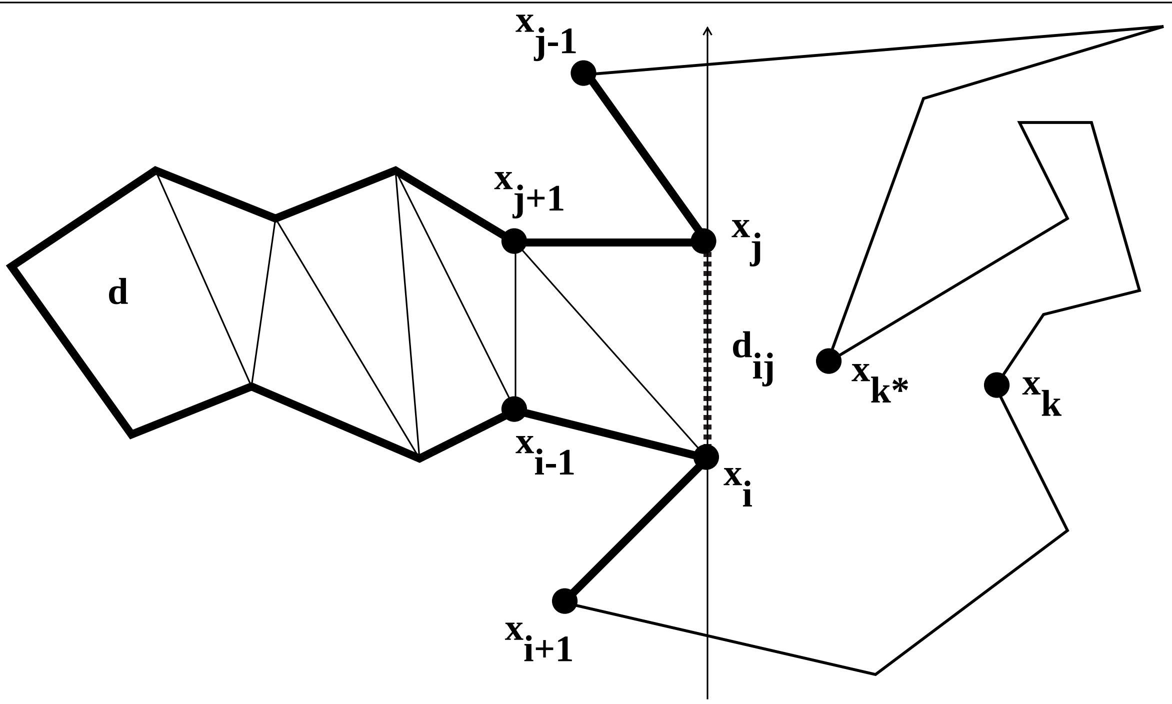


Fig. 12: Case 2.4.2: The vertices x_{i+1} and x_{j-1} do not re-enter the triangulated sleeve and neither does the chain CH[x_{i+1},..., x_{j-1}].

its dependence on the triangulation by taking the extreme values of t_0 over all triangulations of the given polygon. Accordingly we define two new measures of the shape of a polygon: T_{max} = max{t_0} and T_{min} = min{t_0}, where maximization and minimization is carried out over all triangulations. Thomas Shermer [Sh1] has shown that for a given polygon P, T_{max} and a triangulation exhibiting T_{max} can be computed in O(T(n)) time where T(n) is the time taken to obtain any triangulation of P. He has also shown that T_{min} can be computed in O(n^3) time and O(n^2) space. It would be interesting to determine how useful both measures are in pattern recognition applications and other areas of computational morphology [To4] as well as to determine if a sub-cubic time or sub-quadratic space algorithm exists for computing T_{min}.

6. Acknowledgments

The author would like to thank Hossam ElGindy, Tom Shermer, and Rafe Wenger for stimulating and helpful discussions on this topic as well as Tokiichiro Takahashi of NTT, Japan for pointing out some ambiguities in an earlier draft of this paper as well as his comments on the implementation of the algorithm.

7. Appendix

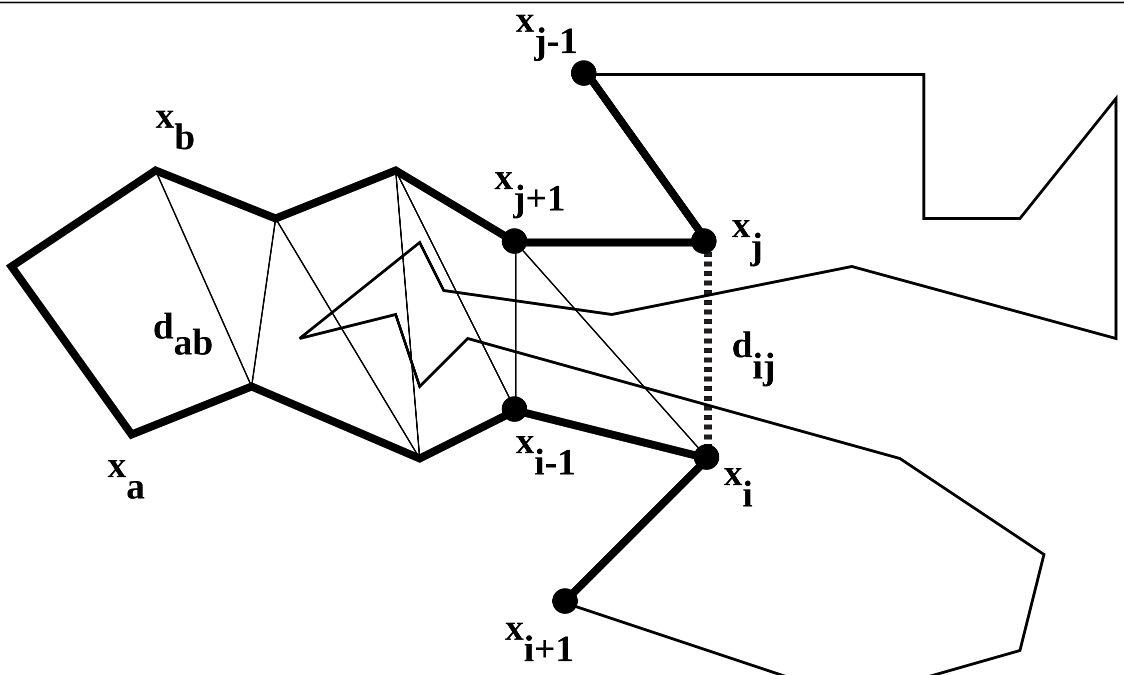


Fig. 11: Case 2.4.1: Neither x_{i+1} nor x_{j-1} re-enter the triangulated sleeve TS[$\mathbf{d}_{ab}$, $\mathbf{d}_{ij}$] but the chain CH[x_{i+1},..., x_{j-1}] does.

which approach is more viable. To assist the programmer interested in this prune and search ear-finding procedure we list it in the Appendix. For a proof of correctness and a complexity analysis the reader is referred to [EET].

On the theoretical side it is interesting to compare the new "sleeve-searching" algorithm proposed here with the only other known adaptive algorithm that is sensitive to the input shape, i.e., the "sinuosity" algorithm of Chazelle & Incerpi [CI]. In particular it is of interest to compare these two algorithms from the point of view of worst-case complexity for certain known classes of simple polygons. Table 1 summarizes the worst-case complexities of both algorithms for *convex*, *anthropomorphic* and *edge-visible* polygons. Note that although t_0 can be as large as O(n) for convex polygons as illustrated in Fig. 4 (a) the algorithm described here will always obtain a value of t_0=0, and hence the complexity on convex polygons is O(n) thus matching the complexity of the sinuosity algorithm. For anthropomorphic polygons "sleeve-searching" is better than "sinuosity" because t_0=0 whereas s=O(n). For edge-visible polygons on the other hand the latter is better although a linear-time algorithm exists [TA].

Measures of the shape of a polygon are of great interest in pattern recognition and computational morphology [To4]. While t_0 may vary too much as a function of the triangulation of a polygon rather than the polygon's shape in order to be a faithful measure of shape, we can remove

$O(n(1+t_0))$ time.

Proof: Since the correctness and linear time complexity of Step 1 of *ALGORITHM* TRIANGULATION follows from *PROCEDURE* DIAGONAL and Lemma 1 we need only concern ourselves with *PROCEDURE* TRIANGULATION. Steps 1, 2 and 1.1 can be done in constant time. The correctness of Step 1.2 (a) follows from Lemma 4 and can be implemented by scanning simultaneously the chain $CH[x_{i+1},...,x_{j-1}]$ and the triangulated sleeve $TS[\mathbf{d}_{ab},\mathbf{d}_{ij}]$. Since the diagonals in $TS[\mathbf{d}_{ab},\mathbf{d}_{ij}]$ are ordered, so are the triangles penetrated by $CH[x_{i+1},..., x_{j-1}]$. Thus if we keep pointers between adjacent triangles as $TS[\mathbf{d}_{ab},\mathbf{d}_{ij}]$ is formed we can easily search for the deepest triangle whenever we have to by advancing either an edge along $CH[x_{i+1},..., x_{j-1}]$ if the previous edge does not intersect the following diagonal in $TS[\mathbf{d}_{ab}, \mathbf{d}_{ij}]$ or advancing a diagonal in $TS[\mathbf{d}_{ab},\mathbf{d}_{ij}]$ as long as it is intersected by the same edge of $CH[x_{i+1},...,x_{j-1}]$. The complexity of this procedure is proportional to the sum of the cardinalities of $CH[x_{i+1},...,x_{j-1}]$ and $TS[\mathbf{d}_{ab},\mathbf{d}_{ij}]$ and in the worst case is $O(n)$.

Consider Step 1.2 (c) and refer to the discussion preceding Lemma 5. We can assume that $x_{j-1} \notin \mathit{int}\,\Delta\, x_i x_j x_{i-1}$ whenever this step is executed because initially $\mathbf{d}_{ij}$ is a diagonal of T(P) and subsequently if this condition were not true it would imply that at the previous diagonal insertion of $\mathbf{d}_{i-1,j}$, $x_{j-1} \in \mathit{int}\,\Delta\, x_j x_{i-1} x_i$ thus contradicting the choice of diagonal $[x_{i-1},x_{j-1}]$ that Step 1.6 would have made. It follows that Lemma 5 implies that Step 1.2 (c) is correct and can be implemented in $O(n)$ time.

The correctness of Steps 1.3 - 1.8 follow straightforwardly from Lemmas 6 and 7 and can each be performed in O(1) time. The arguments for the correctness and complexity of Steps 1.9 and 1.10 are similar to those for Steps 1.1 and 1.2. Finally, Lemmas 2 and 3 ensure the correctness and linear time complexity of Step 1.11.

Every time a diagonal is selected from ***D*** in the outer **While** loop it is possible to do a linear amount of work, in the worst case, only in Steps 1.2, 1.10, and 1.11. All other steps require no more than a O(1) time per step. Furthermore, every time a linear amount of work is done in Steps 1.2, 1.10, and 1.11 a *free* triangle is inserted in T(P) and two diagonals are inserted in ***D***. Since the only time a pair of diagonals is inserted in ***D*** is precisely when a free triangle is inserted in T(P) the outer **While** loop is executed only t_0 times, where t_0 denotes the number of *free* triangles in the triangulation delivered by the algorithm. It follows that the overall complexity of the algorithm is $O(n) + O(nt_0)$. Q.E.D.

5. Concluding Remarks

Concerning the practical issues we remark that the algorithm proposed here has already been implemented by researchers at NTT in Japan for graphics applications and the reader is referred to [YTT] for experimental results. We also remark that a more elegant and perhaps faster implementation of the algorithm proposed here is possible with the recent result of ElGindy, Everett & Toussaint [EET] mentioned in the introduction. In the algorithm presented in the present paper a diagonal is first found which decomposes P into two subpolygons. *PROCEDURE* TRIANGULATION is subsequently called twice and applied to each of the two subpolygons. In [EET] it is shown that an *ear* of P can be found in linear time by a quick prune and search procedure. Thus using the ear-finding procedure of [EET] in Step 1 of *ALGORITHM* TRIANGULATION affords only one call to *PROCEDURE* TRIANGULATION. It remains to be determined experimentally

```
    1.4  If $x_{j-1},x_j,x_i$ is a left turn and $x_{i+1},x_i,x_j$ is a left turn (Case 2.2 in Fig. 10) then apend $[x_{j-1},x_i]$ into TS[$\mathbf{d}_{ab},\mathbf{d}_{ij}$] to create TS[$\mathbf{d}_{ab},\mathbf{d}_{i,j-1}$]; $j \leftarrow j-1$ and go to 1.1.

    1.5  If $x_{j-1},x_j,x_i$ is a left turn and $x_{i+1},x_i,x_j$ is a right turn (Case 2.3 in Fig. 10) then:

    1.6        If $x_{j-1} \in$ int $\Delta\, x_{i+1}x_ix_j$ (Case 2.3.1 in Fig. 10) then apend $[x_{j-1},x_i]$ into TS[$\mathbf{d}_{ab},\mathbf{d}_{ij}$] to create TS[$\mathbf{d}_{ab},\mathbf{d}_{i,j-1}$]; $j \leftarrow j-1$ and go to 1.1.
    1.7        If $x_{i+1} \in$ int $\Delta\, x_ix_jx_{j-1}$ (Case 2.3.2 in Fig. 10) then append $[x_j,x_{i+1}]$ into TS[$\mathbf{d}_{ab},\mathbf{d}_{ij}$] to create TS[$\mathbf{d}_{ab},\mathbf{d}_{i+1,j}$]; $i \leftarrow i+1$ and go to 1.1.

    1.8        If $x_{i+1} \notin$ int $\Delta\, x_ix_jx_{j-1}$ and $x_{j-1} \notin$ int $\Delta\, x_{i+1}x_ix_j$ (Case 2.3.3 in Fig. 10) then arbitrarily select for appending either (a) $[x_j,x_{i+1}]$ or (b) $[x_{j-1},x_i]$ into TS[$\mathbf{d}_{ab},\mathbf{d}_{ij}$]. If (a) is chosen then $i \leftarrow i+1$; If (b) is chosen then $j \leftarrow j-1$. Go to 1.1.

    1.9  If $x_{j-1},x_j,x_i$ is a right turn and $x_{i+1},x_i,x_j$ is a left turn (Case 2.4 in Fig. 10) then:

    1.10       If CH[$x_{i+1},...,x_{j-1}$] intersects $\mathbf{d}_{ij}$ (Case 2.4.1 in Fig. 10) then (a) determine the deepest triangle $\Delta\, x_rx_sx_t$ in TS[$\mathbf{d}_{ab},\mathbf{d}_{ij}$] (see Fig. 11) that contains vertices of CH[$x_{i+1},...,x_{j-1}$]; (b) determine which vertex $x_k$ of CH[$x_{i+1},...,x_{j-1}$] is visible to both $x_r$ and $x_s$; (c) insert diagonals $\mathbf{d}_{sk}=[x_s,x_k]$ and $\mathbf{d}_{kr}=[x_k,x_r]$ into T(P) and $\boldsymbol{D}$; (d) delete all the diagonals in TS[$\mathbf{d}_{ab},\mathbf{d}_{ij}$] starting from and including $\mathbf{d}_{ij}$ and ending with and including $\mathbf{d}_{tr}$, insert all the diagonals remaining in TS[$\mathbf{d}_{ab},\mathbf{d}_{ij}$] into T(P) and Exit.

    1.11       Else (Case 2.4.2 in Fig. 10) (a) find a vertex $x_k$ of CH[$x_{i+1},...,x_{j-1}$] that is visible from $x_i$ and that lies to the right of the directed line through $x_i$ and $x_j$ (see Fig. 12). (b) If $x_k$ is also visible from $x_j$ then insert diagonals $\mathbf{d}_{ik}=[x_i,x_k]$ and $\mathbf{d}_{kj}=[x_k,x_j]$ into T(P) and $\boldsymbol{D}$ and Exit. Else (c) find the vertex $x_{k*} \in$ CH[$x_k,..., x_j$] that lies in the interior of $\Delta\, x_ix_kx_j$ and that minimizes the angle $\angle\, x_{k*}x_jx_i$; insert diagonals $\mathbf{d}_{ik*}=[x_i,x_{k*}]$ and $\mathbf{d}_{k*j}=[x_{k*},x_j]$ into T(P) and $\boldsymbol{D}$.

  End While

End While

End
```

Theorem: *ALGORITHM* TRIANGULATION triangulates a simple polygon P of n vertices in

The Case Structure in the Triangulation-Phase

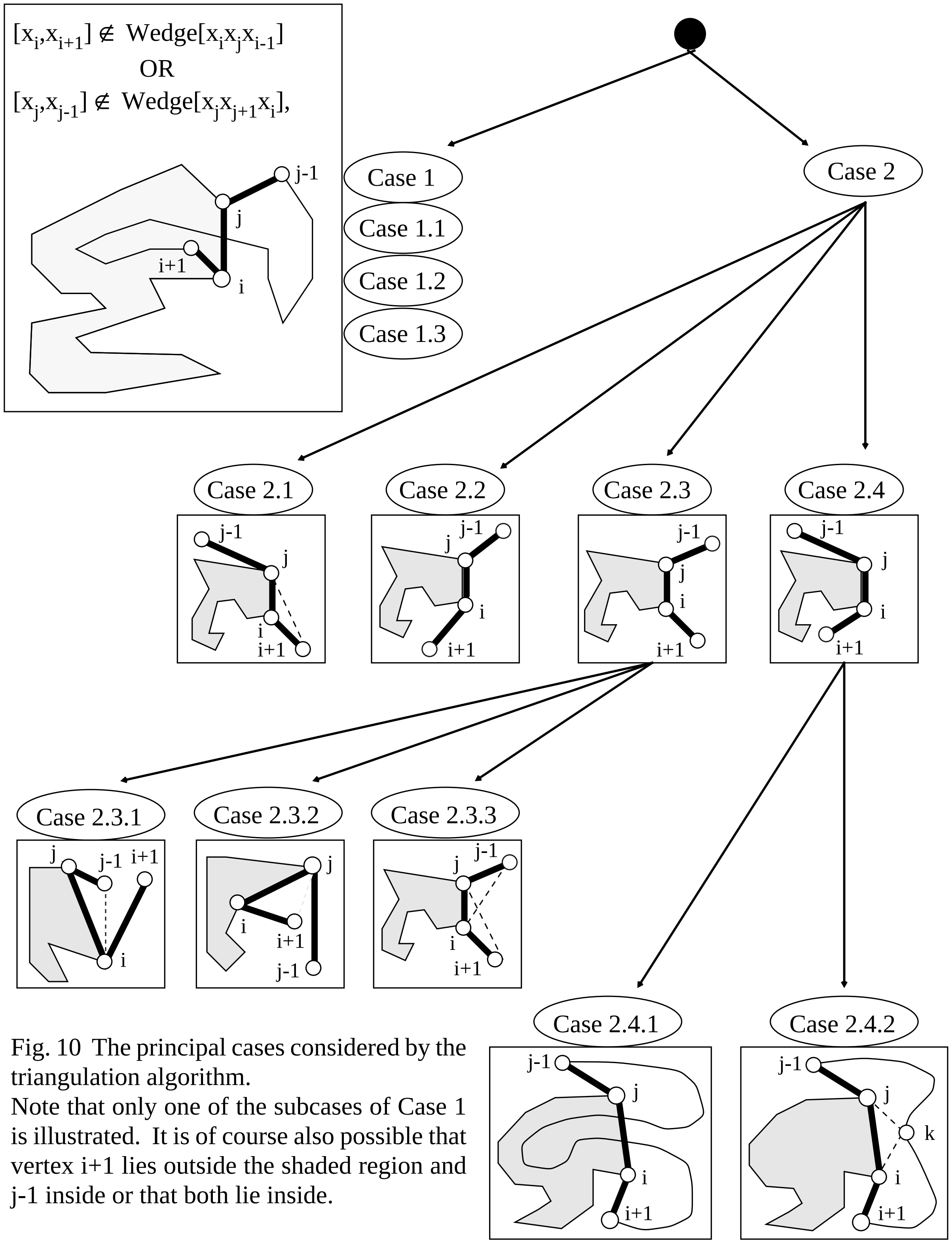


Fig. 10 The principal cases considered by the triangulation algorithm.
Note that only one of the subcases of Case 1 is illustrated. It is of course also possible that vertex i+1 lies outside the shaded region and j-1 inside or that both lie inside.

```
    Step 3:   Triangulate CH[$x_j$,...,$x_i$] using PROCEDURE TRIANGULATION.

End
```

PROCEDURE TRIANGULATION
===

Input: A simple polygon P of n sides oriented in a counterclockwise direction along with a diagonal $\mathbf{d}_{ab}$=[x_a,x_b] inserted in P.

Output: Polygon P with CH[x_a,...,x_b] triangulated.

Comment: {***D*** is a list of diagonals on which the algorithm iterates and is initially empty. T(P) initially consists of P and at the end of execution also contains a triangulation of CH[x_a,...,x_b]}

```
Begin

Initialization Step: D $\leftarrow \mathbf{d}_{ab}$

While D is not empty Do:

1.   Pick a diagonal $\mathbf{d}_{ab}$=[$x_a$,$x_b$] from D and delete it from the list.
2.   i $\leftarrow$ a;  j $\leftarrow$ b.

     While i$\neq$j Do {triangulating a sleeve rooted at $\mathbf{d}_{ab}$}

        1.1   Test whether [$x_i$,$x_{i+1}$] $\in$ Wedge[$x_i x_j x_{i-1}$] or [$x_j$,$x_{j-1}$] $\in$ Wedge[$x_j x_{j+1} x_i$].

        1.2   If either test is violated (Case 1 in Fig. 10) then (a) determine the deepest triangle $\Delta$
              $x_r x_s x_t$ in TS[$\mathbf{d}_{ab}$,$\mathbf{d}_{ij}$] (see Fig. 9) that contains vertices of CH[$x_{i+1}$,...,$x_{j-1}$]; (b) deter-
              mine which vertex $x_k$ of CH[$x_{i+1}$,...,$x_{j-1}$] is visible to both $x_r$ and $x_s$; (c) insert diago-
              nals $\mathbf{d}_{sk}$=[$x_s$,$x_k$] and $\mathbf{d}_{kr}$=[$x_k$,$x_r$] into T(P) and D; (d) delete all the diagonals in
              TS[$\mathbf{d}_{ab}$,$\mathbf{d}_{ij}$] starting from and including $\mathbf{d}_{ij}$ and ending with and including $\mathbf{d}_{tr}$, insert all
              the diagonals remaining in TS[$\mathbf{d}_{ab}$,$\mathbf{d}_{ij}$] into T(P) and Exit.

        1.3.  Else (Case 2 in Fig. 10) If $x_{j-1}$,$x_j$,$x_i$ is a right turn and $x_{i+1}$,$x_i$,$x_j$ is right turn (Case 2.1
              in Fig. 10) then append [$x_{i+1}$,$x_j$] into TS[$\mathbf{d}_{ab}$,$\mathbf{d}_{ij}$] to create TS[$\mathbf{d}_{ab}$,$\mathbf{d}_{i+1,j}$]; i $\leftarrow$ i+1 and
              go to 1.1.
```

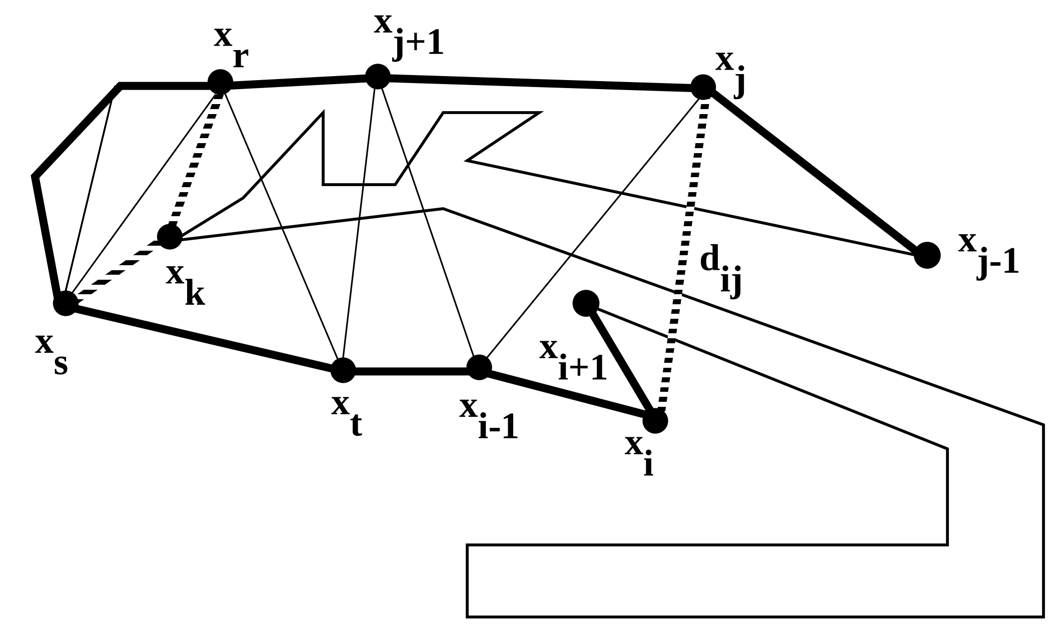


Fig. 9 Illustrating one of the situations in which a portion of the triangulation already completed must be undone. A case when either condition (A) or (B) of Lemma 4 is not satisfied.

int $\Delta\, x_j x_i x_{j-1}$ and $x_{j-1} \notin$ *int* $\Delta\, x_j x_i x_{i+1}$ then either x_{i+1} or x_{j-1} can be chosen to form the triangle with $\mathbf{d}_{ij}$. Clearly we have only a constant number of operations in the case analysis and therefore a diagonal can be found in O(1) time. Q.E.D.

We will now describe the triangulation algorithm in simple outline form. A decision tree illustrating the case analysis considered by the triangulation phase of the algorithm is shown in Fig. 10. In the following TS[$\mathbf{d}_{ab}$,$\mathbf{d}_{cd}$], where a<b<c<d, denotes the triangulated sleeve starting at $\mathbf{d}_{ab}=[x_a,x_b]$ and ending at $\mathbf{d}_{cd}=[x_c,x_d]$.}

ALGORITHM TRIANGULATION

Input: A simple polygon P of n sides oriented in a counterclockwise direction.
Output: Polygon P with n-3 *diagonal*s inserted in P, i.e., T(P).

Begin

Step 1: Find a diagonal $[x_i,x_j]$ of P using *PROCEDURE* DIAGONAL.

Step 2: Triangulate CH$[x_i,...,x_j]$ using *PROCEDURE* TRIANGULATION.

jacent to either of them in P, and can be identified in O(n) time.

Proof: Let the ray emanating from x_r in the direction of x_k intersect $[x_s, x_t]$ at z_k. Then, by the definition of x_k, it follows that $\Delta\, x_r x_s z_k$ is empty. Therefore x_k is visible from both x_r and x_s. Since x_k must be different from x_t by definition, and different from x_{j-1} by assumption, it follows that x_k cannot be adjacent to either x_r or x_s in P. Using point inclusion tests x_k can be easily found in O(n) time. Q.E.D.

We should note here that the selection of x_r rather than x_s as the apex from which to measure the angles $\angle\, x_k x_r x_s$ for selecting x_k is crucial. The reader can easily construct an example similar to that of Fig. 5 (c) to show that with x_s as an apex the resulting extremal vertex selected is not necessarily visible from either x_r or x_s.

In the following we will make extensive use of tests between ordered triplets of vertices x_i,x_j,x_k and will refer to them as either *left turns* or *right turns*. Given an ordered triplet x_i,x_j,x_k we say that it is a *left turn* if x_k lies to the left of the directed line through x_i and x_j in that order. Similarly, we call x_i,x_j,x_k a *right turn* if x_k lies to the right of the directed line through x_i and x_j.

Lemma 6: Let $[x_i,x_{i+1}] \in$ Wedge$[x_i x_j x_{i-1}]$ and let $[x_j,x_{j-1}] \in$ Wedge$[x_j x_{j+1} x_i]$ and define the following two conditions:

(a) $x_{j-1} \in$ *int* $\Delta\, x_j x_i x_{i+1}$,

(b) $x_{i+1} \in$ *int* $\Delta\, x_j x_i x_{j-1}$.

If x_{j-1},x_j,x_i is a *left turn* and x_{i+1},x_i,x_j is *right turn* then it is impossible for conditions (a) and (b) to be satisfied simultaneously.

Proof: Assume $x_{j-1} \in$ *int* $\Delta\, x_j x_i x_{i+1}$. By convexity $\Delta\, x_j x_i x_{i+1}$ must contain $[x_i x_{j-1}]$. It follows that *int* $\Delta\, x_j x_i x_{j-1} \in$ *int* $\Delta\, x_j x_i x_{i+1}$. Therefore $x_{i+1} \notin$ *int* $\Delta\, x_j x_i x_{j-1}$. A similar argument applies in the case we assume $x_{i+1} \in$ *int* $\Delta\, x_j x_i x_{j-1}$. Q.E.D.

Lemma 7: Let $[x_i,x_{i+1}] \in$ Wedge$[x_i x_j x_{i-1}]$ and let $[x_j,x_{j-1}] \in$ Wedge$[x_j x_{j+1} x_i]$. If x_{j-1},x_j,x_i is a *left turn* or x_{i+1},x_i,x_j is a *right turn* then at least one of the vertices from the set $\{x_{i+1},x_{j-1}\}$ forms a triangle with $\mathbf{d}_{ij}$ as base in the triangulation of the sleeve considered so far. Furthermore, such a vertex can be identified in O(1) time.

Proof: Case 1: If x_{j-1},x_j,x_i is a *left turn* and x_{i+1},x_i,x_j is a *left turn* then it follows that $x_{i+1} \notin$ *int* $\Delta\, x_j x_i x_{j-1}$. Therefore x_{j-1} forms a valid triangle with $\mathbf{d}_{ij}$. *Case 2:* If x_{j-1},x_j,x_i is a *right turn* and x_{i+1},x_i,x_j is a *right turn* then it follows that $x_{j-1} \notin$ *int* $\Delta\, x_j x_i x_{i+1}$. Therefore x_{i+1} forms a valid triangle with $\mathbf{d}_{ij}$. *Case 3:* If x_{j-1},x_j,x_i is a *left turn* and x_{i+1},x_i,x_j is a *right turn* then by Lemma 6 we have only three subcases to consider. *Sub-Case 3.1:* If $x_{i+1} \in$ *int* $\Delta\, x_j x_i x_{j-1}$ then x_{i+1} is the desired vertex. *Sub-Case 3.2:* If $x_{j-1} \in$ *int* $\Delta\, x_j x_i x_{i+1}$ then x_{j-1} is the desired vertex. *Sub-Case 3.3:* If $x_{i+1} \notin$

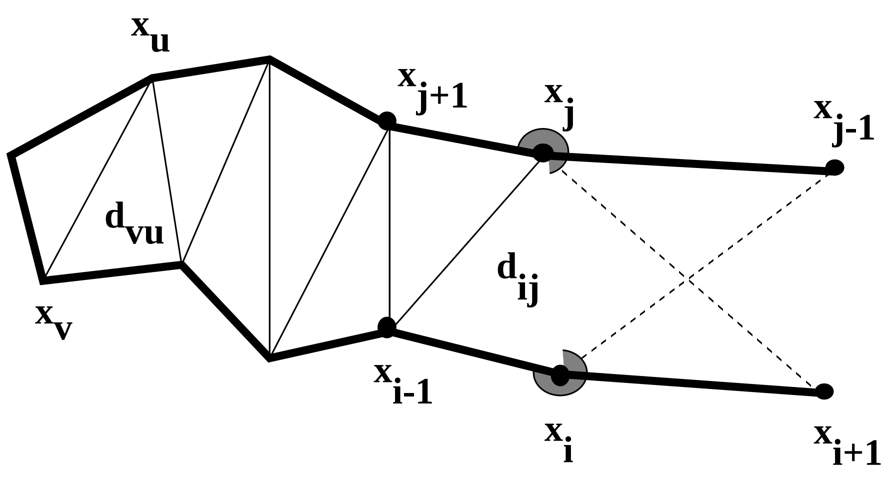


Fig. 8 The general strategy is to insert diagonals in constant time per diagonal as long as the polygon appears to be a *sleeve*.

will ensure the desired outcome. Assume for the moment that the chain $CH[x_j,...,x_i]$ of P has been triangulated as a sleeve (denoted by $T(CH_{ij})$) using O(1) time per diagonal insertion and that it is discovered at the next step that either condition A ($[x_i,x_{i+1}] \in$ Wedge$[x_ix_jx_{i-1}]$) or condition B ($[x_j,x_{j-1}] \in$ Wedge$[x_jx_{j+1}x_i]$) is violated and refer to Fig. 9. This implies that $CH[x_i,...,x_j]$ either intersects $\mathbf{d}_{ij}$ and possibly at least one other diagonal of $T(CH_{ij})$, does not intersect $\mathbf{d}_{ij}$ but possibly at least one other diagonal of $T(CH_{ij})$ or it does not intersect any diagonal of $T(CH_{ij})$ in which case it lies completely in the triangle of $T(CH_{ij})$ determined by $\mathbf{d}_{ij}$. In either of the three cases it follows that $\mathbf{d}_{ij}$ cannot be a diagonal of T(P) and some diagonals (at least $\mathbf{d}_{ij}$) must be removed from $T(CH_{ij})$. Since $T(CH_{ij})$ is a sleeve the diagonals, as well as the triangles they flank, are ordered. Furthermore, since the dual of $T(CH_{ij})$ is a chain it follows that all but the last triangle in $T(CH_{ij})$ must have precisely one of its edges as an edge of P. Let $\Delta\, x_rx_sx_t$ denote the "deepest" triangle (furthest from $\mathbf{d}_{ij}$ in $T(CH_{ij})$) the interior of which is intersected by $CH[x_i,...,x_j]$. Without loss of generality assume that $[x_s,x_t]$ is the edge of $\Delta\, x_rx_sx_t$ that is also an edge of P and that $[x_{i-1},x_i]$ is the edge of $\Delta\, x_{i-1}x_ix_j$ that is also an edge of P. Therefore $CH[x_i,...,x_j]$ properly intersects $[x_r,x_t]$ but does not intersect $[x_r,x_s]$. Furthermore, assume that $x_{j-1} \notin \mathit{int}\,\Delta\, x_ix_jx_{i-1}$. We now show that among the vertices of $CH[x_i,...,x_j]$ that lie in the interior of $\Delta\, x_rx_sx_t$ there is at least one vertex x_k that is visible from both x_r and x_s and is not adjacent to either of them in P thus affording the insertion of two permanent diagonals $[x_r,x_k]$ and $[x_k,x_s]$ and forming a *free* triangle $\Delta\, x_rx_sx_k$. Furthermore, all this can be done in O(n) time.

Lemma 5: Let V_{rs} denote the set of vertices of P that lie in the interior of $\Delta\, x_rx_sx_t$ and let $x_k \in V_{rs}$ be the vertex that minimizes the angle $\angle\, x_kx_rx_s$. Then x_k is visible from both x_r and x_s, is not ad-

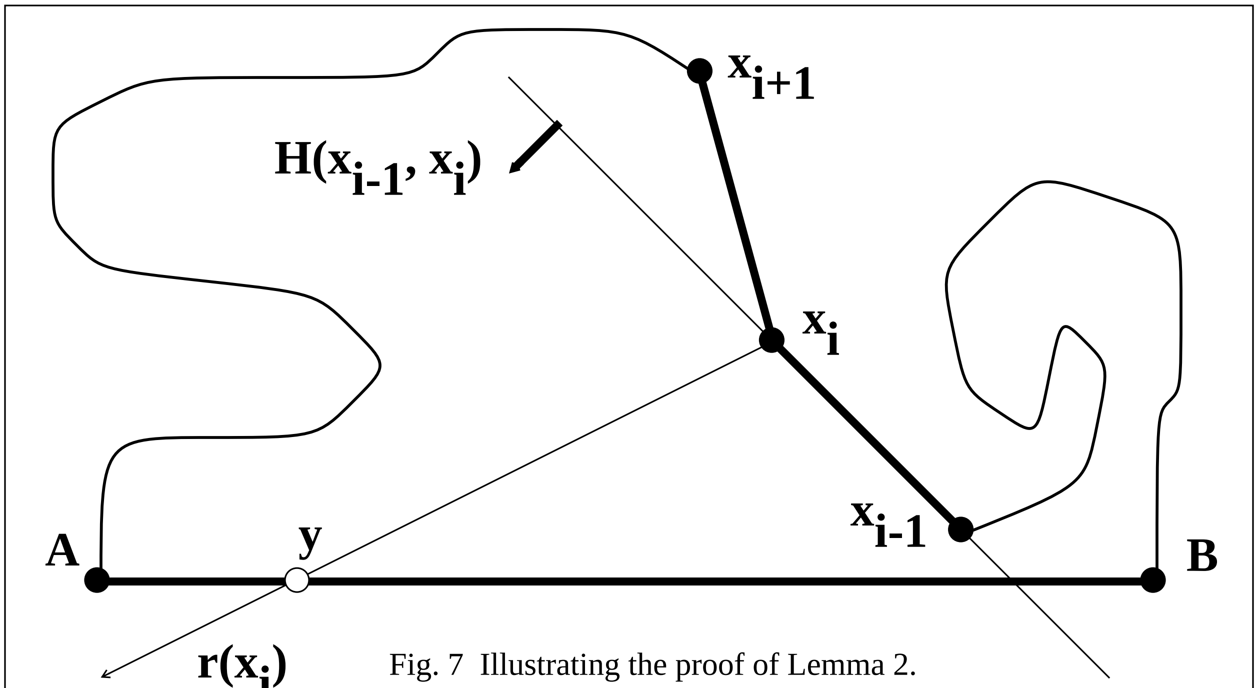


Fig. 7 Illustrating the proof of Lemma 2.

refore the polygon $[x_v,...,x_i,x_j,...,x_u]$ has already been triangulated. Note however that all of this triangulation may not be possible in P and a portion of it may have to be undone at some later stage in the execution of the algorithm. Let Wedge$[x_ix_jx_{i-1}]$ denote the region of the plane swept out by a ray anchored at x_i as it sweeps in a clockwise manner starting from a position colinear with x_j and ending in a position colinear with x_{i-1} and refer to Fig. 9. In order to add a new diagonal we test to determine if $[x_i,x_{i+1}]$ lies in Wedge$[x_ix_jx_{i-1}]$, (we refer to this as condition A), or $[x_j,x_{j-1}]$ lies in Wedge$[x_jx_{j+1}x_i]$, (we refer to this as condition B). We then have the following lemmas.

Lemma 4: Let $\mathbf{d}_{ij}$ be a line segment joining x_i to x_j in P. If either condition A *or* condition B

(A) $[x_i,x_{i+1}] \in$ Wedge$[x_ix_jx_{i-1}]$,

(B) $[x_j,x_{j-1}] \in$ Wedge$[x_jx_{j+1}x_i]$,

does *not* hold then $\mathbf{d}_{ij}$ cannot be a diagonal of any triangulation of P.

Proof: If both conditions (A) and (B) are violated then by the Jordan Curve Theorem it follows that $\mathbf{d}_{ij}$ is an *external* diagonal of P or CH$[x_i,...,x_j]$ intersects $\mathbf{d}_{ij}$ at least twice. If only one of conditions (A) or (B) is exclusively true then CH$[x_i,...,x_j]$ intersects $\mathbf{d}_{ij}$ at least once. In all three cases $\mathbf{d}_{ij}$ is invalidated as a candidate for a diagonal of T(P). Q.E.D.

As we mentioned earlier, at some steps in the execution of the triangulation phase of the algorithm it may be required to undo a portion of the triangulation constructed thus far. Such an action involves the possibility of doing a linear amount of work and therefore, in all such situations, we must be sure that a *free* triangle will be added in T(P). Lemma 5 below together with lemma 3

x_j lies in $H(x_{i-1},x_i)$ and furthermore, such a diagonal can be identified in O(n) time.

Proof: Let x_i be a *concave* vertex of P and refer to Fig. 7 for illustration. As in Lemma 1 we construct a ray $r(x_i)$ that bisects the internal angle at x_i and find the intersection point y of this ray with *bd*(P). By construction x_i and y are visible and $y \in H(x_{i-1},x_i)$. Actually in this case it is also true that $y \in H(x_i,x_{i+1})$. Therefore if y is a vertex of P we are done. Therefore assume y lies in the interior of some edge [A,B] of P and consider the triangle $\Delta\, yBx_i$. We proceed in a manner similar to that described in the proof of Lemma 1 to look for a vertex in $CH[B,...,x_{i-1}]$ other than x_{i-1} that is visible from x_i with the added requirement that such a vertex x_j not only lie in *int*($\Delta\, yBx_i$) but also in $H(x_{i-1},x_i)$. If such a vertex is found we are done. Therefore assume that no such vertex exists for $CH[B,...,x_{i-1}]$ and consider $\Delta\, Ayx_i$. We determine whether a vertex of $CH[x_i,...,A]$ lies in *int*($\Delta\, Ayx_i$). If not then A is visible from x_i. Furthermore, since $L[x_i,x_{i-1}]$ intersects *int*[A,B] it follows that *int*($\Delta\, Ayx_i$) $\in H(x_{i-1},x_i)$. Therefore $A=x_j$ and we have our desired diagonal. On the other hand, if vertices of $CH[x_i,...,A]$ do lie in $\Delta\, Ayx_i$ then we proceed as in the proof of Lemma 1 to find a vertex x_j visible to x_i and exit with this diagonal. It is straightforward to verify that all the steps can be performed in O(n) time. Q.E.D.

Lemma 3: Let x_i and x_{i-1} be two adjacent *concave* vertices in a simple polygon P. Then there exists a vertex x_k, $k \neq i,i-1$ such that $[x_i,x_k,x_{i-1}]$ forms a triangle in a triangulation of P and furthermore, x_k can be identified in O(n) time.

Proof: Lemma 2 implies that x_i admits a diagonal of P, $[x_i,x_k]$ such that $x_k \in H(x_{i-1},x_i)$ and such that x_k can be found in O(n) time. If x_{i-1} is visible from x_k we are done. If not we scan $CH[x_k,...,x_{i-1}]$ to determine which of its vertices lie in $\Delta\, x_kx_{i-1}x_i$ and of these we select the vertex x_{k*} that minimizes the angle $\angle\, x_{k*}x_{i-1}x_i$ which lies in $\angle\, x_kx_{i-1}x_i$. This can be done in O(n) time. Since x_{i-1} is a concave vertex it follows that x_{i-2} cannot lie in $H(x_{i-1},x_i)$ and cannot be a candidate for either x_k or x_{k*}, thus ensuring that $[x_i,x_{k*}]$ is not an edge of P. Therefore in either case we are guaranteed that x_k or x_{k*} are visible from both x_i and x_{i-1} and together they form a *free* triangle in some triangulation of P. Q.E.D.

Any diagonal **d** of P partitions P into two sub-polygons P_1 and P_2. The triangulation algorithm we will describe begins by finding a diagonal **d** using *Procedure* DIAGONAL given in section 3 and subsequently, starting at **d** triangulates one of the sub-polygons in a single pass. A second pass starting at **d** in the opposite direction triangulates the second sub-polygon. We will refer to the step used to find **d** as the *initialization* phase and the subsequent steps as the *triangulation* phase.

Consider for the moment that at some stage in the execution of the triangulation phase of the algorithm, diagonal $\mathbf{d}_{ij}$ has been inserted between vertices x_i and x_j and refer to Fig. 8. Let $CH[x_i,..., x_j]$ denote the portion of the polygonal boundary of $P = [x_1,x_2,...,x_n]$ from x_i to x_j in a counterclockwise orientation. Assume that the diagonal **d** added connects vertices x_u and x_v. The-

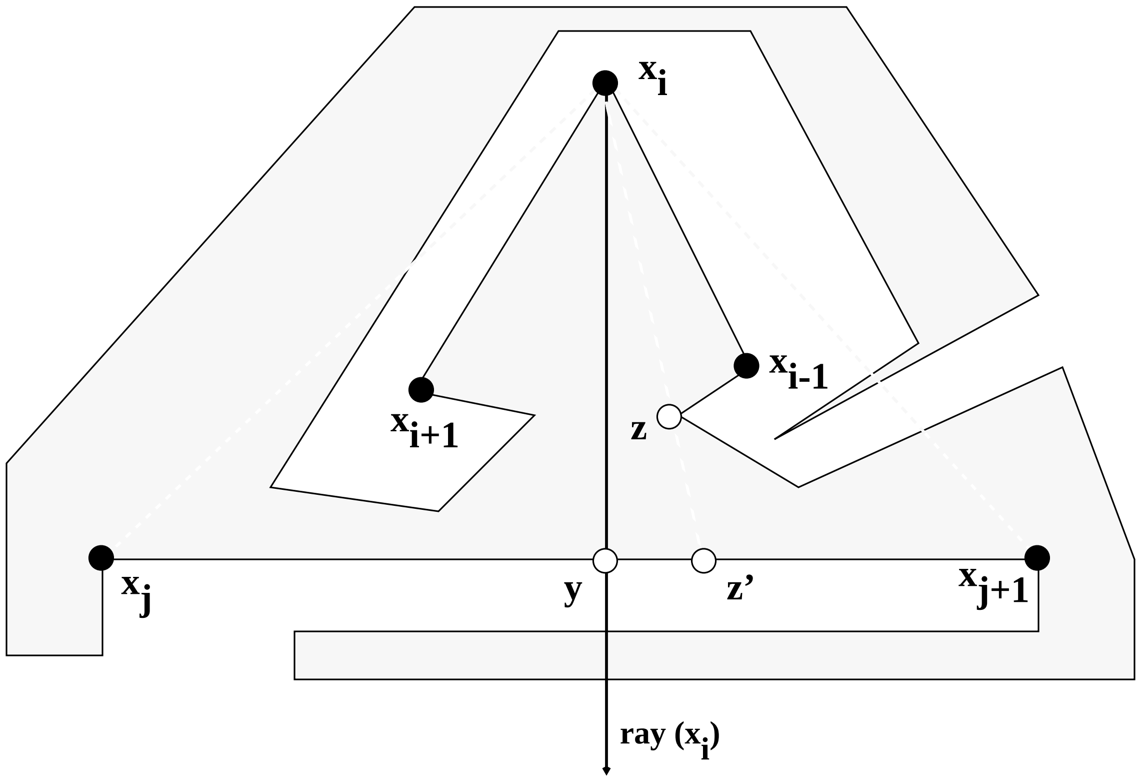


Fig. 6 Illustrating *Procedure* DIAGONAL.

performs a "blind" search for sleeves. Initially the algorithm finds a diagonal **d** and proceeds to attempt to triangulate the polygon in both directions starting from the initial diagonal assuming that the polygon is in fact a sleeve as "viewed" from **d**. If it detects at some point in time that the assumed sleeve it is triangulating is no longer a sleeve the algorithm backtracks undoing the triangulation until the triangulated portion is a *bona-fide* sleeve. At this stage two new diagonals are inserted (identifying a triangle corresponding to a node of degree three in the dual tree of the triangulation completed thus far) and the algorithm begins again recursively from these two new diagonals which themselves will appear in the final triangulation. Before we describe the algorithm in more detail we present a series of lemmas we will need to establish the correctness and complexity of the algorithm.

It is well known and clear from *Step 10* of *Procedure* DIAGONAL that a *convex* vertex of a simple polygon P does not always admit a diagonal in T(P). Much less well known is the result that every *concave* vertex on the other hand does admit a diagonal as the following lemma establishes. In fact we prove a stronger property. In the following we denote by $H(x_i,x_{i+1})$ the open half-plane to the left of the directed line $L[x_i,x_{i+1}]$. Here $L[x_i,x_{i+1}]$ denotes the line colinear with x_i and x_{i+1} and the direction is determined by orienting from x_i to x_{i+1}.

Lemma 2: Let x_i be a *concave* vertex of a polygon P. Then P admits a diagonal $[x_i,x_j]$ such that

```
Step 2:   Construct a ray at $x_i$, ray($x_i$), that bisects the interior of $\angle x_{i-1}x_ix_{i+1}$.

Step 3:   Find the first intersection point of ray($x_i$) with bd(P). Let y be the intersection point
          on edge $[x_j,x_{j+1}]$; If y is a vertex of P, Exit with $[x_i,y]$ as the diagonal.

Step 4:   Construct the triangle $[x_i,y,x_{j+1}]$.

Step 5:   For all $j+1 < k < i$ if $x_k$ lies in triangle $[x_i,y,x_{j+1}]$ label $x_k$ as $x^*_k$.
          If there are no labeled vertices, Exit with $[x_i,x_{j+1}]$ as the diagonal.

Step 6:   For all labeled vertices compute $\angle yx_ix^*_k$ and select that vertex (call it z)
          that minimizes this angle.
          If $z \neq x_{i-1}$, Exit with $[x_i,z]$ as the diagonal.

Step 7:   Construct the triangle $[x_j,y,x_i]$.

Step 8:   For all $j > k > i$ if $x_k$ lies in triangle $[x_j,y,x_i]$ label $x_k$ as $x^*_k$.
          If there are no labeled vertices, Exit with $[x_i,x_j]$ as the diagonal.

Step 9:   For all labeled vertices compute $\angle yx_ix^*_k$ and select that vertex (call it w)
          that minimizes this angle.
          If $w \neq x_{i+1}$, Exit with $[x_i,w]$ as the diagonal.

Step 10:  Exit with $[x_{i-1},x_{i+1}]$ as the diagonal.
```

End

Proof of Lemma 1: Procedure DIAGONAL establishes that a diagonal can always be found. Therefore let us consider its complexity. A convex vertex can always be found in O(n) time since any extreme vertex of P is convex. Thus it suffices to pick the vertex with maximum y coordinate for example. Steps 2,4,7 and 10 are constant time operations. The first intersection point y in step 3 can be found in O(n) time by simply scanning all the edges of P in the order in which they occur and testing each for intersection with *ray*(x_i). The complexity of step 5 is clearly O(n). Its correctness follows from the Jordan Curve Theorem and the fact that $[x_i,y]$ is a chord of P. The complexity of step 6 is clearly also linear and its correctness follows from the fact that by construction $int[x_i,y,z'] \cap bd(P) = \emptyset$, where z' is the intersection of the line colinear with $[x_i,z]$ that intersects $[x_j,x_{j+1}]$. Steps 8 and 9 are similar to 5 and 6. Q.E.D.

4. The New Triangulation Algorithm

A triangulated polygon can be viewed as the "gluing" together of a set of *sleeves*. A *sleeve* is a triangulated polygon whose dual-tree is a chain. The new algorithm proposed here in effect

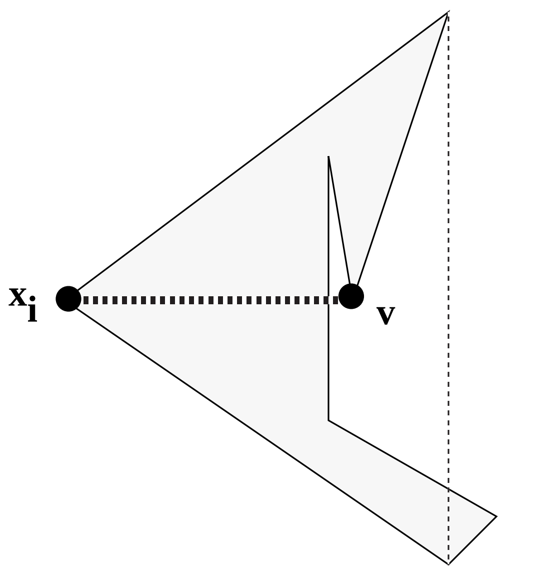


Fig. 5 (b) The closest vertex v to x_i need not form a diagonal of P.

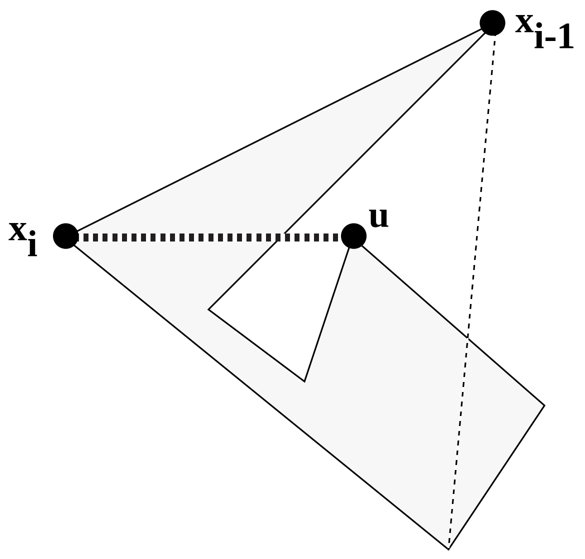


Fig. 5 (c) The vertex u that has smallest $\angle x_{i-1}x_iu$ need not form a diagonal.

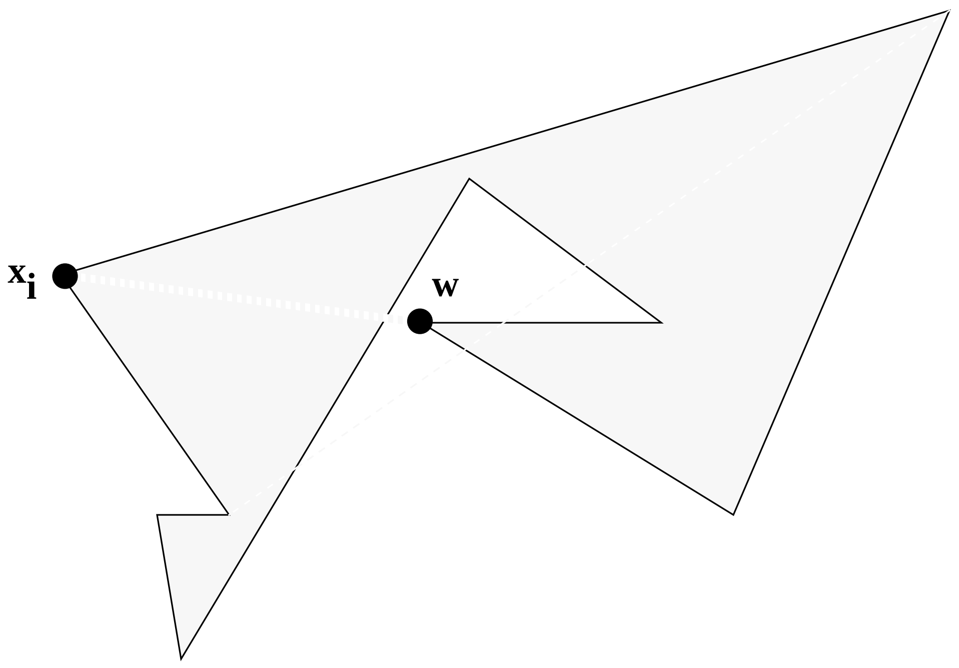


Fig. 5 (d) The vertex w with the second least abscissa after x_i need not form a diagonal of P.

tex in *int*[x_{i-1},x_i,x_{i+1}] whose Euclidean distance to x_i is smallest (vertex v in Fig. 5 (a)) and insert diagonal [x_i,v]. Recursively continue this procedure on each resulting subpolygon until each subpolygon thus created is a triangle. A counter-example to this algorithm is given in Fig. 5 (b).

A proof of lemma 1 appears in Knopp's *Funktionentheorie* [Kn1]. During translation F. Bagemihl discovered that the original proof was incorrect and inserted a proof of his own, also incorrect as it turns out. Bagemihl chose the vertex *p* in *int*[x_{i-1},x_i,x_{i+1}] such that the $\angle x_{i-1} x_i\, p$ is smallest (vertex u in Fig. 5 (a)). A counterexample to this proof is given in Fig. 5 (c). Honsberger gives a similar incorrect proof [Hon]. We note here that the new German edition of *Funktionentheorie* has a new proof [Kn2]. Several other incorrect proofs have also appeared. See for example Forder [Fo] and Cairns [Ca]. Cairns chose the vertex *p* in *int*[x_{i-1},x_i,x_{i+1}] such that *p* has the second least abscissa after x_i (vertex w in Fig. 5 (a)). A counterexample to this situation is given in Fig. 5 (d). For a discussion on several correct and incorrect proofs the reader is referred to the paper by Chung-Wu Ho [Ho]. Our proof given below is a constructive version of Levy's proof [Le1] and will also establish that such a diagonal can be found in O(n) time. We present it in the form of an algorithm called *PROCEDURE* DIAGONAL (refer to Fig. 6).

PROCEDURE DIAGONAL

Input: A simple polygon P oriented in a counterclockwise direction.
Output: Polygon P with a *diagonal* inserted in P.

Begin

Step 1: Find a convex vertex x_i of P.

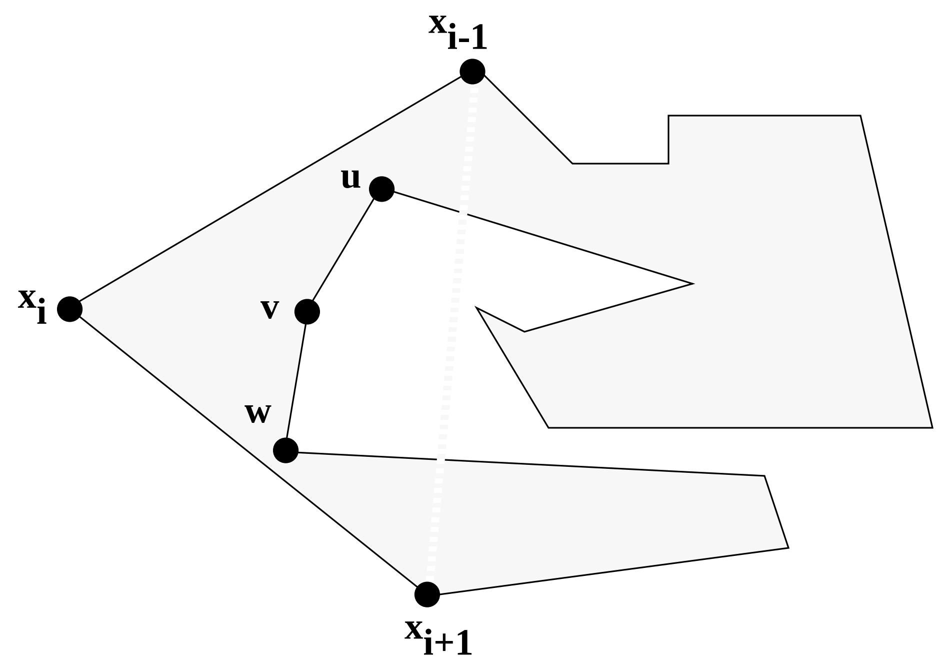
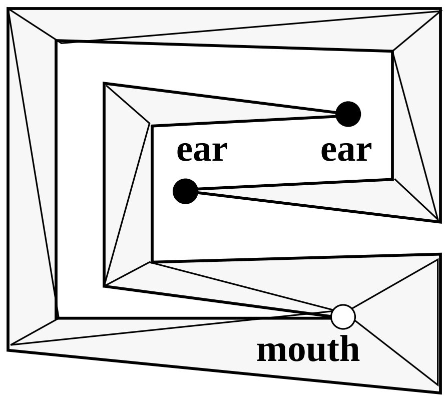
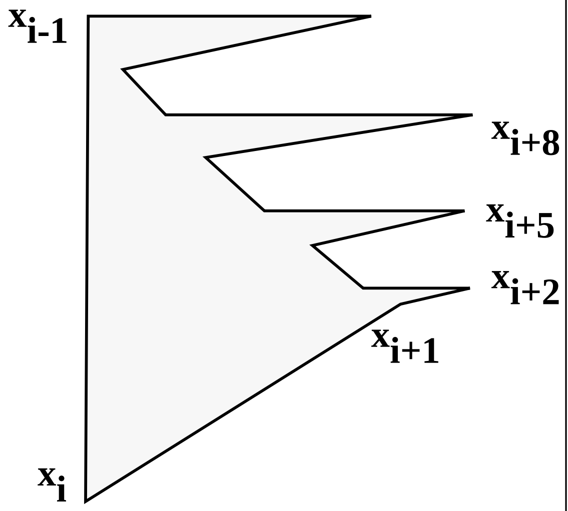


Fig. 5 (a) Illustrating some incorrect "proofs" of Lemma 1.



(c) An *anthropomorphic* polygon admits many triangulations but all of them have $t_0 = 0$.

Fig. 4



(d) A polygon that admits only one triangulation and $t_0 = O(n)$.

Consider now triangulating this polygon. A blue vertex x_k is not visible from any vertex of P non-adjacent to x_k. Therefore all blue vertices are ears of P. Therefore all triangulations of P must contain the blue vertices as ears of T(P). It remains to triangulate P* = $[x_i,x_{i+1},x_{i+3},x_{i+4},x_{i+6},x_{i+7},...,x_{i-4},x_{i-3},x_{i-1}]$. However, all the vertices of this polygon other than x_i form a concave chain and thus the only way to triangulate P* is to join x_i to all the vertices on this chain other than x_{i-1} and x_{i+1}. This results in the only possible triangulation with a value of $t_0 = O(n)$.

3. Finding a Diagonal in a Simple Polygon

A fundamental lemma that is often used to prove by induction that a polygon triangulation always exists (which surprisingly is not true in three dimensions [RS]) and which forms the backbone of most triangulation algorithms concerns the existence of diagonals.

Lemma 1: Every simple n-gon P with $n \geq 4$ can be partitioned into two sub-polygons in O(n) time by inserting some diagonal of P.

Lemma 1 is elementary and quite straightforward to prove but since, surprisingly, so many published proofs of it are incorrect, and since the result is a corner stone in almost all triangulation algorithms we will include a detailed constructive proof of it below. But first we take a small detour through some of the most frequent “trap-doors” encountered.

When students of computational geometry are first given the task of designing an algorithm for triangulating a simple polygon they often propose the following (and in fact published) procedure. (refer to Fig. 5 (a)) Find the vertex of P which has minimum x-coordinate; call it x_i. If *int*$[x_{i-1},x_i,x_{i+1}]$ contains no other vertices of P then insert the diagonal $[x_{i-1},x_{i+1}]$. Otherwise find the ver-

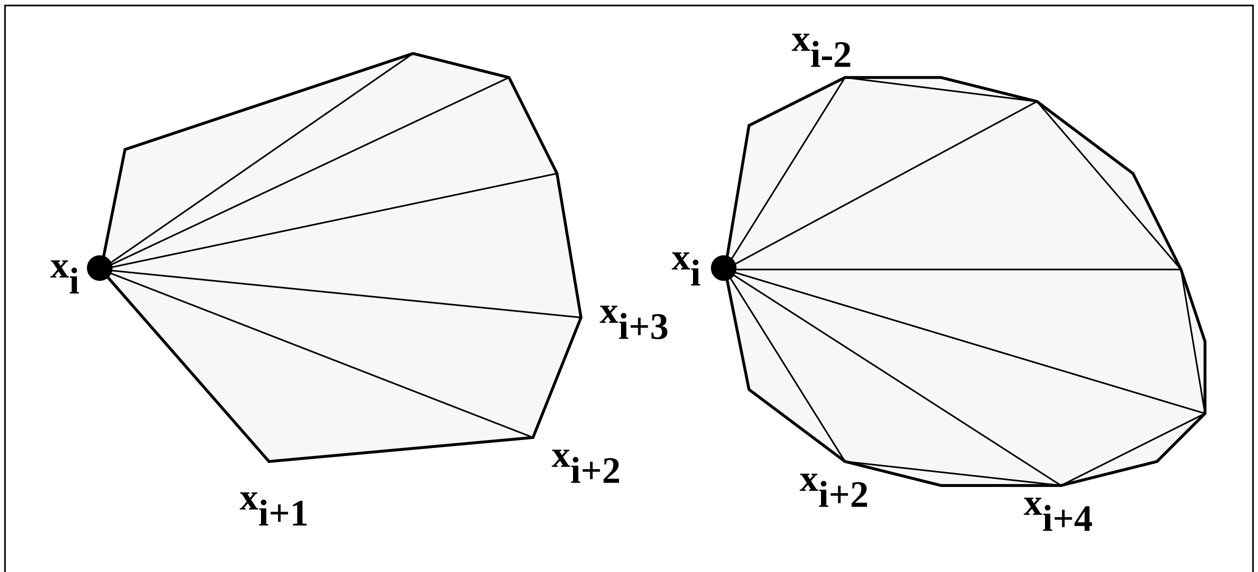


Fig. 4 (a) (b) Two very different triangulations of a convex polygon.

classes since they are relevant to a complexity-based comparison of the algorithm proposed here with other adaptive algorithms.

Consider the *anthropomorphic* polygon illustrated in Fig. 4 (c). A simple polygon P is called *anthropomorphic* provided it contains precisely two ears and one *mouth* [To2]. A *principal* vertex x_i of a simple polygon P is called a *mouth* if the diagonal $[x_{i-1},x_{i+1}]$ is an *external* diagonal, i.e., the interior of $[x_{i-1},x_{i+1}]$ lies in the exterior of P. Surprisingly, although anthropomorphic polygons are quite structured in some ways, they are quite general in other ways. Fig. 4 (c) illustrates an uncomplicated anthropomorphic polygon. Shermer [Sh2] has considered the problem of generating more complex anthropomorphic polygons. The crucial aspect for our purpose here is that for an *anthropomorphic* polygon $t_0 = 0$ for all its triangulations. This is so because the *dual-tree* of every triangulation of a two-ear polygon is a *chain*, i.e., a tree in which all its nodes are of degree either one or two. This suggests that the extreme values of t_0 over all triangulations of a given polygon may be more appropriate measures of the shape of a polygon in certain applications. In this case, for example, since the minimum value of t_0 for any convex polygon is zero, both polygons in Figs. 4 (a) and (b) would have the same shape complexity.

It is also possible that a polygon will admit only a single triangulation and that its value of $t_0 = O(n)$. Such a family is illustrated in Fig. 4 (d). Let the vertices x_{i-1}, x_i, x_{i+1} have coordinates (0,1), (0,-1), and (1,0), respectively and insert edges $[x_{i-1},x_i]$ and $[x_i,x_{i+1}]$. Place two thirds of the remaining n-3 vertices on *C*, the smaller of the two arcs of a unit circle centered at (1,1). Starting with the first of these vertices, connect with an edge each adjacent alternating pair and call these vertices “red.” Finally, place the remaining vertices, called “blue,” sufficiently far from this circular arc *C* such that each of them is connected to a pair of adjacent red vertices on *C* not yet connected to each other such that no blue vertex is visible from x_i. Recall that two points x,y in P are visible from each other if the line segment [x,y] lies in P.

borescence or “branchiness.” Is a tree like a *palm* tree or more like an *oak*? Is it locally very different and smoother than it is globally or is it self-similar in the way *fractals* [Ma] are? There are a variety of methods of quantitatively measuring such fragmentation. The *degree* of a node in a tree is an integer that indicates the number of edges emanating from that node. In a tree that is the dual of a polygon triangulation the nodes are all of degree one, two, or three. For our purposes a good measure of the amount of branching in a tree is its number of nodes of degree three. Let t_i denote the number of triangles in a triangulated polygon T(P) that share i edges with P. It is clear that t_0, which we also refer to as the number of “free” triangles in T(P), corresponds to the number of nodes of degree three in the dual tree of T(P). Thus t_0 is a very natural measure of the complexity of a triangulation. This is not to say that it is necessarily a good measure of the *shape-complexity* of P. Consider for example a convex polygon triangulated in two distinct ways illustrated in Figs. 4 (a) and (b). In Fig. 4 (a) the polygon is triangulated by adding edges from an *anchor* vertex x_i to x_j for j=i+2, i+3,..., i-2. This procedure yields a triangulation with $t_0 = 0$. In Fig. 4 (b) the triangulation is obtained by first connecting the *anchor* vertex to all alternating vertices starting at x_{i+2}, i.e., x_i is joined to x_j for j=i+2, i+4, i+6,..., i-2, and subsequently adding edges between all pairs of vertices (x_k, x_{k+2}) for k=i+2, i+4,..., i-4. This procedure, unlike the previous method, yields a triangulation with $t_0 = O(n)$. From this example it may appear at first glance that the value of t_0 is a property exclusively of the algorithm used to triangulate the polygon and that therefore it plays a somewhat artificial role in measuring the complexity of the algorithm particularly if the algorithm makes no explicit attempt to yield a triangulation that minimizes t_0. However this is in fact not true. Nevertheless the relation between t_0 and the shape of the input polygon is a subtle one. Furthermore, there exist classes of polygons with restricted shapes for which all their triangulations have a value of t_0 either exclusively zero or exclusively O(n). We will now illustrate two such

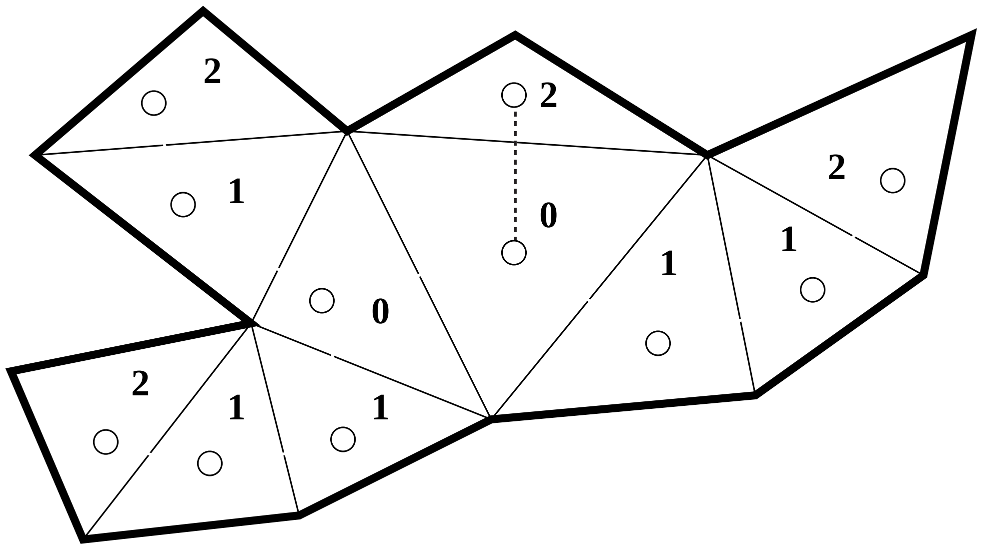


Fig. 3 A triangulation (solid lines) of a simple polygon P, its *dual tree* (dotted lines), and the integer value t_i associated with each triangle.

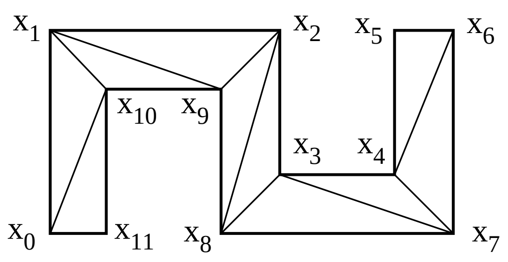


Fig. 2 Illustrating ALGORITHM Triangulate(P).

```
3.          if triangle (PRED($x_j$),$x_j$,SUCC($x_j$)) contains no vertex of R then
4.             return true
5.          else return false
6.       else return false
End IsAnEar
```

In May 1990 Bernard Chazelle finally showed that a simple polygon of n vertices could be triangulated in O(n) time [Ch2]. This discovery is a significant theoretical breakthrough. As a result there is not much merit from the *theoretical time-complexity* point of view in proposing algorithms that are adaptive and only sometimes run in O(n) time unless they contribute also to a *new* theoretical perspective. However, Chazelle's linear time algorithm appears to be difficult to program and thus it is not clear presently if it will have practical consequences. In this light the adaptive algorithm just described and the one proposed in this paper constitute contributions to the practical efficiency of triangulating polygons. These algorithms are easy to describe and program and they run fast in practice.

In this paper we describe a new algorithm for triangulating a simple n-sided polygon. The algorithm runs in time $O(n(1+t_0))$, with $t_0 < n$. The quantity t_0 measures the complexity of the triangulation delivered by the algorithm. More precisely t_0 is the number of triangles in the output triangulation obtained that share zero edges with the input polygon and is related to the *shape-complexity* of the polygon. Although the worst-case complexity of the algorithm is $O(n^2)$, for several classes of polygons it runs in linear time. The practical advantages of the algorithm are that it is extremely simple and does not require sorting or the use of balanced tree structures. On the theoretical side it is of interest because it is the first polygon triangulation algorithm whose *computational* complexity is a function of the *output* complexity. Section 2 discusses a new measure that we propose as the complexity of a triangulation of a polygon. The algorithm is presented in Sections 3 & 4 and some concluding remarks are offered in Section 5.

2. A Measure of the Complexity of a Triangulation

The graph theoretical dual of every polygon triangulation is a tree (see Fig. 3). The nature of a tree suggests a rather natural measure of its shape complexity, namely, its fragmentation, ar-

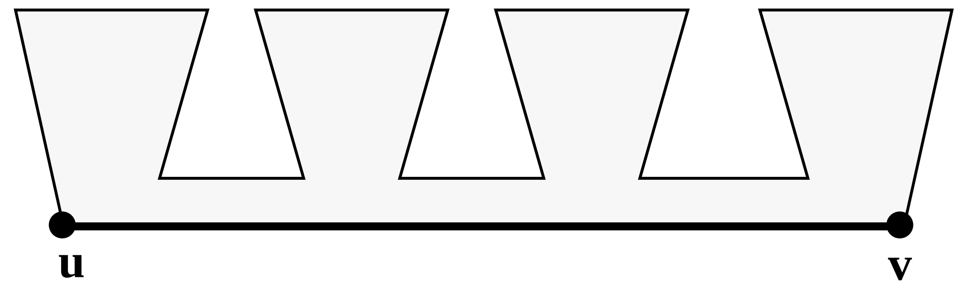


Fig. 1(b): A polygon *edge-visible* from uv with a sinuosity of O(n).

tested is x_6. It is found to be an ear and cut. Again x_4 is tested and this time it is an ear so it is cut. The remaining vertices will be cut in the order x_7, x_3, x_8,x_2, x_9, x_1.

ALGORITHM Triangulate(P): The algorithm takes as input a simple polygon P = $[x_1,x_2,...,x_n]$, stored as a doubly linked circular list. SUCC(x_i) and PRED(x_i) indicate the successor and predecessor of x_i respectively. The algorithm produces a set D of diagonals comprising a triangulation of P. R is a set containing all the concave vertices of P. IsAnEar(P,R,x_i) is a function which returns true if x_i is an ear in polygon P and false otherwise.

```
1. $x_i \leftarrow x_2$;
2. while ($x_i$ is not equal to $x_0$) do
3.     if (IsAnEar(P,R,PRED($x_i$)) and P is not a triangle then       {PRED($x_i$) is an ear.}
4.           D ← D ∪ (PRED(PRED($x_i$)),$x_i$)     {Store a diagonal.}
5.           P ← P - PRED($x_i$)     {Cut the ear.}
6.           if $x_i \in$ R and $x_i$ is a convex vertex then {$x_i$ has become convex.}
7.                 R ← R -$x_i$
8.           if PRED($x_i$) ∈ R and PRED($x_i$) is a convex vertex then {PRED($x_i$) has become
9.                 R ← R - PRED($x_i$)                                        convex.}
10.          if (PRED($x_i$) = $x_0$) then {SUCC($x_0$) was cut.}
11.                $x_i$ ← SUCC($x_i$)   {Advance the scan.}
12.    else $x_i$ ← SUCC($x_i$)     {PRED($x_i$) is not an ear or P is a triangle. Advance the scan.}
13. end while
END Triangulate
```

```
FUNCTION IsAnEar(P,R,$x_i$)
1.  if R = ∅ then return true {P is a convex polygon}
2.  else if $x_j$ is a convex vertex then
```

Finally we mention a new adaptive algorithm discovered recently [KET] that is based on the Graham scan. The Graham scan is a fundamental backtracking technique in computational geometry which was originally designed to compute the convex hull of a set of points in the plane [Gr] and has since found application in several different contexts. In [KET] it is shown how to use the Graham scan to triangulate a simple polygon in O(kn) time where k-1 is the number of concave vertices in P. Although the worst case running time of the algorithm is $O(n^2)$ and hence not as good asymptotically as the algorithm of Hertel & Mehlhorn, it is much easier to implement and is therefore of practical interest. In fact, together with the algorithm presented in this paper it is probably the best way to go in practice. A simple test to determine for a given polygon what the value of k is will determine which algorithm to use. If k is small use the [KET] algorithm, if it is large use the algorithm proposed in this paper. For completeness and availability we include a full description of this algorithm. For a proof of correctness and a complexity analysis the reader is referred to [KET].

The algorithm adapts the Graham scan in the following manner. The vertices of the polygon are scanned in order starting with x_2. At each step the current vertex is tested to see if it is the top of an ear. If it is not the top of an ear then the current vertex is advanced. If it is the top of an ear then the ear is cut off; that is, a diagonal is added to the triangulation and a vertex is deleted from the polygon. The current vertex is not advanced in this case except in the special case that the ear is the vertex following x_0. This prevents x_0 from being cut as an ear.

To illustrate the execution of the algorithm consider the polygon in Fig. 2. Initially, the algorithm tests x_1 and determines that it is not an ear (note that this is equivalent to testing whether x_2 is the top of an ear). The scan is advanced through x_2, x_3, x_4 and x_5 at which time x_5 is determined to be an ear. Next x_5 is cut and then x_4 is tested and found not to be an ear. The next vertex

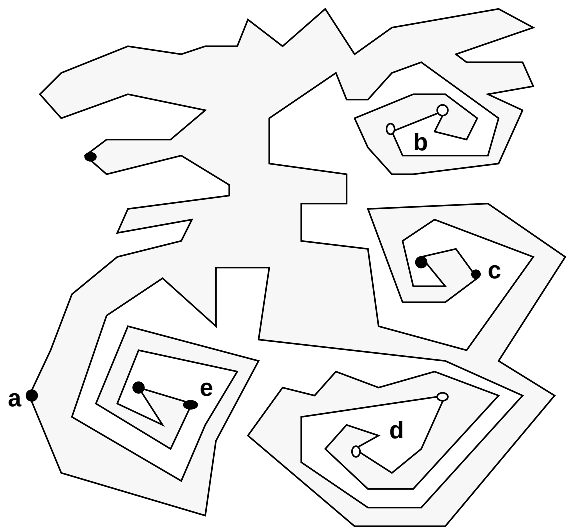


Fig. 1(a): This fairly complicated-looking polygon has a sinuosity of only five. Scanning starts at **a** in a clockwise fashion.

polygons [LC], *weakly-externally-visible* polygons [El], *palm-shaped* polygons [ET], and *anthropomorphic* polygons [To2]. In yet another approach to the problem researchers designed adaptive algorithms that run fast in many situations. Hertel & Mehlhorn [HM] have described a sweep-line based algorithm that performs better the fewer reflex vertices it has. The running time of their method is O(n + r log r) where r denotes the number of reflex vertices of P. Hertel & Mehlhorn's algorithm takes the first step towards obtaining an adaptive algorithm sensitive to the *shape* of the polygon. Unfortunately r is not a truly relevant measure of the shape complexity. To see this it is sufficient to realize that given any polygon of no matter what shape it is a trivial matter to insert n vertices (one between every original pair) and pull them an infinitesimal amount towards the interior of the polygon. Such a transformation will make r proportional to n without changing the basic shape of the polygon.

Chazelle and Incerpi [CI] took a further step to achieve a time complexity that more faithfully reflects the *shape complexity* of the polygon. They describe a triangulation algorithm that runs in time O(n log s) with s < n. The quantity s measures the *sinuosity* of the polygon, i.e., the number of times the polygon's boundary alternates between complete spirals of opposite orientation. Unlike r, s has the advantage that in many practical situations it is very small or a constant even for very winding polygons. Consider the motion of a straight line $L[x_i,x_{i+1}]$ passing through edge $[x_i,x_{i+1}]$ as i goes from 1 to n-1. Every time $L[x_i,x_{i+1}]$ reaches the vertical position in a clockwise (respectively counter-clockwise) manner we increment (respectively decrement) a *winding-counter* by one. $L[x_i,x_{i+1}]$ is said to be *spiraling* (respectively *anti-spiraling*) if the winding counter is never decremented (respectively incremented) twice in succession. In this way the polygon may be decomposed easily in O(n) time into spiraling and anti-spiraling polygonal chains. An example of a polygon with a sinuosity of five is shown in Fig. 1(a). Note that a new polygonal chain is restarted only when the previous chain ceases to be spiraling or anti-spiraling. The *sinuosity* s of P is defined as the number of polygonal chains thus obtained.

The Chazelle-Incerpi algorithm is much more interesting theoretically than the algorithm of Hertel & Mehlhorn because of the implications it has on the complexity of triangulating different known classes of polygons. Since r, the number of reflex vertices, is independent of whether a polygon is *monotonic*, *star-shaped*, *edge-visible* or whatever, Hertel & Mehlhorn's algorithm can run in O(n log n) time for these classes of polygons for which linear time algorithms are known. On the other hand *star-shaped* polygons have a sinuosity of one and thus the Chazelle-Incerpi algorithm runs in linear time for these polygons. Furthermore the algorithm makes no use of the *kernel* of P. In [SV] and [WS] a point in the *kernel* is required and this implies a non-trivial (although linear time) effort. For a completely different and extremely simple algorithm for triangulating a *star-shaped* polygon without making use of the *kernel* of P see [ET1] or [ET2]. However, the sinuosity is not completely satisfactory as a measure of the shape complexity. It has the disconcerting property that it can vary by an order of magnitude depending on the orientation of the input polygon. Consider the *edge-visible* polygon illustrated in Fig. 1(b). Recall that a polygon P is edge visible if there exists an edge [u,v] of P such that for each point x in P there exists a point y in [u,v] such that the line segment [x,y] lies in P. The sinuosity for the polygon in Fig. 1(b) is O(n) and thus the Chazelle-Incerpi algorithm runs in O(n log n) time on this polygon whereas a linear-time algorithm exists [TA]. Furthermore by rotating the polygon through an angle of 90 degrees the sinuosity reduces to O(1). This represents an order of magnitude change in the *sinuosity* of P for no change in the *shape* of P (naturally we assume shape is invariant under translation and rotation).

non-overlapping triangles (their interiors do not intersect) without adding new vertices. Mathematicians have been interested in constructive proofs (algorithms) of the existence of triangulations for simple polygons as early as 1911 [Le]. The “algorithm” of Lennes [Le] works by recursively inserting diagonals between pairs of vertices of P and runs in $O(n^2)$ time. Since then this type of “algorithm” has reappeared in a score of papers and text books during the past seventy years very often and surprisingly containing fundamental errors. See the paper by Chung-Wu Ho [Ho] for a series of counter-examples to published triangulation “proofs.” A rather different inductive proof was offered more recently by Meisters [Me]. He proposed a method based on searching for “ears” and “cutting” them off. We call a vertex x_i of polygon P a *principal* vertex provided that no vertex of P lies in the interior of the triangle $[x_{i-1},x_i,x_{i+1}]$ or in the interior of the diagonal $[x_{i-1},x_{i+1}]$. A *principal* vertex x_i of a simple polygon P is called an *ear* if the diagonal $[x_{i-1},x_{i+1}]$ that bridges x_i lies entirely in P. We say that two ears x_i and x_j are *non-overlapping* if $int[x_{i-1},x_i,x_{i+1}] \cap int[x_{j-1},x_j,x_{j+1}] = \varnothing$. The following *Two-Ears* Theorem was proved by Meisters [Me].

Theorem: (the *Two-Ears* Theorem, Meisters [Me]) Except for triangles every simple polygon P has at least two *non-overlapping ears*.

A straightforward implementation of this idea leads to a complexity of $O(n^3)$. However, it was recently discovered that a prune-and-search technique will actually find an ear in linear time thus yielding an $O(n^2)$ implementation of Meisters’ algorithm [EET]. A good subpolygon of a simple polygon P, denoted by GSP, is a subpolygon whose boundary differs from that of P in at most one edge. A *proper ear* of a good subpolygon GSP is an ear of GSP which is also an ear of P. One of the key observations in [EET] is that a good subpolygon has at least one proper ear. The strategy of their algorithm is as follows. Given a polygon P on n vertices, split it in O(n) time into two subpolygons such that one of these subpolygons is a good subpolygon with at most $\lfloor n/2 \rfloor + 1$ vertices. This splitting step is the crucial step in the algorithm. Subsequently, apply the algorithm recursively to this good subpolygon which is guaranteed to have a proper ear. The worst case running time of the algorithm is given by the recurrence $T(n) = cn + T(\lfloor n/2 \rfloor + 1)$, where c is a constant, which has solution $T(n) \in O(n)$.

The first algorithm to break the $O(n^2)$ upper bound was that of Garey, Johnson, Preparata & Tarjan [GJPT]. Their algorithm runs in time O(n log n) which is the time required by the first step to decompose the polygon into monotone sub-polygons. Then they apply an algorithm for triangulating monotone polygons in linear time. Note that a simpler linear-time algorithm for the latter problem is now available [To1]. An alternate decomposition method with the same complexity appears in [FM]. An entirely different *divide-and-conquer* approach by Chazelle [Ch1] also achieves an O(n log n) upper bound. Finally this upper bound was reduced even further by Tarjan & Van Wyk [TV]. With very complicated and sophisticated data structures they are able to triangulate a simple polygon in O(n log log n) time. However, recently the same complexity was demonstrated using simple data structures [KKT].

Until May 1990 [Ch2]one of the most outstanding open problems in computational geometry has been to determine if a simple polygon can be triangulated in O(n) time. As an alternative some researchers searched for large classes of polygons that can be triangulated in linear time. Such classes include *monotone* polygons [GJPT],[To1], *star-shaped* polygons [SV],[WS], *edge-visible* polygons [TA], *spiral* polygons [FP],[T3], *L-convex* polygons [EAT], *intersection-free*

Efficient Triangulation of Simple Polygons

Godfried Toussaint

School of Computer Science
McGill University
3480 University Street
Montreal, Quebec
Canada H3A 2A7

ABSTRACT

This paper considers the topic of efficiently triangulating a simple polygon with emphasis on practical and easy-to-implement algorithms. It also describes a new *adaptive* algorithm for triangulating a simple n-sided polygon. The algorithm runs in time $O(n(1+t_0))$, with $t_0 < n$. The quantity t_0 measures the *shape-complexity* of the *triangulation* delivered by the algorithm. More precisely t_0 is the number of triangles contained in the triangulation obtained that share zero edges with the input polygon and is, furthermore, related to the shape-complexity of the *input* polygon. Although the worst-case complexity of the algorithm is $O(n^2)$, for several classes of polygons it runs in linear time. The practical advantages of the algorithm are that it is simple and does not require sorting or the use of balanced tree structures. On the theoretical side it is of interest because it is the first polygon triangulation algorithm the *computational* complexity of which is a function of the *output* complexity. As a side benefit we introduce a new measure of the complexity of a polygon triangulation that should find application in other contexts as well.

Key words: polygon, algorithm, triangulation, computational geometry, geometric complexity

1. Introduction

We are concerned with triangulating a special type of polygon in the Euclidean plane E^2 referred to as a *simple* (also *Jordan*) polygon. For any integer $n \geq 3$, we define a *polygon* or *n-gon* in the Euclidean plane E^2 as the figure $P = [x_1,x_2,...,x_n]$ formed by n points $x_1,x_2,...,x_n$ in E^2 and n line segments $[x_i,x_{i+1}]$, i=1,2,...,n-1, and $[x_n,x_1]$. The points x_i are called the *vertices* of the *polygon* and the line segments are termed its *edges*. A polygon P is called a *simple* polygon provided that no point of the plane belongs to more than two edges of P and the only points of the plane that belong to precisely two edges are the vertices of P. A simple polygon has a well defined interior and exterior. We will follow the convention of including the interior of a polygon when referring to P.

Our problem is that of constructing a *triangulation* of P, i.e., decomposing P into a set of