Generating Random Orthogonal Polygons*

Ana Paula Tomás[1] and António Leslie Bajuelos[2]

[1] DCC-FC & LIACC, University of Porto, Portugal
apt@ncc.up.pt
[2] Department of Mathematics / R&D Unit "Mathematics and Applications"
University of Aveiro, Portugal
leslie@mat.ua.pt

Abstract. We propose two different methods for generating random orthogonal polygons with a given number of vertices. One is a polynomial time algorithm and it is supported by a technique we developed to obtain polygons with an increasing number of vertices starting from a unit square. The other follows a constraint programming approach and gives great control on the generated polygons. In particular, it may be used to find all n-vertex orthogonal polygons with no collinear edges that can be drawn in an $\frac{n}{2} \times \frac{n}{2}$ grid, for small n, with symmetries broken.

1 Introduction

Besides being of theoretical interest, the generation of random geometric objects has applications that include the testing and verification of time complexity for computational geometry algorithms, as observed by Zhu et al. in [15] and Auer and Held in [2]. This has also been a major motivation for us. In particular, we needed a sufficiently large number of varied orthogonal polygons to carry out an experimental evaluation of the algorithm proposed in [13] for solving the Minimum Vertex Guard problem for arbitrary polygons. This problem is that of finding a minimum set G of vertices of the given polygon P such that each point in the interior of P is visible from at least a vertex in G, belonging to the *Art Gallery problems* [8, 16].

Polygons are one of the fundamental building blocks in geometric modelling and they are used to represent a wide variety of shapes and figures in computer graphics, vision, pattern recognition, robotics and other computational fields. Some recent publications address uniform random generation of simple polygons with *given vertices*, in the sense that a polygon will be generated with probability $\frac{1}{T}$ if there exist a total of T simple polygons with such vertices [2, 15]. Since no polynomial time algorithm is known to solve the problem, researchers either try to use heuristics [2] or restrict the problem to certain classes of polygons such as monotone [15] or star-shaped [11].

* This work has been partially supported by funds granted to LIACC through *Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia (FCT)* and *Programa POSI*, and by R&D Unit "Mathematics and Applications" (Univ. of Aveiro) through *Programa POCTI, FCT, co-financed by EC fund FEDER.*

Programs for generating random orthogonal polygons have been developed in LISP by O'Rourke et al. to perform experimental tests in the context of [10] and independently by M. Filgueiras in Tcl/Tk (personal communication, 2003). Recently, O'Rourke kindly made available his program to us. The main idea behind it is to construct such a polygon via growth from a seed *cell* (i.e., unit square) in a board, gluing together a given number of cells that are selected randomly using some heuristics. Filgueiras' method shares a similar idea though it glues rectangles of larger areas and allows them to overlap. O'Rourke observed that there is no easy way to control the final number of vertices of the polygon with his program and that what is controlled directly is the area of the polygon, i.e., the number of cells used. However the number of cells may change because the program transforms the generated polygon when it has (non-overlapping) collinear edges to remove such collinearity.

Our contribution. We propose two different methods for generating random orthogonal polygons with *a given number of vertices*. The first one is a polynomial time algorithm. It is supported by a result we prove in this paper that every orthogonal polygon may be obtained from the unit cell by employing a sequence of Inflate-Cut transformations. The other method follows a constraint programming approach, and explores the fact that the generation problem may be modelled naturally as a constraint satisfaction problem in finite domains (CSP). It gives control on other characteristics of polygons, though it is computationally much more expensive. We have not yet tried to classify the degree of randomness of the polygons constructed by our methods.

The paper is structured as follows. In section 2, we introduce basic concepts and results about simple polygons and we describe how to obtain generic orthogonal polygons from standardized orthogonal polygons (that we call grid n-ogons). Then, in sections 3 and 4, we present our two methods for generating grid n-ogons. We conclude giving some implementation details and empirical results in section 5.

2 Background and Terminology

A *simple polygon* P is a region of a plane enclosed by a finite collection of straight line segments forming a simple cycle. Non-adjacent segments do not intersect and two adjacent segments intersect only in their common endpoint. These intersection points are the *vertices* of P and the line segments are the *edges* of P. This paper deals only with simple polygons, so that we call them just polygons, in the sequel. A vertex is called *convex* if the interior angle between its two incident edges is at most π; otherwise it is called *reflex* (or *concave*). We use r to represent the number of reflex vertices of P. A polygon is called *orthogonal* (or *rectilinear*) if its edges are either horizontal or vertical (i.e., if edges meet at right angles). O'Rourke [9] has shown that $n = 2r + 4$ for every n-vertex orthogonal polygon (n-*ogon*, for short). So, orthogonal polygons have an even number of vertices.

Every n-vertex polygon P is well defined by the sequence of its vertices $v_1, \ldots, v_n$, given in counterclockwise (CCW) order. P will be locally to the left of any edge e_i traversed from v_i to v_{i+1}. All index arithmetic will be modulo n, though we shall use n instead of 0, and $e_i = \overline{v_i v_{i+1}}$.

We shall now describe how to obtain generic orthogonal polygons from standardized orthogonal polygons, that we call grid n-ogons.

2.1 Classes of Orthogonal Polygons

Definition 1. *We say that an n-ogon P is in general position iff every horizontal and vertical line contains at most one edge of P, i.e., iff P has no collinear edges. For short, we call "grid n-ogon" each n-ogon in general position defined in a $\frac{n}{2} \times \frac{n}{2}$ square grid.*

Lemma 1. *Each grid n-ogon has exactly one edge in every line of the grid.*

Each n-ogon not in general position may be mapped to an n-ogon in general position by ϵ-perturbations, for a sufficiently small constant $\epsilon > 0$. Consequently, we may restrict generation to n-ogons in general position.

Every n-ogon P in general position may be identified with a unique grid n-ogon, as follows. We consider that the northwest point of the grid has coordinates $(1, 1)$. We order the horizontal edges of P by top-to-bottom sweeping and we order its vertical edges by left-to-right sweeping. If we then assign to each edge its order number, P gets identified with a single grid n-ogon, as shown in Fig. 1. If $\mathcal{V}_{i-1}$ and $\mathcal{H}_i$ are the numbers given to the vertical edge e_{i-1} and to

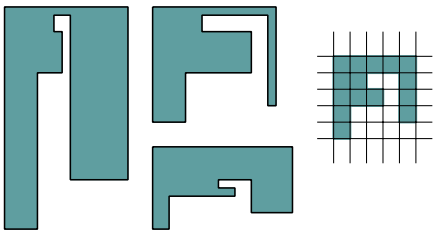

Fig. 1. Identifying three n-ogons with a single grid n-ogon, for $n = 12$.

its consecutive horizontal edge e_i, then $(\mathcal{V}_{i-1}, \mathcal{H}_i)$ is the vertex that corresponds to vertex v_i in the grid n-ogon. Each grid n-ogon identifies a class of n-ogons. Given a grid n-ogon we may create an n-ogon that is an instance of its class by randomly spacing the grid lines though preserving their ordering. When we analysed the program by O'Rourke, we found that this idea was already there. The number of classes may be further reduced if we group grid n-ogons that are identical under symmetries of the square. In this way, the grid n-ogons in Fig. 2 represent the same class. We shall come back to this issue in sections 4 and 5.

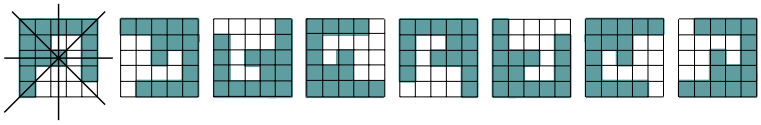

Fig. 2. Eight grid n-ogons that are identical under symmetries of the square. From left to right, we see images by clockwise rotations of 90°, 180° and 270°, by flips wrt horizontal and vertical axes and flips wrt positive and negative diagonals.

3 The Inflate-Cut Algorithm

Fig. 3 illustrates a technique we developed to obtain grid $(n+2)$-ogons from a given grid n-ogon, that we called Inflate-Cut. The idea is that Inflate

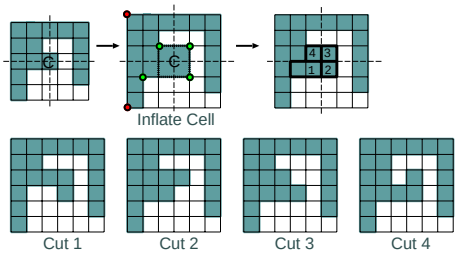

Fig. 3. Illustrating Inflate-Cut transformation. The two rectangles defined by the center of C and the vertices of the leftmost vertical edge $((1,1),(1,7))$ cannot be cut, so that there remain the four possibilities shown.

grows the area of the grid n-ogon P constructed up to a given step and then Cut removes a rectangle from it to create a new grid ogon, with $n+2$ vertices (i.e., with exactly one more reflex vertex). So, let $v_i = (x_i, y_i)$, for $i = 1, \ldots, n$ be the vertices of P and C be a unit cell in its interior. Let (p, q) be the coordinates of the northwest corner of C.

The Inflate *transformation.* The result of applying Inflate to P using C is a new n-vertex orthogonal polygon $\tilde{P}$ with vertices $\tilde{v}_i = (\tilde{x}_i, \tilde{y}_i)$ given by $\tilde{x}_i = x_i$ if $x_i \leq p$ and $\tilde{x}_i = x_i + 1$ if $x_i > p$, and $\tilde{y}_i = y_i$ if $y_i \leq q$ and $\tilde{y}_i = y_i + 1$ if $y_i > q$, for $i = 1, \ldots, n$. Two lines will be free in the new grid, namely $x = p + 1$ and $y = q + 1$. They intersect in the center of inflated C, that will be the new reflex vertex, when the Cut transformation is as we establish now.

The Cut *transformation.* Let $\tilde{v}_C = (p+1, q+1)$. Compute the four intersection points defined by the straight lines $x = p+1$ and $y = q+1$ and the boundary of $\tilde{P}$

(i.e., the points where the four horizontal and vertical rays that start at $\tilde{v}_C$ first intersect the boundary of $\tilde{P}$). Select one of these intersection points at random, and call it $\tilde{s}$. Let $\tilde{v}_m$ be one of the two vertices on the edge of $\tilde{P}$ that contains $\tilde{s}$. The rectangle defined by $\tilde{v}_C$ and $\tilde{v}_m$ may be cut if and only if it contains no vertex of $\tilde{P}$ except $\tilde{v}_m$. If the rectangle may be cut, let $\tilde{s}'$ be $\tilde{v}_C + (\tilde{v}_m - \tilde{s})$, and remove $\tilde{v}_m$ from $\tilde{P}$ inserting instead $\tilde{s}$, $\tilde{v}_C$, $\tilde{s}'$ if this sequence in CCW order (or $\tilde{s}'$, $\tilde{v}_C$, $\tilde{s}$, otherwise). If the rectangle cannot be cut, try to use in a similar way either the other extreme of the edge that contains $\tilde{s}$ or the remaining intersection points. CUT fails for C when no rectangle may be cut, as shown in Fig. 4.

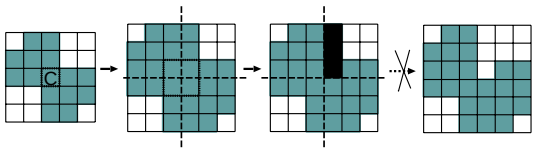

Fig. 4. CUT fails for cell C. We get a grid ogon if we remove the black rectangle, but this cut violates the preconditions of CUT, since the rectangle has two vertices of P.

The INFLATE-CUT algorithm, given in Fig. 5 generates a random grid n-ogon from the unit square (i.e., from the 4-ogon) using INFLATE and CUT. The random

Input: *The number of vertices n (must be even and* ≥ 4*).*
Output: *A random grid n-ogon P.*

r := n/2-2;
$P := \{(1,1),(1,2),(2,2),(2,1)\}$*; /* (i.e., the unit square) */*
while *r > 0* **do**
  repeat
    Select one grid cell C in the interior of P (at random);
    Apply INFLATE *using C to obtain* $\tilde{P}$*;*
    Apply CUT *to remove a rectangle (at random) from* $\tilde{P}$*;*
  while CUT *fails for C;*
  $P := \tilde{P}$*; r := r-1;*
end

Fig. 5. The INFLATE-CUT algorithm.

selections of C and of rectangles must avoid loops. There is always one cell that may be inflated to continue the process. In fact, CUT never fails when the unit cell C has an edge that is part of an edge of P.

The Inflate-Cut algorithm may be adapted to generate all grid n-ogons, for a given n, instead of one at random. Proposition 1 states the completeness of Inflate-Cut. Its proof is not immediate, as indicated by the example in Fig. 6.

Proposition 1. *For each grid $(n+2)$-ogon, there is a grid n-ogon that yields it by* Inflate-Cut*, for all even $n \geq 4$.*

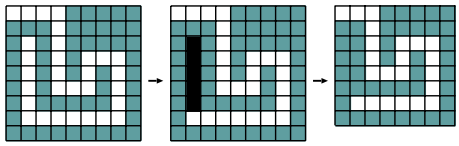

Fig. 6. The inverse of Inflate-Cut transformation: finding a grid n-ogon that yields a given grid $(n+2)$-ogon. The rightmost polygon is the unique grid 16-ogon that gives rise to this 18-ogon, if we employ Inflate-Cut.

The major ideas of our proofs are illustrated in Fig. 7. They were inspired by work about convexification of simple polygons, and, in particular, by a recent paper by O. Aichholzer et al. [1]. This topic was introduced by Paul Erdös [4] who conjectured that a nonconvex polygon would become convex by repeatedly reflecting all its *pockets* simultaneously across their *lids*. This conjecture was later corrected by Nagy [12], who showed the Erdös-Nagy theorem that requires that pockets be flipped one by one.

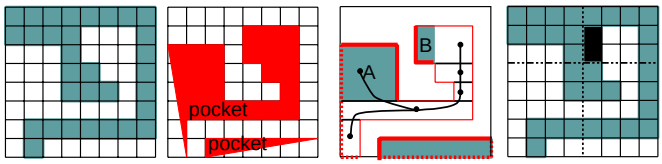

Fig. 7. The two leftmost grids show a grid 18-ogon and its pockets. The shaded rectangles A and B are either leaves or contained in leaves of the tree associated to the horizontal partitioning of the largest pocket. The rightmost polygon is an inflated grid 16-ogon that yields the represented grid 18-ogon, if Cut removes rectangle B.

Any nonconvex polygon P has at least one concavity, or *pocket*. A pocket is a maximal sequence of edges of P disjoint from its convex hull except at the endpoints. The pocket's *lid* is the line segment joining its endpoints. Pockets of n-ogons, together with their lids, define simple polygons that are almost orthogonal except for an edge. We may render them orthogonal by introducing two *artificial edges*, as shown in Fig. 7. Let Π be the decomposition of an orthogonalized

pocket into rectangles obtained by extending the horizontal edges incident to its reflex vertices until they hit the boundary. Then, we consider the *dual graph* of Π, that captures the adjacency relation between pieces of Π. Its nodes are the pieces of Π and its edges connect adjacent pieces. Such a graph is always a tree and every leaf that does not contain the artificial horizontal edge contains (or is itself) a rectangle that might have been removed by CUT. Now, at most one leaf contains the artificial horizontal edge. Moreover, if the tree consists of a single node (rectangle), it trivially meets the conditions of CUT.

Though, this proof shares ideas of a proof of Meisters' *Two-Ears Theorem* [7] by O'Rourke, we were not aware of this when we wrote it. Another concept – *mouth* – helped us refine CUT and reduce the number of possible ways of generating a given $(n+2)$-ogon.

Definition 2. *(Toussaint [14]) A vertex v_i of P is said to be a principal vertex provided that no vertex of P lies in the interior of the triangle (v_{i-1}, v_i, v_{i+1}) or in the interior of the diagonal (v_{i-1}, v_{i+1}). A principal vertex v_i of a simple polygon P is called mouth if the diagonal (v_{i-1}, v_{i+1}) is an external diagonal, i.e., the interior of (v_{i-1}, v_{i+1}) lies in the exterior of P.*

We are not aware of any work that uses this concept as we do. Nevertheless, INFLATE-CUT is somehow doing the reverse of an algorithm given by Toussaint in [14] that computes the convex hull of a polygon globbing-up mouths to successively remove its concavities

For orthogonal polygons, we would rather define *rectangular mouths*, saying that a reflex vertex v_i is a rectangular mouth of P iff the interior of the rectangle defined by (v_{i-1}, v_i, v_{i+1}) is contained in the exterior of P and contains no other vertices of P. To justify the correction of our technique, we observe that when we apply CUT to obtain a grid $(n+2)$-ogon, the vertex $\tilde{v}_C$ (that was the center the inflated grid cell C) is always a *rectangular mouth* of the resulting $(n+2)$-ogon. In sum, we may rephrase the *One-Mouth Theorem* by Toussaint [14] and show Proposition 2.

Proposition 2. *Each grid n-ogon has at least one rectangular mouth, for $n \geq 6$.*

4 A Constraint Programming Approach

The method we have described can be easily adapted to generate all grid n-ogons, for small values of n. For instance, by applying INFLATE-CUT to all cells in the polygon and considering all possibilities of cutting rectangles, for each cell. It is difficult to see how to eliminate symmetric solutions, or even if that is possible. Clearly, we would like to eliminate the possibility of generating them. The number of grid n-ogons grows exponentially with n, so that it does not make sense the comparison of solutions to filter out symmetric ones. To make matters worse, the same n-ogon may be generated from a unit square by performing similar transformations, though in a different order. It is easy to locally detect and break some symmetries: if a given n-ogon is symmetric, then we would possibly not apply INFLATE-CUT to cells that are in symmetric positions.

A major advantage of the following approach is the ability to control the generated polygons by imposing constraints in addition to a basic core that translates the notion of grid n-ogon. In particular, it is possible to completely break symmetry during search through constraints. Several approaches and techniques have been proposed (e.g, [5, 6]) for similar purposes, in recent years.

For certain applications, as for example, to check conjectures, it is also important to restrict the generated polygons (e.g., to constrain their area, the number of consecutive reflex vertices, the length of their edges, etc). This second approach gives this kind of control.

4.1 A Simple Constraint Satisfaction Model

Let us define a grid n-ogon by giving the sequence

$$(X_1, Y_1),\ (X_1, Y_2),\ (X_2, Y_2),\ (X_2, Y_3),\ (X_3, Y_3),\ \ldots, (X_k, Y_k),\ (X_k, Y_1)$$

of its vertices, for example, in CCW order, with $k = \frac{n}{2}$. So, to generate grid n-ogons we have to find ordered sequences $X_1, X_2, \ldots, X_k$ and $Y_1, Y_2, \ldots, Y_k$, such that $Y_i \in 1..k$, $X_i \in 1..k$, for all i, with all X_i's and Y_i's distinct (for the polygon to be in general position). Additional constraints must be imposed to guarantee that non-adjacent edges will not intersect. As there is a single edge per line, we just have to avoid intersections between each vertical edge and all horizontal edges that are non-adjacent to it. When the vertices are ordered as above, $e_{1+2j} = \overline{(X_{1+j}, Y_{1+j}), (X_{1+j}, Y_{2+j})}$ and $e_{2+2j} = \overline{(X_{1+j}, Y_{2+j}), (X_{2+j}, Y_{2+j})}$ define the polygon's edges, for $j = 0, \ldots, k-1$, being $k+1 \equiv 1$. For all j, the vertical edge e_{1+2j} does not intersect any non-adjacent horizontal edge $e_{2+2j'}$ if and only if (1) holds for $j' \in \{0, \ldots, k-1\} \setminus \{j-1, j\}$,

$$\frac{X_{1+j} - X_{1+j'}}{X_{2+j'} - X_{1+j'}} \notin [0,1] \quad \vee \quad \frac{Y_{2+j'} - Y_{1+j}}{Y_{2+j} - Y_{1+j}} \notin [0,1] \tag{1}$$

To encode these constraints as finite domain constraints, we get more disjunctive constraints

$$\frac{A-B}{C-B} \notin [0,1] \equiv (C > B \wedge (A \leq B \vee A \geq C)) \vee (C < B \wedge (A \geq B \vee A \leq C))$$

the model being actually declaratively simple but computationally too complex.

5 Experimental Evaluation

We have developed prototype implementations of the Inflate-Cut method and of this constraint programming model. For Inflate-Cut, we may show that our implementation (in the C language) uses linear space in the number of vertices and runs in quadratic time in average. Fig. 8 shows some empirical results. To achieve this, it keeps the vertices of P in a circular doubly linked list, its area (i.e., the number of cells) and the total number of cells of P in each horizontal

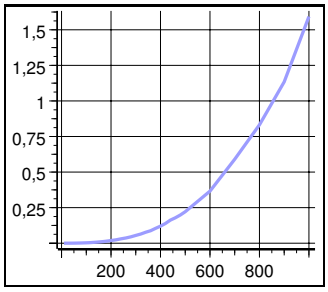

Fig. 8. Average time performance of our implementation of INFLATE-CUT. It yields a random grid 1000-ogon in 1.6 seconds in average (AMD Athlon Processor at 900 MHz).

grid line, this also in a linked list. To compute the coordinates (p, q) that identify cell C, it first uses these counters to find row q and then left-to-right and top-to-bottom sweeping techniques to find column p and the four delimiting edges. Moreover, it first checks whether CUT will succeed for C and performs explicitly INFLATE only if it does. To check whether a rectangle may be cut it performs a rotational sweep of the vertices of P.

As concerns the second method, we developed a prototype implementation using CLP(FD) [3] for SICStus Prolog together with a simple graphical interface written in Tcl/Tk to output the generated polygons and interact with the generator. This program may be used to generate either an n-ogon at random or all n-ogons by backtracking, for very small n.

To break symmetries we added additional constraints to the CSP model described above. First we required that $X_1 = 1$. This means that amongst all sequences that are circular permutations of $v_1, \ldots, v_n$, we choose the one that defines v_1 as the highest vertex in the leftmost vertical edge of P. Then, we require that $P \preceq_{LEX} \sigma(P)$, for all $\sigma \in \mathrm{D}_4$, where D_4 is the group defined by the eight symmetries of the square grid and $\preceq_{LEX}$ is the lexicographic order. If we refer back to Fig. 2, this means that only the second polygon there would be generated. On the other hand, if we add $P = \sigma(P)$, for all σ in a subgroup S of D_4, the generator yields polygons that exhibit all the symmetries in S. Our implementation has been mainly used by us to test some conjectures about orthogonal polygons (for $n < 20$) and to construct symmetrical n-ogons (for $n < 90$). Global constraints were used to implement this ordering constraint and those preventing intersections of non-adjacent edges. Nevertheless, we think that efficiency could still be improved by exploring geometric features of the solutions to the problem and adding redundant constraints to reduce the search.

Since the number of polygons grows exponentially with n, and we need polygons with a larger number of vertices, the empirical results indicate that it may be impractical to use our second method to produce a relevant tests set to the algorithm described in [13]. So, INFLATE-CUT algorithm has some important advantages.

6 Conclusions

Two different methods were proposed for generating random orthogonal polygons with a given number of vertices. One allows for great control on the form of polygons, but is computationally too expensive. The other is a polynomial time algorithm and seems more adequate for the generation of random polygons, provided no special features are sought, apart from the desired number of vertices. Similar ideas may be explored to generate random simple polygons.

Acknowledgement. We would like to thank Domingos Cardoso for detecting a flaw in our former proof of Proposition 1.

References

1. Aichholzer, O., Cortés, C., Demaine, E. D., Dujmovic, V., Erickson, J., Meijer, H., Overmars, M., Palop, B., Ramaswawi, S., Toussaint, G. T.: Flipturning polygons. Discrete & Computational Geometry **28** (2002), 231–253.
2. Auer, T., Held, M.: Heuristics for the generation of random polygons. In Fiala, F., Kranakis, E., Sack, J.-R. (Eds), Proc. of 8th Canadian Conf. Computational Geometry (CCCG'96), Carleton University Press (1996) 38–44.
3. Carlsson M., Ottosson G., Carlson B.: An open-ended finite domain constraint solver. In Proc. of PLILP'97, LNCS 1292, 191-206, Springer-Verlag, 1997.
4. Erdös, P.: Problem number 3763. American Mathematical Monthly **42** (1935) 627.
5. Gent, I. P. , Harvey, W., Kelsey, T.: Groups and constraints – symmetry breaking during search. In P. Van Hentenryck (Ed.), Proc. of CP'02, LNCS 2470, Springer-Verlag (2002) 415-430.
6. McDonald, I., Smith, B.: Partial symmetry breaking. In P. Van Hentenryck (Ed.), Proc. of CP'02 LNCS 2470, Springer-Verlag (2002) 431-445.
7. Meisters, G. H.: Polygons have ears. American Mathematical Monthly **82** (1975) 648-651.
8. O'Rourke, J.: Art gallery theorems and algorithms. Oxford University Press (1987).
9. O'Rourke, J.: An alternate proof of the rectilinear art gallery theorem. J. of Geometry **21** (1983) 118–130.
10. O'Rourke, J., Pashchenko, I., Tewari, G.: Partitioning orthogonal polygons into fat rectangles. In Proc. 13th Canadian Conference on Computational Geometry (CCCG'01) (2001) 133-136.
11. Sohler, C.: Generating random star-shaped polygons. In Proc. 11th Canadian Conference on Computational Geometry (CCCG'99) (1999) 174–177.
12. Sz.-Nagy, B.: Solution of problem 3763. American Mathematical Monthly **46** (1939) 176–177.
13. Tomás, A. P., Bajuelos, A. L., Marques, F.: Approximation algorithms to minimum vertex cover problems on polygons and terrains. In P.M.A Sloot et al. (Eds): Proc. of ICCS 2003, LNCS 2657, Springer-Verlag (2003) 869-878.
14. Toussaint, G. T.: Polygons are anthropomorphic. American Mathematical Monthly **122** (1991) 31–35.
15. Zhu, G., Sundaram, G., Snoeyink, J., Mitchell, J.: Generating random polygons with given vertices. Computational Geom. Theory and Appl. **6**(5) (1996) 277–290.
16. Urrutia, J.: Art gallery and illumination problems. In J.-R. Sack and J. Urrutia, editors, Handbook on Computational Geometry. Elsevier (2000).