Japan J. Indust. Appl. Math. (2017) 34:407–428
DOI 10.1007/s13160-017-0258-8

ORIGINAL PAPER

Area 4

Use of simple polygonal chains in generating random simple polygons

Ali Nourollah[1] · **Mohsen Movahedinejad**[1]

Received: 26 June 2015 / Revised: 7 April 2017 / Published online: 1 July 2017

Abstract The main motivation for generating random simple polygons is to produce test instances for geometric algorithms. In this paper three new algorithms are proposed to generate random simple polygons. A point set in a two dimensional plane is the input, and a simple polygon is the output of the problem. At first a new algorithm to convert any kind of simple polygonal chains into simple polygons is presented and the correctness of the algorithm is proved. Then three new algorithms are presented to produce random simple polygonal chains from the given point set. The first algorithm is capable of producing 2^n simple polygonal chains. The second algorithm works by the concept of divide and conquer and the third algorithm is the most complete and produces all the possible simple polygonal chains. The worst time complexities of these three chain generation algorithms are $O(n^2)$, $O(n^2)$ and $O(n^3)$ respectively and the time complexity of the conversion algorithm is $O(n*l)$, where $1 < l < n$. The polygon generation algorithm works in this way that first each simple polygonal chain generation algorithms are applied over the point set and then the generated chains are converted to simple polygons. The number of different simple polygons generated by each of three algorithms is compared with the well-known algorithms and the experimental results show that the third algorithm produces more polygons rather than the well-known *2-opt move* algorithm. The first algorithm acts better than the second algorithm, where both act better than *steady Growth*.

✉ Ali Nourollah
nourollah@aut.ac.ir

Mohsen Movahedinejad
m.movahedinejad@srttu.edu

[1] Shahid Rajaee Teacher Training University, Tehran, Iran

Keywords Simple polygon generation · Simple polygonal chain · Computational geometry

Mathematics Subject Classification 68U05 · 68W20 · 05C10

1 Introduction

The problem of generating random simple polygons uniformly has taken attention of lots of researchers because of its theoretical importance and applications. There is no polynomial time algorithm to generate all the possible polygons uniformly at random and nobody has proved that there can be no such algorithm, which is the reason for the theoretical interest among the researchers. In recent years, some researchers have worked on heuristics to generate simple polygons which are not uniformly distributed or are restricted to a certain class of polygons such as monotone or star-shaped polygons [1–4].

One of the applications of generating random simple polygons is to evaluate the correctness of the algorithms which are hard to be evaluated by white box methods [5,6]. In the case of evaluating such algorithms, a lot of random polygons are given to the algorithm. If the distribution of the generated polygons is uniform, and the algorithm results are correct for all the inputs, it could be sure that the algorithm is correct with a high degree of certainty. This process is called the black box test [5,6]. So if there is an algorithm to generate simple polygons at random with uniform distribution, it would be possible to evaluate the correctness of the algorithms easily. The other application of generating random simple polygons is their use to present a wide variety of shapes and figures in computer graphics, machine vision, pattern recognition, robotics and other computational fields [7–10].

Since 1992, the generation of geometric objects has become an interesting research topic to the researchers for its different applications. Epstein [11] studied random generation of triangulation. Zhu designed an algorithm to generate x-monotone polygons uniformly at random on a given set of points [1]. A heuristic [12] for generating simple polygons was investigated in 1991. The 2-Opt Move heuristic was first proposed to solve the traveling salesman problem by J.van Leeuwen et al. [13]. In 1996, Thomas Auer et al. [2] presented a study of all heuristics present at that time and reported a variety of comparisons among them.

The paper follows as in the second section the algorithm to convert a chain into simple polygon is covered. In the third, fourth and fifth sections the three heuristics to generate random simple polygonal chains are discussed. In the sixth section, time complexities of the algorithms are analyzed, and experimental results are covered. The seventh section is devoted to conclusions.

2 The algorithm to convert a simple polygonal chain into a simple polygon

The notations used in this paper are defined before explaining the algorithms, which are as follows. The input point sets which lies in a two-dimensional plane are shown

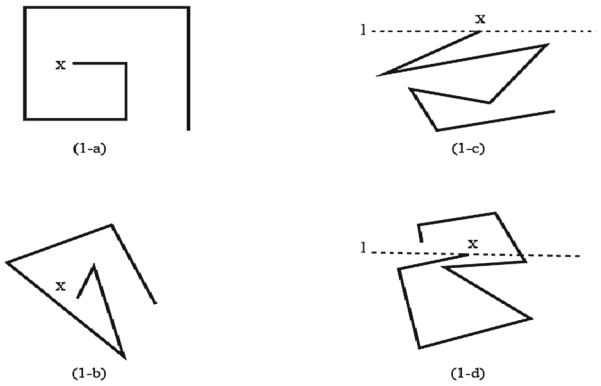

Fig. 1 **a** The head vertex X is in the semi-circle, **b** the head vertex X is not in a semi-circle

with $S = \{p_1, p_2, \ldots, p_n\}$, where n is the number of input points. The input points are supposed to lay in general positions for simplicity. A polygonal chain is specified by a sequence of points $(q_1, q_2, \ldots, q_n)$ called its vertices [14] and a simple polygonal chain is one in which only consecutive segments intersect and only at their endpoints [14]. In this paper, a segment is shown with a bar over two vertices. The vertices with degree one are called the heads of the chain. We define the concept of semi-circle and say a head vertex lies in a semi-circle in the chain if any line passes through this vertex has at least two intersection points with the chain, as in Fig. 1a and b the head vertex X lies in semi-circle because any arbitrary line passing through X has two or more than two intersection with the chain. In Fig. 1c and d the head vertex X is not in a semi-circle, because there is at least one line that has no or just one intersection with the chain. In Fig. 1c the dashed line l has no intersection with the chain and in Fig. 1d the dashed line l has just one intersection with the chain, so the head vertex X is not in a semi-circle. The words chain or polygonal chain would refer to a simple polygonal chain if seen anywhere in the paper.

The proposed algorithm converts any kind of simple polygonal chain into a simple polygon. In a polygonal chain the degree of the first and the last nodes is one and the other nodes have the degree of two. The nodes with degree one are selected. We call them P_{first} and P_{last} and imagine a hypothetical segment between P_{first} and P_{last}. If the segment $\overline{P_{first}P_{last}}$ does not intersect any edges of the polygonal chain, P_{first} would be connected to P_{last} and the algorithm finishes. If there are some edges which have intersection with the segment $\overline{P_{first}P_{last}}$, the intersection point for each edge and the distance between the intersection points and P_{first} and P_{last} are calculated. Then the closest edge to P_{first} and P_{last} are selected. Next, one of the heads P_{first} or P_{last} is selected at random. The selected head is called X and the other is called Y. The polygonal chain is traversed from X until the closest segment of the polygonal chain to X which intersected with the segment $\overline{P_{first}P_{last}}$ is obtained. The vertices of this segment are called M and N where M is the vertex seen first on the traverse. Now the segment $\overline{\text{MN}}$ is omitted from the chain, and the segment $\overline{\text{XN}}$ is added instead. So, a new polygonal chain is produced and Y and M are the heads of this chain. Then, if M and Y are visible, these two heads are connected to each other and the algorithm

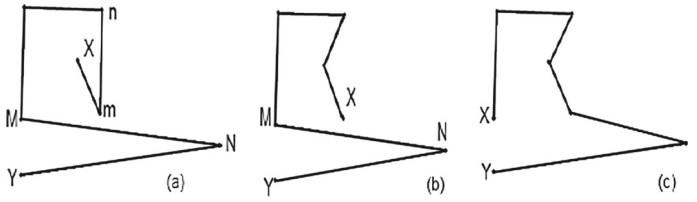

Fig. 2 In The case where adding an edge to the chain is impossible, a new recursion is called again

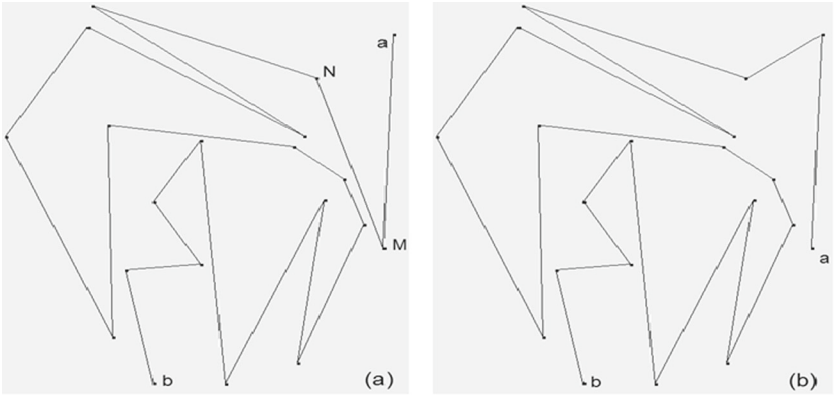

Fig. 3 **a** The input chain, **b** the chain produced after one recursion

finishes, otherwise the program calls itself recursively with the newly produced chain. The proposed algorithm is continued for the newly generated chain recursively until we get a chain whose heads can see each other, and thus the algorithm is finished.

In some cases, it is impossible to add a segment to the chain because the vertices of that segment do not see each other. In Fig. 2a, according to the algorithm, we have to connect X to N and remove $\overline{\text{MN}}$, but X and N are not visible. In this case, we call another recursion of the algorithm with the same chain, but the heads of chain change to X and N. In this case, the traverse begins from X. $\overline{mn}$ intersects with $\overline{\text{XN}}$, so $\overline{mn}$ is removed, and $\overline{\text{X}n}$ is added to the chain. Now, a new polygonal chain is produced. X and Y are the heads of the new chain, Fig. 2b. Starting the traverse from X, $\overline{\text{MN}}$ is removed, $\overline{\text{XN}}$ is added to the chain, and the final chain is produced, Fig. 2c.

Figure 3 through Fig. 5 show the steps of the algorithm over a twenty-point set. Figure 3a shows the input chain. Points a and b are the heads of the chain and a is selected as X at random. The closest intersection to a is the segment $\overline{\text{MN}}$, so the segment $\overline{\text{MN}}$ is omitted and the segment $\overline{aN}$ is added. Figure 3b shows the newly produced chain after one step.

Figure 4a shows the polygonal chain produced in the previous step where the point b is selected as X and $\overline{\text{MN}}$ is the closest segment to b. According to the algorithm, $\overline{\text{MN}}$ is removed and $\overline{bN}$ is added to the chain. Figure 4b shows the newly generated chain.

Figures 5a–c shows the last three recursions of the algorithm. In each figure one intersection between a and b is removed, and a new chain is generated. Each newly generated chain is the input of a new recursion until a and b are visible to each other,

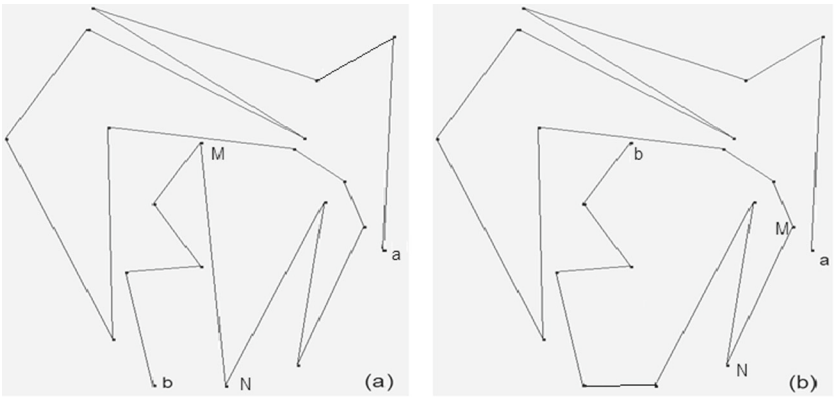

Fig. 4 **a** The chain produced in the second step, **b** the new chain generated from **4a**

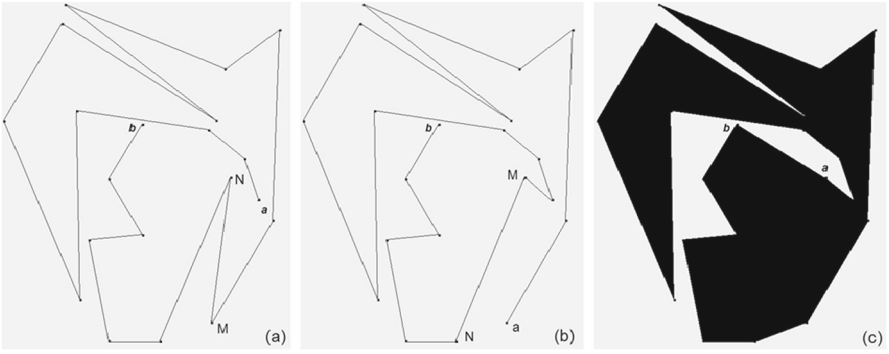

Fig. 5 Three last recursions which cause the generation of polygon

so a and b connect to each other and the polygon is generated. Figure 6 shows the pseudo code written for this algorithm.

Theorem I *The algorithm ConvertChainToPolygon always terminates.*

Proof This algorithm terminates, if the ModifyChain function terminates. The termination criterion of this function happens when the set β is empty. The proof is based on two cases. We demonstrate that in both cases the termination criterion happens.

(a). The condition of line 9 (i.e. visible(X,N)) satisfies.

In this case $\overline{\text{MN}}$ is deleted from the chain and $\overline{\text{XN}}$ is added to the chain, so the new heads of chain change to M and Y. It is clear that $\overline{\text{MY}}$ never intersects $\overline{\text{MN}}$ and $\overline{\text{XN}}$, because $\overline{\text{XN}}$ is always on one side of $\overline{\text{MN}}$ and $\overline{\text{MY}}$ is on the other side. So, in each step one intersection is deleted, and the new added segment will not be a member of β anymore. After each recursion, the position of heads of the chain change, so the number of members of β may increase or decrease. Since the new added segment will not be a member of β, so all the β members belong to the origin chain and since the number of edges of the chain is limited to n, in the worst case, the algorithm will finish after n recursion. The below examples explain this case clearly.

Algorithm I: ConvertChainToPolygon(C,count,f,l)
Input: C is a chain, count is the number of points, f and l are the heads of the chain.
Output: a simple polygon.
1. ModifyChain(C,count,f,l)
2. connect f to l and return the generated polygon.

function ModifyChain(C,count,f,l)
Input: C is a chain, count is the number of points, f and l are the heads of the chain.
Output: a simple polygonal chain where its heads are visible to each other.
1. β .
2. Traverse the chain to find the segments which intersect with segment $\overline{fl}$ and add them into β.
3. **if** β==$\emptyset$ **then**
4. **return** C
5. **else**
6. Choose one of the points f and l at random, and call it X, and call the other one Y.
7. Traverse the chain from X, and find the closest intersecting segment to X, and call this segment $\overline{\text{MN}}$.
8. M is the first point seen on the traverse.
9. **If** visible(X,N) **then**
10. $C \leftarrow C$ - $\overline{\text{MN}}$
11. $C \leftarrow C$ + $\overline{\text{XN}}$
12. **return** ModifyChain(C,count,M,Y).
13. **else**
14. $C \leftarrow$ ModifyChain(C,count,X,N).
15. **return** ModifyChain(C,count,M,Y).
16. **endif**
17. **endif**
18.**end function**

Fig. 6 The algorithm to convert simple polygonal chains into simple polygons

According to the algorithm and Fig. 7, if X and N are visible, the segment $\overline{MN}$ is deleted and the segment $\overline{XN}$ is added to the chain and another recursive function is called for the new chain. Figure 7a is a chain and after one step of the algorithm Fig. 7b is produced. As you see in Fig. 7a there are three edges intersecting $\overline{XY}$, which is reduced to one in Fig. 7b. In fact both the deleted and added segments do not intersect the new $\overline{XY}$. It is because $\overline{XN}$ is always in one side of $\overline{MN}$ and $\overline{MY}$, is on the other side and this cause a decrease in the number of intersections by at least one.

In Fig. 8a there is just one segment intersecting $\overline{XY}$, while after one step this number is increased to 6, Fig. 8b. The main idea is that the newly added segment cd is not among the new intersections. The new intersections belong to chain and the number of the edges of the chain is limited to n. In each step one intersection is deleted and the new added segment will not be an intersection anymore. So, in ModifyChain function, if line 9 always satisfies during the execution, the algorithm will terminate in less than n steps.

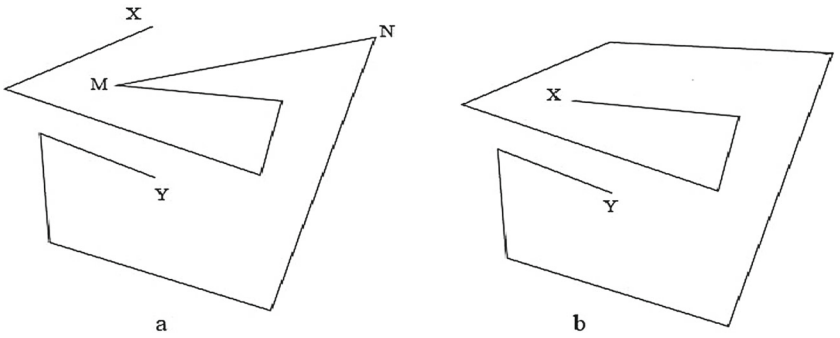

Fig. 7 **a** A polygonal chain, **b** the new polygonal chain generated after one step

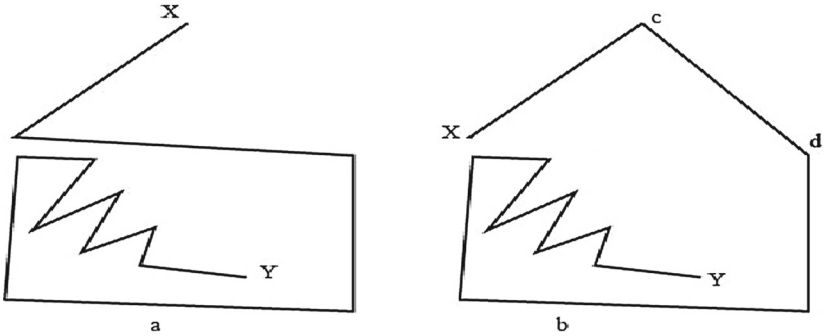

Fig. 8 **a** A polygonal chain with one incident, **b** the new polygonal chain generated after one step with 6 incidents

(b). The condition of line 9 (i.e. visible(X,N)) does not satisfy.

In this case, lines 13–16 of ModifyChain function will be executed. This situation may happen when the head vertex which the traverse begins from there is in a semicircle. To prove that the algorithm terminates, we should prove that this situation also leads to the termination of the algorithm. If X and N are not visible, the recursive function of line 14 will be called, where the heads of the chain are changed and a smaller part of the chain is considered. For this recursion, if the condition of line 9 satisfies, the termination of the algorithm can be proved using (a). But if it does not satisfy, another recursion of line 14 will be executed, and each time this recursive function is executed the considered chain will be smaller until its length will be one or two where there would be no intersection and it will meet the termination criterion. It can be concluded from cases (a) and (b) that the Algorithm I terminates. □

In Fig. 9a the function ConvertChainToPolygon(C,8,G,L) is called which the traverse begins from L. L and H are not visible and the function ConvertChainToPolygon(C,7,L,H) is called and traverse begins from L again. In this case L and K are visible, so they connect and the function ConvertChainToPolygon(C,7,N,H) is called, Fig. 9b. The points N and J are not visible and the function ConvertChainToPolygon(C,5,N,J) is called. N and L are visible and they connect to each other, Fig. 9c. M and J are visible and they connect to each other, Fig. 9d. K and H are visible and

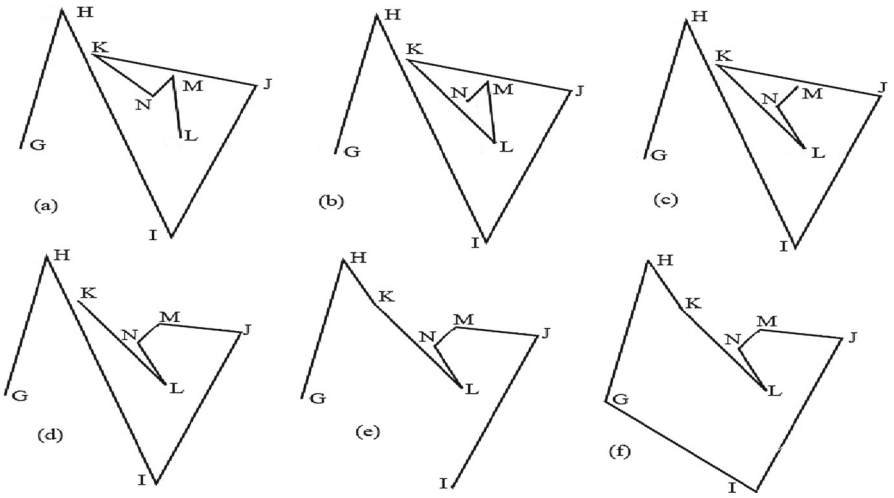

Fig. 9 The situation to call smaller chains

they connect, Fig. 9e and ultimately the points G and I are visible and the algorithm finishes.

Output of each step of the algorithm is a simple polygonal chain, so the output of the ultimate step would be a simple polygonal chain with this difference that its heads are visible. It is clear if we connect these two heads the output will be a simple planner polygon.

Lemma 1 *Assume u represents the number of removed segments in the algorithm of converting a polygonal chain into a polygon. u will never be greater than n.*

In each step of the algorithm, one edge of the chain is removed, and another edge is added to the chain. To prove the lemma, two situations are considered:

1. All the edges removed from the chains belong to the origin chain (input chain). In this case, the lemma is proved easily, because the origin chain has *n-1* edges, and if all are removed, *u* will not be greater than *n*.
2. Some removed edges do not belong to the origin chain. If the heads of the chain do not lie in a semi-circle, it is impossible that the added edges be removed from the chain in any step of the algorithm because the added edge never lies between the new heads. In fact, the added segment lies on one side of the removed edge, and the line between new heads lies on the other side. In Fig. 10 the chain heads are not in a semi-circle. $\overline{\mathrm{MN}}$ is removed, and $\overline{P_{last}N}$ is added to the chain. Because $\overline{P_{last}N}$ is on the right of $\overline{\mathrm{MN}}$, and $\overline{\mathrm{M}P_{first}}$ is on the left of $\overline{\mathrm{MN}}$, $\overline{P_{last}N}$ has no intersection with $\overline{\mathrm{M}P_{first}}$.

In some chains, one or both of the heads lie in a semi-circle. In such chains, it is possible to remove an edge which was added in previous steps. Figure 11 shows a chain with this characteristic. In Fig. 11 *c* lies in a semi-circle, because any line which pass through *c* cuts the chain in more than one point. In Fig. 11 according to

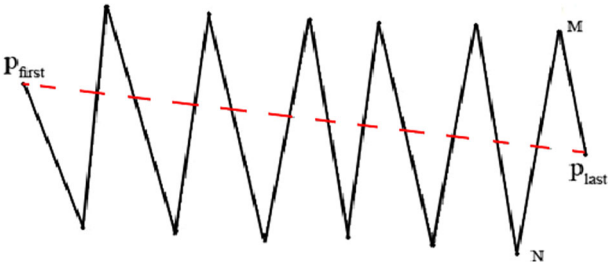

Fig. 10 The worst time complexity is $O(n^3)$

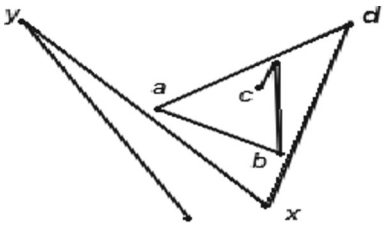

Fig. 11 The head c lies inside a semi-circle

the algorithm, if traverse begins from c, the segment $\overline{ac}$ is added in first step and $\overline{ab}$ is removed. But in the second step, $\overline{ac}$ is removed, and $\overline{ab}$ is added to the chain again.

When one or both of the heads lie in a semi-circle, it seems it is possible to add a lot of edges and then remove them, so the number of recursions would be more than n. But in fact, the deletion of an added edge just happens once for each head in a semi-circle, and for each semi-circle there would be at least one edge such that has no intersection with the connection line of two heads, so the prove is done. In Fig. 11, $\overline{dx}$ is the segment which never lies between two heads of the chain and is never omitted from the chain.

3 The first algorithm to produce simple polygonal chain

This is an incremental algorithm and in each step, a simple polygonal chain is produced. The algorithm works as in each step one point is added to the chain such that the remaining points of the points set can see the heads of the generated chain, and so in each step the chain can grow to reach to the final chain. Figure 12 shows the pseudo code of the algorithm. At first the rightmost point of S is selected and named b. The chosen point is added to the chain C. This point is the starting point of the chain and is removed from S. Next an array with n cells is defined and is initiated with random values from the set {*'max'*,*'min'*}. In Fig. 12, in each step of the loop, one vertex is added to the chain. When the value of the ith element of the array is *'min'*, the 9th line is executed, and the point m is selected from S such that $\widehat{abm}$ gets its minimum value counter clock wise and if the value of the ith element of the array is *'max'*, line 11 is executed and m is chosen from S such that $\widehat{abm}$ gets its maximum value counter clock wise. The chain C and variables a and b get their new values in line 16. Figure 13 shows the process of producing a chain in a set with 9 points.

Algorithm II: $ChainGeneration_1(S,n)$
input: let S={p_1,p_2,...,p_n } and n = number of points.
output: C=(q_1,q_2,...,q_n) shows the polygonal chain.
1. C .
2. let α be an array of size n, and it is initiated by 'max' and 'min' randomly.
3. select the rightmost point, and name it b.
4. $q_1 \leftarrow b$, remove b from S.
5. generate a random point on the right of b,and name it a.
6. **for** i$\leftarrow$ 2 **to** n
7. **do**
8. **if** α[i]='min' **then**
9. $q_i \leftarrow argmin_{m\in S}\ \widehat{abm}$
10. **else**
11. $q_i \leftarrow argmax_{m\in S}\ \widehat{abm}$
12. **endif**
13. **endif**
14. add q_i to the end of chain C, and remove it from S.
15. let $b \leftarrow q_i$ and $a \leftarrow b$.
16. **endfor**
17. **return** C.

Fig. 12 The pseudo code of first algorithm to generate random simple chain

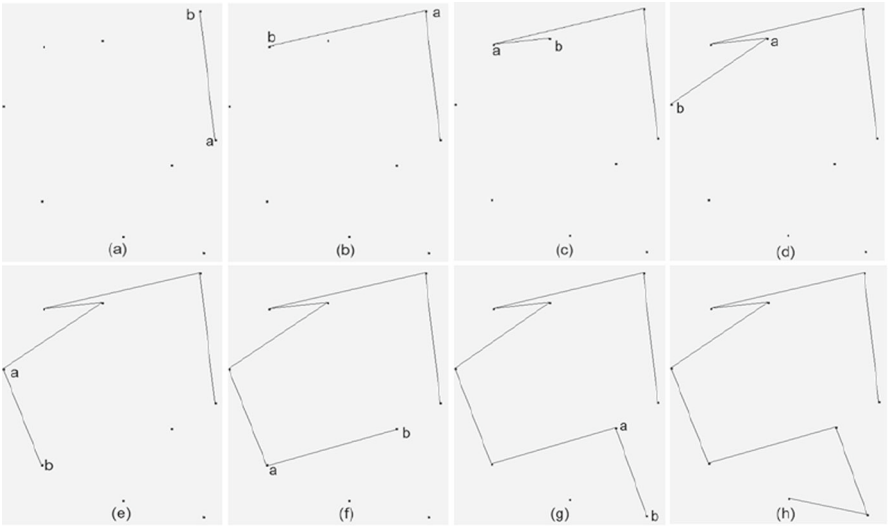

Fig. 13 The process of producing a polygonal chain with the given array as ['min', 'min', 'max', 'min', 'min', 'max', 'max', 'min']

In Fig. 13, array A is equal to [*'min'*, *'min'*, *'max'*, *'min'*, *'min'*, *'max'*, *'max'*, *'min'*]. In the first step, the rightmost point is selected and called *b*. A random point is produced on the right of *b* and called *a*. Since the first element of A is *'min'*, the point *m* which is a member of *S* is selected such that $\widehat{abm}$ gets its minimum value counter clock wise. *m* is added to the chain as the second vertex and is removed from *S*. *a* and *b* get their new values. Figure 13a shows the produced chain in the first step. The

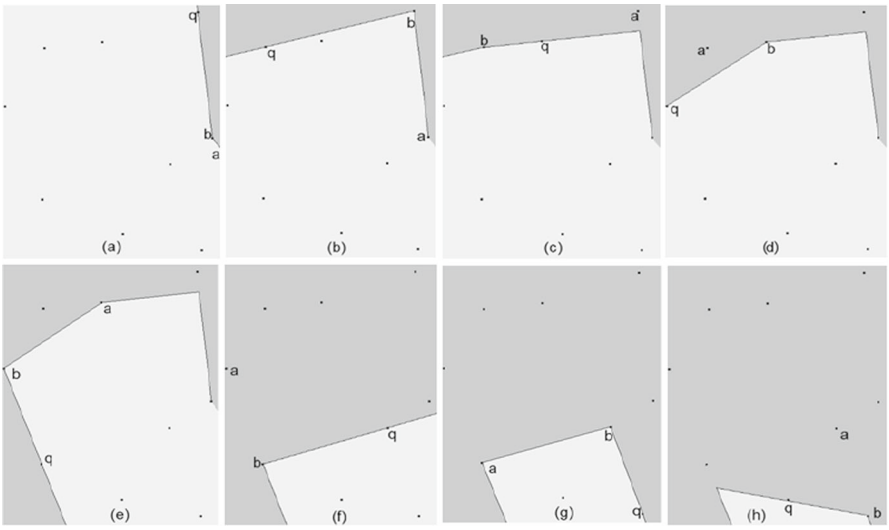

Fig. 14 The space belonging to S is restricted in each step

second element of A is *'min'*, so like the previous step, m is selected from S such that $\widehat{abm}$ gets its minimum value. Next S, a, and b are updated according to the pseudo code. Figure 13b shows the produced chain in this step. In the third step, the value of A is *'max'*, so m is selected such that $\widehat{abm}$ gets its maximum value. Figure 13c shows the related chain of this step. This process continues until there are no more points in S. Figure 13d–h show the generated chains of each step.

With the policy of choosing the minimum or the maximum angle in *CCW* order in each step, the point set is practically divided into two regions. The removed points from S belong to one region and the points in S are in the other. In Fig. 14, the points over the lines or in the darker spaces are the points removed from S and the points in the lighter colored spaces belong to S. Figure 14 shows how in each step, the problem space is divided. When q is selected, it has the minimum or maximum angle with the last segment of the chain in *CCW* order, so there would be no point of S inside $\widehat{abq}$. This means that the selected point belongs to convex hull of S. In each step some space is added to the darker space which has only one point inside. The new point added to chain C can always see the whole points of S, because the point q belongs to convex hull of S and a convex hull point can see the whole points of S. As in Fig. 14, the point q can see the points of S in every step. So we can add any point of S to the chain at any step. This proves that the algorithm terminates and the output of the algorithm is always a simple polygonal chain.

Figure 14a–h show the process of dividing the point set into two regions respectively. In all the steps, q is the new point added to the chain and can see all the points belonging to S. a and b are the two last points of the chain in each step.

According to the algorithm, to create a chain with n vertices, an array with n element is needed. Each element in the n-element array contains the values 'max' or 'man', so each element has 2 options, and 2^n different combinations are possible where each

one represents a chain. This algorithm produces 2^n simple polygonal chain for a point set with size n.

4 The second algorithm to produce simple polygonal chain

The second algorithm to generate simple polygonal chain is based on the concept of Divide and Conquer. Two variables are used to divide the problem space. The first variable is called θ which takes a real value among zero and 180 and describes the angle which the points are sorted according to that. Another variable is an integer number between 1 and n-1 and is called Z. The sorted points are divided into two groups, the first Z points of the sorted point set and the remaining n-Z points. If number of points in each group becomes less than three, the points in that group are connected to each other and produce a simple polygonal chain. But if the number of points in groups are more than three, division process with new Z and θ is continued recursively. The convex hulls of points in groups do not intersect with each other because the groups are divided with straight lines. This feature helps to merge the chains of groups and make a new chain. Figure 15 shows the divide and conquer approach algorithm to produce simple polygonal chain.

The line 12 in algorithm of Fig. 15 shows the merge function of two polygonal chains. The merge process of two simple polygonal chains converts each chain into simple polygons and then merges the polygons and produces a new polygonal chain.

According to what said, to cover the merge function, an algorithm to convert a polygonal chain into a simple polygon and another algorithm to merge two simple polygons into a simple chain is needed. The algorithm to convert a chain into polygon was discussed in Sect. 2 and each two simple chains are converted into simple polygons by this algorithm. The groups of points are divided with straight lines, so the convex hull of the points in each group does not intersect and therefore no polygons will intersect with other polygons. There is a lemma in [1] indicating that a point outside

AlgorithmIII: $ChainGeneration_2(S,n)$
Input: let $S=\{p_1,p_2,...,p_n\}$ and $n=$ number of points.
Output: $C=(q_1,q_2,...,q_n)$shows the polygonal chain.
1. θ is a random number between zero and 180.
2. Z is a random number between one and n-1.
3. $C \leftarrow$ null.
4. **if** n $\leq$ 2
5. Add the points to C and return C.
6. **else**
7. Sort the points of S in θ direction.
8. $S_1 \leftarrow$ the first Z point of S.
9. $S_2 \leftarrow S$-S_1.
10. C_1=ChainGeneration(S_1,Z)
11. C_2=ChainGeneration(S_2,n-Z)
12 C=MergeTheChain(C_1,Z,C_2,n-Z,θ)
13. **endif**

Fig. 15 The second algorithm to produce simple polygonal chain

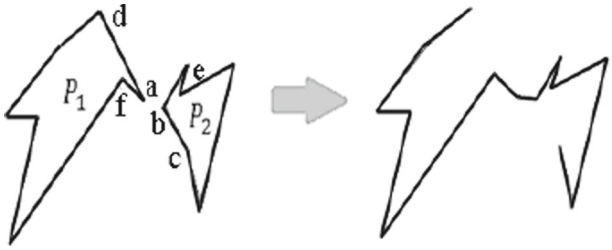

Fig. 16 Merge two polygons to a simple polygonal chain

AlgorithmIV: MergeTheChain(C_1,n_1,C_2,n_2, θ)
Input: C_1 and C_2 as input chain, θ is the dividing angle. n_1 and n_2 are size of chains.
Output:C shows the output polygonal chain.
1. P_1= ConvertChainToPolygon(C_1,n_1,$C_1 first$,C_{1last}).
2. P_2= ConvertChainToPolygon(C_2,n_2,C_{2first},C_{2last}).
3. Array A= the projection value of points over a line with degree θ+90.
4. A= Sort(A)
5. a,b = two consecutive points of A which belong to different polygons.
6. C= P_1 and P_2 where one of the edges which has a as vertex is deleted arbitrarily, one of the edges which has b as vertex is deleted arbitrarily, and a connects to b .

Fig. 17 The merge algorithm of two polygonal chains

of the convex hull of a polygon is visible to at least one edge of that polygon. According to this Lemma we are sure that there is two points from each polygon which are visible.

To find two points which are visible to each other, all the points of two polygons are projected to a line with the angle of θ+90. These groups of points were separated by the help of a line with a random degree of θ. All the points are sorted in an array according to their projection value. Since the points with smaller projection value belong to one polygon and the points with bigger projection value belong to the other polygon, all the consecutive array elements belong to the same polygon except one pair. So, we search the array in O(n) to find a pair of points which one of them belongs to one polygon and the other belongs to the other polygon. These two points are visible for sure, because there is no other point between them. we connect these two points. In each of the merging polygons an edge which the newly connected points is one of its vertices is omitted arbitrarily. This causes to generate a simple polygonal chain from two different polygons.

In Fig. 16 a and b are the closest visible pair from polygons P_1 and P_2, so they are connected to each other. An edge which is incident to a in P_1, and b in P_2 is omitted arbitrarily. These edges are $\overline{ad}$ and $\overline{bc}$ from P_1 and P_2 respectively. Figure 16 shows the two merging polygons and the related generated simple polygonal chain. Now according to what mentioned, the pseudo code to merge two simple polygonal chains would be like Fig. 17.

In each step of Algorithm III the point set is divided into two pieces, so the recursive calls will proceed to the situation that the sets have less than three members, and they connect to generate chains. Since the points of sets are divided by straight lines, the

convex hull of points will not conflict and so according to Lemma in [1] there are points from different sets which are visible. These points can connect to merge the chains according to Algorithm IV, and if this process continues the main chain will be generated by merging smaller chains.

5 The third algorithm to produce random simple polygonal chain

The algorithm works as follow, at first two points are selected from the point set arbitrarily and a segment connects these two points. This segment is a simple polygonal chain. The proposed algorithm is incremental and in each step a point is added to the chain and at last the final simple polygonal chain is generated. The new point is added to one of the heads of the simple polygonal chain. If the added segment has no intersection with previous segments, the generated polygonal chain is simple. But if the new segment has intersections with previous segments, the algorithm will omit these conflicts. Figure 18 shows the pseudo code of the chain generation algorithm and Fig. 19 shows the steps of producing a polygonal chain for a 9 point set case.

Suppose f and l are the heads of the chain and γ is selected from S at random to be added to the chain. If $\overline{f\gamma}$ has no intersection with the chain, f is connected to γ, but if

```
AlgorithmV: ChainGeneration3(C,S,n)
Input: C is an empty chain, let S={p1,p2,...,pn} and n= number of points.
Output: C.
1. γ, f and l are random points from S and are removed from S.
2. f is connected to l and γ, these three points are added to C.
3. while(S is not empty)
4.     f referes to the head of the chain again.
5.     γ is selected at random from S, j←0.
6.     while(γ is not added to C and j < n)
7.         the intersecting segments of C with fγ are added to β, j++.
8.         if (β==∅) do
9.           connect f to γ and remove γ from S, add γ to the first of C.
10.        else
11.            search β to find the closest segment to f, and call this seg-
               ment MN, where M is seen before N on the traverse.
12.            If visible(f,N) do
13.                C←C-MN
14.                C←C+fN
15.            else
16.                ConvertChainToPolygon(C,size of C,f,N). //algorithm
                    of figure 6.
17.            endif
18.        endif
19.    endwhile
20.    if (j==n)
21.        C= ConvertChainToPolygon(C,size of C,f,l).
22.        add γ to polygon and make a new chain as C.
23.        remove γ from S.
24.    endif
25. endwhile
26. return C
```

Fig. 18 The third algorithm to produce simple polygonal chain

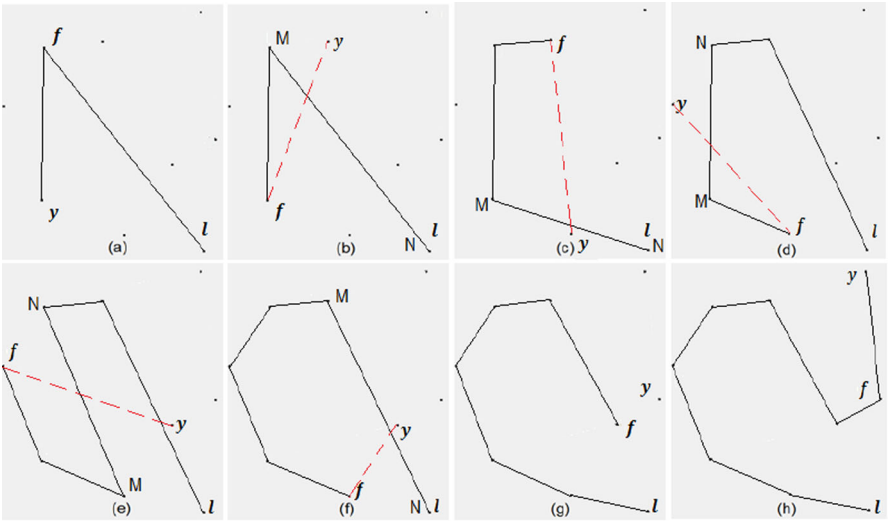

Fig. 19 The process of producing a polygonal chain

$\overline{f\gamma}$ intersects the chain, the chain is traversed from f and $\overline{MN}$ is the closest segment to f which has intersection with $\overline{f\gamma}$. The point M is seen before N in the traverse. The segment $\overline{MN}$ is removed from the chain and $\overline{fN}$ is added. This action leads to a new chain where l and M are its heads. Now if γ can see M the algorithm is finished and γ is connected to M, and the polygonal chain of this step is completed. f refers to γ and the chain heads are again f and l. But if γ cannot see M another recursion of the algorithm for newly generated chain is called. for simplicity the explanation was for situation where γ is going to connect to f, clearly γ can connect to l too).

The mentioned algorithm is capable of producing all the simple polygonal chains. A simple polygonal chain is an order of visiting the vertices or it is a permutation of the vertices, but there should be no conflict among the edges. This algorithm is incremental and since it selects the vertices randomly, it can generate all the possible permutation and chains. It is clear that the set of simple polygonal chains is a subset of chains and this algorithm produces all the chain, so it produces all the simple polygonal chain. The time complexity of the algorithm is $O(n^2 * \mathrm{w})$ where $1 < w < n$, so the worst case time complexity of the algorithm is $O(n^3)$. w shows the number of recursions of the algorithm and as proved in Sect. 2, w is never more than n.

Figure 19a shows the first step of the algorithm, where three points are selected from S at random and are called f, l and γ. f connects to l and also to γ. These points are deleted from S. This is a chain and the edges are added to C. In each step, after updating the chain f refers to the new head of the chain. As in Fig. 19b f and l are pointed to the heads of the chain.

In the second step in Fig. 19b, γ is selected from S at random, but $\overline{MN}$ intersects $\overline{f\gamma}$. So the chain changes as f connects to N and $\overline{MN}$ is removed from the chain. M is the new head of the chain and is called f. Now, f and γ are visible and are connected to each other and the chain in Fig. 19c is created. In Fig. 19c $\overline{MN}$ intersects $\overline{f\gamma}$. $\overline{fN}$ is added to the chain and $\overline{MN}$ is removed from the chain, then f refers to M and as f

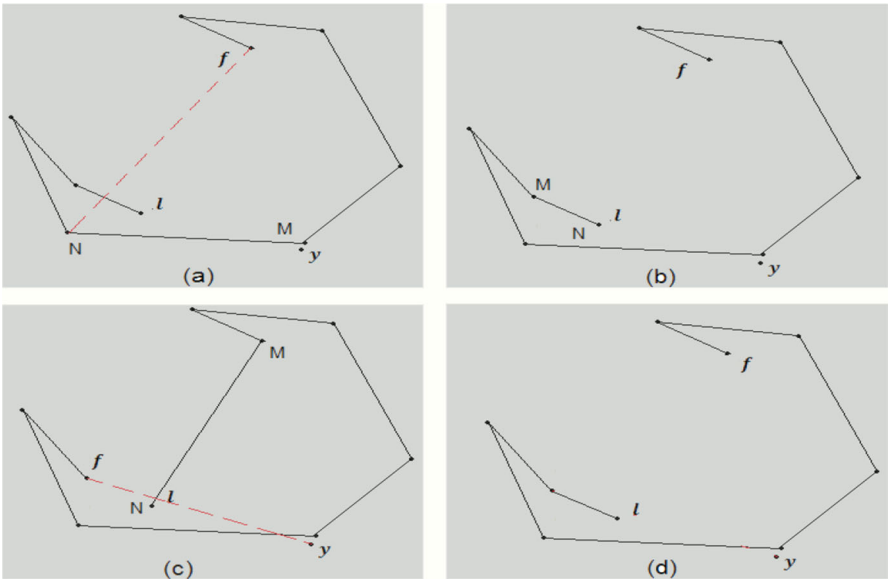

Fig. 20 The case in which the algorithm sticks in loop

and γ are visible they are connected to each other and the new chain with new heads is produced. This chain is shown in Fig. 19d. The actions in Fig. 19d are the same as in Fig. 19b and c, so the chain in Fig. 19e is produced.

In Fig. 19e $\overline{f\gamma}$ intersects two segment of the chain. The closest intersected segment to f is called $\overline{MN}$. Then f connects to N and f refers to M which is the new head of the chain and the chain of Fig. 19f is produced. In Fig. 19f, $\overline{MN}$ intersects $\overline{f\gamma}$, so f connects to N and $\overline{MN}$ is removed and the new head of the chain is called f. Now, f can see γ and these points are connected and again f is the new head of the chain, like Fig. 19g. Figure 19h shows two last steps of the algorithm, where in each step f and γ are visible and the algorithms easily connects these two points.

In some cases the algorithm is traped in loop. This algorithm is like the ConvertChainToPolygon algorithm, but in that algorithm it was almost impossible that the newly added segment and the segment between new heads conflict, and this was the clue for termination. In this algorithm if two events happen simultaneously the algorithm may stuck in a loop. The first event is that the newly added segment and the segment between the head and the random point conflict. The second event is that the head of the chain that the traverse begins from there changes in each iteration. The chain in Fig. 20a converts to the chain in Fig. 20c according to the algorithm, and it is converted to the chain in Fig. 20d which is the same as Fig. 20a. In Fig. 20c the newly added segment and the segment between head and the point conflict, which is the first event($\overline{MN}$ conflicts $\overline{f\gamma}$). If the heads which traverse begins there change in each iteration this process will continue steadily. The second event happens rarely. We know that every random point shoud be added to the chain in at most n step. If the number of steps is more than n we know that it is a loop, we use a counter to distinguish the loop. To overcome this problem, the algorithm of converting chain into

polygon is called for the chain and the chain is converted into a simple polygon. The point y can be added to the generated polygon easily and the result would be a simple polygonal chain.

As you see in Algorithm V, the lines through 7–18 is exactly like the algorithm of converting chain into polygon, which we proved it terminates. These codes lie inside two while loops. The inner loop finishes in n or less steps and after each time this loop finishes an edge is added to the chain. In the case that j is equal to n the chain is converted into a polygon and the random point is added to the polygon and a new chain is produced. The outer loop finishes when there is no more vertices to be added to the chain and since the inner loop adds a vertex to the chain in each step the outer loop will finish after n step. This shows that the algorithm terminates and generates a simple polygonal chain.

6 Experimental results

Since the time complexities of *Space Partitioning* and *Steady Growth* are the same as the time complexities of algorithms based on *ChainGeneration1* and *ChainGeneration2*, which all are of the order of $O(n^2)$ [15], the CPU consumption is used to compare the algorithms. The *2-OptMove* algorithm is the best known algorithm so far, because it produces more polygons rather than other algorithms. It has the time complexity of $O(n^4)$ [15]. In this section it will be shown by experiments that the algorithm based on the *ChainGeneration3* generates more polygons than *2-OptMove*. This algorithm has the time complexity of $O(n^3)$ which is better than *2-OptMove*.

The most important criteria to compare the performance of random simple polygon generation algorithms are the number of different polygons generated and the amount of CPU consumption of each algorithm. The algorithms which generate more polygons with a high degree of uniformity are better. From the other point of view, the algorithms are better which solve the problem with using less CPU resources. In this paper the methods like *2-OptMove, Steady Growth* and *Space Partitioning* are compared with the three proposed algorithms. The ability of algorithms in generating different polygons and the CPU consumption of each algorithm are the criteria of the comparison.

To compare the number of different polygons generated, ten different 10 and 15 point sets are used as samples and the ratio of different polygons generated for each algorithm is compared using these samples. A modified version of *Incremental Construction and Backtracking* algorithm is used to count the total number of simple polygons for each point sets. Table 1 shows the total number of possible simple polygons for each sample.

The ratio of different polygons generated for each algorithm is calculated from the division of number of different polygons generated by the algorithm over the whole number of existing polygons (the values written in Table 1). Let t denote the number of polygons generated by the algorithms and k denotes the total number of simple polygons for each sample. If m represent the number of different simple polygons generated by each algorithms, so the value of $m/(\min(t,k))$ describes the ability of the algorithms in generating different polygons. This value is calculated for all the algorithms and all the samples and the results are depicted in Fig. 21a and b for 10 points

Table 1 Number of simple polygons for 10 and 15 points set problems

Samples	10 points set	15 points set
1	325	87,137
2	742	142,570
3	358	210,425
4	420	128,642
5	157	62,349
6	802	125,551
7	286	89,142
8	527	131,286
9	216	286,428
10	88	175,624

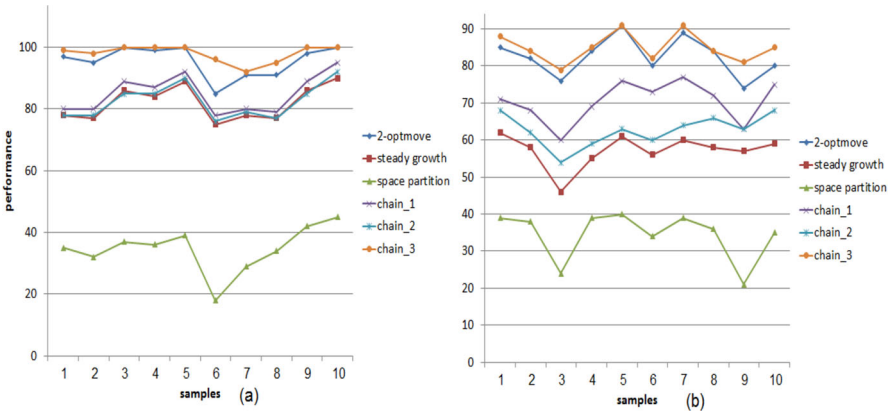

Fig. 21 Comparison of the algorithms on the criteria of number of generated polygons. **a** 10 points set samples, **b** 15 points set samples

and 15 points sets, respectively. In this paper the polygon generation algorithm based on generating polygonal chains and converting them into simple polygons is compared to the previous algorithms. In each sample 5000 simple polygons is generated for each algorithm and the hit rate of the algorithms is calculated based on the formula $m/(\min(t,k))$.

The *2-OptMove* algorithm used to produce more simple polygon than any other algorithms and it has been the best known algorithm so far [2], but as seen in Fig. 21a and b, the number of polygons generated by the algorithm based on the *ChainGeneration3* is more than *2-OptMove* algorithm. As we know, its time complexity is better either.

The algorithm based on the generation of the first kind of chains, produces more polygons rather than the algorithm based on the generation of the second kind of chains, and both of these algorithms have better performance rather than the well-known *Steady Growth* and *Space Partitioning* algorithms.

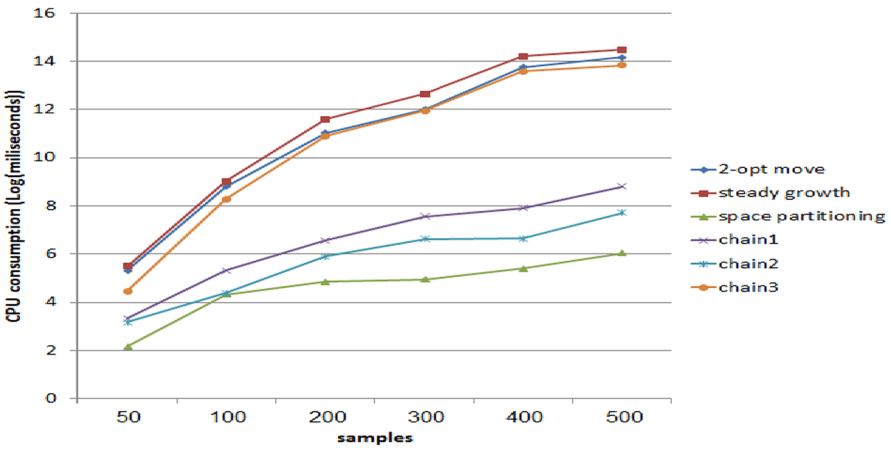

Fig. 22 Comparison of the algorithms on the criteria of CPU consumption

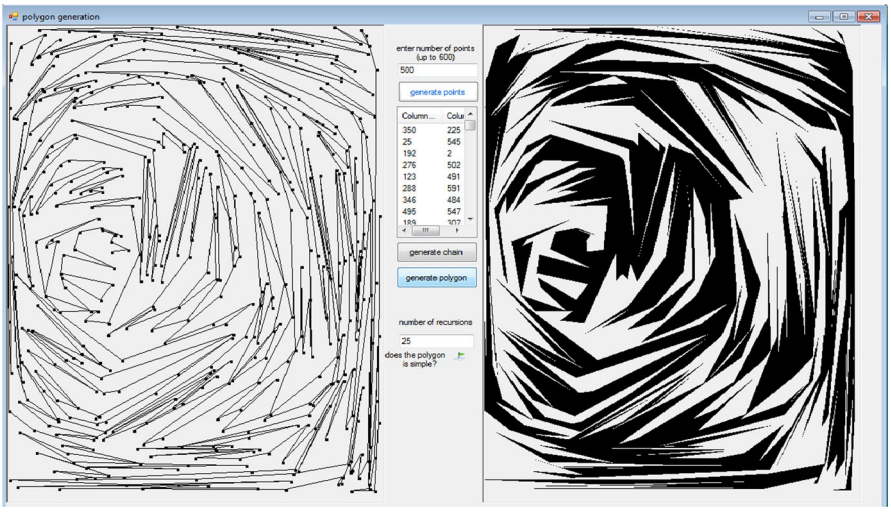

Fig. 23 The chain and the polygon generated by the first algorithm

Another criterion of comparison is CPU consumption of each algorithm. The problems with 50, 100, 200, 300, 400 and 500 points in each point set are considered. Ten random point sets are generated for each of these problem sizes randomly. The CPU elapsed time for each algorithm and each sample is calculated using the Stopwatch class of C# programming language. There are 10 different random point sets for each problem size. For each algorithm, The CPU consumption time is calculated for each point set and their average is assigned to that problem size and algorithm. Figure 22 shows the results for CPU consumption of 6 different algorithms over six problem sizes.

As you see in Fig. 22, *Space Partitioning* is significantly faster than the other Methods. *Steady growth* is the slower one. The methods *Steady Growth, 2-opt move,*

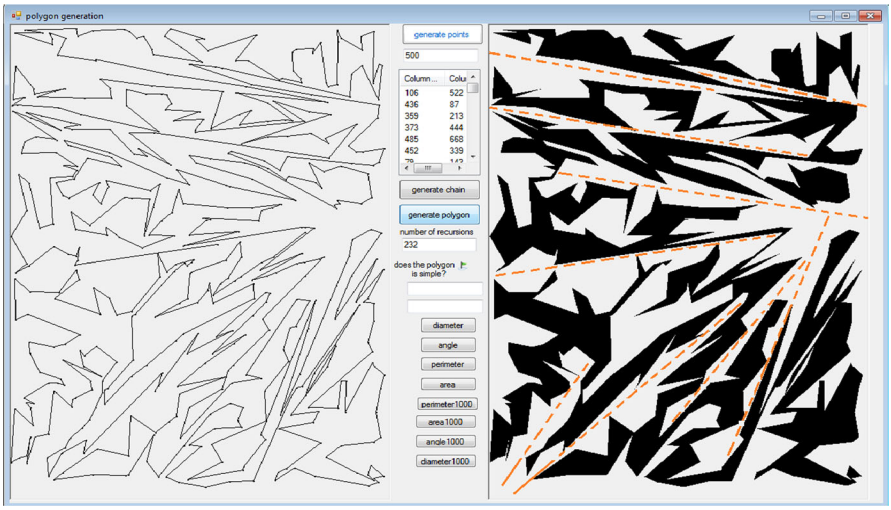

Fig. 24 The chain and the polygon generated by the second algorithm

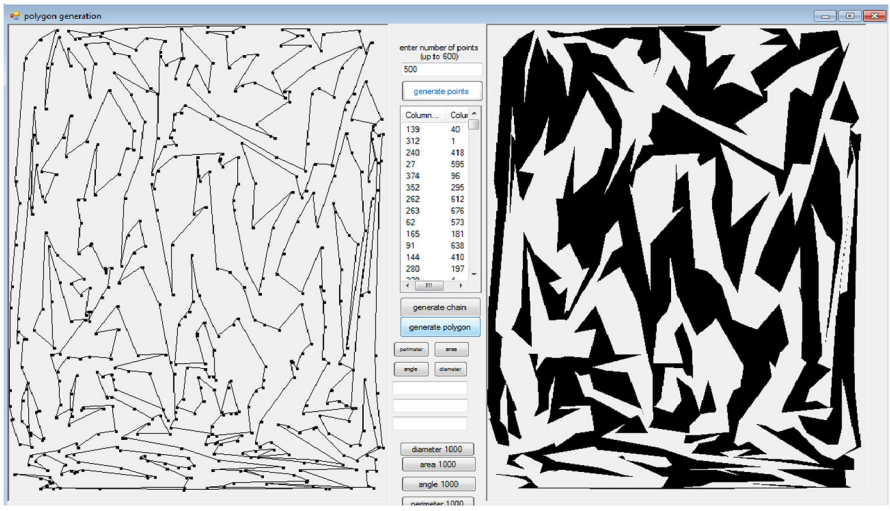

Fig. 25 The chain and the polygon generated by the third algorithm

ConvertChainToPolygon3, ConvertChainToPolygon1, ConvertChainToPolygon2 and *Space Partitioning* are solving the problem with 500 points in 23.1, 18.43, 14.63, 0.45, 0.21 and 0.066 s on average. *ConvertChainToPolygon3* and *2-opt move* are the algorithms which generate more polygons rather than other algorithms and *ConvertChainToPolygon3* is slightly better than the *2-opt move* algorithm in CPU consumption and polygon generation criteria.

Figures 23, 24 and 25 show the polygons generated by the first, second and the third chain generation algorithms, respectively. All the figures show the chain and the polygon generated for a 500 points set problem. In Fig. 23 the sharpness of the angles

is because of choosing the minimum or maximum angle in each step of generating the chain. Dashed lines in Fig. 24 show some of the lines to divide the problem space. These lines are conjectured from the shape of the polygon and are drawn by hand after generating the polygon, so this algorithm produces some special polygons which can be easily divided into different areas by straight lines. Figure 25 shows a 500 points polygon and is generated by converting the chain generated by the third algorithm into simple polygon.

7 Conclusions

The problem of generating random simple polygonal chains is simpler than generating random simple polygons and has fewer constraints. The main idea of this paper is to generate random simple polygonal chains and then to convert them into simple polygons. In this paper three algorithms are proposed to generate random simple polygonal chains, and an algorithm is proposed to convert simple polygonal chains into simple polygons. The first algorithm to generate simple chains is capable of producing 2^n simple polygonal chain. The second algorithm works by the concept of divide and conquer and the third algorithm is the most complete and produces all the possible simple polygonal chains. The worst time complexities of these three chain generation algorithms are $O(n^2)$, $O(n^2)$ and $O(n^3)$ respectively and the time complexity of the conversion algorithm is $O(n*l)$, where $1 < l < n$. The number of different simple polygons generated by each of three algorithms is compared with the well-known algorithms and the experimental results show that the third algorithm produces more polygons rather than the well-known *2-OptMove* algorithm. The first algorithm acts better than the second algorithm, which both the algorithms are better than *Steady Growth* and *Space Partitioning*.

References

1. Zhu, C., Sundaram, G., Snoeyink, J., Mitchel, J.S.B.: Generating random polygons with given vertices. Comput. Geom. Theory Appl. **6**, 277–290 (1996)
2. Auer, T., Held, M.: Heuristics for the generation of random polygons. In: 8th Canadian conference of computational geometry, pp. 38–44 (1996)
3. Dailey, D., Whitfield, D.: Constructing random polygons. In: Proceedings of the 9th ACM SIGITE Conference on Information Technology Education, pp. 119–124. ACM (2008)
4. Christian, S.: Generating random star-shaped polygons. CCCG (1999)
5. Tahat, L. H., et al.: Requirement-based automated black-box test generation. In: Computer software and applications conference, 25th annual international. IEEE (2001)
6. Spillner, A., Linz, T., Schaefer, H.: Software Testing Foundations: A Study Guide for the Certified Tester Exam. Rocky Nook Inc, Santa Barbara (2014)
7. Hughes, J.F., Van Dam, A., Foley, J.D., Feiner, S.K.: Computer Graphics: Principles and Practice. Pearson Education, Upper Saddle River (2014)
8. Crespo, J., Barber, R., Victores, J.G., Jardon, A.: Algorithm for graph visibility obtainment from a map of non-convex polygons. Int. J. Mech. Eng. Robot. Res. **3**(2), 150 (2014)
9. Valveny, E., Delalandre, M., Raveaux, R., Lamiroy, B.: Report on the Symbol Recognition and Spotting Contest. In: Kwon, Y.B., Ogier, J.M. (eds) Graphics Recognition. New Trends and Challenges. Lecture Notes in Computer Science, vol. 7423. Springer, Berlin, Heidelberg (2013)
10. Jha, M.K., McCall, C., Schonfeld, P.: Using GIS, genetic algorithms, and visualization in highway development. ComputerAided Civ. Infrastruct. Eng. **16**(6), 399–414 (2001)

11. Epstein, P., Sack, J.: Generating triangulation at random. ACM Trans. Model. Comput. Simul. **4**(3), 267–278 (1994)
12. O'Rourke, J., Virmani, M.: Generating random polygons. In: Technical report 011,CS Dept., Smith College, Northampton, MA 01063, 3844, (1991)
13. Leeuwen, J.V., Schoone, A.A.: Untangling a travelling salesman tour in the plane. In: 7th conference graph-theoretic concepts in computer science, pp. 87–98 (1982)
14. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer-Verlag TELOS, Santa Clara (2008)
15. Auer, T., Held, M.: RPG-heuristics for the generation of random polygons. In: Proceedings of the 8th Canada Conference on Computational Geometry, pp. 38–44. Ottawa, Canada (1996)