

Generation of Random Triangular Digital Curves Using Combinatorial Techniques

Apurba Sarkar¹(✉), Arindam Biswas², Mousumi Dutt³,
and Arnab Bhattacharya⁴

¹ Department of Computer Science and Technology,
Indian Institute of Engineering Science and Technology, Howrah, India
as.besu@gmail.com

² Department of Information Technology,
Indian Institute of Engineering Science and Technology, Howrah, India
barindam@gmail.com

³ Department of Information Technology,
Indian Institute of Information Technology, Kalyani, India
duttmousumi@gmail.com

⁴ Department of Computer Science and Engineering,
Indian Institute of Technology, Kanpur, India
arnabb@iitk.ac.in

Abstract. This work presents an algorithm to generate simple closed random triangular digital curves of finite length imposed on a background triangular grid. A novel timestamp-based combinatorial technique is incorporated to allow the curve to grow freely without intersecting itself. The algorithm runs in linear time as a fixed set of vertices are consulted to find the next direction and since it does not require backtracking. The proposed algorithm is implemented and tested exhaustively.

Keywords: Random digital curves · Testing of algorithms · Combinatorial technique · Triangular curve

1 Introduction

The problem of generating simple random polygon has attracted lot of researchers not only because of its theoretical interest but also for its application in many areas of computer science. Random polygon has two main areas of application, firstly in testing correctness of geometric and graph algorithms [2, 6, 7] and secondly evaluating efficiency, i.e., estimating consumption of CPU-Time of algorithms that operate on polygons. Sometimes, it is difficult to obtain a large set of practically relevant input data for an algorithm to run on, hence next best choice would be to run the algorithm on random input. To that end an algorithm that generates random polygon can supply random polygon as input to algorithms that expect polygons as input. Random curves may also be useful in graphical applications [4] which intends to generate textures of the nature: like clouds and landforms. In [1, 4, 5, 8, 10, 11] several works on generating random

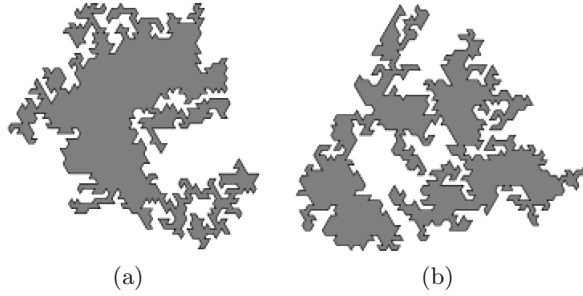


Fig. 1. Two instances of triangular digital random curves with grid length 4

polygon and random curves have been reported. A heuristic approach for generation of simple polygons is presented in [8], which also presents several tests to verify the uniformity of random simple polygon generator. An approach for generating a random x-monotone polygon on a given set of vertices is reported by Zhu et al. [11]. Auer et al. [1] analyzed heuristics to generate simple and star-shaped polygons on a given set of points. Two methods for generating random orthogonal polygon with given set of vertices are presented in [10], one of them uses an inflate-cut technique, whereas the other employs a constraint programming approach and modelled the problem as constraint satisfaction problem. In most of these works, a polygon is generated from a random set of vertices. Two recent work on generating random digital closed curve (4- and 8- connected) is presented in [3] and in [9]. However, to the best of our knowledge, there are no proposed algorithms to generate random closed curves in triangular grid.

A random curve is called triangular when it consists of edges which lie in either \mathbb{L}_0 or \mathbb{L}_{60} or \mathbb{L}_{120} as defined in Definition 1. In this paper, a linear-time algorithm is proposed based on combinatorial technique devoid of any backtracking. The algorithm generates random simple triangular closed curve imposed on the background triangular grid where the resolution of the generated curve are controlled by varying the grid length. A move from a given grid point is chosen randomly from a set of ‘safe’ directions calculated on the basis of the occupancy and orientation of some designated neighbors of the current vertex. Two instances of random triangular curves generated by the proposed algorithm are shown in Fig. 1.

The paper is organized as follows. Required definitions are presented in Sect. 2. The basic principle behind formulation of the combinatorial rules for traversal is discussed in Sect. 3. The method for generation of random digital curves are presented in Sect. 4. Section 5 presents the experimental results with analysis and the conclusion is presented in Sect. 6.

2 Definitions

Definition 1 (*triangular grid*). A triangular grid (henceforth simply referred as grid) $\mathbb{T} := (\mathbb{L}_0, \mathbb{L}_{60}, \mathbb{L}_{120})$ consists of three sets of parallel grid lines, which

are inclined at 0° , 60° , and 120° w.r.t. x-axis. The grid lines in $\mathbb{L}_0, \mathbb{L}_{60}, \mathbb{L}_{120}$ correspond to three distinct coordinates, namely α, β, γ . Three grid lines, one each from $\mathbb{L}_0, \mathbb{L}_{60}, \mathbb{L}_{120}$, intersect at a (real) grid point. The distance between two consecutive grid points along a grid line is termed as grid size, g . A line segment of length g connecting two consecutive grid points on a grid line is called grid edge. A portion of the triangular grid is shown in Fig. 2. It has six distinct regions called sextants, each of which is well-defined by two rays starting from $(0, 0, 0)$. For example, Sextant 1 is defined by the region lying between $\{\beta = \gamma = 0, \alpha \geq 0\}$ and $\{\alpha = \gamma = 0, \beta \geq 0\}$. One of α, β, γ is always 0 in a sextant. For example, $\gamma = 0$ in Sextant 1 and Sextant 4. For a given grid point, p , there are six neighboring UGTs, given by $\{T_i : i = 0, 1, \dots, 5\}$. The three coordinates of p are given by the corresponding moves along a/the shortest path from $(0, 0, 0)$ to p , measured in grid unit. For example, $(1, 2, 0)$ means a unit move along 0° followed by two unit moves along 60° , starting from $(0, 0, 0)$. The grid point p can have six neighbor grid points, whose direction codes are given by $\{d : i = 0, 1, \dots, 5\}$.

Definition 2 (triangular curve). A (finite) closed curve \mathcal{C} imposed on the triangular grid \mathbb{T} is termed as a triangular curve if its sides coincide with lines in $\mathbb{L}_0, \mathbb{L}_{60}$, and \mathbb{L}_{120} . It is represented by the (ordered) sequence of its vertices, which are grid points on the triangular grid. It can also be represented by a sequence of directions and a start vertex. Its interior is defined as the set of points with integer coordinates lying inside it. A triangular curve is said to be simple if it does not intersect itself.

The objective is to generate random simple closed triangular digital curves imposed on the background triangular grid \mathbb{T} where the vertices are grid points.

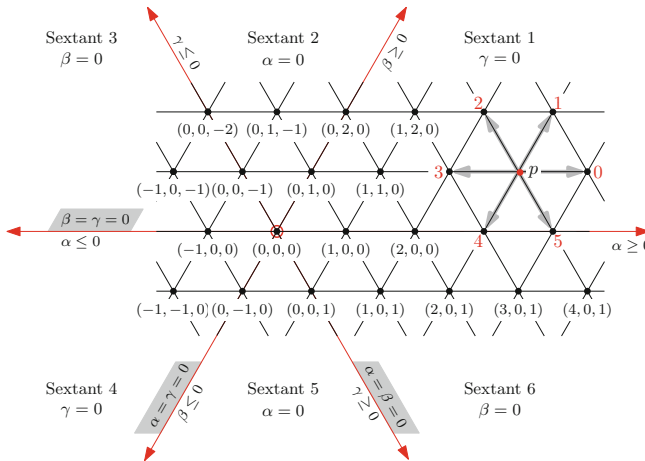


Fig. 2. Portion of a triangular canvas, the UGTs $\{T_0, T_1, \dots, T_5\}$ incident at a grid point p , and the direction codes $\{0, 1, \dots, 5\}$ of neighboring grid points of p

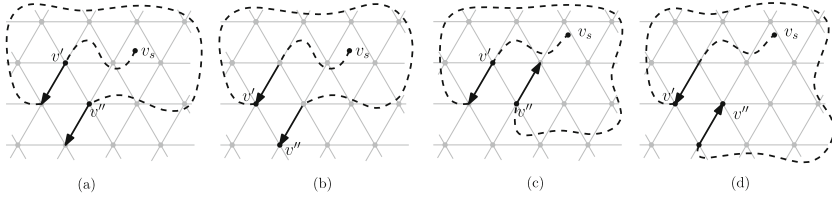


Fig. 3. (a) and (b) curves are in same direction, start of \mathcal{C} is unreachable, (c) and (d) in opposite direction, \mathcal{C} can traverse to its start point

3 Basic Principle

The proposed triangular random digital curve, \mathcal{C} , selects a random direction from a set of “safe” directions, S , calculated from its current position at the grid point $p(i, j, k)$. The set S is computed on the basis of the presence (and orientation) of \mathcal{C} in designated set of vertex $V_d \in N_6(p)$ ¹. The curve grows itself by choosing a random direction from the set S and eventually concludes at the start point, producing a closed non-intersecting curve with its side lying on $\mathbb{L}_0, \mathbb{L}_{60}$, and \mathbb{L}_{120} . To compute S , a combinatorial technique is used which guarantee that \mathcal{C} never moves into a situation where there is no way for it to meet the start vertex without intersecting itself. The proposed algorithm is designed based on the observation that under no condition, either of incoming or outgoing direction of any two grid points on \mathcal{C} , separated by unit grid length will be same as the other. When two points on \mathcal{C} are unit grid length apart, they describe a closed region. The start point may lie inside the closed region or out side of it but in any case if the directions are same, then the \mathcal{C} propagates into a region where the start point does not lie, thus moves into a dead end. As shown in Fig. 3(a) the out going direction at v' and v'' are same and the start point is in the closed region defined by v' and v'' . In Fig. 3(b) the incoming direction at v' and v'' are same and the start point is in the closed region defined by v' and v'' . In either of the cases, the curve can never meet the start point without intersecting itself. On the other hand, if the directions at those two points are different there is always a way for the curve to eventually meet v_s (Fig. 3(c) and (d)). Since there are six possible directions available in triangular grid (Fig. 2), the notion of direction being same or opposite is not so obvious. This is made simpler by considering only two directions namely clockwise and anticlockwise and it is determined by the incoming and outgoing direction at the point p under consideration. Another fact is that when the curve \mathcal{C} is at a particular point p , there could be more than one visited neighbor of p at $N_6(p)$. Out of all those visited neighbors, the one through which \mathcal{C} has entered in $N_6(p)$ is considered to determine S . To find the vertex through which \mathcal{C} has entered in $N_6(p)$, a timestamp t_i is assigned in

¹ Two points p and q are said to be 6-connected in a set S if and only if there exists a sequence $\langle p = p_0, p_1, \dots, p_n = q \rangle \subseteq S$ such that $p_i \in N_6(p_{i-1})$ for $1 \leq i \leq n$. The 6-neighborhood of a point (x, y, z) is given by $N_6(x, y, z) = \{(x', y') : \max(|x - x'|, |y - y'|, |z - z'|) = 1\}$.

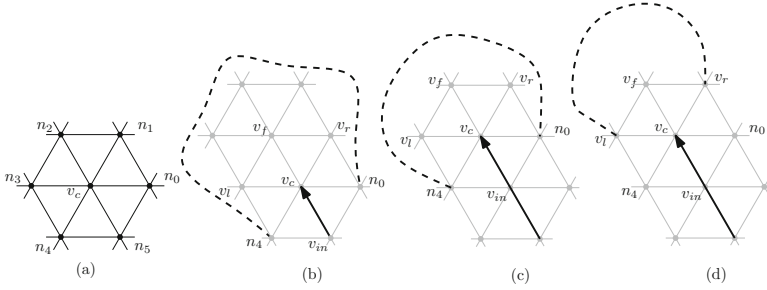


Fig. 4. (a) 6 neighbors (b) trap involving n_4 and n_0 , (c) trap involving n_3 and n_1

increasing order to each vertex v_i as C grows. If v_i is generated earlier than v_j then t_i is less than t_j .

To start with, a random start grid point v_s of the curve is chosen randomly and a timestamp of $t_s = 0$ is assigned to it. The curve grows itself from a given (current) point, v_c , by computing set of safe direction S from v_c based on the occupancy and orientation of C at the earliest visited amongst the designated neighbours of v_c . Then C chooses its next direction of propagation randomly from S , denoted by d_c . As the curve proceeds along d_c to the new grid point from v_c , the timestamp of the new point along d_c is increased by one. The curve repeats this procedure until it reaches to the start point v_s and concludes.

Since random digital triangular curve of finite length is of our interest, the algorithm is designed to execute on a finite canvas which is parallelogram in shape. So, a boundary condition is imposed such that if C takes a clockwise (anticlockwise) turn when it meets the boundary for the first time, then it will take a clockwise (anticlockwise) turn in all subsequent cases whenever it meets the boundary again. The methods for generating triangular digital curves are described in the following section.

4 Generation of Simple Random Triangular Digital Curves

To generate simple random triangular digital curves of finite length following the principle described in Sect. 3, some rules are formulated. This section explains those rules in detail. Time complexity is discussed at the end of this section.

Let the set $N = \{n_0, n_1, \dots, n_5\}$ denotes the six neighbors of a point in triangular grid as shown in Fig. 4(a) and let the neighbor along direction d is n_d . Also, let v_c be the current point, and v_l , v_f , and v_r be the left, front, and right points (Fig. 4(b)). An array \mathbf{b} of size 6 is considered to signify the 6 directions initialized to 1, which means all directions are initially permissible. Also, if a neighbour n_d is already visited then $\mathbf{b}[d]$ is invalidated. However the start vertex is always considered to be not visited, allowing the curve to meet the start vertex whenever it wishes to. The array \mathbf{b} is used throughout to compute the set of

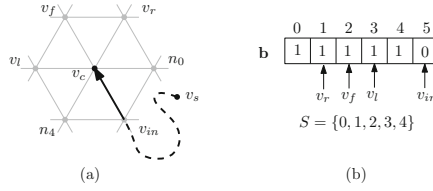


Fig. 5. Case I: (a) None of $\{v_l, v_f, v_r\}$ are visited, (b) Direction array **b**

permissible direction from the current point v_c until the curve concludes at start point v_s .

To generate random triangular digital curves, only the neighbors v_l , v_f , and v_r of v_c are consulted to find S , thus designated set of vertices V_d consist of $\{v_l, v_f, v_r\}$. The reason is to avoid potential traps involving neighbors of the current point, v_c . Without loss of generality, let the curve \mathcal{C} reaches v_c from v_{in} as shown in Fig. 4(b). This means that it has already checked the existence of a possible trap involving either n_4 or v_c or n_0 , whichever is visited earlier, and v_{in} . Now, for a safe move from v_c , possible loop between v_{in} and the earliest visited of $\{v_l, v_f, v_r\}$ need to be checked. Suppose after checking the possible loop described in Fig. 4(b) the curve now reaches to v_c as shown in Fig. 4(c) with the new labels of vertices. Again to make safe move from v_c the curve needs to check possible loop between earliest visited of $\{v_l, v_f, v_r\}$ and v_{in} as shown in Fig. 4(d). Depending on the occupancy and orientation of v_l , v_r , and v_f , there exist three different cases as stated below.

Case I: *None of v_l, v_f and v_r are visited.* The curve can move in any direction except in the direction of the visited neighbors.

For example, in Fig. 5(a), none of $\{v_l, v_f, v_r\}$ are visited. $\mathbf{b}[5]$ is assigned 0 as the curve has come to v_c via n_5 (marked as v_{in} in figure) making it invalid. Thus, the curve can randomly choose one of the directions from $S = \{0, 1, 2, 3, 4\}$ to make further progress, shown as cells with 1 in it (Fig. 5(b)).

Case II: *At least one in $\{v_l, v_f, v_r\}$ is visited. Let v_m from $\{v_l, v_f, v_r\}$ has the lowest timestamp, and v_{m+1} , the grid point next to v_m on \mathcal{C} , lies in $N_6(v_c)$.* For each visited neighbor n_d of v_c , $\mathbf{b}[d]$ is marked as invalid. Then, a traversal is made in **b** from v_m in the direction of v_m to v_{m+1} (the immediate next cell in **b**) upto the farthest visited vertex in $\{v_l, v_f, v_r, v_{in}\} \setminus v_m$ setting all the location encountered in **b** to 0 during the traversal. The indices in **b** with value 1 constitute the set of permissible directions, S .

Figure 6 depicts one such situation in which v_r and v_f is visited, v_r has lower timestamp and its next grid point v_{m+1} (v_f in this case) lies in $N_6(v_c)$. Figure 6(b) shows the initial marking; location $\mathbf{b}[1]$, $\mathbf{b}[2]$ and $\mathbf{b}[5]$ are marked 0 as $v_{in}(= n_5)$, $v_f(= n_2)$ and $v_r(= n_1)$ are already visited. The traversal is made in the direction from v_r to v_{m+1} , in **b** till v_{in} thereby setting $\mathbf{b}[3]=0$ and $\mathbf{b}[4]=0$, others being already set to 0, thus $S = \{0\}$. The array **b** is treated as circular during the traversal. One interesting thing to observe

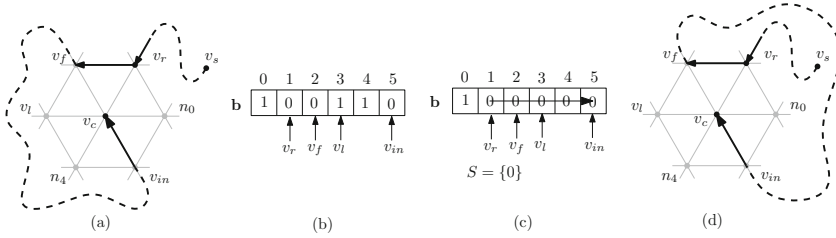


Fig. 6. Illustration of Case II (a) v_r is visited and has lowest timestamp, (b) initial marking of \mathbf{b} (c) final marking of \mathbf{b} (d) v_r is visited: another scenario of (a)

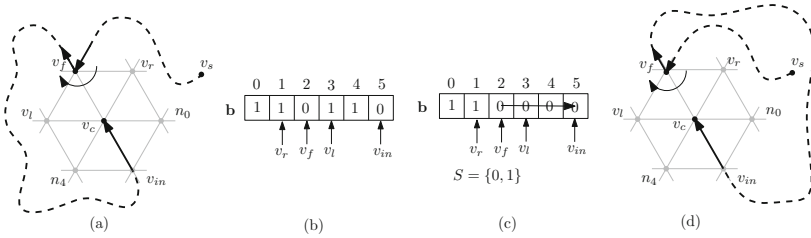


Fig. 7. Illustration of Case III (a) v_f is visited and has clockwise movement (b) Initial marking of \mathbf{b} (c) Final marking of \mathbf{b} (d) another alternative of (a)

is that the curve can traverse to v_c following two paths as shown in Fig. 6 (i) keeps the start point v_s inside (Fig. 6(d)) the closed region defined by v_f and v_c and (ii) keeps the start point v_s outside (Fig. 6(d)) the closed region defined by v_f and v_c . However, in both the cases the only permissible direction is 0.

Case III: *At least one in $\{v_l, v_f, v_r\}$ is visited. Let v_m from $\{v_l, v_f, v_r\}$ has the lowest timestamp, and v_{m+1} , the grid point next to v_m on \mathcal{C} , does not lie in $N_6(v_c)$.* First, the visited neighbours of v_c are marked 0 in \mathbf{b} . Then, a traversal is made from v_m in forward (backward) direction if there is a clockwise (anticlockwise) turn at v_m till the farthest visited vertex in $\{v_l, v_f, v_r, v_{in}\} \setminus v_m$ thereby setting all the location encountered in \mathbf{b} to 0 during the traversal. The indices in \mathbf{b} with value 1 constitute the set of permissible directions, S , from v_c .

In Fig. 7(a) $v_m = v_f$ and $v_{m+1} \notin N_6(v_c)$ and there is an clockwise turn at v_m . Hence, a forward traversal is made from v_f to v_{in} which is the farthest visited vertex in $\{v_l, v_r, v_{in}\}$, as v_l and v_r are not visited at all. Thus, $S = \{0, 1\}$.

Figure 8(a) shows another situation in which all of $\{v_l, v_f, v_r\}$ are visited, v_l has the lowest timestamp and there is an anti-clockwise turn at v_l . Hence, a backward traversal is made from v_l to v_{in} since it is the farthest visited vertex in $\{v_l, v_r, v_{in}\}$ along backward direction. Thus, $S = \{4\}$.

The steps for generating the random curve are captured in the form of pseudo-code as follows

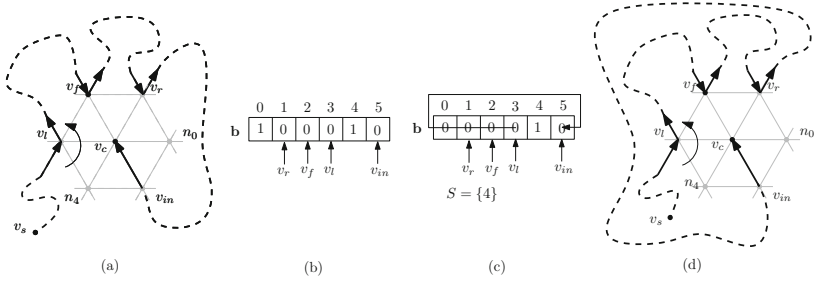


Fig. 8. Another illustration of Case III (a) v_l, v_f, v_r are all visited, v_l has lowest timestamp and has clockwise movement (b) Initial marking of **b** (c) Final marking of **b** (d) another alternative of (a)

Algorithm

1. select the start point v_s randomly;
2. $v_c = v_s, t_s = 0$;
3. **do**
4. compute the set S based on V_d ;
5. randomly select next direction d_c from S ;
6. advance to the next point v_n along d_c ;
7. update v_c i.e
 $t_n = t_c + 1; v_c = v_n$;
8. **while**($v_c \neq v_s$);

where $V_d = \{v_l, v_f, v_r\}$, S is the set of permissible directions, v_c is the current vertex, d_c is incoming direction to v_c , and t_i is the timestamp of the i th vertex.

Time Complexity: Every time the curve progresses it computes the set of safe directions, S , from the current point, v_c . To compute S , a fixed number of vertices (v_l, v_f, v_r) are checked. The number of grid points on the generate curve is of $O(|\mathcal{C}|/g)$ and the computation of S at each grid point requires constant amount of time. Thus, the running time of algorithm is linear in the number of grid points on the length of the generated curve which is given by $O(|\mathcal{C}|/g)$.

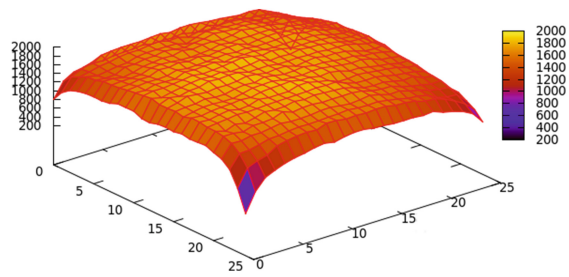


Fig. 9. Test of randomness of the generated curves

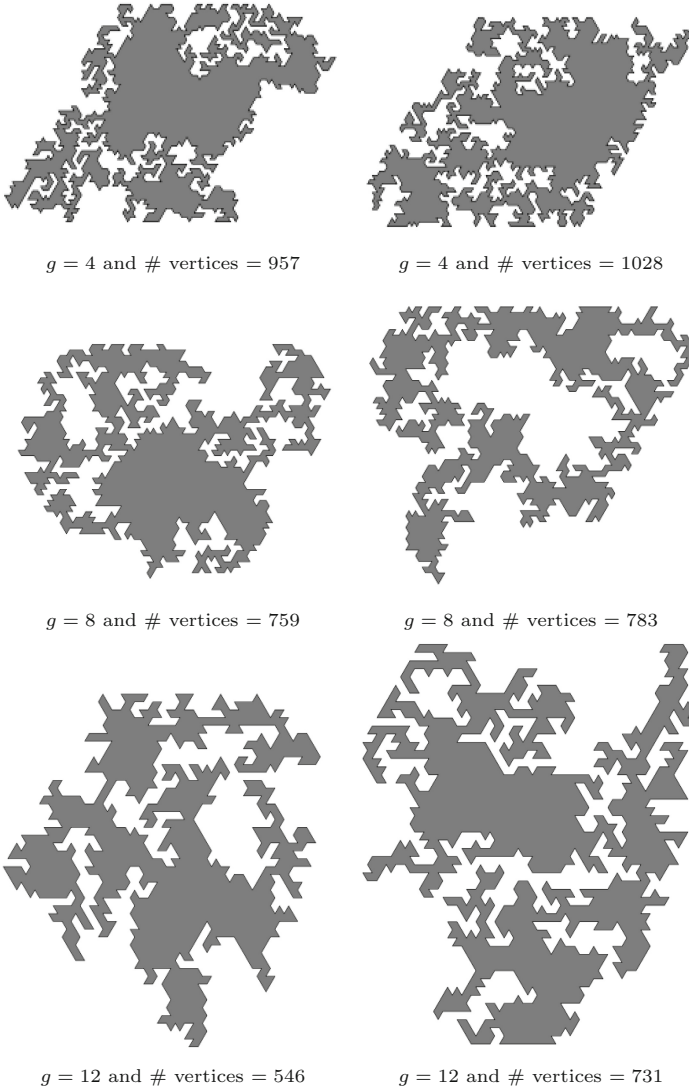


Fig. 10. Instances of triangular digital random curves on a canvas of size 50×50

5 Experimental Results and Analysis

The proposed algorithm is implemented in C in Ubuntu 12.04, 64-bit, kernel version 3.5.0-43-generic, the processor being Intel i5-3570, 3.4 GHz FSB. Six instances of triangular random curves are shown in Fig. 10, first row shows two random curves for grid length, $g = 4$ whereas the second and third rows show two random curves each for $g = 8$ and $g = 12$ respectively. The figures depict

randomness of the curves and also the number of vertices for each curve. The resolution of the curves can be controlled by varying the grid length g .

In order to test the randomness of the algorithm, we have generated 10,000 curves on a fixed canvas size and counted number of times a grid point in the canvas lies on \mathcal{C} . Figure 9 shows the plot with the coordinates of the grid points along the x and y axes whereas the z -axis represents the frequency. It shows that each of the grid points in the canvas has almost equal frequency (flat at the top) indicating the equal probability of occurrences.

6 Conclusions

A combinatorial technique to generate random triangular digital curve is presented in this paper. The algorithm is linear on the length (in number of grid points) of the generated curve and does not require backtracking. The randomness of the generated curve is obvious from the experimental results. In this work, we have used a regular canvas to generate the curve, however, with some modification the algorithm can be used to generate random curve on any arbitrary canvas. The algorithm can further be tuned to generate random paths inside a digital object which can be useful for practical applications.

References

1. Auer, T., Held, M.: Heuristics for the generation of random polygons. In: Proceedings of the Canadian Conference on Computational Geometry, pp. 38–44 (1996)
2. Bhowmick, P., Bhattacharya, B.B.: Fast polygonal approximation of digital curves using relaxed straightness properties. *IEEE Trans. PAMI* **29**(9), 1590–1602 (2007)
3. Bhowmick, P., Pal, O., Klette, R.: Linear-time algorithm for the generation of random digital curves. In: Proceedings of the 2010 Fourth Pacific-Rim Symposium on Image and Video Technology, pp. 168–173 (2010)
4. Dailey, D., Whitfield, D.: Constructing random polygons. In: Proceedings of the 9th ACM SIGITE Conference on Information Technology Education, pp. 119–124 (2008)
5. Epstein, P.: Generating geometric objects at random. Master's thesis, CS Dept., Carleton University, Canada (1992)
6. Klette, R., Rosenfeld, A.: *Digital Geometry: Geometric Methods for Digital Picture Analysis*. Morgan Kaufmann, San Francisco (2004)
7. Rennesson, I., Luc, R., Degli, J.: Segmentation of discrete curves into fuzzy segments. *Electron. Notes Discrete Math.* **12**, 372–383 (2003)
8. Rourke, J., Virmani, M.: Generating random polygons, TR:011. CS Dept. Smith College, Northampton (1991)
9. Sarkar, A., Biswas, A., Dutt, M., Bhattacharya, A.: Generation of random digital curves using combinatorial techniques. In: Ganguly, S., Krishnamurti, R. (eds.) CALDAM 2015. LNCS, vol. 8959, pp. 286–297. Springer, Heidelberg (2015)
10. Tomas, A.P., Bajuelos, A.L.: Generating random orthogonal polygons. In: Conejo, R., Urretavizcaya, M., Pérez-de-la-Cruz, J.-L. (eds.) CAEPIA/TTIA 2003. LNCS (LNAI), vol. 3040, pp. 364–373. Springer, Heidelberg (2004)
11. Zhu, C., Sundaram, G., Snoeyink, J., Mitchell, J.S.B.: Generating random polygons with given vertices. *CGTA* **6**, 277–290 (1996)