An Algorithm to Generate Random Points in Polygon Based on Triangulation

Zhimin Xu$^{1,2,3}$, Li Zhang$^{1,2,3}$ and Rui Ma$^{2,3}$

1 Changjiang Survey, Planning, Design and Research Co., Ltd, Wuhan 430019, China
2 Changjiang Spatial Information Technology Engineering Co., Ltd, Wuhan 430019, China
3 Hubei Province Engineering Technology Research Center for Water Resources Information Perception and Big Data, Wuhan 430019, China
Email: xuzhimin@whu.edu.cn

Abstract. Random point generation plays an important role in assisting random sampling and is widely used in various disciplines. This paper proposes a new algorithm to generate random points in polygon based on Triangulation. Firstly, the polygons are triangulated, and then the number of random points in each triangle is allocated according to the triangle area. Finally, the given number of random points is generated inside the triangle. In the experiment, the results showed that the proposed algorithm is 20 times faster than the algorithm in QGIS software on average. And with the increase of the complexity of the polygon, the advantages of this algorithm are more obvious.

1. Introduction

Random point generation is a basic tool in GIS disciplines. It can generate a specified number of randomly distributed location points within a specified geographic range, and it is important for assisting random sampling. Therefore, random point generation has a wide range of applications in the fields of ecology, forestry, geology, agriculture, meteorology, population, and tourism [1]. Li Xudong studied the theorem of generating evenly distributed random points in ellipses and rectangles, and gave a proof of the theorem and its specific application from a mathematical point of view [2], and proposed an algorithm for generating evenly distributed random points in cuboid space [3]. Cheng Dengbiao improved the traditional problem of uneven distribution of random points generated on a circular surface in the radial direction and added a power P to the random radius factor to make the generated points close to an even distribution [4]. Zhang Haiwen proposed a random point generation algorithm based on polygon rasterization [5]. According to the difference in the spatial relationship between the grid unit and the polygon, random points are generated in the part of the grid unit that falls into the polygon, thereby achieving random point generation within the polygon. In fact, from the perspective of computer graphics, circles, ellipses, and rectangles can be called polygons in graphics drawing. Therefore, the problem of random point generation in any planar shape is ultimately the problem of random point generation in polygons.

This paper proposes an algorithm to generate random points in polygon based on triangulation, and elaborates the principles and key steps of the algorithm, including polygon triangulation, random point allocation, and random point generation within triangles. Compared with the open-source geographic information software QGIS, a comparison experiment was performed to compare the efficiency of the proposed algorithm and QGIS’s algorithm for generating different numbers of random points in polygons of different complexity.
2. Algorithm Flow

The random point generation algorithm based on the triangulation first triangulates the input polygon, then allocates random points proportionally according to the area of each triangle after triangulation, and finally generates a specified number of random points inside each triangle and adjusts the generated random points to meet the specified number of requirements. The algorithm pseudo code is as follows (Table 1):

Table 1. Pseudo Code of the Proposed Algorithm

| Algorithm: | RandomPointsInPolygon |
|------------|-----------------------|
| Input:     | Polygon polygon, int totalNum |
| Output:    | std::vector<Point> vecPts |

1: vecTri ← Triangulation(polygon)
2: totalArea ← vecTri.getTotalArea()
3: For i = 0 to vecTri.size()
4: num ← round(totalNum * vecTri[i].Area / totalArea)
5: pts ← RandomPointsInTriangle(vecTri[i], num)
6: vecPts.push_back(pts)
7: End for
8: AdjustPoints(vecPts)

2.1. Polygon Triangulation

Polygon triangulation is a classic problem in the field of computation geometry. It can decompose any complex polygon into a set of disjoint triangles which fully cover the area of the polygon. The polygon triangulation algorithm is very mature, and it is integrated in many open source software and tool libraries, such as MeshLab, Qhull, PCL, Triangle, CGAL, OSG, etc. In this paper, the class osgUtil::Tessellator in OSG is used to triangulate the input polygon, and then the class osg::TriangleFunctor is used to collect the triangle vertex information of the drawing object.

2.2. Random Point Allocation Based on Area Ratio

As the generated target points are randomly distributed, the larger the triangle area, the greater the probability that the points will fall on the triangle. Therefore, the total number of random points \( N \) generated should be distributed proportionally according to the area of the triangle after triangulation. Let the total area of the polygon is \( S \), and the number of triangles after the triangulation is \( m \). For a triangle of area \( s_i \), the number of points to be generated in the triangle is:

\[
\hat{n}_i = \text{round} \left( \frac{N \times s_i}{S} \right)
\]  

(1)

Where round () is the rounding function.

Due to the rounding operation, \( \sum_{i=1}^{m} \hat{n}_i \) does not necessarily equal \( N \). Therefore, after the random points are generated, a random point adjustment operation is required.

1. If \( \sum_{i=1}^{m} \hat{n}_i = N + 1 \), there are only two triangles with the same area. Delete a point randomly.
2. If \( \sum_{i=1}^{m} \hat{n}_i = N - 1 \), a triangle is randomly selected to add a point in its interior.

2.3. Random Point Generation in Triangle

After triangulation and random point allocation based on area ratio, the problem of generating random points in a triangle is transformed into a problem of randomly generating a specified number of random points in a triangle. As shown in the Figure 1, Let P be a point within triangle \( ABC \), and make parallel lines from point \( P \) to sides \( AB \) and \( AC \), intersecting at points \( E \) and \( F \) respectively. Let \( AE/AB=s \), \( AF/AC=t \), then \( 0<s<1 \), \( 0<t<1 \). Meanwhile,

\[
\overrightarrow{AP} = s \cdot \overrightarrow{AB} + t \cdot \overrightarrow{AC}
\]

(2)
Then,
\[ \overrightarrow{OP} = s \cdot (\overrightarrow{OB} - \overrightarrow{OA}) + t \cdot (\overrightarrow{OC} - \overrightarrow{OA}) \] (3)
So,
\[ \overrightarrow{OP} = (1 - s - t) \cdot \overrightarrow{OA} + s \cdot \overrightarrow{OB} + t \cdot \overrightarrow{OC} \] (4)

![Figure 1. Schematic Diagram of Random Points in a Triangle](image)

When point P is on side BC, \( s = \frac{EP}{AC} \), so \( s + t < 1 \). Therefore, to limit the point P to the triangle, \( s + t < 1 \) must also be satisfied. In fact, when \( s + t > 1 \), let \( s' = 1 - s \) and \( t' = 1 - t \) to be the new coefficients, then \( s' + t' = 2 - s - t < 1 \), which still meets the condition that point P is within the triangle ABC. Therefore, to generate random points in a specified number of triangles, simply perform a loop, call the random number generation function \( \text{rand}(0,1) \) to generate the coefficients \( s \) and \( t \), and substitute it into the formula (4) for calculation. The algorithm pseudo code is as follows (Table 2):

**Table 2. Pseudo Code to Generate Random Points in Triangle**

| Algorithm:          | RandomPointsTriangle       |
|---------------------|---------------------------|
| Input:              | Triangle\( \text{tri} \), int\( \text{n} \) |
| Output:             | std::vector<Point>\( \text{pts} \) |
| 1:                   | \textbf{For} \( i = 1 \) \textbf{to} \( n \) |
| 2:                   | \( s, t \leftarrow \text{rand}(0,1) \) |
| 3:                   | \textbf{If} \( s + t > 1 \) \textbf{then} |
| 4:                   | \( s \leftarrow 1 - s \) |
| 5:                   | \( t \leftarrow 1 - t \) |
| 6:                   | \textbf{End if} |
| 7:                   | \( \text{point} \leftarrow \text{tri.p1 * (1-s-t) + tri.p2 * s + tri.p3 * t} \) |
| 8:                   | \( \text{pts.push_back(point)} \) |
| 9:                   | \textbf{End for} |

3. Random Point Generation Algorithm in QGIS

In order to compare the efficiency of the algorithms proposed in this paper, the random point generation algorithm of QGIS software is selected as a reference. QGIS (Quantum GIS) is an internationally renowned open source, cross-platform geographic information system software [6]. It has the characteristics of friendly interface, high operating efficiency, and active development community. QGIS software also provides a tool for generating random points within a polygon. By reading its source code, it can be seen that the implementation process is as follows: (1) Get the enveloping rectangle of the polygon; (2) Randomly generate a point within the enveloping rectangle; (3) If the point is inside the polygon, save the point; (4) Repeat steps (2) and (3) until the number of saved points reaches the specified amount. In step 3, QGIS uses the GEOS library to judge whether a point is within the polygon. The algorithm pseudo code is as follows (Table 3).
Table 3. Pseudo Code of Random Point Generation Algorithm in QGIS

| Algorithm: | RandomPointsQGIS |
|-----------|------------------|
| Input:    | Polygon polygon, int totalNum |
| Output:   | std::vector<Point> vecPts |
| 1:        | num ← 0 |
| 2:        | While num < totalNum |
| 3:        | e ← polygon.getEnvelop(); |
| 4:        | s, t ← rand(0, 1) |
| 5:        | pt ← Point(e.xmin + s * e.wid, e.ymin + t * env.hei) |
| 6:        | If polygon.Contains(pt) |
| 7:        | vecPts.push_back(pt) |
| 8:        | num ← num + 1 |
| 9:        | End if |
| 10:       | End while |

4. Experiment and Analysis

In the experimental part, we use the algorithm in this paper and the QGIS random point algorithm to generate different numbers of random points (including: 20, 100, 400, 2000) in five polygons with different complexity. Figure 2 shows the five polygons and the effect after generating 2000 random points using the algorithm in this paper. It can be seen from the figure that the results of the random point generation algorithm proposed in this paper meet the expectations.

![Figure 2. Results of 2000 Random Points in Different Polygon](image)

Further statistics are performed on the key performance indicators of the algorithm in this paper and the QGIS algorithm, including the number of iterations, triangulation time, total time consumption, and the corresponding average values. The statistical results are shown in Table 4 and Table 5.

Table 4. Experimental Results of QGIS Random Point Generation Algorithm

| Polygon Num. | 20   | 100  | 400  | 2000 | Average   |
|--------------|------|------|------|------|-----------|
|              | Iterations | Time spent /μs | Iterations | Time spent /μs | Iterations | Time spent /μs | Iterations | Time spent /μs | Iterations | Time spent /μs | Iterations | Time spent /μs |
| 1            | 24   | 5.06 | 125  | 14.96 | 512      | 59.94      | 2558      | 304.79      | 0.14       | 96.19       |
| 2            | 38   | 6.18 | 158  | 22.09 | 685      | 99.00      | 3304      | 481.86      | 0.15       | 152.28      |
| 3            | 36   | 17.52| 171  | 68.75 | 765      | 335.79     | 3829      | 1814.42     | 0.45       | 559.12      |
| 4            | 32   | 27.12| 181  | 162.31| 735      | 706.76     | 3999      | 4145.76     | 0.94       | 1260.49     |
| 5            | 43   | 87.73| 240  | 527.55| 909      | 2097.31    | 4686      | 11375.20    | 2.24       | 3521.95     |
Table 5. Experimental Results of the Proposed Algorithm

| Polygon Num | 20  | 100 | 400 | 2000 | Average |
|-------------|-----|-----|-----|------|---------|
|             | Tri. Time spent (μs) | Time spent (μs) | Tri. Time spent (μs) | Time spent (μs) | Tri. Time spent (μs) | Time spent (μs) | Tri. Time spent (μs) | Time spent (μs) | Tri. Time spent (μs) | Time spent (μs) |
| 1           | 0.13 | 0.56 | 0.10 | 1.09 | 0.11 | 2.92 | 0.12 | 13.78 | 0.12 | 4.59 |
| 2           | 0.40 | 0.64 | 0.32 | 1.15 | 0.32 | 3.17 | 0.34 | 13.99 | 0.34 | 4.74 |
| 3           | 0.58 | 0.83 | 0.58 | 1.50 | 0.58 | 3.62 | 0.62 | 14.41 | 0.59 | 5.09 |
| 4           | 0.78 | 1.04 | 0.79 | 1.95 | 0.79 | 3.97 | 0.81 | 14.88 | 0.80 | 5.46 |
| 5           | 1.32 | 1.55 | 1.30 | 2.56 | 1.30 | 5.55 | 1.32 | 15.98 | 1.31 | 6.41 |

In fact, analysis of the principle of the QGIS random point generation algorithm shows that although the algorithm has a clear idea and simple logic, there are two problems:

1. The algorithm needs to generate random points in the polygon-enclosed rectangle. As the polygon depressions and holes increase, the probability of random points falling within the polygon will decrease, and the number of iterations of the algorithm will increase.

2. Determining whether a point is inside a polygon is the most complicated step in each iteration. Therefore, as the number of iterations increases, the time consumed by the algorithm increases dramatically.

In contrast, the algorithm in this paper effectively avoids the above problems:

1. The polygon triangulation algorithm with a higher complexity in the algorithm is called only once during a random point generation process, and is not affected by the increase in the number of random points;

2. In the main loop iteration steps, each step generates valid points, and there is no invalid iteration; at the same time, the complexity of the random point generation algorithm in the triangle is not high.

Comparing the average of the total time spent in each group, the efficiency of the algorithm in this paper to generate random points in five groups of polygons with different degrees of complexity is 20.96 times, 32.13 times, 109.86 times, 230.75 times, and 549.51 times the QGIS algorithm. Further, the correlation analysis between the number of iterations, the number of generation points, and the algorithm time spent of the two algorithms in polygons with different complexity is shown in Figure 3. It can be seen that as the complexity of the polygon increases, the time consumption of the QGIS algorithm increases faster, while the efficiency of the algorithm in this paper is not significantly affected by the complexity of the polygon. This result also confirms the advantages of the algorithm summarized in the above analysis.

Figure 3. Diagram of Correlation Analysis
5. Conclusions
This paper proposes a random point generation algorithm based on triangulation, and elaborates the key algorithm steps of polygon triangulation, area-based random point allocation, and random point generation within triangle. The efficiency of the algorithm in this paper is compared with the well-known open source software QGIS random point algorithm from the aspects of algorithm principle and experimental results. The results show that the triangle-based random point generation algorithm has obvious efficiency advantages compared with the QGIS algorithm and has great practical value in production practice.

6. Acknowledgments
This work was supported by the National key technologies Research & Development program (2017YFC1502601).

7. References
[1] Chang Kang-tsung. Introduction to Geographic Information Systems (3rd Edition) [M]. Beijing: Tsinghua University Press, 2009: 12-13.
[2] Li, X.D., Zhao, X. J.: Theorem on generating stochastic points uniformly distributed on the cuboid region and its application [J]. Journal of Xihua University · Natural Science, 2011, (05):32-33+42.
[3] Li Xudong, Zhao Xuejiao. Theorems of uniformly distributed stochastic points in rectangles and ellipse and their application [J]. Journal of Chengdu University of Technology (Science & Technology Edition), 2012, (05):555-558.
[4] Cheng Dengbiao. On Improvement and Application of Generating Non-uniformly Distributed Random Points on A Round Face [J]. Journal of Mianyang Normal University, 2013, (08):1-8.
[5] Zhang Haiwen, Shi Liangping, Hao Yangmei. Study on Algorithm for Generating Random Points in Dot Density Map [J]. Computer and Modernization, 2010, (08):22-24.
[6] QGIS.qgspointsample.cpp[EB/OL]. http://www.qgis.org/api/qgspointsample_8cpp_source.html, 2017–08–15.