Constructing random polygons

Conference Paper · January 2008

DOI: 10.1145/1414558.1414592 · Source: DBLP

Deborah Whitfield
Slippery Rock University of Pennsylvania
22 PUBLICATIONS **333** CITATIONS

Constructing Random Polygons

David Dailey
Slippery Rock University
Computer Science Dept
Slippery Rock, PA 16057
00 1 724-738-2145
david.dailey@sru.edu

Deborah Whitfield
Slippery Rock University
Computer Science Dept
Slippery Rock, PA 16057
00 1 724-738-2935
deborah.whitfield@sru.edu

ABSTRACT

The construction of random polygons is used in psychological research and for the testing of algorithms. With the increased popularity of client-side vector based graphics in the web browser such as seen in Flash and SVG, as well as the newly introduced <canvas> tag in HTML5.0, the use of random shapes for creation of scenes for animation and interactive art requires the construction of random polygons. Generating random polygons in a computationally efficient manner is important particularly in a resource limited environment such as the web browser. This paper presents a random polygon algorithm (RPA) that generates polygons that are random and representative of the class of all n-gons in O(n^2logn) time. Our algorithm differs from other approaches in that the vertices are generated randomly, the algorithm is inclusive (i.e., each polygon has a nonzero probability to be constructed), and it runs efficiently in polynomial time

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems Geometrical problems and computations.

General Terms: Algorithms.

Keywords

polygon, random, n-gon, algorithm, computational geometry

1. INTRODUCTION

With the increased popularity of client-side vector based graphics in the web browser such as seen in Flash and SVG [17], as well as the newly introduced <canvas> tag in HTML5.0 [16], the use of random shapes for creation of scenes for animation and interactive art requires the construction of random polygons. Client-side IT applications use graphics and artwork as backgrounds, eye candy, or as meaningful imagery such as clouds, mountains, and icebergs. Web page design can be more appealing to the user if the artwork is "random" (i.e., the artwork is different every time the user visits the page). Random polygon construction can be seen as useful in a wide range of graphical applications that seek to mimic the textures of nature: clouds, land formations, biological or hydrological phenomena. The applications to both natural science and entertainment are natural to consider

The construction of random polygons is an application used for testing the correctness of algorithms [2]. As Martin Held writes [10] "For testing the correctness of an algorithm, the goal is to generate a diverse set of input data such that all branches of the algorithm will be executed with a high probability. "Hence, it is important to ensure that a technique of generating random instances of problems is in fact random and inclusive.

Generating random polygons (probability of 1/k where k is the number of simple polygonalizations is related to the Simple Polygonalizations Problem 16 [13]. The open NP problem is stated as "Can the number of simple polygonalizations of a set of *n* points in the plane be computed in polynomial time? A *simple polygonalizations* is a simple polygon whose vertices are the points". Heuristics have been developed [2] for generating polygons. This paper describes the Random Polygon Algorithm (RPA) algorithm that generates random polygons from randomly generated points in O(n^2logn). The algorithm is inclusive in that it can generate any polygon and the algorithm is fair and representative in that each polygon has a probability of 1/k to be generated.

1.1 Related Work

A series of experiments on human perception of shapes conducted in the 1950's and 1960's was summarized by Leonard Zusne [19]. In order to experiment with humans' perceptions of shapes, several techniques for measuring shapes as well as for generating random shapes were considered. Random shapes are needed to ensure that the results of experiments are statistically generalizable to the population of all shapes.

One such method, as illustrated in Table 1, involved the following construction: start with a single point. Given a positive integer, n, a collection of n pairs of the form P_i=(alpha$_i$, d_i) is generated in which $0 \leq$ alpha$_i$ $< 2\pi$ represents an angle, and d_i represents a distance from the origin. The numbers alpha$_i$ and d_i are generated randomly. The angles alpha$_i$ are then sorted (in say, clockwise, order) and a polygon is constructed by connecting the points in a clockwise fashion.

Table 1. Construction of a random ngon

Point i	alpha$_i$	d_i
0	5.5	5.5
1	5.3	7.2
2	5.2	3.6
3	5.0	5.0
4	3.9	4.9
5	3.0	4.8
6	2.6	5.0
7	0.3	9.1
8	0.2	8.3

This method results in the star-shaped polygon shown in figure 1 and discussed later in this paper. It should be noted that not all polygons can be constructed using the method illustrated in Table 1

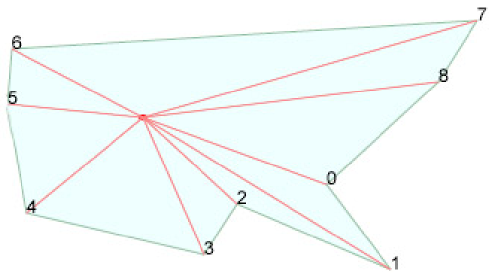

Figure 1. Star shaped polygon of Table 1

.

The polygons constructed are those which, in the terminology of the museum lighting problem[12], can have their edges illuminated (via line-of-sight) by a single lamp placed on the interior of the polygon. Not all polygons can be illuminated with a single lamp. The one in Figure 2 requires three lights. Some require arbitrarily many inside lights. Hence, one is forced to think of other ways to proceed. Chvatal [5] demonstrated in the early 1970's that while all n-gons can be lit with n/3 lights,with the maximum being required in some cases.

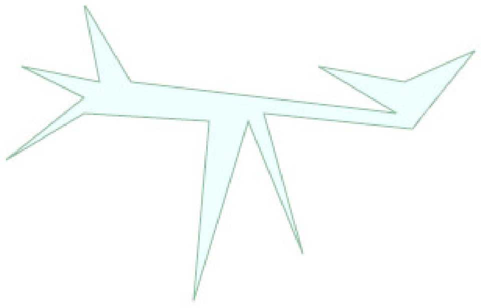

Figure 2. A polygon requiring three lights

It is unknown if counting the number of simple polygons that can be drawn without crossing lines from a set of n points can be computed in polynomial time [13]. Generating a random simple polygon from a set of vertices is a similar problem in computational geometry where researchers have found heuristic methods [2, 15, 18]. Each of these heuristic methods generates polygons that have certain commonalities or idiosyncrasies For example, RPG's implementation of Held's [2] algorithms generate x-monotone(those for which y=f(x) is functional as seen from one point's perspective), star-shaped, and general simple polygons. The ConvexBottom [15] algorithm constructs polygons with a boat-like bottom in $O(n \log n)$ time and the SteadyGrowth algorithm [2] operates in $O(n^2)$ time and uses multiple heuristics that begin with a set of vertices. The 2OptMove [2] algorithm does generate all possible polygons from a given set of vertices in $O(n^4)$ time. Our algorithm does not rely on a given set of vertices, but rather generates the vertices randomly, and does so more efficiently.

The star-shaped polygons discussed earlier are one example of a group of polygons that can be generated in the Random Polygon Generator tool [2]. Star-shaped n-gons can be generated by randomly generating n radii ($0 <= r_i < 360$), sorting the radii, and then generating n lengths ($0 <= l_i <= r_i$,). By "connecting the dots", a star-shaped n-gon is created (recall Table 1). This computationally quite simple method, fails the inclusiveness requirement, since not all n-gons may be constructed.

Another technique to generate all n-gons, would be to position n/3 points (or lights as in the museum lighting problem [12]) randomly in the plane, and then construct a star-shaped polygon from each point. The stars would then be stitched together by removing an edge in each star and connecting the stars. A technique to connect the n/3 stars, although *doable* appears computationally complex since the center of each star was selected randomly (i.e., stars may overlap). The polynomial-time algorithm of Chazelle and Edelsbrunner [4] can be used to determine the edges that intersect. But merely knowing which edges intersect may still leave us with a large number of intersecting edges, hence leaving another polygonal tracing problem almost as large as the first – that is we have not really solved the problem.

Another polygon generation technique utilizes casting. Imagine casting a random 3-dimensional object into a rigid mold and then magically removing the mold without breaking or cutting the mold. Now, randomly slice the mold at any angle. The sliced view would result in at least one random n-gon (consider the corkscrew and its cast; certain cross sections of the cast result in multiple disjoint regions.) However, the n-gon would have a flat bottom. This technique is similar to the one implemented as the ConvexBottom [15] algorithm which constructs polygons in O(n log n) time, but is also representative. Separating the object from its cast [1] is an interesting problem for many reasons. For example, rotations may not allow appendages that are not visible via line of sight to be extracted without breaking. Although two flat bottomed regions may be adjoined and fused along a shared flat bottom, the resultant single-seamed objects are still not representative of all polygons.

A popular approach for connecting n points consists of finding a shortest path through the n points. This well-known traveling salesman approach is NP complete. Once the path is created, then simply connect the start to the finish. Such an approach applied to our problem, would not necessarily produce a simple polygon (there may be crossing edges). While certain variations on the shortest path problems can be solved in polynomial time [7, 14], none appears to produce n-gons as final output.

There are many techniques for solving the well-known convex hull problem including the Graham scan in O (n log n)[9]. Consider the onion skin problem where concentric convex hulls are created from n points (at most n/3 hulls would be created). Randomly select an edge from each hull to remove and stitch the hulls together. However, the number of possible stitchings is large, and since the number of points per hull is not limited to three, the systematic enumeration of the interweaving of these hulls which would seem to be a prerequisite for a representative sample of the possibilities appears to be a complex problem.

The classic Koch curve (figure 3), which is used as an example of a simple fractal curve with finite area and infinite length, is an intriguing way to make an arbitrarily complex polygons [11]. The technique for drawing the Koch curve technique begins with an equilateral triangle and at each recursive step, one trisects each line segment (of length L) and replaces it with a new equilateral triangle, pointing outward from the interior, of length L/3.

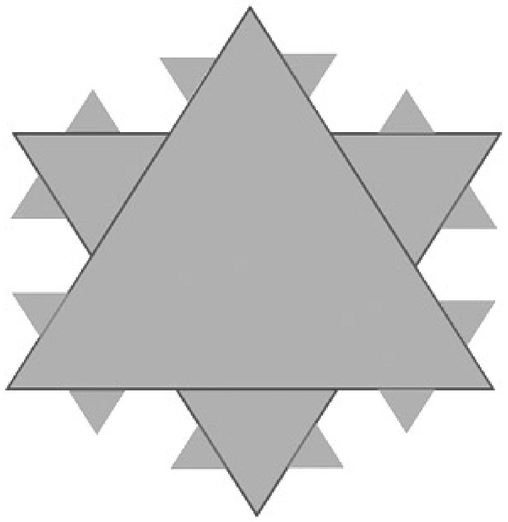

Figure 3. The Koch curve after three generations

The approach presented in this paper for constructing random polygons utilizes this concept. This involves constructing a polygon of n sides by successively choosing one side at a time and subdividing it into two new sides, maintaining care that the two new sides do not intersect any existing edges.

An algorithm that generates closed random polygons must be inclusive. There should be no polygon which cannot be constructed using the method. Alternatively, each polygon should have a nonzero probability of being constructed.
An approach to randomly generating an n-gon in polynomial time is introduced in section 2. The algorithm is followed by an analysis in section 3 and conclusions in section 4.

2. Random Polygon Algorithm

The Random Polygon Algorithm (RPA) introduced in this paper generates a random n-gon in polynomial time. Unlike some approaches to the problem, the polygon is not constructed by starting with n points as input, but instead, the points are generated at random as the polygon is constructed. In addition, the algorithm obeys the stipulations of section 1.2, that it be random and representative of the class of all n-gons.

The algorithm begins by constructing a random triangle. Three random (x,y) points are generated and a polygon P of size three is constructed. Another point is added to P by:

1) determining the region visible to the randomly selected line segment,

2) dividing the visible polygonal region into triangles, and

3) fairly selecting a point from within one triangle as a new vertex to P.

More precisely, the RPA

Step 1. Generate 3 random points (that are not collinear) and draw a polygon P of size 3.

Step 2. FOR n-3 times

a. Select a line segment of P at random with endpoints at vertices V_a and V_b.

b. Determine the visible region of P as described in section 3.1 below. A region visible to both V_a and V_b will be found. This region can be viewed itself as a polygon P'.

c. Randomly select a new point, V_c, within P' by subdividing P' (the visible region) into triangles as described in section 3.2 below.

d. Add V_c to P' by adding the line segments (V_a, V_c) and (V_b, V_c). Overwrite P with P'

2.1 Determining the Visible Region

The goal of this portion of the algorithm is to create a polygon P' that represents the region that is visible (namely reachable by a line without crossing any existing lines) to the two endpoints (V_a and V_b) of the line segment that was removed. Each line segment of P is examined to determine if a portion of P is visible to V_a and V_b. The visible portion is added to P'.

It is assumed that the vertices are stored in a sorted linked list format and the V.next and V.prev are accessible. The vertices are sorted so as to form the n-gon.

a. Set P' initially to be all the vertices in P .

b. Extend, in both directions, the segment from V_a to V_b. The two new line segments will either hit the border or intersect an existing line in P. Where the lines intersect, insert the two new vertices to the ordered list of vertices in P'.

c. Beginning with V_a.next (the vertex adjacent to V_a), continue around all n vertices in P' determining the vertices that are visible to both V_a and V_b. Maintain the lastVisible node (initially set to V_a). For each V_i visited in P', determine if V_i is visible to both V_a and V_b.

If V_i is not visible to both, delete V_i from P'

If it is visible to both, check if V_i.prev an element of P (the n-gon) was visible to both.

Case 1: V_i.prev is visible to both V_a and V_b

Set lastVisible to V_i. Continue with the march.

Case 2: V_i.prev is not visible to either V_a or V_b

Draw the line (V_a, lastVisible) until it intersects P. If the intersection point is visible to both V_a and V_b, then add the intersection point to P'.

Draw the line (V_b, lastVisible) until it intersects P. If the intersection point is visible to both V_a and V_b, then add the intersection point to P'.

Draw the line (V_a, V_i) until it intersects P. If the intersection point is visible to both V_a and V_b, then add the intersection point to P'.

Draw the line (V_b, V_i) until it intersects P. If the intersection point is visible to both V_a and V_b, then add the intersection point to P'.

(Note: at most two of the four intersection points above will be added to P' and two original points of P were removed. P' does not grow, thus there is no additional overhead incurred)

Set lastVisible to V_i. Continue with the march

Case 3, 4: V_i.prev is visible to one of V_a or V_b

Without loss of generality, select either V_a or V_b (call it V_x) that failed the visibility test with V_i.prev.

Draw the line (V_x, V_i) until it intersects P. If the intersection point is visible to both V_a and V_b, then add the intersection point to P'.

Draw the line (V_x, lastVisible) it intersects P. If the intersection point is visible to both V_a and V_b, then add the intersection point to P'.

Set lastVisible to intersection point. Continue with the march.

2.2 Point Selection

This section of the algorithm ensures that there is no bias in the point selected and thus helps to ensure that stipulation b (fair and representative polygons) is met.

a. Begin by dividing P', the visible region, into triangles. There are several techniques in the literature that can be used Epstein's [8] technique runs in $O(n^4)$, whereas Clarkson et. al.'s [6] runs in O(n log n) and Chazelle's algorithm [3] is linear, but difficult to implement.

b. Select one of the T_m triangles using a weighted formula based on the area of each T_i so that each point in P' has an equal probability of being selected.

i. Calculate the area of P' and each triangle(T_i) of P'.

ii. Weight each triangle (W_i) such that W_i = Area(T_i) / Area of P'

iii. Select a triangle with probability directly proportional to its weight.

c. Select a point within the interior of T_i at random and with equal probability. Fold a copy of the triangle onto itself forming a parallelogram, which can then have one of its slanted portions adjoined to the opposite side to make a rectangle. Once a rectangle is formed then generate a point randomly and uniformly over the range of x and y. If the point (x, y) is not within T_i, invert (x,y). Call this point V_c

2.3 Example

Assume an n-gon of size 22 has already be created by the algorithm as seen in Figure 4. To extend the n-gon to size 23, an edge has been randomly selected (edge V_a, V_b in Figure 4).

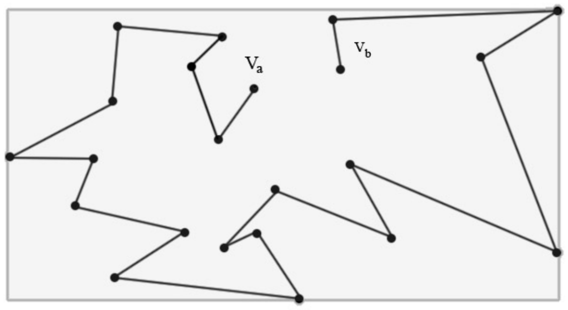

Figure 4. N gon of size 22

To determine the region visible to both V_a and V_b, line segments that extend V_a, V_b are drawn until they intersect with an existing line segment of the 22-gon (see Figure 5.)

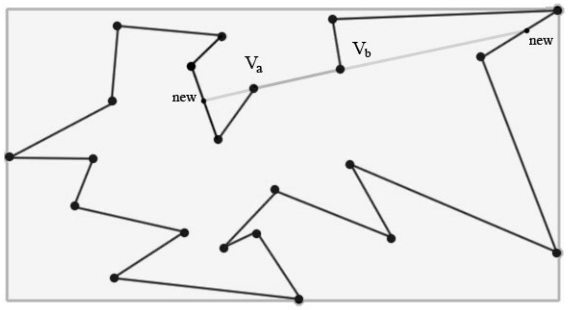

Figure 5. Extending the line segment

From V_a, begin a march around the n-gon. In Figure 6, the vertices that are visible to both V_a and to V_b (1, 9, 12, 14, 16, 17, 18) will cause one of the four cases of Step c to be performed. Notice that vertices 2, 3, and 20 are also "visible", from V_a and V_b, but that this visibility is from outside. (Note that a simple intersection test with the removed segment V_a, V_b is all that is required.) Exterior visible vertices must be walked separately and those vertices may include the bounding box if appropriate for the application. Vertex 1 is visible from both V_a and V_b thus it is included in P'. Vertices 2 through 8 are discarded as they are not visible from both V_a and V_b. Vertex 9 is visible to both vertices, but vertex 8 was not visible from V_a. In Case 4, a line is drawn from V_a to vertex 1 (the last visible vertex) until it intersects an edge (from vertex 8 to vertex 9). This intersection point is visible to V_b, and thus this new vertex is added to P'. Another line is drawn from V_a to vertex 9.

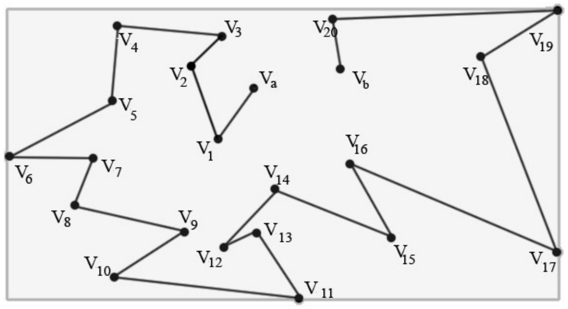

Figure 6. Walking the N-gon

Notice that neither vertex 10 nor 11 is visible to V_a and V_b since vertices 9 and 12 are obstructing their view. So, vertices 10 and 11 are removed from the visible region, P'. When vertex 12 is encountered, the most interesting case, case 2, is performed. The four line segments as seen in Figure 7 are drawn. The four intersections (u_1, u_2, u_3, and u_4) are determined. Only u_2 and u_3 are visible from both V_a and V_b, so these new points are added to the visible region, P'. Notice that P' can never have more edges than the original P as new vertices are added to P' only when other vertices have been removed.

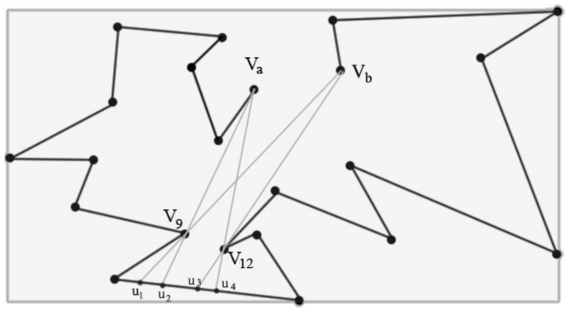

Figure 7. The partial edge

After the march is complete, the visible region has been determined and triangularization of P' is performed.

3. ANALYSIS

The run time analysis is given here followed by proofs of the algorithms inclusiveness and fairness.

3.1 Run-Time Analysis

Step 1 of the algorithm is obviously constant. Step 2 of the algorithm is repeated n times and thus will be bounded by the greater of step 2b (determining the visible region) and step 2c (point selection) as steps 2a and 2d are both constant.

Determining the visible region in step 2b of the algorithm requires numerous intersection and visibility tests. The intersection test is at worst O(n) since the test involves determining whether a point intersects any of the n line segments. Finding the nearest intersection point can be performed by calculating the distances and determining the smallest (still O(n).) The visibility tests involve determining if there are any line segments obstructing the view between the two points in question. Again, n intersection tests are performed in O(n) time. Both steps a and b of determining the visible region are at most O(n). Step c however is repeated n times. Step c contains four cases. Each case utilizes an initial visibility test (O(n).) Case 2 is the most costly; four intersections and eight visibility tests (O (12n).) Thus the worst case running time of determining the visible region is $O(n^2)$.

In step a of point selection, triangulation can be completed in O(n log n) using Clarkson et al technique [6]. Step b, triangle selection is linear in the number of triangles which at most is n/3. Step c, point selection, is constant. Thus, the running time of point selection is bounded by the running time of triangulation. The worst case point selection running time is O(n log n).

Thus, the overall worst case running time of the algorithm is $O(n^3)$. Upon closer inspection, one notices that since step 2 (the most costly) only has $O(n^2)$ operations when adding the nth vertex to the polygon. The march of part c is first performed 4 times, then 5 times,…, then finally n times (overall the sum from 4 to n or n(n+1)/2).

3.2 Fairness Consideration

Many other approaches to finding the next point to add to P were considered by the authors before settling on the one described in section 2.2. As each approach was considered, it was determined that the approach showed some type of favoritism. For example, instead of first triangulating to find a random point inside a polygon an undergraduate student (Mr. Stephan Browarny), suggested the following technique: generate a point (x, y). If it is inside the polygon stop. Otherwise choose randomly from among all lines that pass through both (x,y) and P. Complete this line until it contacts the bounding box on either side of (x,y). The points on the resulting line are then easily partitioned into those inside or outside the polygon. Choose one at random from the interior. This technique, however, shows a bias towards selecting a point close to the center of P.

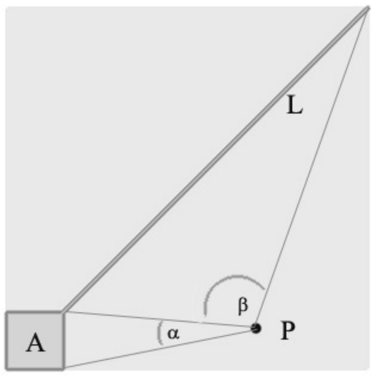

Figure 8. Bias in point selection

Consider polygon Q in Figure 8 that is the union of the square, A, with the diagonally tilted rectangle L (which is a thick line). Since the width of L may be made arbitrarily small, we may construct Q such that the ratio of the area of A to that of L is arbitrarily large. That is, we may make Q so that A contains almost all points of Q, hence, by the fairness consideration, the probability of choosing a point in A may be made arbitrarily close to 1.0. However consider a typical point P. For *almost all* points P, inside the bounding rectangle around Q, it is easy to verify that L subtends a larger angle from P than does A – implying that the probability of choosing a line which intersects L is in fact higher than the likelihood of choosing a line that intersects A. That is, the technique will over-represent points within L. Thus, there is bias towards the center and the techniques is not representative.

3.3 Inclusiveness

It now remains to demonstrate that the polygons generated by RPA are representative of the class of all polygons – i.e., that any polygon may be generated via this method. This follows from the following observations:

Suppose we have an n-gon P. Let the N vertices of the n-gon be partitioned into the concave and convex sets, V and X respectively where |N|=|V|+|X| . Clearly |X| > 2 .

If we can demonstrate that there are always three adjacent vertices m, n and o in N where *mn* and *no* are edges in the polygon in which m and o are mutually visible, then we are certain that P can be derived from a polygon of |N| - 1 sides, since the two edges E(mn) and E(no) could have been derived from a single edge E(mo).

If |V| = 0 then the polygon is convex and any three consecutive nodes are mutually visible. If |V| = 1 then the three vertices of the single concavity are necessarily all mutually visible and we are done.

If |V|>1, choose any concave angle. If its three points are mutually visible then its two line segments may be replaced by one showing the n-gon is derivable from an n-1 gon by RPA. If they are not, then the line of site between the two endpoints of the angle must be interrupted by a part of the n-gon. This part of the n-gon must, in turn, contain a concavity. Repeat this process obtaining a finite sequence of nested concavities. That finite sequence terminates with a last internal concavity which itself, must have no inner blocking concavities. The three points of a concave angle within that unblocked concavity must all be mutually visible, implying that the n-gon could have been derived from an n-1-gon

4. CONCLUSION

This paper has introduced a polynomial time algorithm for generating n-gons that are random and representative of the class of all n-gons. The run time behavior of this algorithm is, in its worst case, O(n^2logn). RPA is faster than other algorithms and is proven to be both fair and inclusive of all polygons.

The original application for which these researchers worked on this problem was, in truth, for entertainment rather than practicality. However, an argument can be made that entertainment is one of the most practical and lucrative aspects of computer graphics. It is certainly the case that being able to construct random shapes quickly on the fly is a useful application in the business of scene-generation, and likely in the realm of algorithm testing as well.

The RPA technique can easily be applied to a graphical context by limiting the random number generator to within the confines of a bounding box. For any size of P, a bounding box can be easily determined by finding the minima and maxima in both the x and y dimensions. The vertices of the bounding box are included in the walk after all of the vertices in P are traversed. In addition, when intersections are determined, the line segments of the bounding box are included.

5. ACKNOWLEDGMENTS

The authors would like to recognize the input of Mr. Stephan Browarny, an alum of Slippery Rock University. Mr. Browarny had many suggestions on how to select the next point.

6. REFERENCES

[1] Ahn, HK, de Berg M., Bose P., Cheng SW., Halperin D., Matousek J.V, and Schwarzkopf O. 2002. Separating an Object from its Cast. Computer- Aided Design. 34, 8 (July 2002), 547-559.

[2] Auer, T. and Held M. 1996. Heuristics for the Generation of Random Polygons, in: 8th Canadian Conference on Computational Geometry (CCCG). Ottawa, Canada, 1996, 38-44.

[3] Chazelle B. 1991. Triangulating a Simple Polygon in Linear Time. Disc. Computational Geomentry 6(1991), 485-524.

[4] Chazelle, B., and H. Edelsbrunner 1992.. An Optimal Algorithm for Intersecting Line Segments in the Plane, Journal of the ACM. 39 (1992), 1-54.

[5] Chvátal V. 1975. A combinatorial theorem in plane geometry, J. Combin. Theory B18 (1975), 39-41.

[6] Clarkson, K.L, Tarjan R.E., and Van Wyk C.J. 1989. A Fast Las Vegas Algorithm for Triangulating a Simple Polygon. Discoveries of Computational Geometry. 4 (1989), 423-432.

[7] Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. Numerische Mathematik. 1 (1959), S. 269–271.

[8] Epstein, P. and Sack J.R. 1994. Generating Triangulations at Random. ACM Transactions on Modeling and Computer Simulation (TOMACS). 4,3 (1994) 267-278.

[9] Graham, R. 1972. An Efficient Algorithm for Determining the Convex Hull of a Finite Point Set. Info. Proc. Letters. 1 (1972), 132-133.

[10] Held, M., 2007. Generation of Random Polygonal Objects, at: http://www.cosy.sbg.ac.at/~held/projects/rpg/rpg.html.

[11] Mandelbrot, Benoit B., The Fractal Geometry of Nature, W.H. Freeman and Co. San Francisco, 1982, 34-57.

[12] Michael, T.S. and Val Pinciu 2003. Computational Geometry: Theory and Applications. 26,3, (Elsevier Science Publishers B. V. Amsterdam, The Netherlands, The Netherlands 2003). 247 – 258.

[13] Mitchell, J.S.B. and O'Rourke J. 2001. Computational Geometry Column 42. International Journal of Computational Geometry Applications. 11,5 (2001) 573-582.

[14] Prim, R. C. 1957. Shortest connection networks and some generalisations, in: Bell System Technical Journal, 36 (1957), 1389–1401.

[15] Toussaint, G.T., Sitaru V. and Ruso T. 1996. Drawing Simple Polygons through Point Sets at: http://cgm.cs.mcgill.ca/~godfried/teaching/cg-web.html

[16] World Wide Web Consortium: HTML 5: W3C Editor's Draft, September 22: The Canvas, 2007: http://www.w3.org/html/wg/html5/#the-canvas

[17] Zhou, Z. 2007. Developing BREW Applications with Qualcomm's SVG Solution SVG Open, Tokyo, September 2007.

[18] Zhu, C., Sundaram G., Snoeyink J. and Mitchell J.S.B. 1996. Generating Random Polygons with Given Vertices Computational Geometry Theory and Application. 6 (1996) 277-290.

[19] Zusne, L., 1970. Visual Perception of Form, Academic Press, New York, 1970.