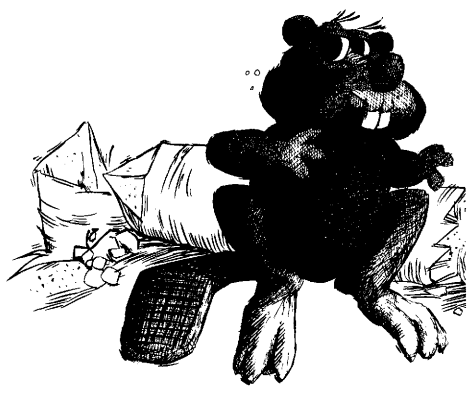


Turtle Polygons

Joseph Culberson
Gregory Rawlins

Data Structuring Group
CS-84-41

November, 1984

Turtle Polygons

Joseph Culberson
Gregory Rawlins

Data Structuring Group
Department of Computer Science
University of Waterloo
Waterloo,Ontario
N2L 3G1 Canada

ABSTRACT

In this paper we present an algorithm to create simple polygons with a particular sequence of exterior angles, given only the sequence of angles. The algorithm has worst case time complexity $O(Dn)$, where n is the number of angles and D is dependent on the angles. As a bonus, the algorithm proves an interesting converse of the ancient theorem that the sum of the exterior angles of a simple polygon is 2π radians.

1. Introduction

A turtle, in the sense of [4] , placed in the plane either walks directly ahead in a straight line or makes a turn to the right or left through some fixed angle $\theta \in (0,\pi)$. If the turtle is given a sequence of "turn left" (l) and "turn right" (r) commands, can the turtle execute the sequence of turns so as to return to its starting point without at any time intersecting its path? Clearly if this happens then its path describes a simple polygon all of whose interior angles are $\pi \pm \theta$.

Let $\#l$ ($\#r$) designate the total number of left (right) turns the turtle is asked to make. If the path described is a simple polygon then $\theta = 2\pi/(\#l - \#r)$, since the sum of the turning angles of any simple polygon must be 2π. From now on we assume that the basic angle of turn is π/D , D an integer greater than 1. We denote this angle by θ. Also, we assume that we may cyclically shift any turning sequence since this just corresponds to rotating the path drawn.

Observe that if $\theta = \pi/D$ and the turtle makes only left turns (or only right turns) then after $2D$ of them it will be back where it started (assuming equal distances between turns). Of course, in this case it would have described a regular polygon with $2D$ sides in the plane. Now if we allow exactly *one* right turn there must be one more left turn to match it, since $\#l = \#r + 2D$. Also, observe that after a right turn followed by a left turn the turtle is oriented in exactly the same direction as before, the only difference being that it has moved some distance in the plane.

Given a sequence of turns we describe a way to pair up the right turns with the excess left turns so that the turtle effectively traces out a regular polygon of $2D$ sides with some "detours" to match some of the right turns. Given this technique we will describe an algorithm to solve the stated problem, then generalize it to handle a much larger class of turning sequences.

2. The *rl*-Algorithm

Before giving the algorithm in detail, it is helpful to consider a simple example. Consider the turning sequence $\sigma = (l,r,l,l,l,l,l,l)$ with turning angle $\theta=\pi/3$. The turtle begins at the origin facing along the positive x-axis and it generates a closed nonintersecting path by attempting to draw an edge of unit length before turning left or right θ degrees and drawing the next edge.

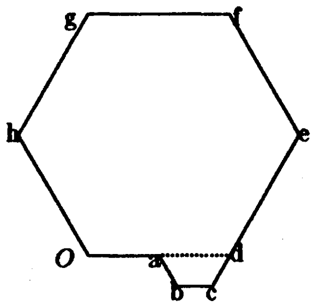


Figure 1.a
$\sigma=(l,r,l,l,l,l,l,l)$

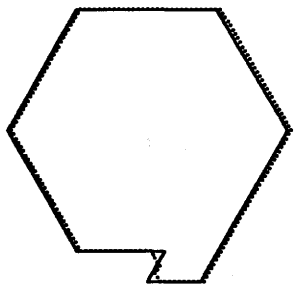


Figure 1.b
$\sigma=(1l,2r,2l,1l,1l,1l,1l,1l)$

Thus, in figure 1.a, it starts at O and proceeds towards d. It is drawing the edge which has endpoints corresponding to the first l (in σ) at its left end, and the r at its right end. If there were no r's in σ, then the regular hexagon $Odefgh$ would be the desired path. However, the r indicates that the turtle must make a right turn. To facilitate this detour, the turtle stops short of its original destination (d) at the point a. The ratio of Oa to Od is designated R and is 1/2 in this example.

To keep its path from being self-intersecting, when the turtle makes a right turn the length of subsequent edges of the detour are reduced by an appropriate scale factor S. After a right turn, a single left turn points the turtle back in its original direction. Thus we think of a (r,l) pair as forming a completed detour. The scale factor is so chosen that $ab=bc$ and c lies on an extension of the side ed. At this point the vertices O,a,b,c correspond to the first four turns in σ.

After completing the detour, the turtle moves to d, and continues its tour from there (i.e. just as if it had not encountered an (r,l) pair in between). If the r had been followed by another r in this example, then the turtle would have been forced to make a sub-detour on the edge ab. (Of course, another l would also be required to complete that detour). Alternatively, if σ had been $(l,r,l,r,l,...)$ then the turtle would have made a sub-detour on the edge bc. This nicely illustrates the pairing idea; after any r we may either have another r or an l. If the r is followed by an l then they are paired together and represent one detour. If on the other hand the r is followed by another r then we must first find an l to pair with the *second* r before we can pair the first r. Logically, an (r,l) pair corresponds to a pair of left and right brackets, and the process just described corresponds to finding a "proper nesting" of the brackets.

Algorithm 1 (RL)

```
INPUT:
    A positive integer $D$ and a sequence $\sigma$, of $r$'s and $l$'s, with $\#l=\#r+2D$.
OUTPUT:
    A sequence of co-ordinates of the vertices of a polygon with turning
    sequence $\sigma$.
begin
    {Parse $\sigma$ into properly nested pairs of $r$'s and $l$'s.}
        Initialize Stack to empty;
        Scanning $\sigma$ from left to right (wrapping around if necessary)
        if current unpaired turn is $r$
        then push current location in $\sigma$ on Stack
        else if current unpaired turn is $l$ and Stack not empty
        then pop top location and pair with current location;
        {This process halts when all $r$'s are paired. Since $\#l$'s $>$ $\#r$'s this step
        must halt in at most two scans of $\sigma$.}
        Select one of the $2D$ unpaired $l$'s as the first vertex. Cyclically shift $\sigma$
        so that this vertex is at the start;
    {Generate the vertices of the polygon using the recursive procedure Edge. }
        Calculate the basic turning angle $\theta=\pi/D$ and the scale factor
        $S=(1-R)/(2\cos(\theta)+1)$;
        {$R$ is the ratio of the actual distance travelled to the intended
        distance, and is fixed.}
        Index = 2;
        { Index is an index to $\sigma$ indicating the next vertex to be drawn. The
        first vertex is assumed to be at the origin.}
        Edge(Origin,1,0,2D,Index);
end.
```

Procedure Edge(*Start*,*Length*,*Turns*,*Number*,*var Index*)

{Edge draws *Number* edges ($Number=2$ or $Number=2D$), with recursive calls if a right turn is encountered in σ. Essentially, Edge attempts to draw a regular polygon with $2D$ sides; however, it may have to make (recursive) detours on any of the $2D$ edges. The first edge begins at the point *Start*. Each edge is of length *Length* (or $Length \times R$ if it ends at a right turn). The first edge has direction $Turns \times \theta$, that is, it is *Turns* turns away from the horizontal. Subsequent (non-recursive) edges have directions $(Turns+i)\theta$, $1 \leq i < N$. *Index* is incremented after each edge is drawn. }

```
begin
    for $i=0$ to $N-1$ do;
    begin
        {Compute the Start for the next iteration.}
        Start := Start + a distance of Length in the direction $(Turns+i)\theta$;
        if $\sigma[Index]=r$
        then
            {The turtle is forced to make a detour, hence the current edge
            must be truncated. Make the truncation point the
            Currentvertex, which becomes the Start point for the recursive
            call. The scale is reduced by $S$, the direction is one turn to the
            right relative to the current direction, and the recursive call
            draws 2 edges (to complete the detour).}
            Currentvertex := the end of a line from Start in the direction
            $(Turns+i)\theta$ with length $Length \times R$;
            output Currentvertex;
            Index:=Index+1;
            Edge(Currentvertex , $S \times Length$ , $Turns+i-1$ , 2,Index);
        else
            output Start;
            Index:=Index+1;
    end;
end.
```

It is straightforward to show that step 1 of the *rl*-algorithm properly pairs the *r*'s and *l*'s and assigns a unique *l* to each *r*. Further, exactly $2D$ *l*'s are left unpaired in this process and these correspond in an obvious way to the vertices of a regular polygon with $2D$ sides. Between any two consecutive unpaired *l*'s there will be a (possibly empty) balanced subsequence of *r*'s and *l*'s which Edge draws as "detours" on the side corresponding to those two unpaired *l*'s.

It is a simple exercise in analytic geometry to show that the choice $R=1/2$ and $S=(1-R)/(1+2\cos(\theta))$ guarantees that the polygon drawn by the *rl*-algorithm is closed and non self-intersecting. We sketch the proof in the following four steps: (1) and (2) hold by construction, (3) and (4) follow easily.

(1) Each detour is an isosceles trapezoid with base angle θ and side lengths $(S/(1-R))$ times the length of the base. For example, in Figure 2.a, *bedc* is a detour on the edge *ac* and $dc=ed=be=(S/(1-R))\times bc$, $bc=(1-R)\times ac$, $ed \| bc$ and $\angle ebc=\angle bcd=\theta$.

(2) A sub-detour on a detour with base length x has base length $x \times S$. Hence the side lengths of any nested set of detours form a geometric progression with common ratio S. For example, in Figure 2.a, the sub-detour *fhge* on the detour *bedc* has base length $ef=bc\times S$, similarly the sub-detours on *fhge* each have base length $ef\times S=bc\times S^2$.

(3) Since the algorithm is recursive, it is only necessary to prove that a sequence of nested detours does not intersect the side of the regular polygon on which it is drawn. (All other cases are similar except for a change in scale and orientation).

(4) Observe that $eb \| gi$, $ab \| ge$ and that we may add more detours onto the edge ij. The resulting polygon is non-intersecting only if

$$\begin{aligned} eb &\geq gh+hi+\cdots \\ \Leftrightarrow ac\times S &\geq ac\times S^2+ac\times S^3+\cdots \\ \Leftrightarrow S &\geq \sum_{i=2}^{\infty} S^i \Leftrightarrow S\leq 1/2 \Leftrightarrow R+\cos(\theta)\geq 1/2 \end{aligned}$$

When $D\geq 3$, $0\leq\theta\leq\pi/3$, $\cos(\theta)\geq 1/2$ and hence any $R\in(0,1)$ will do. When $D=2$, $\theta=\pi/2$ $\cos(\theta)=0$ hence we must choose $R\in[1/2,1)$.

We observe in passing that although the particular scale factor chosen guarantees simplicity it may not be the "weakest" possible scale necessary; i.e. it may be shrinking the lengths of recursive edges more than is necessary (they decrease exponentially with S). A more intelligent algorithm could well rectify this.

3. The General Algorithm

It is easy to generalize the problem to that of drawing a simple polygon where the turtle may be asked to turn some *integral multiple* of the basic turning angle. Turning sequences will now be made up of multipliers (m_i) times an l or an r to indicate a turn of $m_i\times\theta$ to the left or right $(m_i\in[1,D-1])$. For example, the rl-algorithm applies only to turning sequences where each $m_i=1$.

To handle the general case we will create a copy of σ which has m_i l's (or r's) in sequence wherever σ has a turn of $m_i\times l$ (or $m_i\times r$). Then draw (using the rl-algorithm) a simple polygon with that turning sequence (the rl-polygon). Finally we create the actual turning angles required by "merging" the appropriate edges of the rl-polygon.

Consider the turning sequence $\sigma = (1l,2r,2l,1l,1l,1l,1l,1l,1l)$ of Figure 1.b. First the $2r$ is replaced by the pair r,r and the $2l$ by the pair l,l, then the resulting sequence is plotted using the rl-algorithm (dotted line polygon in Figure 1.b). A representative polygon with turning sequence σ is then created by projecting the appropriate edges of the rl-polygon to their intersection point, effectively merging the angles to form the correct angles (straight line polygon in figure 1.b). Observe that the two polygons in Figure 1.b are the same except that one edge of the dotted polygon has been "pushed in" and another "pushed out". This is the result of the re-merging of the two l's back into one $2l$ turn and the two r's back into one $2r$ turn.

Algorithm 2

```
INPUT:
    A general turning sequence $\sigma$.
OUTPUT:
    A sequence of co-ordinates of the vertices of a polygon with turning
    sequence $\sigma$.
begin
    {Construct an $rl$-sequence for $\sigma$.}
    Beginning with an empty $rl$-sequence,
    for each multiplier $m$ in $\sigma$,
        if a left turn then append $m$ $l$'s to the sequence
        else append $m$ $r$'s to the sequence;
    Use the $rl$-algorithm to generate a representative polygon of the resulting
    $rl$-sequence;
    {Construct the appropriate turning angles.}
        Initialize next vertex $N=1$;
        for each multiplier $m$ in $\sigma$,
            output the intersection point of the edges ending at $N$ and
            $N+m$;
            {That is, find the intersection of the edges corresponding to the
            pairs of turns in positions $(N-1,N)$ and $(N+m-1,N+m)$.
            Note that this effectively deletes $m-1$ edges from the $rl$-
            polygon.}
            $N=N+m$;
end.
```

We must now show that the "merging" process does not introduce any intersections. There are only two types of merges, those which merge l's and those which merge r's. Let us call the former "external" merges and the latter "internal" merges. Figure 2.b shows examples of both types. Here the rl-algorithm had to make two recursive detours in drawing the detour $blmn$ (the sub-detour $dikl$ and the sub-subdetour $fghi$). Then (in the merging phase) the vertices b, d and f were merged to the vertex c (an internal merge), and the vertices g, h and k were merged to the vertex j (an external merge). We will sketch the proof for internal merges only since the same argument applies to externals by "exchanging" the interior and exterior of the polygon.

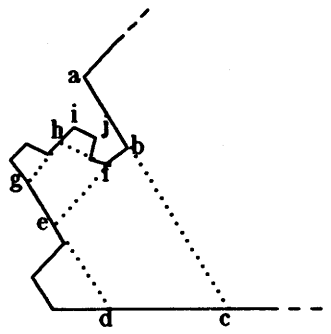


Figure 2.a
A nested set of detours

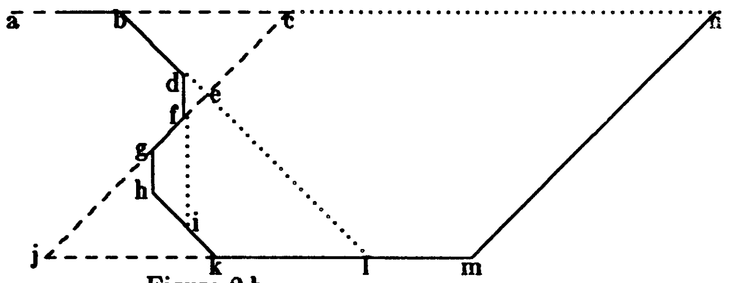


Figure 2.b
Examples of merges

Consider the internal merge on edges *ab* and *gf* shown in Figure 2.b. Here $D=4$ and we are merging the maximum possible number of vertices (3). Since we are merging $D-1$ angles, the exterior angle produced at the intersection point is θ (in Figure 2.b $\angle bcf=\theta=\pi/4$). This means that the second edge of the merge is parallel to the last edge of the base detour (in Figure 2.b $gf \| mn$). Hence the merge cannot produce an intersection. Suppose now that *bd* is used as the "start edge" of a $D-1$ merge. Now the intersection point must lie on *dl* by the same argument as before (since the other extended edge must be parallel to *kl*).

In general, for any sequence of $D-1$ consecutive r's, there corresponds a sequence of $D-1$ edges, the last of which is parallel to the last drawn edge of the detour on the first edge of the sequence. Since the extension of the last edge of the sequence must lie in the detour (proved previously) then there can be no intersections caused by merging the first and last edges of the sequence. If we merge less than $D-1$ edges then the extension is clearly less. Finally, it is easy to see that an internal merge cannot intersect another internal merge; similarly, an internal merge cannot intersect an external merge.

To give worst case time bounds we choose the usual (RAM) model of computation in which any arithmetic operation costs $O(1)$ time and overall execution time is proportional to the number of times the input is scanned. Suppose that σ is an arbitrary turning sequence of length n and $\theta=\pi/D$. Each step of the general algorithm costs $O(Dn)$ (since the rl-algorithm is given a string of length at most $(D-1)n$ and Edge examines each vertex exactly once). Hence the overall worst case cost is $O(Dn)$. Note that the normal input to the algorithm may not necessarily be turning sequences with explicit multipliers. The sequence of interior angles of the polygon to be drawn is sufficient. To convert all we need do is find the lowest common denominator (D) of the angles in the sequence (this costs $O(n \log D)$).

4. Discussion

In the polygons drawn by the algorithm deeply nested edges rapidly shrink in length with the consequence that some polygons have an "unnatural" look. Although it is, of course, impossible to formalize precisely what is meant by "unnatural" it seems reasonable to say that a polygon looks "natural" if all its edges are of approximately equal length (alternately, we could require that the edge lengths follow a Gaussian distribution). It is an interesting unsolved problem to characterize those polygons for which it is *possible* for the edges to be of exactly equal length.

We say that an angle is *rational* if it can be expressed as a rational number times π radians. The algorithm constitutes a proof of the theorem that for any finite sequence of rational angles summing to 2π, there exists a *simple* polygon with that turning sequence. Thus, not only must the turning angles of a simple polygon sum to 2π, but also, if a sequence of angles is rational and sum to 2π then some simple polygon has that sequence of angles.

Note that we may think of a turning sequence with $\theta=\pi/D$ as a *necklace* where beads may be coloured in any of $2(D-1)$ colours (the integers $1-D..D-1$, except zero, represent the colours) and the numbers of beads of each colour is constrained so that the sum of the colours is $2D$. This observation, together with the techniques in [1], can be used to create an "encyclopaedia" of representatives of each non-isomorphic (in terms of angle sequence) simple polygon on n angles. This collection can be used to provide insights into or counter-examples of geometric conjectures. Indeed, the desire for just such a collection was the main reason for this research.

As was pointed out by O'Rourke [3] the technique outlined in this paper may be used to generate "random" simple polygons to test geometric algorithms.

The results in this paper suggest the following avenues of further research:

The Construction Problem:

Prove a lower bound of $\Omega(Dn)$ for the problem of drawing simple polygons or show that $O(Dn)$ can be improved. Find a more intelligent algorithm which draws more "natural" polygons in a reasonable time.

The "Spline" Problem:

We can generalize the drawing problem to that of drawing a *polygonal curve* between two given points such that the turning angles of the curve is a given turning sequence. Further, by analogy with splines, we could ask for a polygonal curve which passes through prefixed "knots" and has a fixed turning sequence.

The Decision Problem:

Suppose we generalize turning sequences even more by allowing a turn of *any* angle at any vertex (still with the restriction that the angles sum to 2π). For which strings of these arbitrary turning angles is it possible to construct a simple polygon? The only progress we have made on this problem is the observation that any finite nonempty set of irrational angles, I, can only occur in at most $|I|-1$ distinct turning sequences in which all the other angles are rational.

The Identification Problem:

> Characterize those polygons whose edge lengths can be made equal. Generalize to those polygons whose edge lengths each belong to some finite set.

Although this paper solves the problem of drawing a polygon given a particular angle sequence it should be observed that it pays no attention to the lengths of the edges. If we restrict the problem to not only drawing a polygon with a particular angle sequence but also with edge lengths chosen from some finite set we know of no polynomial time algorithm. On the other hand, if we add the condition that all the vertices must land on lattice points ([2]), or that all the edge lengths be integral ([7]), then the problem is still not fully solved. (It seems that there is far more to geometric figures than meets the eye!).

Acknowledgments

Sack ([5] see also [6] §3.1) independently came up with an idea similar to the pairing scheme used in the *rl*-algorithm for the special case of rectilinear polygons ($D=2$). O'Rourke [3] also solved the rectilinear problem but in a completely different way.

We wish to thank Darrell Raymond and Professor Daniel Field for their careful proofreading and useful comments. In particular, Professor Field suggested the "spline" problem. We also wish to thank Professor Derick Wood for improving an earlier version of this paper and for his continuing support and encouragement.

This work supported in part by National Sciences and Engineering Research Council of Canada Grant No. A-5692 and a NSERC 1967 science scholarship.

References

[1] Fredericksen, H. and Maiorana, J.; "Necklaces of beads in *k* colours and *k*-ary DeBruijn sequences", *Discrete Math.* **23** (1978) pp. 207-210.

[2] Honsberger, R.; "Semi-regular lattice polygons", *Two Year College Math. J.* **13** (1982) pp. 36-44.

[3] O'Rourke, J.; Personal communication.

[4] Papert, S.; *"Mindstorms: Children, Computers and powerful ideas"*, Basic Books, New York, 1980.

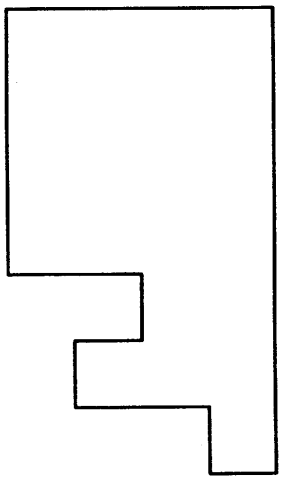
[5] Sack, J.-R.; Personal communication.

[6] Sack, J.-R.; "Rectilinear Computational Geometry", Ph.D. thesis, McGill University, School of Computer Science, May 1984.

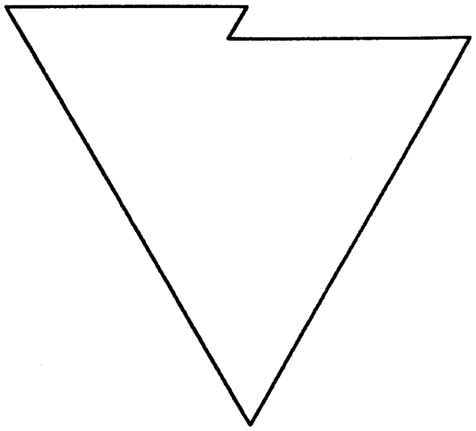
[7] Stewart, B.M. and Herzog, F.; "Semi-regular plane polygons of integral type", *Israel J. of Math.* **11** (1972) pp. 31-52.

Appendix

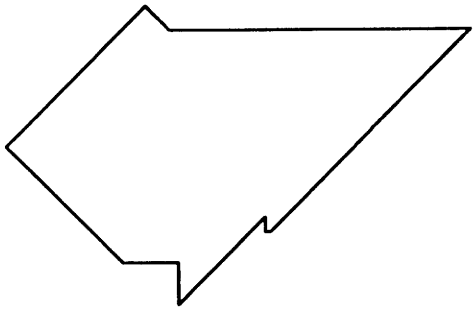
Here are some example polygons drawn by the algorithm. We have made the multipliers positive or negative to indicate left and right and dropped the l's and r's.



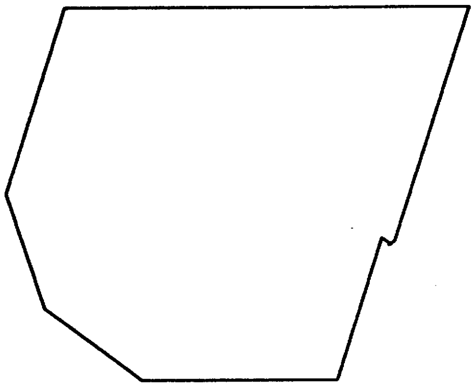
$(1,-1,-1,1,1,-1,1,1,1,1), \theta=\pi/2$



$(2,2,-2,2,2), \theta=\pi/3$



$(1,-2,3,-3,2,1,3,-1,2,2), \theta=\pi/4$



$(1,2,-3,-2,4,1,3,2,1,1), \theta=\pi/5$