Morphing TSP Art

David Swart
Waterloo, Ontario Canada,
dmswart1@gmail.com

Abstract

TSP Art is a technique to represent an image by tracing out a solution to the Travelling Salesman Problem. It results in a distinctive and attractive aesthetic, consisting of a simple closed curve with varying densities. This paper explores a straightforward method to smoothly transition between two TSP solutions resulting in interesting animations. I extend this method to open curves and to curves on the surface of a sphere. Finally, I present examples of how this technique is used to fill in between key TSP Art frames of an animation.

Introduction

TSP Art, well described by Bosch and Kaplan [7], involves creating images by judiciously placing cities, and then drawing a path that visits all the cities and returns to the beginning with as short a path as possible. Since all TSP Art is made up of a single non-intersecting loop (a *simple closed curve*), it would be interesting to see one of these loops smoothly and continuously morph into another piece of TSP art without crossing itself (see Figure 1). In this paper I take existing ideas from Computer Graphics and Differential Geometry to write an easy-to-implement algorithm that morphs the especially difficult curves that are TSP Art

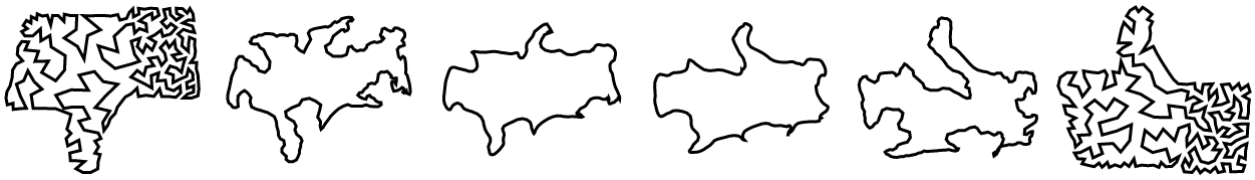

Figure 1: *Morphing the simple closed curve of one TSP Art piece into another.*

The task of smoothly interpolating one shape into another is an active research field and very useful in computer graphics. There are many papers with titles in the set {“polygon”, “closed planar curve”, “shape”} × {“tweening”, “morphing”, “metamorphosis”, “blending”} each with their own particular problem constraints. Papers by Sederberg and Greenwood [10] and by Gotsman and Surazhsky [5] present two different approaches but are more complex than we need here, as they include constraints to do things like maintain a triangulation of the interior, or find appropriate point correspondences between the images.

The new method closely resembles multi resolution blending by Goldstien and Gotsman [4]. Both their method and this method smooth the initial and target curves into regular polygons using the idea of *curvature flow* (see the discussion of the SMOOTH subroutine below). Goldstein and Gotsman then combine the two animations at the smoothed ends, and then “undo the smoothing steps” in a manner of speaking. Anyone familiar with this earlier work will see that our method conceptually does the same thing but in a slightly different manner described in the discussion of the TIGHTEN subroutine below.

This paper is organized as follows: I describe the morphing algorithm, first at a high level with pseudocode, and then we examine two subroutines SMOOTH and TIGHTEN and I show some example runs. Following this, I describe implementation details such as what programming environment I used and some ideas for improving performance. I show how we can modify this algorithm to work with curves on the surface of a sphere, and on open curves. Finally, we will look at some fun results.

The Morphing Algorithm

We quickly describe our algorithm first and then follow this with a pseudocode listing, followed by a more in depth discussion of each stage.

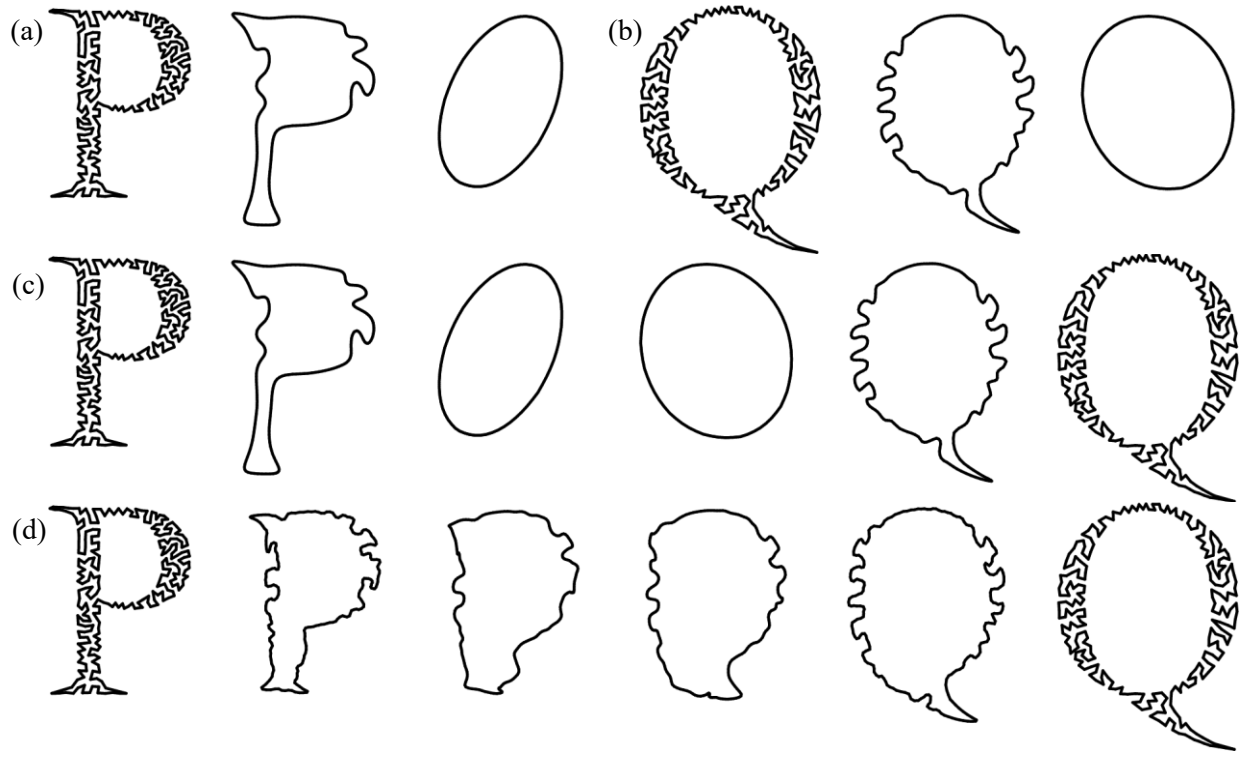

Figure 2: *(a,b) SMOOTHing two polygons into eliptical shapes; (c) reversing the second smoothing animation and appending it to the first; (d) TIGHTENing the entire animation.*

The morphing algorithm takes as input two TSP Art tours P and Q as two lists of 2D points (see Kaplan and Bosch [6] for how these might be obtained). We want P and Q to have the same direction and the same number of points. So if necessary, reverse the order of P (and / or Q) such that both are clockwise. Then, add midpoints to the longest segment of P (or Q) until they each have the same number of points n. Then rotate the indices of Q until the locations of the points of Q most closely match the positions of the corresponding points of P. SMOOTH P and Q into an elliptical shape in a series of animated frames. Calculate each new frame by moving each point to the average of its neighbors and itself, taking care not to introduce any crossings (Figure 2a-b). Then reverse the Q animation and concatenate it to the end of the animation of P (Figure 2c). Finally, apply a TIGHTEN step to the points of the animation similar the SMOOTH step but in the time dimension, taking care not to introduce any crossings (Figure 2d). For reference, Table 1 has a more precise pseudocode listing of this algorithm.

The SMOOTH Subroutine

The goal of the SMOOTH routine is to calculate an animation that turns a tour into an elliptical shape, or more precisely into an *affine regular polygon*. Differential Geometry has a useful tool to do this very thing called *mean curvature flow*: a process that asymptotically turns a smooth curve into an ellipse in a well behaved way. It accomplishes this by moving each point of a curve by an amount proportional to the amount

Table 1: *Pseudo-code of the morphing algorithm. Indices of points are to be interpreted modulo n.*

```
1   SMOOTH(T):  // calculate animation that turns a tour T into an ellipse
2     F_0 ← T;  μ_0 ← avg position of F_0;  δ_0 ← avg distance between μ_0 and the points F_0
3     iterate on i:
4       F_i ← (p_i,1, …, p_i,n), where p_i,j = { (p_i−1,j−1 + p_i−1,j + p_i−1,j+1) / 3,  if no crossing is introduced
5                                             { p_i−1,j,                                 otherwise
6       μ_i ← avg position of F_i;  δ_i ← average distance between μ_i and the points of F_i
7       F_i ← ((F_i−μ_i)×δ_i-1/δ_i)+μ_i−1
8     until F_i ≈ F_i-1
9     return (F_0, F_1, …, F_i)

10  TIGHTEN(A): // smooth the points of an animation in the time dimension
11    repeat until manually halted:
12      for each frame F_i in A that is not a beginning or end frame:
13        F_i ← (p_i,0, …, p_i,n), where p_i,j = { (p_i−1,j + p_i,j + p_i+1,j) / 3,  if no crossing is introduced
14                                              { p_i−1,j,                          otherwise
15    return A

16  MORPH(P, Q):  // calculate an animation that smoothly transitions tour P into tour Q
17    make P and Q clockwise
18    add a midpoint to the longest segment of P (or Q) until they are the same size: n points
19    shift the indexes of Q until ∑_i∈0…n |p_i − q_i| is minimized.
20    (P_0, …, P_f) ← SMOOTH(P)
21    (Q_0, …, Q_g) ← SMOOTH(Q)
22    A ← TIGHTEN(P_0, …, P_f, Q_g, …, Q_0)
```

of curvature at that point. The calculation in Line 4 has a theoretical basis described by Chow and Glickenstein [2] who approximate this process for polygons. Those interested in the powerful things that geometric flow can do should read the excellent paper by Crane et al. [1] which uses a slightly different kind of flow to smooth out surfaces quickly. Or readers can look at SMOOTH's opposite: an algorithm that takes smooth curves and makes wiggly mazes [8].

Unless we take precautions, SMOOTH would cause our curve to eventually shrink into a point. To keep things at the same scale, we renormalize the points after each step. Line 7 makes sure the scale and position of the points remain constant from frame to frame. The SMOOTH routine stops when we get a self-similar figure: an affine regular polygon.

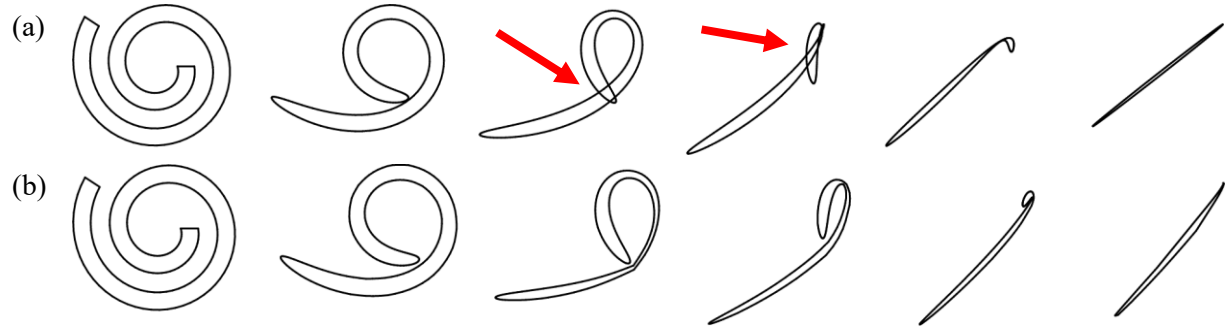

Figure 3. *(a) A self intersection; (b) the animation after avoiding crossings.*

Using mean curvature flow to shape our curves does not guarantee that our edges will not cross (Figure 3a). So we can simply add the following check: if moving the point would result in an edge crossing within the current frame, then the point stays put (see Line 5). This appears to effectively prevent edge crossings, at least for non-contrived examples, even though it is not guaranteed to do so (Figure 3b).

The TIGHTEN Subroutine

Figure 2c shows us a polygon P morphing into an ellipse first and then morphing into a polygon Q. Technically this satisfies the problem we set out to solve and yet it seems unsatisfying. We want to have the animation look like it is going directly from P to Q without any intermediate steps.

To fix this, we can run the animation through a routine called TIGHTEN. It might help to think of the frames of the animation as cross sections of a tube of stretchy fabric shaped like P and Q on each end and like an ellipse in the middle. The goal then is to simulate the movement of each point of this stretchy tube based on the points ahead and behind in (time). As with SMOOTH we avoid introducing self-intersections and we get the satisfying transition that we see in Figure 2d.

This TIGHTEN step is analogous to the second part of the algorithm by Goldstien and Gotsman [4] where they recover the high frequency details as they add back in more details.

Implementation

This algorithm is simple enough to implement in JavaScript which has the benefit of requiring only a web browser to run it. In order to display the results, I relied on the D3 library [3] to update and animate the paths of an SVG image in the browser window. As an added bonus, D3's Voronoi module was useful for generating the original TSP art tours.

I did notice that when n gets too high (~ 400 pts) the algorithm gets *slow*. By slow I mean greater than one minute on a single core of an Intel Xeon @ 2.80GHz. Without getting into a complexity analysis of the algorithm here, I will just note that the run time seemed to behave between $O(n^2)$ and $O(n^3)$ for observed values of $n < 5000$. The following performance improvements were called for.

In order to pare down the number of frames being sent to TIGHTEN, I modified SMOOTH to just save the animation frames when the cumulative movement of all the points exceeds some threshold. A nice side effect is that this makes the animation appear to move with a more constant speed, instead of something that moves quickly at the beginning, and then decreases in speed.

A performance improvement for SMOOTH is obtained by noticing that when the curve is smooth enough (average angle < 5°) we reduce the points by deleting every other one. This is similar to the multiresolution technique used in numerical analysis. This allows us to work on a reduced set of points as representative of the entire thing. The "missing" cities get added back in when needed by interpolating between the undeleted points. Of course, any number of polygon simplification routines such as the Ramer–Douglas–Peucker algorithm [9] would suffice – though there may be more to keep track of.

Open curves. If we wanted to morph two polygonal chains (i.e., not just closed loops), we merely have to set the endpoints of the polygonal chains fixed from frame to frame, and then skip the normalization step. What ends up happening then is SMOOTH turns the given curve into a line segment rather than an elliptical shape (see Figure 4a). If TIGHTEN was pulling a tube of stretchy fabric in our analogy before, now it is pulling a rectangular sheet with satisfying results as well (see Figure 4b).

Figure 4: *(a) Turning an open curve into a line segment with SMOOTH; (b) Applying TIGHTEN to two open curves.*

On the surface of a sphere. Another variation I personally find very intriguing is to move the entire problem to the surface of a sphere. To do this we start by listing our points as 3D points that are projected to the unit sphere. Then, throughout the algorithm, whenever a point is moved, we just normalize it to project it back to the surface. New routines were used to calculate the distance between points on the surface of a sphere, the angle between two arcs, whether two arcs intersect, etc. One interesting consequence to moving to a sphere is that the normalization step of SMOOTH (Line 7) no longer needed to factor in the δ values that kept the curve from shrinking. Figure 5 shows a spherical TSP Art morph.

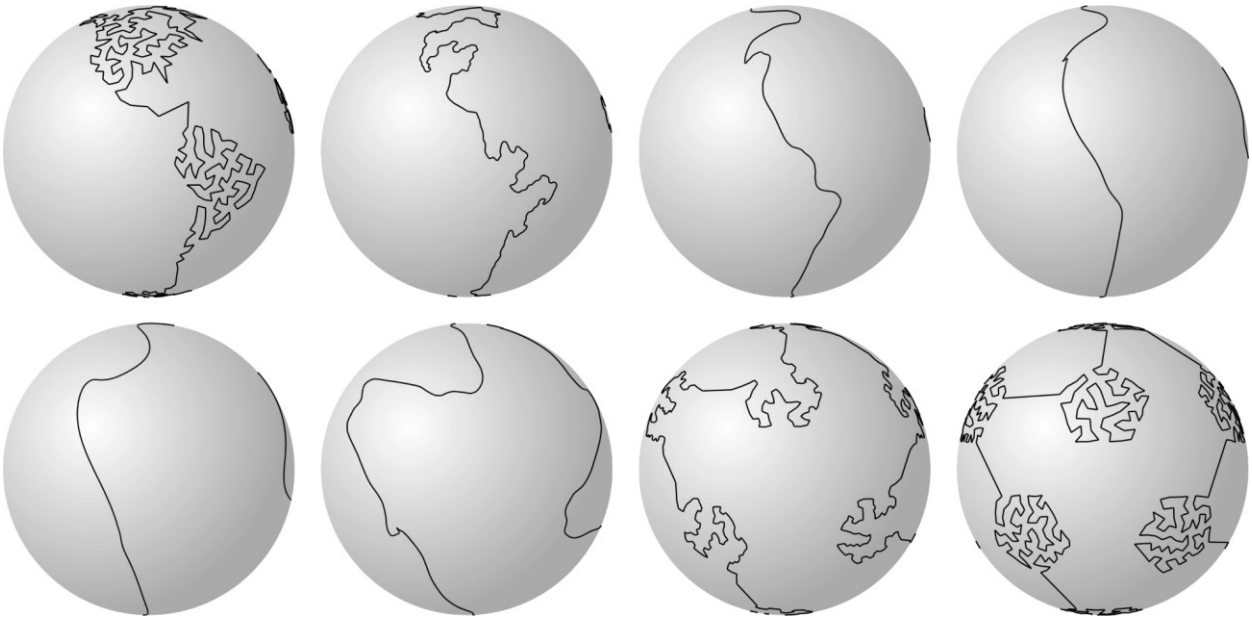

Figure 5: *TSP Art on a sphere, morphing from a globe to a soccer ball.*

Concluding Remarks

We have tools in hand to create a novel animation medium. A natural thing to do is to make a .gif. The animation shown in Figure 1, for example, is a cute novelty that can be shared on social media. I am also making an animated short film, although at the time of this writing, the animation is still incomplete. When it is complete, it will appear on the website dmswart.com. In the meantime, Figure 6 shows selected still frames.

An anonymous reviewer suggested a great idea: if we render the curve, substituting the time dimension for the third spatial dimension, the result is an attractive design for a sculpture, evocative of Segerman and Irving's developing fractal curves [6]. See Figure 7.

Finally, I cannot claim that every transitional frame maintains the same aesthetics as TSP Art: they are not as crinkly, and they do not exhibit that constant distance between adjacent lines that give TSP Art its maze-like quality. Future work could address these issues.

References

[1] Keenan Crane, Ulrich Pinkall, and Peter Schröder, *Robust Fairing via Conformal Curvature Flow* ACM Trans. Graph., Vol. 32, No. 4 (2013).

[2] Bennett Chow and David Glickenstein, *Semidiscrete Geometric Flows of Polygons* The American Mathematical Monthly Vol. 114, No. 4 (Apr., 2007). Pages 316–328.

[3] D3.js Data-Driven Documents `https://d3js.org/` Accessed January 31, 2017.

[4] Eli Goldstein and Craig Gotsman. *Polygon Morphing Using a Multiresolution Representation.* Graphics Interface '95. Pages 247–254.

[5] Craig Gotsman, Vitaly Surazhsky *Guaranteed intersection-free polygon morphing* Computers & Graphics 25 (2001). Pages 67–75.

[6] Geoffrey Irving and Henry Segerman *Developing fractal curves* Journal Of Mathematics And The Arts Vol. 7 , Iss. 3-4,2013.

[7] Craig S. Kaplan and Robert Bosch *TSP Art* Renaissance Banff: Mathematics, Music, Art, Culture (2005) Editors Reza Sarhangi and Robert V. Moody. Pages 301–308 (Available online at `https://archive.bridgesmathart.org/2005/bridges2005-301.html`

[8] Hans Pedersen and Karan Singh *Organic labyrinths and mazes* in NPAR '06 Proceedings of the 4th international symposium on Non-photorealistic animation and rendering. Pages 79–86.

[9] David Douglas & Thomas Peucker, *Algorithms for the reduction of the number of points required to represent a digitized line or its caricature*, The Canadian Cartographer 10(2), (1973). Pages 112–122.

[10] Thomas W. Sederberg and Eugene Greenwood *A Physically Based Approach to 2-D Shape Blending* in SIGGRAPH '92 Proceedings of the 19th annual conference on Computer graphics and interactive techniques. Pages 25–34.

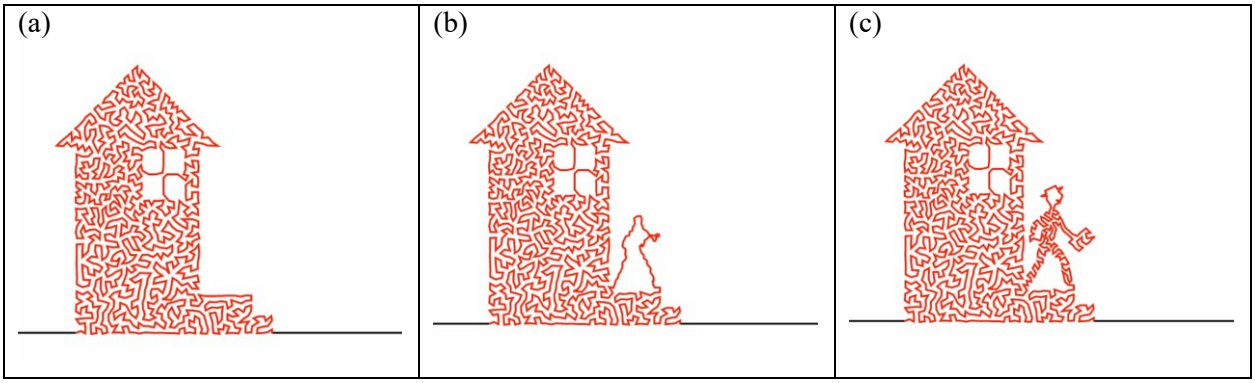

Figure 6: *Two key frames and an in-between frame: The travelling salesman starts his journey.*

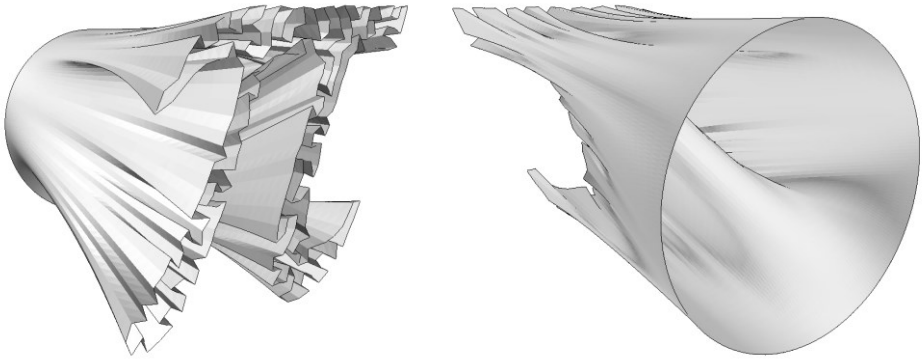

Figure 7: *Two views of a hypothetical sculpture with a circle on one side, and the symbol π on the other.*