Compact simplification of Bézier splines

Jeremy Tan

October 31, 2015

Abstract

Algorithms that convert from raster to vector graphics are well-known, but so far there has not been one that compacts the vector output while retaining accuracy as much as possible. We introduce a two-phase heuristic algorithm for this purpose based on three principles: closeness of fit, preservation of corners and minimisation of nodes used. In the first phase, the irregularities are removed by a construction that produces a sequence of points approximating the simplification. This is the input for the second phase, which emulates human approaches to produce the final output. We note how the two parts can be executed simultaneously.

1 Introduction

The *tracing* of raster images, converting them to a vector basis, is a common operation in computer graphics. Programs that do this, such as Peter Selinger's potrace,[Sel] must balance simplicity with accuracy. In general they focus on the latter property and generate large outputs, especially when the input is a photograph of a real-world object or scene, because of small and local variations in the source that the tracing program copies over. Since large outputs take longer to render and are harder to edit manually, a simplifier to suppress these variations before display would be desired, but such algorithms that accept Bézier curves have never been published in the literature.

Indeed, even when public software is considered, very little is known about how to achieve this. The simplifying function of Inkscape accumulates errors when repeated, as shown in Figure 1. It is therefore only a smoothing function, despite its name. In this paper we construct a functioning algorithm for the purpose based on heuristics.

2 Terminology

A cubic Bézier curve, or just *cubic*, is a parametric curve defined by four points $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$, taking a parameter t in the interval [0, 1]. Its equation and

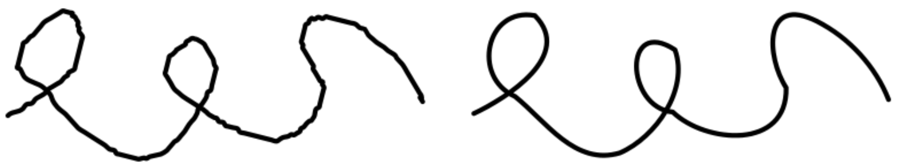

Figure 1: A freehand path before (left) and after 10 consecutive applications of Inkscape's *Simplify* function. The two loops had no clear corners at their tops at first but acquired them after simplification. The one in the right loop is especially prominent, showing that the function is deficient.

shorthand are:

$$\begin{aligned}\mathbf{B}(t) &= \sum_{i=0}^{3}\binom{3}{i}(1-t)^{3-i}t^i\mathbf{P}_i \\ &= (1-t)^3\mathbf{P}_0 + (1-t)^2t\mathbf{P}_1 + (1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \\ &= \langle \mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3\rangle (t)\end{aligned}$$

The curve starts at $\mathbf{B}(0) = \mathbf{P}_0$ and ends at $\mathbf{B}(1) = \mathbf{P}_3$, the starting and ending *nodes* respectively. The other two control points $\mathbf{P}_1$ and $\mathbf{P}_2$ are called *handles* and define the curve's direction at its ends, though they do not lie on the curve in general. It is common in diagrams of cubics to draw a line from $\mathbf{P}_0$ to $\mathbf{P}_1$ and from $\mathbf{P}_3$ to $\mathbf{P}_2$ for clarity.

A *path* is a sequence $\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_{n-1}$ of these curves such that the ending node of one curve is identified with the starting node of the next. This may also apply to $\mathbf{B}_{n-1}$ and $\mathbf{B}_0$, in which case the path is *closed*; otherwise it is *open*. A point on curve $\mathbf{B}_i$ with parameter t may be denoted by the ordered pair (i, t). A point $\mathbf{p}_1 = (i_1, t_1)$ is *greater* than another distinct point $\mathbf{p}_2 = (i_2, t_2)$ if either $i_1 > i_2$ or $i_1 = i_2$ and $t_1 > t_2$, and *lesser* otherwise.

Relative to a node shared by two paths, the angle between the incoming handle ($\mathbf{P}_2$) of the previous path and the outgoing handle ($\mathbf{P}_1$) of the next may be 180°, in which case the node is *smooth*; otherwise the node is *cusp*. The ends of an open path are considered cusp.

These definitions are illustrated in Figure 2.

3 What is simplified?

The description of a path as "simplified" is subjective and based more on aesthetics than measurable quantities. Just as how multiple cubic Bézier curves may be drawn through three points,[Kam, §25] there are multiple perceptions of which paths are simplified and which are not. For this paper we adopt the following principles:

1. The simplified path should use as few segments as possible and should have the same endpoints as the initial path if it is open.

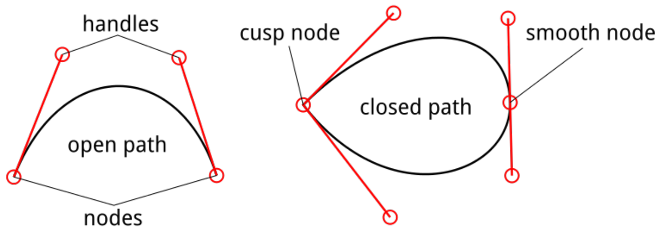

Figure 2: Parts of a path. Circles denote control points of the Bézier curves.

2. Cusp nodes should be present in the output if and only if the input changes its general direction abruptly there.

3. The maximum distance of any point on the original path from its simplification should be less than some threshold.

The first principle's reasons have already been discussed – fewer nodes in a path results in less time spent rendering it – and it may be trivially evaluated. The second keeps corners as they are while the third maintains the status of the output path as a simplification *approximating* the original. For evaluating these two we use the following visualisation.

Suppose a circle, whose radius corresponds to the desired accuracy, moves along the different cubics in a path. It defines an envelope, which may be likened to a river's course. The second principle dictates that cusp nodes of the simplified path lie where the *thalweg* of this "river", the continuous line at maximum distance from the envelope's edges, makes a sharp turn. The third states that this simplified path, like the original, lies entirely within the envelope.

However, directly computing the envelope – an offset path – is slow and expensive. Instead we will approximate the thalweg with a *sequence* of points, constructing the simplified path to fit only this.

4 Ribosomes and the point sequence

The approximated thalweg $\mathbf{T} = \{\mathbf{T}_0, \mathbf{T}_1, \mathbf{T}_2, \dots \mathbf{T}_{N-1}\}$, with N points, depends on a smoothness parameter δ and is generated by several processes called *ribosomes* operating on sections of the path. Each ribosome, given a starting and ending point on the path $\mathbf{p}_{start}$ and $\mathbf{p}_{end}$, operates according to the following algorithm.

Algorithm A Ribosome method of generating a sequence of points approximating the thalweg of a path at radius δ

1. Initialise the output list with $\mathbf{p}_{start}$.

2. Draw a circle with radius δ centred on the last point of the output list, then find all its intersections $\mathbf{i}$ with the path such that $\mathbf{p}_{start} < \mathbf{i} < \mathbf{p}_{end}$.
3. If there is exactly one such intersection $\mathbf{i}$, append it to the output list and return to step 2.
4. If there are no intersections, append $\mathbf{p}_{end}$ to the output list and return the list in its entirety.
5. If there are two or more intersections:
    (a) Sort the intersections in ascending order $\mathbf{i}_1, \ldots, \mathbf{i}_k$, then append $\mathbf{i}_{k+1} = \mathbf{p}_{end}$ to form a list of $k+1$ points.
    (b) Start k new ribosomes, where the nth ribosome starts at $\mathbf{i}_n$ and ends at $\mathbf{i}_{n+1}$ for $1 \leq n \leq k$. For all these ribosomes but the last, delete the last point of the list they return, then append the concatenation of these lists in ascending order of their first point to the current output list and return the extended list.

Initially there is one ribosome operating on the entire path; the output of this process is $\mathbf{T}$ and will be sent to the second phase, that of fitting a curve to $\mathbf{T}$.

The processes are called ribosomes because they progress along the linear path and return parts of $\mathbf{T}$, analogous to ribosomes in cell biology that progress along the bases of an mRNA molecule and return a protein. The circles drawn, the black circles in Figure 3, skip over and thereby remove irregularities in the path that are smaller than δ in radius from consideration. Because the starting of new ribosomes subdivides the path into disjoint and strictly shorter paths, step 4 must eventually be invoked, preventing further generation and hence guaranteeing the algorithm's termination.

5 Path fitting and stressing

$\mathbf{T}$, the approximated thalweg, is an ordered list of at least two points to which the output simplified path will be fitted to. Observe that when $\mathbf{T}$ contains four points $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$ it is always possible to find *one* cubic that passes through the points in order, or two real numbers $0 \leq \lambda \leq \mu \leq 1$ such that there exist two points $\mathbf{P}_\lambda$ and $\mathbf{P}_\mu$ with

$$\begin{aligned}\langle \mathbf{P}_0, \mathbf{P}_\lambda, \mathbf{P}_\mu, \mathbf{P}_3 \rangle (\lambda) &= \mathbf{P}_1 \\ \langle \mathbf{P}_0, \mathbf{P}_\lambda, \mathbf{P}_\mu, \mathbf{P}_3 \rangle (\mu) &= \mathbf{P}_2\end{aligned}$$

To see this, set λ and μ as arbitrary constants and note that the resulting system has two independent sets of two linear equations in two unknowns, which can be solved easily. Similarly, by fitting two points to a line or three points to a quadratic Bézier curve and elevating the curve order, there must exist a cubic when $\mathbf{T}$ contains two or three points.

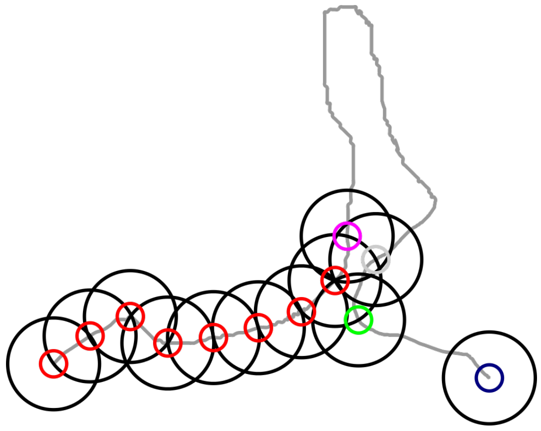

Figure 3: Generation of the approximated thalweg $\mathbf{T}$ for a path (shown in grey). The rightmost red circle intersects the path at three greater points – fuchsia, light grey and lime circles centred on $\mathbf{i}_0$, $\mathbf{i}_1$ and $\mathbf{i}_2$ respectively. The points of $\mathbf{T}$ generated by ribosomes operating between these intersections and the path's end (purple circle) will be appended to the red circles' centres to form $\mathbf{T}$.

When $\mathbf{T}$ contains more than four points, one cubic may not suffice. For example, a set containing four points on the x-axis and a fifth point not lying on the common line does not admit a fit, since the y-component of the cubic equation is overconstrained. However, this suggests a procedure of fitting points of $\mathbf{T}$ to one cubic until the error exceeds a second threshold ϵ, at which point a new cubic will be started, the process iterated until no points remain to be processed. This "sliding window" method forms the base operation of the second phase, which generates a path such that the furthest distance from any point in $\mathbf{T}$ to the path is minimised and less than ϵ. The value of ϵ obviously depends on δ, but should not be set equal or greater than δ because the approximating structure – whether it be a point set or curve – should become better as the algorithm progresses, not worse. In particular, we recommend $\epsilon = \frac{1}{2}\delta$.

Here we define a notion of chord distance for a point $\mathbf{P}_i$ amongst a set of points $\{\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_{n-1}\}$:

$$\chi(\mathbf{P}_i) = \begin{cases} 0 & \text{if } i = 0 \\ 1 & \text{if } i = n-1 \\ \frac{\sum_{k=1}^{i} \|\mathbf{T}_k - \mathbf{T}_{k-1}\|}{\sum_{k=1}^{n-1} \|\mathbf{T}_k - \mathbf{T}_{k-1}\|} & \text{otherwise} \end{cases}$$

This corresponds to the "chord length method" described in [Pas].

Algorithm B Generation of a path that fits $\mathbf{T}$ with maximum error ϵ

1. Initialise a *window* of consecutive points in $\mathbf{T}$ as $\{\mathbf{T}_0, \mathbf{T}_1\}$. Mark these two points as *processed* and the remaining points in the path as unprocessed.

2. Fit a *current cubic* $\mathbf{C}$ to the window based on the number of points in the window as follows:

   (a) If the window contains two points $\{\mathbf{T}_i, \mathbf{T}_{i+1}\}$, construct a linear interpolation:

   $$\mathbf{C} = \left\langle \mathbf{T}_0, \frac{2}{3}\mathbf{T}_0 + \frac{1}{3}\mathbf{T}_1, \frac{1}{3}\mathbf{T}_0 + \frac{2}{3}\mathbf{T}_1, \mathbf{T}_1 \right\rangle$$

   (b) If the window contains three points $\{\mathbf{T}_i, \mathbf{T}_{i+1}, \mathbf{T}_{i+2}\}$, fit a quadratic curve exactly to the three points, taking the parameter of $\mathbf{T}_{i+1}$ as $\chi(\mathbf{T}_{i+1})$, then elevate the curve to a cubic as explained in [Kam, §10]. Similarly, if the window contains four points $\{\mathbf{T}_i, \mathbf{T}_{i+1}, \mathbf{T}_{i+2}, \mathbf{T}_{i+3}\}$, fit a cubic curve exactly with the parameters of $\mathbf{T}_{i+1}$ and $\mathbf{T}_{i+2}$ as $\chi(\mathbf{T}_{i+1})$ and $\chi(\mathbf{T}_{i+2})$ respectively.

   (c) If the window contains five or more points $\{\mathbf{T}_{start}, \ldots, \mathbf{T}_{end}\}$, save the current $\mathbf{C}$, then *stress* a new $\mathbf{C}$ to fit the window's points (explained below). If the furthest distance of any point in the window from this new $\mathbf{C}$ is greater than ϵ, write the saved $\mathbf{C}$ to output and set the window as $\{\mathbf{T}_{end-1}, \mathbf{T}_{end}\}$, with the new $\mathbf{C}$ constructed as in step 2(a).

3. If there are still unprocessed points, extend the window by one point to the right (i.e. if the window ends at $\mathbf{T}_z$, the extended window ends at $\mathbf{T}_{z+1}$), mark the new point as processed and go back to step 2. If not, write $\mathbf{C}$ to output and end the algorithm.

Stressing is the heuristic part of this algorithm; it is based on the process humans use to perform curve fitting, which is to move only the handles until a good fit is perceived and inserting a node if a satisfactory fit cannot be found.

Algorithm S Method of stressing a cubic to fit points $\{\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3, \ldots, \mathbf{I}_N\}$, where $N \geq 5$

1. Construct the $\left\lfloor \frac{N-1}{2} \right\rfloor$ cubics that fit the lists of points

   $$\{\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_{N-1}, \mathbf{I}_N\}$$
   $$\{\mathbf{I}_1, \mathbf{I}_3, \mathbf{I}_{N-2}, \mathbf{I}_N\}$$
   $$\vdots$$
   $$\left\{\mathbf{I}_1, \mathbf{I}_{\left\lfloor \frac{N+1}{2} \right\rfloor}, \mathbf{I}_{\left\lfloor \frac{N}{2} \right\rfloor + 1}, \mathbf{I}_N\right\}$$

   and then take the average of the $\mathbf{P}_1$ and $\mathbf{P}_2$ handles of these cubics, $\overline{\mathbf{P}_1}$ and $\overline{\mathbf{P}_2}$, to form the initial cubic

   $$\mathbf{K} = \left\langle \mathbf{I}_1, \overline{\mathbf{P}_1}, \overline{\mathbf{P}_2}, \mathbf{I}_N \right\rangle.$$

2. For each pair of points $\mathbf{I}_2/\mathbf{I}_{N-1}, \mathbf{I}_3/\mathbf{I}_{N-2}, \ldots, \mathbf{I}_{\left\lfloor \frac{N+1}{2} \right\rfloor}, \mathbf{I}_{\left\lfloor \frac{N}{2} \right\rfloor + 1}$ in order:

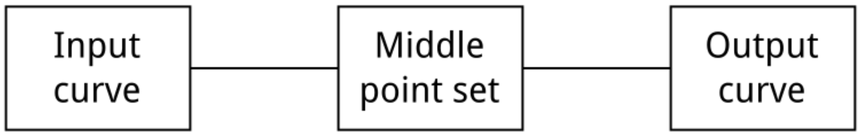

Figure 4: Pipeline for the curve simplification algorithm.

(a) Project the points onto $\mathbf{K}$ and note the two vectors from these projections to the points themselves. If the point earlier in the sequence is $\mathbf{I}_S$ and that later in the sequence $\mathbf{I}_E$, while their projections are $\mathbf{J}_S$ and $\mathbf{J}_E$ respectively, then the two vectors are $\mathbf{v}_1 = \mathbf{I}_S - \mathbf{J}_S$ and $\mathbf{v}_2 = \mathbf{I}_E - \mathbf{J}_E$.

(b) Set $\mathbf{P}_1$ of $\mathbf{K}$ as $\mathbf{P}_1 + k\mathbf{v}_1$ and $\mathbf{P}_2$ as $\mathbf{P}_2 + k\mathbf{v}_2$, where k is a positive real number.

3. Repeat step 2 a total of M times; the output cubic is $\mathbf{K}$ after these operations have been completed.

Hence the stressing algorithm is determined by two parameters: the number of iterations M over the middle (not end) points and the factor k to multiply the projection vector by. It was found in testing that $M = 5$ and $k = 2.5$ provided the best balance between speed and accuracy.

6 Implementation, discussion and optimisations

The basic curve fitting algorithm of phase two is a greedy one, in that it does not consider whether moving nodes to the left (i.e. shortening one cubic while lengthening the next) will improve the fit. Corners are never explicitly marked either and the second phase may miss what may look as an obvious corner to the human eye because the error obtained in fitting the cubic to the point immediately beyond the corner may be less than ϵ. Hence we discuss some optimisations and improvements to our algorithm.

6.1 Marking corners on the fly

Existing corner detectors such as Moravec's and Förstner's require a raster image as input, not a discrete point set as given here. It would also be desirable if the algorithm was implemented as a *pipeline* in which the second phase simply waits for thalweg points from the first phase, as shown in Figure 4. This parallel implementation is suited for applications using this algorithm since it saves time.

Hence we use a far simpler corner detector, inserted before step 2(a) of Algorithm B. Given a current point – the newly added point of the window in Algorithm S – if there are at least three points before it we construct the circumcircle of this three points, or the common line if those points are collinear. This requires only three operations as described in [Kam, §35]. The shortest distance from the current point to this circle or line is then taken, and if this is greater than a third threshold δ_c the previous point is deemed a corner. When this happens, the saved $\mathbf{C}$ is written to output and a new window and curve are constructed, as in the latter part of step 2(c). We recommend $\delta_c = \epsilon$ for the sake of simplicity.

6.2 Reverse curve fitting

Because the curve fitter is point-based, it is very easy to reverse the process, such as to improve the fit of previous cubics. Given the sequences of points corresponding to two such cubics already written to output, Algorithm B can be applied to the *reverse* of the combined sequence until the sum of the fitting errors $\epsilon_1 + \epsilon_2$ does not decrease further.

References

[Kam] Mike Kamermans. *A Primer on Bézier Curves.* Retrieved 23 October 2015.

[Pas] Tim A. Pastva. "Bézier Curve Fitting".

[Sel] "Peter Selinger: Potrace". Retrieved 16 August 2015.