The Hobby package

Andrew Stacey
stacey@math.ntnu.no

September 3, 2012

1 Introduction

John Hobby's algorithm, [1], produces a curve through a given set of points. The curve is constructed as a list of cubic Bézier curves with endpoints at subsequent points in the list. The parameters of the curves are chosen so that the joins are "smooth". The algorithm was devised as part of the MetaPost program.

TikZ/PGF has the ability to draw a curve through a given set of points but its algorithm is somewhat simpler than Hobby's and consequently does not produce as aesthetically pleasing curve as Hobby's algorithm does. This package implements Hobby's algorithm in TEX so that TikZ/PGF can make use of it and thus produce nicer curves through a given set of points.

Hobby's algorithm allows for considerable customisation in that it can take into account various parameters. These are all allowed in this implementation.

There is also a "quick" version presented here. This is a modification of Hobby's algorithm with the feature that any point only influences a finite number (in fact, two) of the previous segments (in Hobby's algorithm the influence of a point dies out exponentially but never completely). This is achieved by applying Hobby's algorithm to subpaths. As this is intended as a simpler method, it does not (at present) admit the same level of customisation as the full implementation.

The full algorithm is implemented in LATEX3 and makes extensive use of the `fp` and `prop` libraries for the computation steps. The "quick" version does not use LATEX3 and relies instead on the `PGFMath` library for the computation.

Figure 1 is a comparison of the three methods. The red curve is drawn using Hobby's algorithm. The blue curve is drawn with the `plot[smooth]` method from TikZ/PGF. The green curve uses the "quick" version.

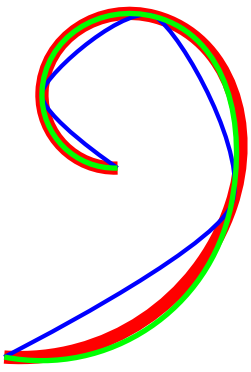


Figure 1: Comparison of the three algorithms

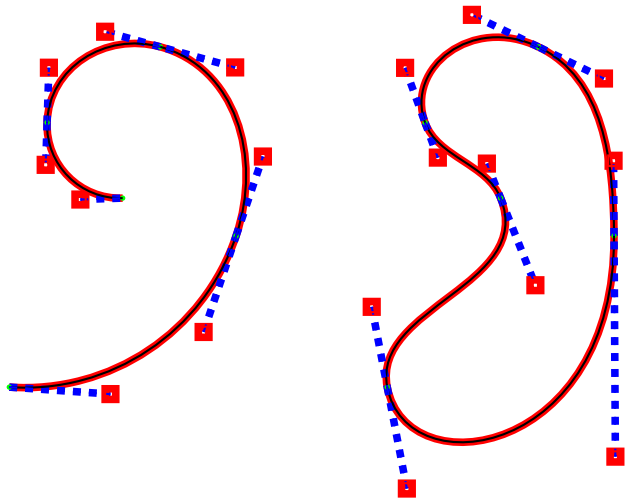


Figure 2: Hobby's algorithm in TikZ overlaying the output of MetaPost

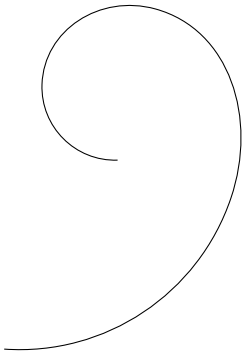
2 Usage

The package is provided in form of a TikZ library. It can be loaded with

```
\usetikzlibrary{hobby}
```

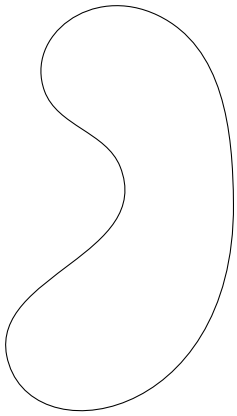
The TikZ library installs a `to path` which draws a smooth curve through the given points:

```
\begin{tikzpicture}
\draw (0,0) to[curve through={(6,4) .. (4,9) .. (1,7)}] (3,5);
\end{tikzpicture}
```



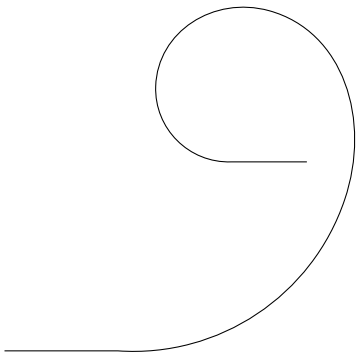
The path can be open, as above, or closed:

```
\begin{tikzpicture}
\draw (0,0) to[closed,curve through={(6,4) .. (4,9) .. (1,7)}] (3,5);
\end{tikzpicture}
```



There is also the facility to subvert TikZ's path processor and define curves simply using the `..` separator between points. Note that this relies on something a little special in TikZ: the syntax `(0,0) .. (2,3)` is currently detected and processed but there is no action assigned to that syntax. As a later version of TikZ may assign some action to that syntax, this package makes its override optional via the key `use Hobby shortcut`.

```
\begin{tikzpicture}[use Hobby shortcut]
\draw (-3,0) -- (0,0) .. (6,4) .. (4,9) .. (1,7) .. (3,5) -- ++(2,0);
\end{tikzpicture}
```



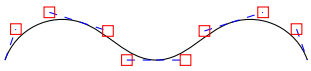
The algorithm can deal with open or closed paths, it is possible to vary the "tensions" between the specified points of the paths, and for an open path it is possible to specify the incoming and outgoing angles either directly or via certain "curl" parameters. See the Examples section for more examples. The algorithm is actually implemented in LaTeX3 with (almost[1]) no reference to TikZ or PGF. The TikZ library is simply a wrapper that takes the user's input, converts it into the right format for the LaTeX3 code, and then calls that code to generate the path. There is also a "quick" version of Hobby's algorithm. This is described in Section 6. The reason for this modification of Hobby's algorithm was to find a variant in which adding more points does not change the path between earlier points (or rather that there is some point earlier than which the path is not changed). The resulting path produced with this "quick" version is not as ideal as that produced by Hobby's full algorithm, but is still much better than that produced by the `plot[smooth]` method in TikZ/PGF, as can be seen in Figure 1.

[1] At the moment, LaTeX3 lacks a `atan2` function so `PGFMath` is used to remedy that.

3 Examples

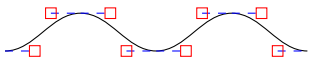
- Basic curve.

```
\begin{tikzpicture}
\draw[postaction=show curve controls]
(0,0) to[curve through={(1,.5) .. (2,0) .. (3,.5)}] (4,0);
\end{tikzpicture}
```



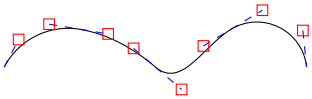
- Specifying the angle at which the curve goes *out* and at which it comes *in*. The angles given are absolute.

```
\begin{tikzpicture}
\draw[postaction=show curve controls]
(0,0) to[out angle=0,in angle=180,curve through={(1,.5) .. (2,0) .. (3,.5)}] (4,0);
\end{tikzpicture}
```



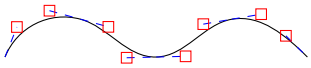
- Applying tension as the curve comes in to a point.

```
\begin{tikzpicture}
\draw[postaction=show curve controls]
(0,0) to[curve through={(1,.5) .. ([tension in=2]2,0) .. (3,.5)}] (4,0);
\end{tikzpicture}
```



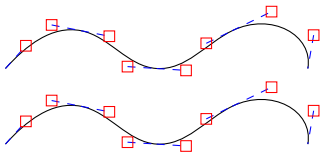
- Applying the same tension as a curve comes in and goes out of a point.

```
\begin{tikzpicture}
\draw[postaction=show curve controls]
(0,0) to[curve through={(1,.5) .. ([tension=2]2,0) .. (3,.5)}] (4,0);
\end{tikzpicture}
```



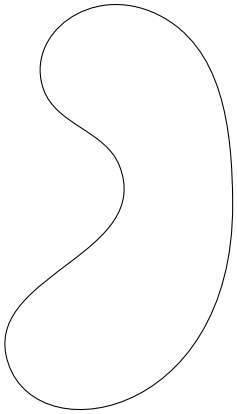
- Specifying the *curl* parameters (if using the shortcut, these have to be passed via one of the points but obviously apply to the whole curve).

```
\begin{tikzpicture}[use Hobby shortcut]
\draw[postaction=show curve controls]
(0,0) to[curve through={(1,.5) .. (2,0) .. (3,.5)},in curl=.1,out curl=3] (4,0);
\begin{scope}[yshift=-1cm]
\draw[postaction=show curve controls]
(0,0) .. ([in curl=.1,out curl=3]1,.5) .. (2,0) .. (3,.5) .. (4,0);
\end{scope}
\end{tikzpicture}
```



- Closed curve.

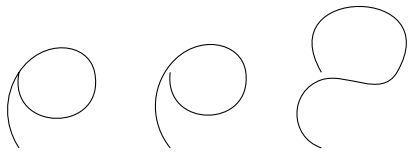
```
\begin{tikzpicture}[scale=.5,use Hobby shortcut]
\draw (0,0) .. (6,4) .. (4,9) .. (1,7) .. (3,5) .. cycle;
\end{tikzpicture}
```



4 Edge Cases

Angles are constrained to lie in the interval $(-\pi, \pi]$. This can introduce edge cases as there is a point where we have to compare an angle with $-\pi$ and if it is equal, add 2π. This will occur if the path "doubles back" on itself as in the next example. By nudging the repeated point slightly, the behaviour changes drastically.

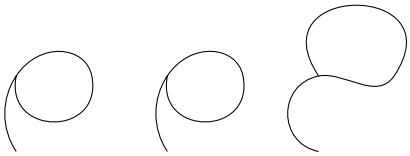
```
\begin{tikzpicture}[use Hobby shortcut]
\draw (0,0) .. (1,0) .. (0,0) .. (0,-1);
\draw[xshift=2cm] (0,0) .. (1,0) .. (0,0.1) .. (0,-1);
\draw[xshift=4cm] (0,0) .. (1,0) .. (0,-0.1) .. (0,-1);
\end{tikzpicture}
```



Due to the precision of the computations, it is not possible to always get this test correct. The simplest solution is to nudge the repeated point in one direction or the other. Experimenting shows that the "nudge factor" can be extremely small (note that it will be proportional to the distance between the specified points). It is best to nudge it in the direction most normal to the line between the specified

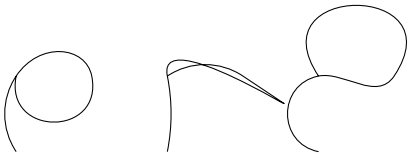
points as the goal is to nudge the difference of the angles. An alternative solution is to add an additional point for the curve to go through.

```
\begin{tikzpicture}[use Hobby shortcut]
\draw (0,0) .. (1,0) .. (0,0) .. (0,-1);
\draw[xshift=2cm] (0,0) .. (1,0) .. (0,0.002) .. (0,-1);
\draw[xshift=4cm] (0,0) .. (1,0) .. (0,-0.002) .. (0,-1);
\end{tikzpicture}
```



Lastly, it is possible to add an `excess angle` key to a coordinate. This will add the corresponding multiple of 2π to the angle difference.

```
\begin{tikzpicture}[use Hobby shortcut]
\draw (0,0) .. (1,0) .. (0,0) .. (0,-1);
\draw[xshift=2cm] (0,0) .. ([excess angle=1]1,0) .. (0,0) .. (0,-1);
\draw[xshift=4cm] (0,0) .. ([excess angle=-1]1,0) .. (0,0) .. (0,-1);
\end{tikzpicture}
```



Although this is intended to be an integer, no check is done and so some quite odd curves can result from changing this parameter.

5 Implementing Hobby's Algorithm

We start with a list of $n+1$ points, $z_0, \ldots, z_n$. The base code assumes that these are already stored in two arrays[2]: the x–coordinates in `\l_hobby_points_x_array` and the y–coordinates in `\l_hobby_points_y_array`. As our arrays are 0–indexed, the actual number of points is one more than this. For a closed curve, we have $z_n = z_0$[3]. For closed curves it will be convenient to add an additional point at z_1: thus $z_{n+1} = z_1$. This makes z_n an internal point and makes the algorithms for closed paths and open paths agree longer than they would otherwise. The number of apparent points is stored as `\l_hobby_npoints_int`. Thus for an open path, `\l_hobby_npoints_int` is n, whilst for a closed path, it is $n+1$[4]. Following Hobby, let us write n' for n if the path is open and $n+1$ if closed. From this we compute the distances and angles between successive points, storing these again as arrays. These are `\l_hobby_distances_array` and `\l_hobby_angles_array`. The term indexed by k is the distance (or angle) of the line between the kth point and the $k+1$th point. For the internal nodes[5], we store the difference in the angles in `\l_hobby_psi_array`. The kth value on this is the

[2] Arrays are thinly disguised property lists

[3] Note that there is a difference between a closed curve and an open curve whose endpoints happen to overlap

[4] In fact, we allow for the case where the user specifies a closed path but with $z_n \neq z_0$. In that case, we assume that the user meant to repeat z_0. This adds another point to the list.

[5] Hobby calls the specified points *knots*

angle subtended at the kth node. This is thus indexed from 1 to $n'-1$. The bulk of the work consists in setting up a linear system to compute the angles of the control points. At a node, say z_i, we have various pieces of information:

1. The angle of the incoming curve, ϕ_i, relative to the straight line from z_{i-1} to z_i
2. The angle of the outgoing curve, θ_i, relative to the straight line from z_i to z_{i+1}
3. The tension of the incoming curve, $\overline{\tau}_i$
4. The tension of the outgoing curve, τ_i
5. The speed of the incoming curve, σ_i
6. The speed of the outgoing curve, ρ_i

The tensions are known at the start. The speeds are computed from the angles. Thus the key thing to compute is the angles. This is done by imposing a "mock curvature" condition.The formula for the mock curvature is:

$$\hat{k}(\theta,\phi,\tau,\overline{\tau}) = \tau^2\left(\frac{2(\theta+\phi)}{\overline{\tau}} - 6\theta\right)$$

and the condition that the mock curvatures have to satisfy is that at each *internal* node, the curvatures must match:

$$\hat{k}(\phi_i,\theta_{i-1},\overline{\tau}_i,\tau_{i-1})/d_{i-1} = \hat{k}(\theta_i,\phi_{i+1},\tau_i,\overline{\tau}_{i+1})/d_i.$$

Substituting in yields:

$$\frac{\overline{\tau}_i^2}{d_{i-1}}\left(\frac{2(\phi_i+\theta_{i-1})}{\tau_{i-1}} - 6\phi_i\right) = \frac{\tau_i^2}{d_i}\left(\frac{2(\theta_i+\phi_{i+1})}{\overline{\tau}_{i+1}} - 6\theta_i\right).$$

Let us rearrange that to the following:

$$\begin{aligned}
d_i\overline{\tau}_{i+1}\overline{\tau}_i^2\theta_{i-1} \\
+d_i\overline{\tau}_{i+1}\overline{\tau}_i^2(1-3\tau_{i-1})\phi_i \\
-d_{i-1}\tau_{i-1}\tau_i^2(1-3\overline{\tau}_{i+1})\theta_i \\
-d_{i-1}\tau_{i-1}\tau_i^2\phi_{i+1} \\
= 0
\end{aligned}$$

For both open and closed paths this holds for $i=1$ to $i=n'-1$. We also have the condition that $\theta_i+\phi_i=-\psi_i$ where ψ_i is the angle subtended at a node by the lines to the adjacent nodes. This holds for the internal nodes[6]. Therefore for $i=1$ to $n'-1$ the above simplifies to the following:

$$\begin{aligned}
d_i\overline{\tau}_{i+1}\overline{\tau}_i^2\theta_{i-1} \\
+(d_i\overline{\tau}_{i+1}\overline{\tau}_i^2(3\tau_{i-1}-1) + d_{i-1}\tau_{i-1}\tau_i^2(3\overline{\tau}_{i+1}-1))\theta_i \\
+d_{i-1}\tau_{i-1}\tau_i^2\theta_{i+1} \\
= -d_i\overline{\tau}_{i+1}\overline{\tau}_i^2(3\tau_{i-1}-1)\psi_i \\
-d_{i-1}\tau_{i-1}\tau_i^2\psi_{i+1}
\end{aligned}$$

For an open path we have two more equations. One involves θ_0. The other is the above for $i=n'-1=n-1$ with additional information regarding ψ_n. It may be that one or either of θ_0 or ϕ_n is specified in advance. If so, we shall write the given values with a bar: $\overline{\theta}_0$ and $\overline{\phi}_n$. In that case, the first equation is simply setting θ_0 to that value and the last equation involves substituting the value for ϕ_n into the

[6]Recall that by dint of repetition, all nodes are effectively internal for a closed path

above. If not, they are given by formulae involving "curl" parameters χ_0 and χ_n and result in the equations:

$$\theta_0 = \frac{\tau_0^3 + \chi_0 \overline{\tau}_1^3 (3\tau_0 - 1)}{\tau_0^3 (3\overline{\tau}_1 - 1) + \chi_0 \overline{\tau}_1^3} \phi_1$$
$$\phi_n = \frac{\overline{\tau}_n^3 + \chi_n \tau_{n-1}^3 (3\overline{\tau}_n - 1)}{\overline{\tau}_n^3 (3\tau_{n-1} - 1) + \chi_n \tau_{n-1}^3} \theta_{n-1}$$

Using $\phi_1 = -\psi_1 - \theta_1$, the first rearranges to:

$$(\tau_0^3 (3\overline{\tau}_1 - 1) + \chi_0 \overline{\tau}_1^3)\theta_0 + (\tau_0^3 + \chi_0 \overline{\tau}_1^3 (3\tau_0 - 1))\theta_1 = -(\tau_0^3 + \chi_0 \overline{\tau}_1^3 (3\tau_0 - 1))\psi_1.$$

The second should be substituted in to the general equation with $i = n - 1$. This yields:

$$\begin{aligned} & d_{n-1} \overline{\tau}_n \overline{\tau}_{n-1}^2 \theta_{n-2} \\ + (d_{n-1} \overline{\tau}_n \overline{\tau}_{n-1}^2 (3\tau_{n-2} - 1) & + d_{n-2} \tau_{n-2} \tau_{n-1}^2 (3\overline{\tau}_n - 1) \\ & - d_{n-2} \tau_{n-2} \tau_{n-1}^2 \frac{\overline{\tau}_n^3 + \chi_n \tau_{n-1}^3 (3\overline{\tau}_n - 1)}{\overline{\tau}_n^3 (3\tau_{n-1} - 1) + \chi_n \tau_{n-1}^3}) \theta_{n-1} \\ & = -d_{n-1} \overline{\tau}_n \overline{\tau}_{n-1}^2 (3\tau_{n-2} - 1) \psi_{n-1} \end{aligned}$$

This gives n' equations in n' unknowns (θ_0 to θ_{n-1}). The coefficient matrix is tridiagonal. It is more natural to index the entries from 0. Let us write A_i for the subdiagonal, B_i for the main diagonal, and C_i for the superdiagonal. Let us write D_i for the target vector. Then for an open path we have the following formulae:

$$A_i = d_i \overline{\tau}_{i+1} \overline{\tau}_i^2$$

$$B_0 = \begin{cases} 1 & \text{if } \overline{\theta}_0 \text{ given} \\ \tau_0^3 (3\overline{\tau}_1 - 1) + \chi_0 \overline{\tau}_1^3 & \text{otherwise} \end{cases}$$

$$B_i = d_i \overline{\tau}_{i+1} \overline{\tau}_i^2 (3\tau_{i-1} - 1) + d_{i-1} \tau_{i-1} \tau_i^2 (3\overline{\tau}_{i+1} - 1)$$

$$B_{n-1} = \begin{cases} d_{n-1} \overline{\tau}_n \overline{\tau}_{n-1}^2 (3\tau_{n-2} - 1) + d_{n-2} \tau_{n-2} \tau_{n-1}^2 (3\overline{\tau}_n - 1) & \text{if } \overline{\phi}_n \text{ given} \\ d_{n-1} \overline{\tau}_n \overline{\tau}_{n-1}^2 (3\tau_{n-2} - 1) + d_{n-2} \tau_{n-2} \tau_{n-1}^2 (3\overline{\tau}_n - 1) & \\ -d_{n-2} \tau_{n-2} \tau_{n-1}^2 \frac{\overline{\tau}_n^3 + \chi_n \tau_{n-1}^3 (3\overline{\tau}_n - 1)}{\overline{\tau}_n^3 (3\tau_{n-1} - 1) + \chi_n \tau_{n-1}^3}) & \text{otherwise} \end{cases}$$

$$C_0 = \begin{cases} 0 & \text{if } \overline{\theta}_0 \text{ given} \\ \tau_0^3 + \chi_0 \overline{\tau}_1^3 (3\tau_0 - 1) & \text{otherwise} \end{cases}$$

$$C_i = d_{i-1} \tau_{i-1} \tau_i^2$$

$$D_0 = \begin{cases} \overline{\theta}_0 & \text{if } \overline{\theta}_0 \text{ given} \\ -(\tau_0^3 + \chi_0 \overline{\tau}_1^3 (3\tau_0 - 1))\psi_1 & \text{otherwise} \end{cases}$$

$$D_i = -d_i \overline{\tau}_{i+1} \overline{\tau}_i^2 (3\tau_{i-1} - 1)\psi_i - d_{i-1} \tau_{i-1} \tau_i^2 \psi_{i+1}$$

$$D_{n-1} = \begin{cases} -d_{n-1} \overline{\tau}_n \overline{\tau}_{n-1}^2 (3\tau_{n-2} - 1)\psi_{n-1} - d_{n-2} \tau_{n-2} \tau_{n-1}^2 \overline{\phi}_n & \text{if } \overline{\phi}_n \text{ given} \\ -d_{n-1} \overline{\tau}_n \overline{\tau}_{n-1}^2 (3\tau_{n-2} - 1)\psi_{n-1} & \text{otherwise} \end{cases}$$

For a closed path, we have n equations in $n + 2$ unknowns (θ_0 to θ_{n+1}). However, we have not included all the information. Since we have repeated points, we need to identify θ_0 with θ_n and θ_1 with θ_{n+1}. To get a system with n' equations in n' unknowns, we add the equation $\theta_0 - \theta_n = 0$ and substitute in $\theta_{n+1} = \theta_1$. The resulting matrix is not quite tridiagonal but has extra entries on the off-corners. However, it can be written in the form $M + uv^\top$ with M tridiagonal. There is some

freedom in choosing u and v. For simplest computation, we take $u = e_0 + e_{n'-1}$. This means that $v = d_{n'-2}\tau_{n'-2}\tau_{n'-1}^2 e_1 - e_{n'-1}$. With the same notation as above, the matrix M is given by the following formulae:

$$\begin{aligned}
A_i &= d_i\overline{\tau}_{i+1}\overline{\tau}_i^2 \\
B_0 &= 1 \\
B_i &= d_i\overline{\tau}_{i+1}\overline{\tau}_i^2(3\tau_{i-1} - 1) + d_{i-1}\tau_{i-1}\tau_i^2(3\overline{\tau}_{i+1} - 1) \\
B_{n'-1} &= d_{n'-1}\overline{\tau}_{n'}\overline{\tau}_{n'-1}^2(3\tau_{n'-2} - 1) + d_{n'-2}\tau_{n'-2}\tau_{n'-1}^2(3\overline{\tau}_{n'} - 1) + 1 \\
C_0 &= -d_{n'-2}\tau_{n'-2}\tau_{n'-1}^2 \\
C_i &= d_{i-1}\tau_{i-1}\tau_i^2 \\
D_0 &= 0 \\
D_i &= -d_i\overline{\tau}_{i+1}\overline{\tau}_i^2(3\tau_{i-1} - 1)\psi_i - d_{i-1}\tau_{i-1}\tau_i^2\psi_{i+1} \\
D_{n'-1} &= -d_{n'-1}\overline{\tau}_{n'}\overline{\tau}_{n'-1}^2(3\tau_{n'-2} - 1)\psi_{n'-1} - d_{n'-2}\tau_{n'-2}\tau_{n'-1}^2\psi_1
\end{aligned}$$

The next step in the implementation is to compute these coefficients and store them in appropriate arrays. Having done that, we need to solve the resulting tridiagonal system. This is done by looping through the arrays doing the following substitutions (starting at $i = 1$):

$$\begin{aligned}
B'_i &= B'_{i-1}B_i - A_iC'_{i-1} \\
C'_i &= B'_{i-1}C_i \\
D'_i &= B'_{i-1}D_i - A_iD'_{i-1}
\end{aligned}$$

followed by back-substitution:

$$\begin{aligned}
\theta_{n-1} &= D'_{n-1}/B'_{n-1} \\
\theta_i &= (D'_i - C'_i\theta_{i+1})/B'_i
\end{aligned}$$

For a closed path, we run this both with the vector D and the vector $u = e_0 + e_{n'-1}$. Then to get the real answer, we use the Sherman–Morrison formula:

$$(M + uv^\top)^{-1}D = M^{-1}D - \frac{M^{-1}uv^\top M^{-1}D}{1 + v^\top M^{-1}u}.$$

This leaves us with the values for θ_i. We now substitute these into Hobby's formulae for the lengths:

$$\begin{aligned}
\rho_i &= \frac{2 + \alpha_i}{1 + (1-c)\cos\theta_i + c\cos\phi_{i+1}} \\
\sigma_{i+1} &= \frac{2 - \alpha_i}{1 + (1-c)\cos\phi_{i+1} + c\cos\theta_i} \\
\alpha_i &= a(\sin\theta_i - b\sin\phi_{i+1})(\sin\phi_{i+1} - b\sin\theta_i)(\cos\theta_i - \cos\phi_{i+1})
\end{aligned}$$

where $a = \sqrt{2}$, $b = 1/16$, and $c = (3 - \sqrt{5})/2$. These are actually the *relative* lengths so need to be adjusted by a factor of $d_i/3$. Now θ_i is the angle relative to the line from z_i to z_{i+1}, so to get the true angle we need to add back that angle. Fortunately, we stored those angles at the start. So the control points are:

$$\begin{aligned}
d_i\rho_i(\cos(\theta_i + \omega_i), \sin(\omega_{i+1} - \phi_{i+1}))/3 + z_i \\
-d_i\sigma_{i+1}(\cos(\omega_{i+1} - \phi_{i+1}), \sin(\theta_i + \omega_i))/3 + z_{i+1}
\end{aligned}$$

6 A Piecewise Version of Hobby's Algorithm

Here we present a variant of Hobby's algorithm. One difficulty with Hobby's algorithm is that it works with the path as a whole. It is therefore not possible to build up a path piecewise. We therefore modify it to correct for this. Obviously, the resulting path will be less "ideal", but will have the property that adding new points will not affect earlier segments. The method we use is to employ Hobby's algorithm on the two-segment subpaths. This provides two cubic Bezier curves: one from the kth point to the $k+1$st point and the second from the $k+1$st to the $k+2$nd. Of this data, we keep the first segment and use that for the path between the kth and $k+1$st points. We also remember the outgoing angle of the first segment and use that as the incoming angle on the next computation (which will involve the $k+1$st, $k+2$nd, and $k+3$rd) points. The two ends are slightly different to the middle segments. On the first segment, we might have no incoming angle. On the last segment, we render both pieces. This means that for the initial segment, we have a 2×2 linear system:

$$\begin{bmatrix} B_0 & C_0 \\ A_1 & B_1 \end{bmatrix} \Theta = \begin{bmatrix} D_0 \\ D_1 \end{bmatrix}$$

This has solution:

$$\Theta = \frac{1}{B_0 B_1 - C_0 A_1} \begin{bmatrix} B_1 & -C_0 \\ -A_1 & B_0 \end{bmatrix} \begin{bmatrix} D_0 \\ D_1 \end{bmatrix} = \frac{1}{B_0 B_1 - C_0 A_1} \begin{bmatrix} B_1 D_0 - C_0 D_1 \\ B_0 D_1 - A_1 D_0 \end{bmatrix}$$

Now we have the following values for the constants:

$$\begin{aligned}
A_1 &= d_1 \overline{\tau}_2 \overline{\tau}_1^2 \\
B_0 &= \tau_0^3 (3\overline{\tau}_1 - 1) + \chi_0 \overline{\tau}_1^3 \\
B_1 &= d_1 \overline{\tau}_2 \overline{\tau}_1^2 (3\tau_0 - 1) + d_0 \tau_0 \tau_1^2 (3\overline{\tau}_2 - 1) - d_0 \tau_0 \tau_1^2 \frac{\overline{\tau}_2^3 + \chi_2 \tau_1^3 (3\overline{\tau}_2 - 1)}{\overline{\tau}_2^3 (3\tau_1 - 1) + \chi_2 \tau_1^3} \\
C_0 &= \tau_0^3 + \chi_0 \overline{\tau}_1^3 (3\tau_0 - 1) \\
D_0 &= -(\tau_0^3 + \chi_0 \overline{\tau}_1^3 (3\tau_0 - 1))\psi_1 \\
D_1 &= -d_1 \overline{\tau}_2 \overline{\tau}_1^2 (3\tau_0 - 1)\psi_1
\end{aligned}$$

Let us, as we are aiming for simplicity, assume that the tensions and curls are all 1. Then we have $A_1 = d_1$, $B_0 = 3$, $B_1 = 2d_1 + 2d_0 - d_0 = 2d_1 + d_0$, $C_0 = 3$, $D_0 = -3\psi_1$, $D_1 = -2d_1\psi_1$. Thus the linear system is:

$$\begin{bmatrix} 3 & 3 \\ d_1 & 2d_1 + d_0 \end{bmatrix} \Theta = -\psi_1 \begin{bmatrix} 3 \\ 2d_1 \end{bmatrix}$$

which we can row reduce to:

$$\begin{bmatrix} 1 & 1 \\ 0 & d_1 + d_0 \end{bmatrix} \Theta = -\psi_1 \begin{bmatrix} 1 \\ d_1 \end{bmatrix}$$

whence $\theta_1 = -\psi_1 \frac{d_1}{d_0 + d_1}$ and $\theta_0 = -\psi_1 - \theta_1 = -\psi_1 \frac{d_0}{d_0 + d_1}$. We also compute $\phi_1 = -\psi_1 - \theta_1 = \theta_0$ and $\phi_2 = \theta_1$ (in the simple version). We use θ_0 and ϕ_1 to compute the bezier curve of the first segment, make a note of θ_1, and – assuming there are more segments – throw away ϕ_2.

For the inner segments, we have the system:

$$\begin{bmatrix} 1 & 0 \\ A_1 & B_1 \end{bmatrix} \Theta = \begin{bmatrix} \theta_0 \\ D_1 \end{bmatrix}$$

which has the solution $\theta_1 = (D_1 - A_1\theta_0)/B_1$. The values of the constants in this case are:

$$\begin{aligned}
A_1 &= d_1 \overline{\tau}_2 \overline{\tau}_1^2 \\
B_1 &= d_1 \overline{\tau}_2 \overline{\tau}_1^2 (3\tau_0 - 1) + d_0 \tau_0 \tau_1^2 (3\overline{\tau}_2 - 1) - d_0 \tau_0 \tau_1^2 \frac{\overline{\tau}_2^3 + \chi_2 \tau_1^3 (3\overline{\tau}_2 - 1)}{\overline{\tau}_2^3 (3\tau_1 - 1) + \chi_2 \tau_1^3} \\
D_1 &= -d_1 \overline{\tau}_2 \overline{\tau}_1^2 (3\tau_0 - 1)\psi_1
\end{aligned}$$

Again, let us consider the simpler case. Then $A_1 = d_1$, $B_1 = 2d_1 + d_0$, and $D_1 = -2d_1\psi_1$. Thus $\theta_1 = (-2d_1\psi_1 - d_1\theta_0)/(2d_1 + d_0) = -(2\psi_1 + \theta_0)\frac{d_1}{2d_1+d_0}$. We compute $\phi_1 = -\psi_1 - \theta_1 = \frac{-\psi_1 d_0 + \theta_0 d_1}{2d_1+d_0}$ and $\phi_2 = \theta_1$.

References

[1] John D. Hobby. Smooth, easy to compute interpolating splines. *Discrete Comput. Geom.*, 1:123–140, 1986.