

The Hobby package

Andrew Stacey

`stacey@math.ntnu.no`

v1.2 from 2013/04/12

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. 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. As this is intended as a simpler method, it does not (at present) admit the same level of customisation as the full implementation. The “quick” algorithm is described in full in Section 5.

The full algorithm is implemented in \LaTeX 3 with (almost*) no reference to TikZ or PGF. It makes extensive use of the `fp` and `prop` libraries for the computation steps. The TikZ library is simply a wrapper that takes the user’s input, converts it into the right format for the \LaTeX 3 code, and then calls that code to generate the path. The “quick” version does not use \LaTeX 3 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

*At the moment, \LaTeX 3 lacks a `atan2` function so `PGFMath` is used to remedy that.

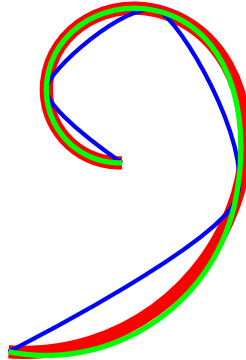


Figure 1: Comparison of the three algorithms

TikZ/PGF. The green curve uses the “quick” version. Figure 2 compares the implementation with that given by MetaPost.

2 Usage

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

```
\usetikzlibrary{hobby}
```

There are a variety of ways of specifying the data to the algorithm to generate the curve.

2.1 As a to path.

The key `curve through={<points>}` installs a `to path` which draws a smooth curve through the given points. The points should be specified as a list which can be optionally separated by dots. The purpose of allowing the dots is to make it simpler to switch between the `to path` method and the `shortcut` method (described in Section 2.2). However, note that the two methods are not completely synonymous due to how one can specify options so care must still be taken when switching.

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

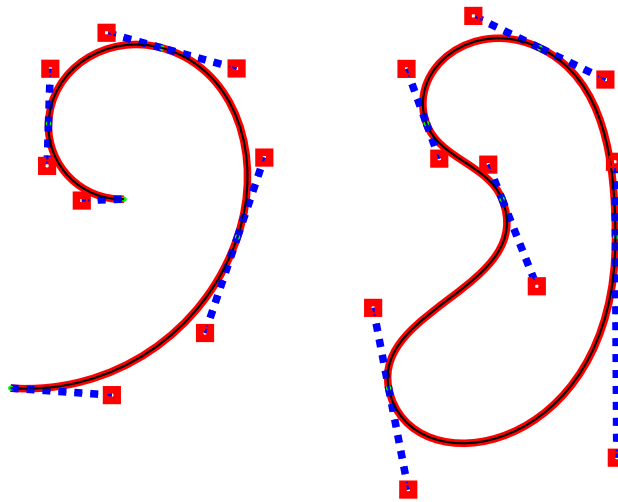
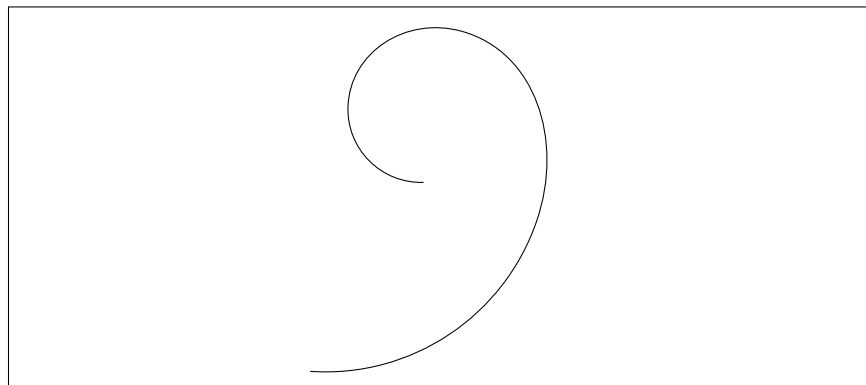
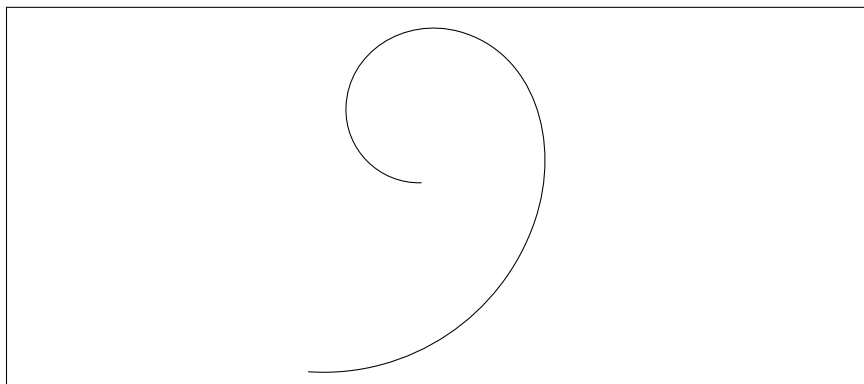


Figure 2: Hobby's algorithm in TikZ overlaying the output of MetaPost

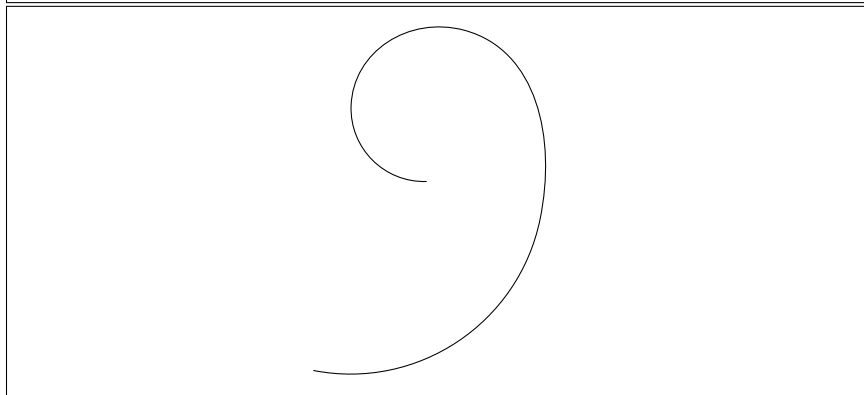


```
\begin{tikzpicture}[scale=.5]
\draw (0,0) to[curve through={(6,4) (4,9) (1,7)}] (3,5);
\end{tikzpicture}
```



There is a corresponding key `quick curve through={<points>}` which uses the “quick” algorithm. Again, the dots are optional.

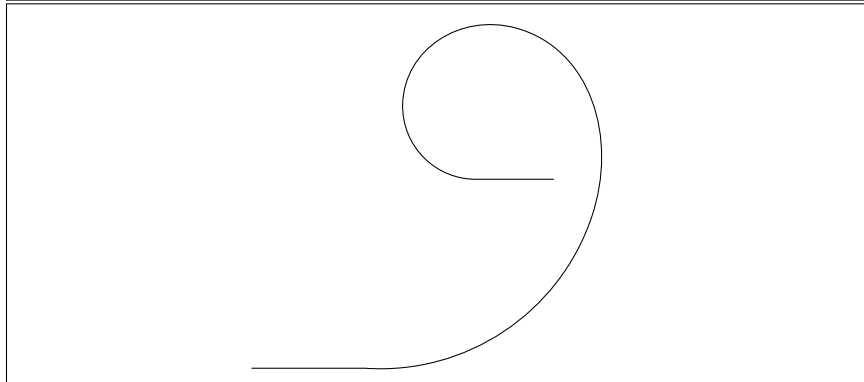
```
\begin{tikzpicture}[scale=.5]
\draw (0,0) to[quick curve through={(6,4) .. (4,9) ..
(1,7)}] (3,5);
\end{tikzpicture}
```



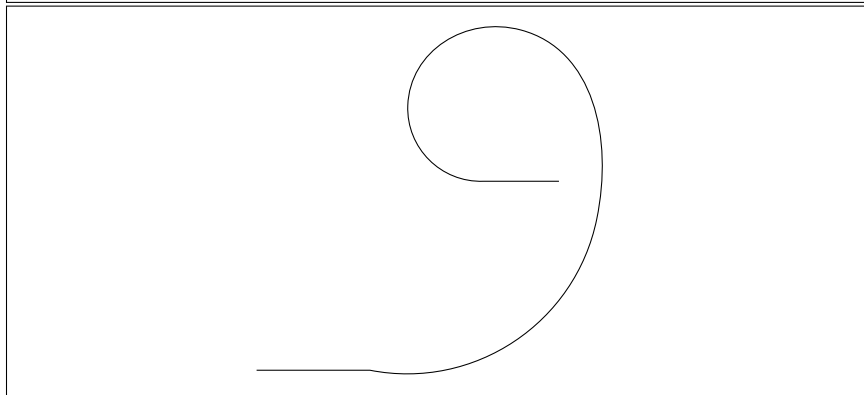
2.2 The shortcut Method

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` (which can be set globally if so desired).

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



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



2.3 The Plot Handler Method

The algorithms can also be used via the `plot handler` syntax. This library registers three plot handlers: `hobby`, `closed hobby`, and `quick hobby`. The first is an open curve through the points using the full algorithm, the second is a closed curve, and the third uses the quick algorithm (and is thus an open curve).

```

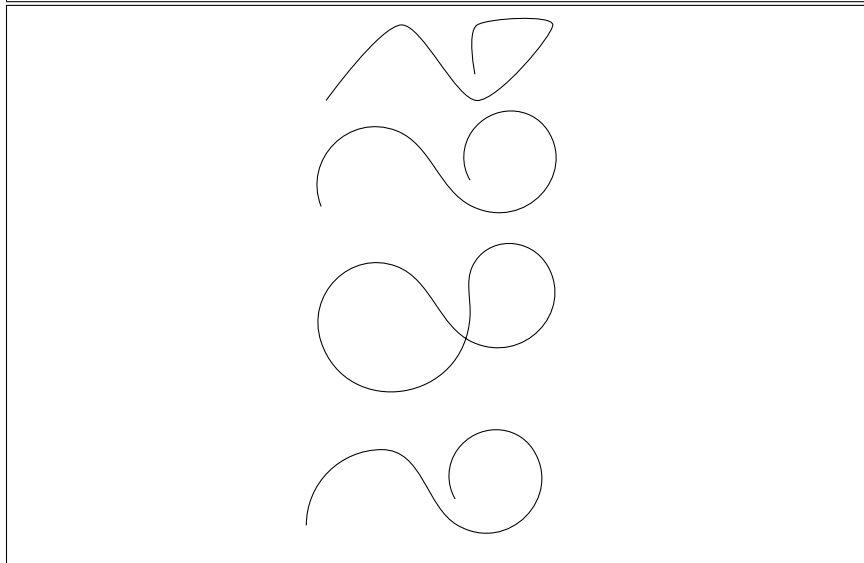
\tikz[smooth] \draw plot coordinates {(0,0) (1,1) (2,0)
(3,1) (2,1) (10:2cm)};

\tikz[hobby] \draw plot coordinates {(0,0) (1,1) (2,0)
(3,1) (2,1) (10:2cm)};

\tikz[closed hobby] \draw plot coordinates {(0,0) (1,1)
(2,0) (3,1) (2,1) (10:2cm)};

\tikz[quick hobby] \draw plot coordinates {(0,0) (1,1)
(2,0) (3,1) (2,1) (10:2cm)};

```

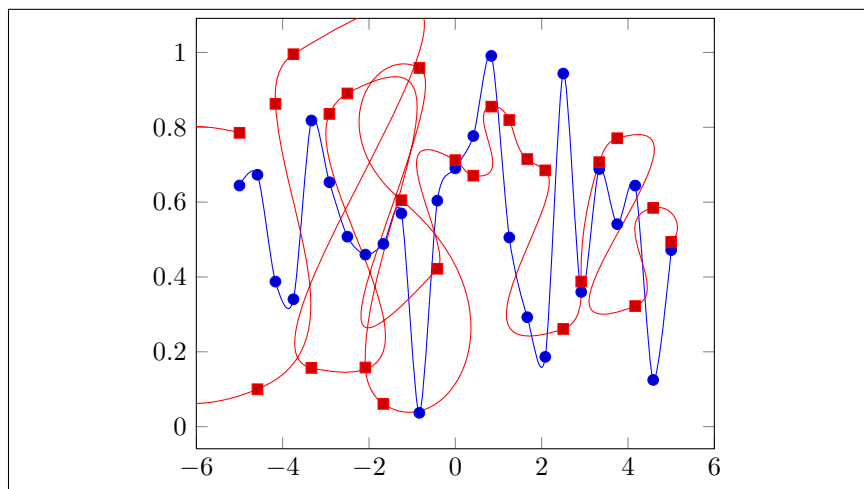


This has the side effect that these can be used with the `pgfplots` package. However, the Hobby algorithm is designed to draw a curve in 2D-space and does not take into account the fact that when plotting a graph then the two dimensions are treated differently.

```

\begin{tikzpicture}
\begin{axis}
\addplot+[smooth] {rnd};
\addplot+[hobby] {rnd};
\end{axis}
\end{tikzpicture}

```



3 Customisation

There are various ways to customise the path generated by the Hobby algorithms. The full algorithm has a variety of parameters which can be varied to produce different paths through the same points. These vary from specifying that the path be open or closed, to specifying “tensions” at each point to change how the path approaches or leaves it.

3.1 Algorithm Customisations

The main algorithm (i.e., not the “quick” variant) 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. When using the `to path` specification, the parameters can be specified before or after the `curve through` key or as options to the coordinates. When using the `shortcut` specification, the parameters can be given on the path or on coordinates.

On occasion, it is ambiguous which curve an option belongs to. This is most likely if a coordinate happens to belong to two curves, or if a coordinate is parsed before TikZ knows that it is constructing a curve using this library. The simplest solution is to move the option to a place where there is no ambiguity. Other solutions to this problem will be detailed later.

Let us start with the customisations to the algorithm.

- Basic curve.

```

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

```

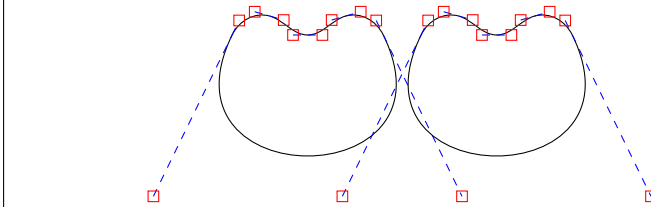


- The path can be open, as above, or closed.

```

\begin{tikzpicture}[scale=.5]
\draw[postaction=show curve controls]
(0,0) to[closed,curve through={(1,.5) .. (2,0) ..
(3,.5)}] (4,0);
\draw[xshift=5cm,use Hobby shortcut,postaction=show
curve controls]
([closed]0,0) .. (1,.5) .. (2,0) .. (3,.5) .. (4,0);
\end{tikzpicture}

```

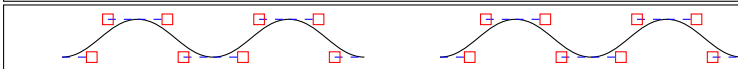


- Specifying the angle at which the curve goes *out* and at which it comes *in*. The angles given are absolute.

```

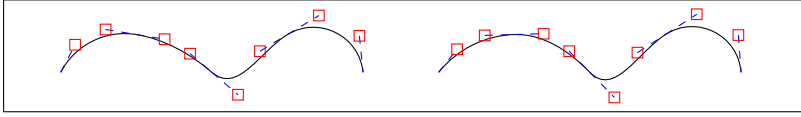
\begin{tikzpicture}
\draw[postaction=show curve controls]
(0,0) to[out angle=0,in angle=180,curve
through={(1,.5) .. (2,0) .. (3,.5)}] (4,0);
\draw[xshift=5cm,use Hobby shortcut,postaction=show
curve controls]
([out angle=0,in angle=180]0,0) .. (1,.5) .. (2,0)
.. (3,.5) .. (4,0);
\end{tikzpicture}

```



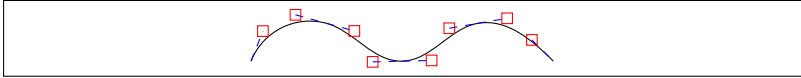
- Applying tension as the curve comes in to a point.

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

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

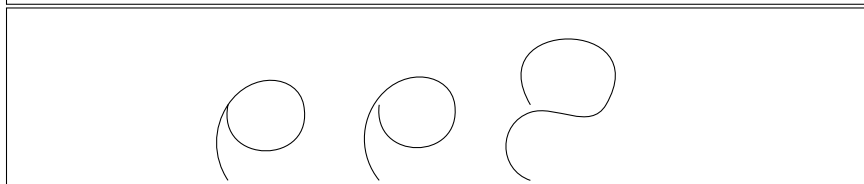


3.2 Edge Cases

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,

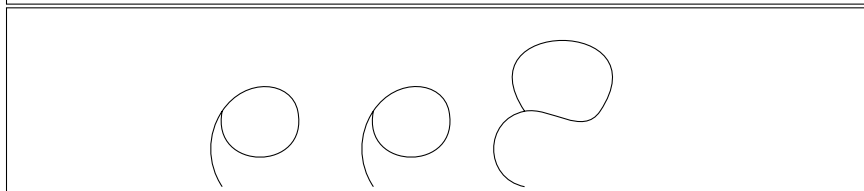
add 2π . This will occur if the path “doubles back” on itself as in the next example. By nudging the repeated point slightly, the behaviour changes drastically.

```
\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}
```



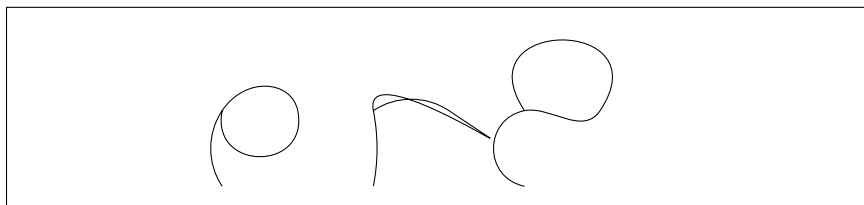
Due to the precision of the computations, it is not possible to always get this test correct. The simplest solution is to nudge the repeated point in one direction or the other. Experimenting shows that the “nudge factor” can be extremely small (note that it will be proportional to the distance between 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}
```

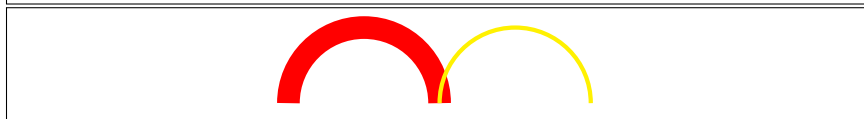


Although this is intended to be an integer, no check is done and so some quite odd curves can result from changing this parameter.

3.3 Reusing Paths

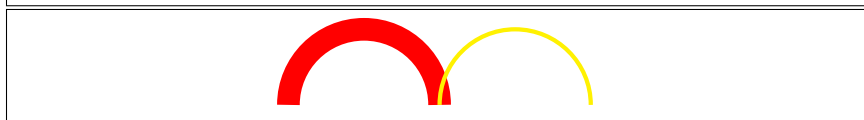
Although the (full) algorithm has good theoretical computation time, using \TeX for its implementation does not provide for fast runs. The externalisation library of TikZ/PGF can be used to save whole pictures, but it can be useful to save a generated path within a single `tikzpicture` for later use within that same picture. The implementation allows for this by separating the generation of the path from its use.

```
\begin{tikzpicture}
\draw[line width=3mm,red,use Hobby shortcut,save Hobby
      path={saved}] (0,0) .. (1,1) .. (2,0);
\draw[xshift=2cm,ultra thick,yellow] (0,0) [restore and
      use Hobby path={saved}{}];
\end{tikzpicture}
```



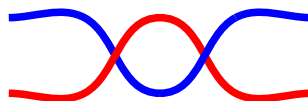
Note that the key `restore and use Hobby path` is given *after* the initial `(0,0)`. This is because by default the path generated by the Hobby algorithm does not start with an explicit `moveto` since that is the standard behaviour of all of PGF's path construction macros. So the `(0,0)` ensures that our path is well-formed by issuing an initial `moveto`. An alternative would be to use the key `disjoint` which does add an initial `moveto`.

```
\begin{tikzpicture}
\draw[line width=3mm,red,use Hobby shortcut,save Hobby
      path={saved}] (0,0) .. (1,1) .. (2,0);
\draw[xshift=2cm,ultra thick,yellow,restore and use
      Hobby path={saved}{disjoint}];
\end{tikzpicture}
```



An example of where this is useful is in drawing knot diagrams. When so doing, it is sometimes convenient to draw a path (or segment of a path) twice in order to get the under/over crossings correct. For this situation, it can be useful to designate certain parts of the path as **blank**, whereby we mean to redraw them later. The point of a blank segment of a curve is that it is still taken into account when computing the algorithm but is left blank when it comes to rendering. A path can then be redrawn with the blank/non-blank segments reversed. As it might be desired to have only some blank segments drawn the second time, there are two types of blank. Only a **soft** blank will be reversed in these circumstances.

```
\begin{tikzpicture}[use Hobby shortcut,line
width=1mm,rotate=90]
\draw[blue,save Hobby path={left}]
([out angle=90,in angle=-90]1,0) .. (1,1) ..
([blank=soft]0,2) .. (1,3) .. (1,4);
\draw[red] ([out angle=90,in angle=-90]0,0) .. (0,1) ..
(1,2) .. (0,3) .. (0,4);
\draw[blue,restore and use Hobby
path={left}{disjoint,invert soft blanks}];
\end{tikzpicture}
```



This can be taken a step further. The generated data can be saved to the **aux** file and read back in, avoiding the need to regenerate it on each run. To engage this facility, the Hobby path has to be named (via **save Hobby path**) and the key **Hobby externalise** (or **Hobby externalize**) must be given in a context that applies (such as on the path or on the surrounding scope).

The relevant keys are the following.

- **use previous Hobby path[=<options>]**. This (re)uses the previously generated Hobby path. As all the data is globally stored, this can technically be in a different **tikzpicture**. The **<options>** will be applied, in so far as they are options that can be applied after the algorithm has run.
- **save Hobby path=<name>**. Saves a path for later use. The path is saved in a global macro so can be reused in another picture.
- **restore Hobby path=<name>**. This restores the named Hobby path (if it exists). It does not *use* it. After this key, **use previous Hobby path** will use the restored path.
- **restore and use Hobby path={<name>}{<options>}**. This restores the named path and uses it with **<options>** applied.

- **Hobby externalise** or **Hobby externalize**. This puts in place the code for saving the generated data to the **aux** file. On subsequent runs, it uses the saved data rather than the current data. For a curve to make use of this, it has to be named via the **save Hobby path** key. So to regenerate the data, either delete the **aux** file or remove the **save Hobby path** key for one compilation run.

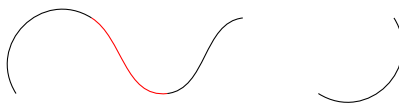
The options that can be applied are those that affect the rendering of the curve but not its generation. When the curve is rendered (or *used*, in the above parlance), \TeX steps along the coordinates of the generated curve and carries out an action for each piece. This action can be modified after the curve has been generated. The action will be one of:

- Move to the end point (ignoring the control points).
- Draw a Bezier curve to the end point through the control points.
- Draw a Bezier curve to the end point through the control points and then move to the end point.

The last is subtle: the move doesn't actually go anywhere but it “breaks” the curve at the designated point. In particular, a later **cycle** would return to this point (or a later break) rather than to the start of the curve.

These actions are triggered by the keys **blank** and **break**. Each should be specified to the coordinate at the *end* of the segment under consideration. The **blank** key can be given the argument **soft**. The effect of this is seen when the key **invert soft blanks** is used. This swaps the drawing action so that non-blank segments are skipped and *soft* blanks are drawn. Non-soft-blank segments are still not drawn.

```
\begin{tikzpicture}[use Hobby shortcut]
\draw (0,0) .. (1,1) .. ([blank=soft]2,0) .. (3,1) ..
    ([blank]4,0) .. (5,1);
\draw[red,use previous Hobby path={invert soft
    blanks,disjoint}];
\end{tikzpicture}
```

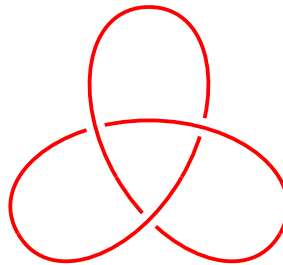


As a more practical application, consider the following rendering of a trefoil knot.

```

\begin{tikzpicture}[
  use Hobby shortcut,
  every path/.style={
    line width=1mm,
    white,
    double=red,
    double distance=.5mm
  }
]
\draw ([closed]0,2) .. ([blank=soft]210:.5) .. (-30:2) ..
([blank=soft]0,.5) .. (210:2) .. ([blank=soft]-30:.5);
\draw[use previous Hobby path={invert soft
  blanks, disjoint}];
\end{tikzpicture}

```

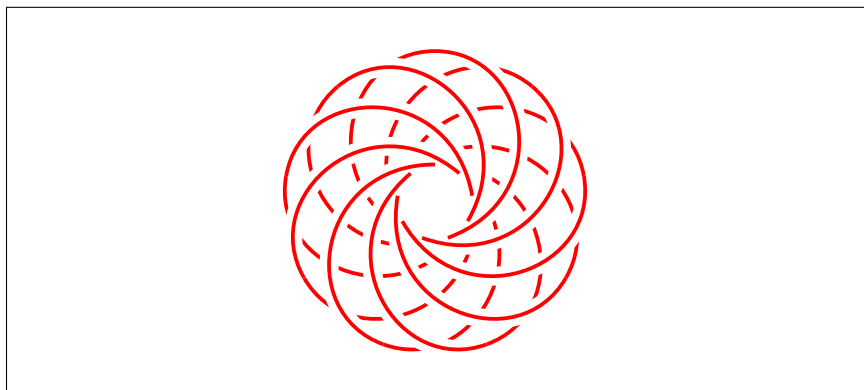


This could easily be generalised using the `\foreach` command, as demonstrated in the next code.

```

\begin{tikzpicture}[
  use Hobby shortcut,
  every path/.style={
    line width=1mm,
    white,
    double=red,
    double distance=.5mm
  }
]
\def\nfoil{9}
\draw ([closed]0,2)
\foreach \k in {1,...,\nfoil} {
  .. ([blank=soft]90+360*\k/\nfoil-180/\nfoil:-.5) ..
  (90+360*\k/\nfoil:2)
};
\draw[use previous Hobby path={invert soft
  blanks, disjoint}];
\end{tikzpicture}

```



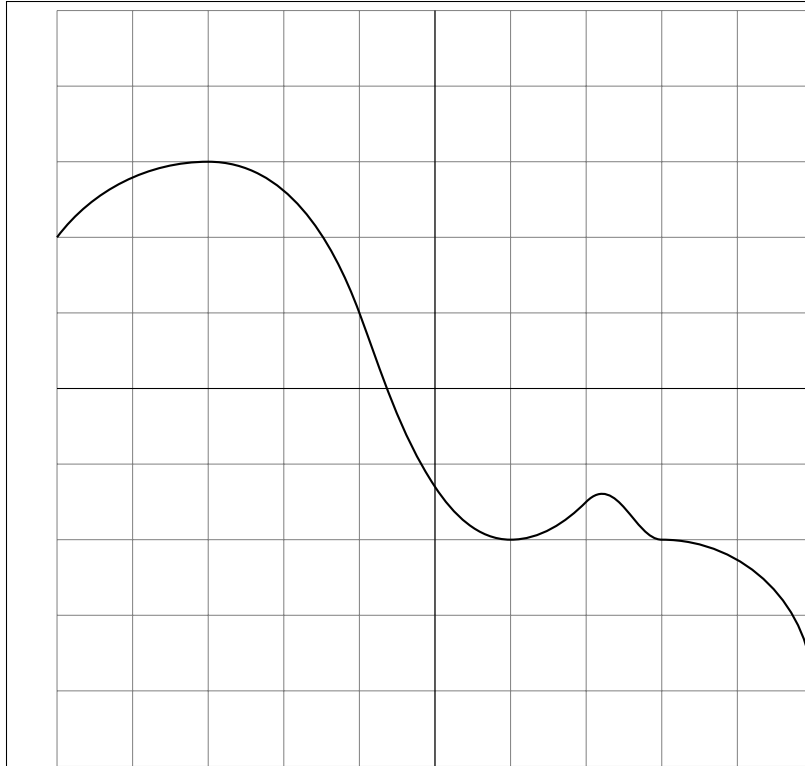
3.4 Breaking the Path

One issue with the shortcut notation is that it is not possible (using this notation) to have two sets of curves following directly on from each other because there is no clear demarcation of the boundary. To make this possible, there is a key `Hobby action`, which installs an action to be taken after the point has been processed. The general key `Hobby action={code}` can install arbitrary code. Probably the more useful variant is `Hobby finish` which runs the algorithm on the points gathered so far. An example of the use of this is to make it possible to specify tangencies at certain points. Technically, once a tangent direction has been specified, the Hobby algorithm splits the set of points there and works on each piece separately. The following key implements this, the technicalities are due to the fact that the tangent angle has to be used twice: once to specify the angle of the path coming in to that point and once to specify the angle of the path coming out. Note that specifying the tangent vector at every point means that the algorithm is not actually being used. However, Hobby's formulae for the lengths of the control points is still being used.

```

\begin{tikzpicture}[
  use Hobby shortcut ,
  tangent/.style={%
    in angle={180+#1}},
  Hobby finish ,
  designated Hobby path=next ,
  out angle=#1,
  },
]
\draw[help lines] (-5,-5) grid (5,5);
\draw (-5,0) -- (5,0) (0,-5) -- (0,5);
\draw[thick] (-5,2) .. ([tangent=0]-3,3) .. (-1,1) ..
  (0,-1.3) .. %
  ([tangent=0]1,-2) .. ([tangent=45]2,-1.5) ..
  ([tangent=0]3,-2) .. (5,-4);
\end{tikzpicture}

```



4 Implementing Hobby's Algorithm

We start with a list of $n+1$ points, z_0, \dots, z_n . The base code assumes that these are already stored in two arrays[†]: 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 $z_n = z_0$ [‡]. 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 $n+1$ [§]. 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 k th point and the $k+1$ th point. For the internal nodes[¶], we store the difference in the angles in `\l_hobby_psi_array`. The k th value on this is the angle subtended at the k th 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, $\bar{\tau}_i$
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:

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

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

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

[†]Arrays are thinly disguised property lists.

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

[§]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.

[¶]Hobby calls the specified points *knots*.

Substituting in yields:

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

Let us rearrange that to the following:

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

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^{||}. Therefore for $i = 1$ to $n' - 1$ the above simplifies to the following:

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

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: $\bar{\theta}_0$ and $\bar{\phi}_n$. 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 above. If not, they are given by formulae involving “curl” parameters χ_0 and χ_n and result in the equations:

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

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

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

^{||}Recall that by dint of repetition, all nodes are effectively internal for a closed path.

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

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

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:

$$\begin{aligned}
A_i &= d_i\bar{\tau}_{i+1}\bar{\tau}_i^2 \\
B_0 &= \begin{cases} 1 & \text{if } \bar{\theta}_0 \text{ given} \\ \tau_0^3(3\bar{\tau}_1 - 1) + \chi_0\bar{\tau}_1^3 & \text{otherwise} \end{cases} \\
B_i &= d_i\bar{\tau}_{i+1}\bar{\tau}_i^2(3\tau_{i-1} - 1) + d_{i-1}\tau_{i-1}\tau_i^2(3\bar{\tau}_{i+1} - 1) \\
B_{n-1} &= \begin{cases} d_{n-1}\bar{\tau}_n\bar{\tau}_{n-1}^2(3\tau_{n-2} - 1) + d_{n-2}\tau_{n-2}\tau_{n-1}^2(3\bar{\tau}_n - 1) & \text{if } \bar{\phi}_n \text{ given} \\ d_{n-1}\bar{\tau}_n\bar{\tau}_{n-1}^2(3\tau_{n-2} - 1) + d_{n-2}\tau_{n-2}\tau_{n-1}^2(3\bar{\tau}_n - 1) \\ - d_{n-2}\tau_{n-2}\tau_{n-1}^2 \frac{\bar{\tau}_n^3 + \chi_n\tau_{n-1}^3(3\bar{\tau}_n - 1)}{\bar{\tau}_n^3(3\tau_{n-1} - 1) + \chi_n\tau_{n-1}^3} & \text{otherwise} \end{cases} \\
C_0 &= \begin{cases} 0 & \text{if } \bar{\theta}_0 \text{ given} \\ \tau_0^3 + \chi_0\bar{\tau}_1^3(3\tau_0 - 1) & \text{otherwise} \end{cases} \\
C_i &= d_{i-1}\tau_{i-1}\tau_i^2 \\
D_0 &= \begin{cases} \bar{\theta}_0 & \text{if } \bar{\theta}_0 \text{ given} \\ -(\tau_0^3 + \chi_0\bar{\tau}_1^3(3\tau_0 - 1))\psi_1 & \text{otherwise} \end{cases} \\
D_i &= -d_i\bar{\tau}_{i+1}\bar{\tau}_i^2(3\tau_{i-1} - 1)\psi_i - d_{i-1}\tau_{i-1}\tau_i^2\psi_{i+1} \\
D_{n-1} &= \begin{cases} -d_{n-1}\bar{\tau}_n\bar{\tau}_{n-1}^2(3\tau_{n-2} - 1)\psi_{n-1} - d_{n-2}\tau_{n-2}\tau_{n-1}^2\bar{\phi}_n & \text{if } \bar{\phi}_n \text{ given} \\ -d_{n-1}\bar{\tau}_n\bar{\tau}_{n-1}^2(3\tau_{n-2} - 1)\psi_{n-1} & \text{otherwise} \end{cases}
\end{aligned}$$

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 $v = d_{n'-2}\tau_{n'-2}\tau_{n'-1}^2e_1 - e_{n'-1}$. With the same notation as above,

the matrix M is given by the following formulae:

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

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 $i = 1$):

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

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:

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

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

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

$$\text{where } \alpha_i = a(\sin \theta_i - b \sin \phi_{i+1})(\sin \phi_{i+1} - b \sin \theta_i)(\cos \theta_i - \cos \phi_{i+1})$$

and $a = \sqrt{2}$, $b = 1/16$, 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:

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

5 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 two-segment subpaths. When applied to a two-segment subpath, the algorithm provides two cubic Bezier curves: one from the k th 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 k th 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:

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

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:

$$A_1 = d_1 \bar{\tau}_2 \bar{\tau}_1^2$$

$$B_1 = d_1 \bar{\tau}_2 \bar{\tau}_1^2 (3\tau_0 - 1) + d_0 \tau_0 \tau_1^2 (3\bar{\tau}_2 - 1) - d_0 \tau_0 \tau_1^2 \frac{\bar{\tau}_2^3 + \chi_2 \tau_1^3 (3\bar{\tau}_2 - 1)}{\bar{\tau}_2^3 (3\tau_1 - 1) + \chi_2 \tau_1^3}$$

$$D_1 = -d_1 \bar{\tau}_2 \bar{\tau}_1^2 (3\tau_0 - 1) \psi_1$$

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 $\phi_2 = \theta_1$. Then we store θ_1 for the next iteration.

The actual curves are then produced from the angles using the same formulae for the lengths of the control points as in the main algorithm.

At the last stage, we render both segments of the generated curve.

References

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