The Hobby package: code

Andrew Stacey
loopspace@mathforge.org

1.6 from 2014-08-11

1 Implementation

1.1 Main Code

We use LaTeX3 syntax so need to load the requisite packages

```
\RequirePackage{expl3}
\RequirePackage{xparse}
\RequirePackage{pml3array}
\ExplSyntaxOn
```

```
\cs_generate_variant:Nn \fp_set:Nn {Nx}
\cs_generate_variant:Nn \tl_if_eq:nnTF {VnTF}
\cs_generate_variant:Nn \tl_if_eq:nnTF {xnTF}
```

1.1.1 Initialisation

We declare all our variables.

Start with version and date, together with a check to see if we've been loaded twice (fail gracefully if so).

```
\tl_clear:N \l_tmpa_tl
\tl_if_exist:NT \g__hobby_version
{
  \tl_set:Nn \l_tmpa_tl {
    \ExplSyntaxOff
    \tl_clear:N \l_tmpa_tl
    \endinput
  }
}
\tl_use:N \l_tmpa_tl

\tl_new:N \g__hobby_version
\tl_new:N \g__hobby_date
\tl_set:Nn \g__hobby_version {1.6}
\tl_set:Nn \g__hobby_date {2014-08-11}
\DeclareDocumentCommand \hobbyVersion {}
{
  \tl_use:N \g__hobby_version
}
\DeclareDocumentCommand \hobbyDate {}
{
  \tl_use:N \g__hobby_date
}
```

The function for computing the lengths of the control points depends on three parameters. These are set to $a = \sqrt{2}$, $b = 1/16$, and $c = \frac{3-\sqrt{5}}{2}$.

```
\fp_new:N \g_hobby_parama_fp
\fp_new:N \g_hobby_paramb_fp
\fp_new:N \g_hobby_paramc_fp
\fp_gset:Nn \g_hobby_parama_fp {2^.5}
\fp_gset:Nn \g_hobby_paramb_fp {1/16}
\fp_gset:Nn \g_hobby_paramc_fp {(3-5^.5)/2}
```

Now we define our objects for use in generating the path.

\l_hobby_closed_bool

\l_hobby_closed_bool is true if the path is closed.

```
\bool_new:N \l_hobby_closed_bool
```

\l_hobby_disjoint_bool

\l_hobby_disjoint_bool is true if the path should start with a moveto command.

```
\bool_new:N \l_hobby_disjoint_bool
```

\l_hobby_save_aux_bool

\l_hobby_save_aux_bool is true if when saving paths then they should be saved to the aux file.

```
\bool_new:N \l_hobby_save_aux_bool
\bool_set_true:N \l_hobby_save_aux_bool
\DeclareDocumentCommand \HobbyDisableAux {}
{
  \bool_set_false:N \l_hobby_save_aux_bool
}
```

\l_hobby_points_array

\l_hobby_points_array is an array holding the specified points on the path. In the LaTeX3 code, a "point" is a token list of the form x = <number>, y = <number>. This gives us the greatest flexibility in passing points back and forth between the LaTeX3 code and any calling code. The array is indexed by integers beginning with 0. In the documentation, we will use the notation z_k to refer to the kth point.

```
\array_new:N \l_hobby_points_array
```

l_hobby_points_x_array

\l_hobby_points_x_array is an array holding the x–coordinates of the specified points.

```
\array_new:N \l_hobby_points_x_array
```

l_hobby_points_y_array

\l_hobby_points_y_array is an array holding the y–coordinates of the specified points.

```
\array_new:N \l_hobby_points_y_array
```

\l_hobby_actions_array

\l_hobby_actions_array is an array holding the (encoded) action to be taken out on the segment of the path ending at that point.

```
\array_new:N \l_hobby_actions_array
```

\l_hobby_angles_array

\l_hobby_angles_array is an array holding the angles of the lines between the points. Specifically, the angle indexed by k is the angle in radians of the line from z_k to z_{k+1}.

```
\array_new:N \l_hobby_angles_array
```

_hobby_distances_array

\l_hobby_distances_array is an array holding the distances between the points. Specifically, the distance indexed by k, which we will write as d_k, is the length of the line from z_k to z_{k+1}.

```
\array_new:N \l_hobby_distances_array
```

obby_tension_out_array

\l_hobby_tension_out_array is an array holding the tension for the path as it leaves each point. This is a parameter that controls how much the curve "flexes" as it leaves the point. In the following, this will be written τ_k.

```
\array_new:N \l_hobby_tension_out_array
```

hobby_tension_in_array

\l_hobby_tension_in_array is an array holding the tension for the path as it arrives at each point. This is a parameter that controls how much the curve "flexes" as it gets to the point. In the following, this will be written $\overline{\tau}_k$.

```
\array_new:N \l_hobby_tension_in_array
```

l_hobby_matrix_a_array

\l_hobby_matrix_a_array is an array holding the subdiagonal of the linear system that has to be solved to find the angles of the control points. In the following, this will be denoted by A_i. The first index is 1.

```
\array_new:N \l_hobby_matrix_a_array
```

l_hobby_matrix_b_array

\l_hobby_matrix_b_array is an array holding the diagonal of the linear system that has to be solved to find the angles of the control points. In the following, this will be denoted by B_i. The first index is 0.

```
\array_new:N \l_hobby_matrix_b_array
```

l_hobby_matrix_c_array

\l_hobby_matrix_c_array is an array holding the superdiagonal of the linear system that has to be solved to find the angles of the control points. In the following, this will be denoted by C_i. The first index is 0.

```
\array_new:N \l_hobby_matrix_c_array
```

l_hobby_matrix_d_array

\l_hobby_matrix_d_array is an array holding the target vector of the linear system that has to be solved to find the angles of the control points. In the following, this will be denoted by D_i. The first index is 1.

```
\array_new:N \l_hobby_matrix_d_array
```

l_hobby_vector_u_array

\l_hobby_vector_u_array is an array holding the perturbation of the linear system for closed paths. The coefficient matrix for an *open* path is tridiagonal and that means that Gaussian elimination runs faster than expected ($O(n)$ instead of $O(n^3)$). The matrix for a closed path is not tridiagonal but is not far off. It can be solved by perturbing it to a tridiagonal matrix and then modifying the result. This array represents a utility vector in that perturbation. In the following, the vector will be denoted by u. The first index is 1.

```
\array_new:N \l_hobby_vector_u_array
```

bby_excess_angle_array

\l_hobby_excess_angle_array is an array that allows the user to say that the algorithm should add a multiple of 2π to the angle differences. This is because these angles are wrapped to the interval $(-\pi, \pi]$ but the wrapping might go wrong near the end points due to computation accuracy. The first index is 1.

```
\array_new:N \l_hobby_excess_angle_array
```

\l_hobby_psi_array

\l_hobby_psi_array is an array holding the difference of the angles of the lines entering and exiting a point. That is, ψ_k is the angle between the lines joining z_k to z_{k-1} and z_{k+1}. The first index is 1.

```
\array_new:N \l_hobby_psi_array
```

\l_hobby_theta_array

\l_hobby_theta_array is an array holding the angles of the outgoing control points for the generated path. These are measured relative to the line joining the point to the next point on the path. The first index is 0.

```
\array_new:N \l_hobby_theta_array
```

\l_hobby_phi_array

\l_hobby_phi_array is an array holding the angles of the incoming control points for the generated path. These are measured relative to the line joining the point to the previous point on the path. The first index is 1.

```
\array_new:N \l_hobby_phi_array
```

`\l_hobby_sigma_array`

`\l_hobby_sigma_array` is an array holding the lengths of the outgoing control points for the generated path. The units are such that the length of the line to the next specified point is one unit.

```
\array_new:N \l_hobby_sigma_array
```

`\l_hobby_rho_array`

`\l_hobby_rho_array` is an array holding the lengths of the incoming control points for the generated path. The units are such that the length of the line to the previous specified point is one unit.

```
\array_new:N \l_hobby_rho_array
```

`l_hobby_controla_array`

`\l_hobby_controla_array` is an array holding the coordinates of the first control points on the curves. The format is the same as for `\l_hobby_points_array`.

```
\array_new:N \l_hobby_controla_array
```

`l_hobby_controlb_array`

`\l_hobby_controlb_array` is an array holding the coordinates of the second control points on the curves. The format is the same as for `\l_hobby_points_array`.

```
\array_new:N \l_hobby_controlb_array
```

`\l_hobby_matrix_v_fp`

`\l_hobby_matrix_v_fp` is a number which is used when doing the perturbation of the solution of the linear system for a closed curve. There is actually a vector, v, that this corresponds to but that vector only has one component that needs computation.

```
\fp_new:N \l_hobby_matrix_v_fp
```

`\l_hobby_tempa_fp`

`\l_hobby_tempa_fp` is a temporary variable of type `fp`.

```
\fp_new:N \l_hobby_tempa_fp
```

`\l_hobby_tempb_fp`

`\l_hobby_tempb_fp` is a temporary variable of type `fp`.

```
\fp_new:N \l_hobby_tempb_fp
```

`\l_hobby_tempc_fp`

`\l_hobby_tempc_fp` is a temporary variable of type `fp`.

```
\fp_new:N \l_hobby_tempc_fp
```

`\l_hobby_tempd_fp`

`\l_hobby_tempd_fp` is a temporary variable of type `fp`.

```
\fp_new:N \l_hobby_tempd_fp
```

`\l_hobby_temps_fp`

`\l_hobby_temps_fp` is a temporary variable of type `fp`.

```
\fp_new:N \l_hobby_temps_fp
```

`\l_hobby_in_curl_fp`

`\l_hobby_in_curl_fp` is the “curl” at the end of an open path. This is used if the angle at the end is not specified.

```
\fp_new:N \l_hobby_in_curl_fp
\fp_set:Nn \l_hobby_in_curl_fp {1}
```

`\l_hobby_out_curl_fp`

`\l_hobby_out_curl_fp` is the “curl” at the start of an open path. This is used if the angle at the start is not specified.

```
\fp_new:N \l_hobby_out_curl_fp
\fp_set:Nn \l_hobby_out_curl_fp {1}
```

`\l_hobby_in_angle_fp`

`\l_hobby_in_angle_fp` is the angle at the end of an open path. If this is not specified, it will be computed automatically. It is set to `\c_inf_fp` to allow easy detection of when it has been specified.

```
\fp_new:N \l_hobby_in_angle_fp
\fp_set_eq:NN \l_hobby_in_angle_fp \c_inf_fp
```

\l_hobby_out_angle_fp

\l_hobby_out_angle_fp is the angle at the start of an open path. If this is not specified, it will be computed automatically. It is set to \c_inf_fp to allow easy detection of when it has been specified.

```
\fp_new:N \l_hobby_out_angle_fp
\fp_set_eq:NN \l_hobby_out_angle_fp \c_inf_fp
```

\l_hobby_npoints_int

\l_hobby_npoints_int is one less than the number of points on the curve. As our list of points starts at 0, this is the index of the last point. In the algorithm for a closed curve, some points are repeated whereupon this is incremented so that it is always the index of the last point.

```
\int_new:N \l_hobby_npoints_int
```

\l_hobby_draw_int

```
\int_new:N \l_hobby_draw_int
```

A “point” is a key-value list setting the x-value, the y-value, and the tensions at that point. Using keys makes it easier to pass points from the algorithm code to the calling code and vice versa without either knowing too much about the other.

```
\keys_define:nn {hobby / read in all} {
  x .fp_set:N = \l_hobby_tempa_fp,
  y .fp_set:N = \l_hobby_tempb_fp,
  tension~out .fp_set:N = \l_hobby_tempc_fp,
  tension~in .fp_set:N = \l_hobby_tempd_fp,
  excess~angle .fp_set:N = \l_hobby_temps_fp,
  break .tl_set:N = \l_tmpb_tl,
  blank .tl_set:N = \l_tmpa_tl,
  tension .meta:n = { tension~out=#1, tension~in=#1 },
  break .default:n = false,
  blank .default:n = false,
  invert~soft~blanks .choice:,
  invert~soft~blanks / true .code:n = {
    \int_gset:Nn \l_hobby_draw_int {0}
  },
  invert~soft~blanks / false .code:n = {
    \int_gset:Nn \l_hobby_draw_int {1}
  },
  invert~soft~blanks .default:n = true,
  tension~out .default:n = 1,
  tension~in .default:n = 1,
  excess~angle .default:n = 0,
  in~angle .fp_gset:N = \l_hobby_in_angle_fp,
  out~angle .fp_gset:N = \l_hobby_out_angle_fp,
  in~curl .fp_gset:N = \l_hobby_in_curl_fp,
  out~curl .fp_gset:N = \l_hobby_out_curl_fp,
  closed .bool_gset:N = \l_hobby_closed_bool,
  closed .default:n = true,
  disjoint .bool_gset:N = \l_hobby_disjoint_bool,
  disjoint .default:n = true,
  break~default .code:n = {
    \keys_define:nn { hobby / read in all }
    {
      break .default:n = #1
    }
  },
  blank~default .code:n = {
    \keys_define:nn { hobby / read in all }
    {
      blank .default:n = #1
    }
  },
}
```

There are certain other parameters that can be set for a given curve.

```
\keys_define:nn { hobby / read in params} {
  in~angle .fp_gset:N = \l_hobby_in_angle_fp,
  out~angle .fp_gset:N = \l_hobby_out_angle_fp,
  in~curl .fp_gset:N = \l_hobby_in_curl_fp,
  out~curl .fp_gset:N = \l_hobby_out_curl_fp,
  closed .bool_gset:N = \l_hobby_closed_bool,
  closed .default:n = true,
  disjoint .bool_gset:N = \l_hobby_disjoint_bool,
  disjoint .default:n = true,
  break~default .code:n = {
    \keys_define:nn { hobby / read in all }
    {
      break .default:n = #1
    }
  },
  blank~default .code:n = {
    \keys_define:nn { hobby / read in all }
    {
      blank .default:n = #1
    }
  },
  invert~soft~blanks .choice:,
  invert~soft~blanks / true .code:n = {
    \int_gset:Nn \l_hobby_draw_int {0}
  },
  invert~soft~blanks / false .code:n = {
    \int_gset:Nn \l_hobby_draw_int {1}
  },
  invert~soft~blanks .default:n = true,
}
```

\hobby_distangle:n

Computes the distance and angle between successive points. The argument given is the index of the current point. Assumptions: the points are in \l_hobby_points_x_array and \l_hobby_points_y_array and the index of the last point is \l_hobby_npoints_int.

```
\cs_set:Nn \hobby_distangle:n {
  \fp_set:Nn \l_hobby_tempa_fp {
    (\array_get:Nn \l_hobby_points_x_array {#1 + 1})
    - (\array_get:Nn \l_hobby_points_x_array {#1})}

  \fp_set:Nn \l_hobby_tempb_fp {
    (\array_get:Nn \l_hobby_points_y_array {#1 + 1})
    - (\array_get:Nn \l_hobby_points_y_array {#1})}

  \fp_set:Nn \l_hobby_tempc_fp { atan ( \l_hobby_tempb_fp, \l_hobby_tempa_fp ) }
  \fp_veclen:NVV \l_hobby_tempd_fp \l_hobby_tempa_fp \l_hobby_tempb_fp

  \array_push:Nx \l_hobby_angles_array {\fp_to_tl:N \l_hobby_tempc_fp}
  \array_push:Nx \l_hobby_distances_array {\fp_to_tl:N \l_hobby_tempd_fp}
  }
```

\fp_veclen:NVV

Computes the length of the vector specified by the latter two arguments, storing the answer in the first.

```
\cs_new:Nn \fp_veclen:Nnn {
  \fp_set:Nn #1 {((#2)^2 + (#3)^2)^.5}
}
\cs_generate_variant:Nn \fp_veclen:Nnn {NVV}
```

\hobby_ctrllen:Nnn Computes the length of the control point vector from the two angles, storing the answer in the first argument given.

```
\cs_new:Nn \hobby_ctrllen:Nnn {
  \fp_set:Nn #1 {(2 - \g_hobby_parama_fp
    * ( sin(#2) - \g_hobby_paramb_fp * sin(#3) )
    * ( sin(#3) - \g_hobby_paramb_fp * sin(#2) )
    * ( cos(#2) - cos(#3) ) )
    / ( 1 + (1 - \g_hobby_paramc_fp) * cos(#3) + \g_hobby_paramc_fp * cos(#2))}
}
\cs_generate_variant:Nn \hobby_ctrllen:Nnn {NVV}
```

by_append_point_copy:n This function adds a copy of the point (numbered by its argument) to the end of the list of points, copying all the relevant data (coordinates, tension, etc.).

Originally from Bruno Le Foch on TeX-SX.

```
\cs_new_protected:Npn \hobby_append_point_copy:n #1
  {
    \hobby_append_point_copy_aux:Nn \l_hobby_points_array {#1}
    \hobby_append_point_copy_aux:Nn \l_hobby_points_x_array {#1}
    \hobby_append_point_copy_aux:Nn \l_hobby_points_y_array {#1}
    \hobby_append_point_copy_aux:Nn \l_hobby_tension_in_array {#1}
    \hobby_append_point_copy_aux:Nn \l_hobby_tension_out_array {#1}
    \hobby_append_point_copy_aux:Nn \l_hobby_excess_angle_array {#1}
    \hobby_append_point_copy_aux:Nn \l_hobby_actions_array {#1}
  }
\cs_new_protected:Npn \hobby_append_point_copy_aux:Nn #1#2
  { \array_gpush:Nx #1 { \array_get:Nn #1 {#2} } }
```

\hobby_gen_path: This is the curve generation function. We assume at the start that we have an array containing all the points that the curve must go through, and the various curve parameters have been initialised. So these must be set up by a wrapper function which then calls this one. The list of required information is:

1. \l_hobby_points_x_array
2. \l_hobby_points_y_array
3. \l_hobby_tension_out_array
4. \l_hobby_tension_in_array
5. \l_hobby_excess_angle_array
6. \l_hobby_in_curl_fp
7. \l_hobby_out_curl_fp
8. \l_hobby_in_angle_fp
9. \l_hobby_out_angle_fp
10. \l_hobby_closed_bool
11. \l_hobby_actions_array

```
\cs_new:Nn \hobby_gen_path:
{
```

For much of the time, we can pretend that a closed path is the same as an open path. To do this, we need to make the end node an internal node by repeating the z_1 node as the z_{n+1}th node. We also check that the last (z_n) and first (z_0) nodes are the same, otherwise we repeat the z_0 node as well.

```
\bool_if:NT \l_hobby_closed_bool {
```

Are the x-values of the first and last points different?

```
  \fp_compare:nTF {(\array_get:Nn \l_hobby_points_x_array {0})
    =
    (\array_top:N \l_hobby_points_x_array)}
  {
```

No, so compare the y-values. Are the y-values of the first and last points different?

```
    \fp_compare:nF {
      \array_get:Nn \l_hobby_points_y_array {0}
      =
      \array_top:N \l_hobby_points_y_array
    }
  {
```

Yes, so we need to duplicate the first point, with all of its data.

```
    \hobby_append_point_copy:n {0}
  }
  }
  {
```

Yes, so we need to duplicate the first point, with all of its data.

```
    \hobby_append_point_copy:n {0}
  }
```

Now that we are sure that the first and last points are identical, we need to duplicate the first-but-one point (and all of its data).

```
    \hobby_append_point_copy:n {1}
}
```

Set \l_hobby_npoints_int to the number of points (minus one).

```
\int_gset:Nn \l_hobby_npoints_int {\array_length:N \l_hobby_points_y_array}
```

At this point, we need to decide what to do. This will depend on whether we have any intermediate points.

```
\int_compare:nNnTF {\l_hobby_npoints_int} = {0} {
```

Only one point, do nothing

```
}
{
  \int_compare:nNnTF {\l_hobby_npoints_int} = {1} {
```

Only two points, skip processing. Just need to set the incoming and outgoing angles

```
\hobby_distangle:n {0}
\fp_compare:nF { \l_hobby_out_angle_fp == \c_inf_fp }
{
  \fp_set:Nn \l_hobby_tempa_fp { \l_hobby_out_angle_fp
    - \array_get:Nn \l_hobby_angles_array {0}}
```

We want to ensure that these angles lie in the range $(-\pi, \pi]$. So if the angle is bigger than π, we subtract 2π. (It shouldn't be that we can get bigger than 3π - check this)

```
    \fp_compare:nT {\l_hobby_tempa_fp > \c_pi_fp }
    {
      \fp_sub:Nn \l_hobby_tempa_fp {2 * \c_pi_fp}
    }
```

Similarly, we check to see if the angle is less than $-\pi$.

```
    \fp_compare:nT {\l_hobby_tempa_fp < -\c_pi_fp }
    {
      \fp_add:Nn \l_hobby_tempa_fp {2 * \c_pi_fp}
    }
  \array_put:Nnx \l_hobby_theta_array {0} {\fp_to_tl:N \l_hobby_tempa_fp}
    \fp_compare:nT { \l_hobby_in_angle_fp == \c_inf_fp }
    {
%^^A      \fp_mul:Nn \l_hobby_tempa_fp {-1}
      \array_put:Nnx \l_hobby_phi_array {1}{ \fp_to_tl:N \l_hobby_tempa_fp}
    }
   }
\fp_compare:nTF { \l_hobby_in_angle_fp == \c_inf_fp }
{
  \fp_compare:nT { \l_hobby_out_angle_fp == \c_inf_fp }
  {
    \array_put:Nnx \l_hobby_phi_array {1} {0}
    \array_put:Nnx \l_hobby_theta_array {0} {0}
  }
}
{
  \fp_set:Nn \l_hobby_tempa_fp { - \l_hobby_in_angle_fp + \c_pi_fp
+ (\array_get:Nn \l_hobby_angles_array {0})}
  \fp_compare:nT {\l_hobby_tempa_fp > \c_pi_fp }
  {
    \fp_sub:Nn \l_hobby_tempa_fp {2 * \c_pi_fp}
  }
  \fp_compare:nT {\l_hobby_tempa_fp < -\c_pi_fp }
  {
    \fp_add:Nn \l_hobby_tempa_fp {2 * \c_pi_fp}
  }

  \array_put:Nnx \l_hobby_phi_array {1}
  {\fp_to_tl:N \l_hobby_tempa_fp}
  \fp_compare:nT { \l_hobby_out_angle_fp == \c_inf_fp }
    {
%^^A      \fp_mul:Nn \l_hobby_tempa_fp {-1}
      \array_put:Nnx \l_hobby_theta_array {0}{ \fp_to_tl:N \l_hobby_tempa_fp}
    }
}

  }
  {
```

Got enough points, go on with processing

```
    \hobby_compute_path:
  }
  \hobby_build_path:
}
}
```

\hobby_compute_path: This is the path builder where we have enough points to run the algorithm.

```
\cs_new:Nn \hobby_compute_path:
{
```

Our first step is to go through the list of points and compute the distances and angles between successive points. Thus d_i is the distance from z_i to z_{i+1} and the angle is the angle of the line from z_i to z_{i+1}.

```
\int_step_function:nnnN {0} {1} {\l_hobby_npoints_int - 1} \hobby_distangle:n
```

For the majority of the code, we're only really interested in the differences of the angles. So for each internal point we compute the differences in the angles.

```
  \int_step_inline:nnnn {1} {1} {\l_hobby_npoints_int - 1} {
    \fp_set:Nx \l_hobby_tempa_fp {
    \array_get:Nn \l_hobby_angles_array {##1}
    - \array_get:Nn \l_hobby_angles_array {##1 - 1}}
```

We want to ensure that these angles lie in the range $(-\pi, \pi]$. So if the angle is bigger than π, we subtract 2π. (It shouldn't be that we can get bigger than 3π - check this.)

```
    \fp_compare:nTF {\l_hobby_tempa_fp > \c_pi_fp }
    {
      \fp_sub:Nn \l_hobby_tempa_fp {2 * \c_pi_fp}
    }
    {}
```

Similarly, we check to see if the angle is less than $-\pi$.

```
    \fp_compare:nTF {\l_hobby_tempa_fp <= -\c_pi_fp }
    {
      \fp_add:Nn \l_hobby_tempa_fp {2 * \c_pi_fp}
    }
    {}
```

The wrapping routine might not get it right at the edges so we add in the override.

```
\array_get:NnNTF \l_hobby_excess_angle_array {##1} \l_tmpa_tl {
  \fp_add:Nn \l_hobby_tempa_fp {2 * \c_pi_fp * \l_tmpa_tl}
  }{}
    \array_put:Nnx \l_hobby_psi_array {##1}{\fp_to_tl:N \l_hobby_tempa_fp}
  }
```

Next, we generate the matrix. We start with the subdiagonal. This is indexed from 1 to $n - 1$.

```
  \int_step_inline:nnnn {1} {1} {\l_hobby_npoints_int - 1} {
    \array_put:Nnx \l_hobby_matrix_a_array {##1} {\fp_to_tl:n {
       \array_get:Nn \l_hobby_tension_in_array {##1}^2
      * \array_get:Nn \l_hobby_distances_array {##1}
      * \array_get:Nn \l_hobby_tension_in_array {##1 + 1}
  }}
}
```

Next, we attack main diagonal. We might need to adjust the first and last terms, but we'll do that in a minute.

```
  \int_step_inline:nnnn {1} {1} {\l_hobby_npoints_int - 1} {

  \array_put:Nnx \l_hobby_matrix_b_array {##1} {\fp_to_tl:n
{(3 * (\array_get:Nn \l_hobby_tension_in_array {##1 + 1}) - 1) *
 (\array_get:Nn \l_hobby_tension_out_array {##1})^2 *
(\array_get:Nn \l_hobby_tension_out_array {##1 - 1})
* ( \array_get:Nn \l_hobby_distances_array {##1 - 1})
+
(3 * (\array_get:Nn \l_hobby_tension_out_array {##1 - 1}) - 1)
* (\array_get:Nn \l_hobby_tension_in_array {##1})^2
* (\array_get:Nn \l_hobby_tension_in_array {##1 + 1})
* (\array_get:Nn \l_hobby_distances_array {##1})}
}
}
```

Next, the superdiagonal.

```
  \int_step_inline:nnnn {1} {1} {\l_hobby_npoints_int - 2} {

  \array_put:Nnx \l_hobby_matrix_c_array {##1} {\fp_to_tl:n
{(\array_get:Nn \l_hobby_tension_in_array {##1})^2
* (\array_get:Nn \l_hobby_tension_in_array {##1 - 1})
* (\array_get:Nn \l_hobby_distances_array {##1 - 1})
}}

}
```

Lastly (before the adjustments), the target vector.

```
  \int_step_inline:nnnn {1} {1} {\l_hobby_npoints_int - 2} {

  \array_put:Nnx \l_hobby_matrix_d_array {##1} {\fp_to_tl:n
{
- (\array_get:Nn \l_hobby_psi_array {##1 + 1})
* (\array_get:Nn \l_hobby_tension_out_array {##1})^2
* (\array_get:Nn \l_hobby_tension_out_array {##1 - 1})
* (\array_get:Nn \l_hobby_distances_array {##1 - 1})
- (3 * (\array_get:Nn \l_hobby_tension_out_array {##1 - 1}) - 1)
* (\array_get:Nn \l_hobby_psi_array {##1})
* (\array_get:Nn \l_hobby_tension_in_array {##1})^2
* (\array_get:Nn \l_hobby_tension_in_array {##1 + 1})
* (\array_get:Nn \l_hobby_distances_array {##1})
}
}
}
```

Next, there are some adjustments at the ends. These differ depending on whether the path is open or closed.

```
\bool_if:NTF \l_hobby_closed_bool {
```

Closed path

```
\array_put:Nnx \l_hobby_matrix_c_array {0} {\fp_to_tl:n {
- (\array_get:Nn \l_hobby_distances_array {\l_hobby_npoints_int - 2})
* (\array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int - 2})
* (\array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int - 1})^2
}}

\array_put:Nnn \l_hobby_matrix_b_array {0} {1}
\array_put:Nnn \l_hobby_matrix_d_array {0} {0}

\array_put:Nnx \l_hobby_matrix_b_array {\l_hobby_npoints_int - 1} {\fp_to_tl:n {
(\array_get:Nn \l_hobby_matrix_b_array {\l_hobby_npoints_int - 1})
+ 1
}}

 \array_put:Nnx \l_hobby_matrix_d_array {\l_hobby_npoints_int - 1} {\fp_to_tl:n {
- (\array_get:Nn \l_hobby_psi_array {1})
* (\array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int -1})^2
* (\array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int -2})
* (\array_get:Nn \l_hobby_distances_array {\l_hobby_npoints_int - 2})
- (3 * (\array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int - 2}) - 1)
* (\array_get:Nn \l_hobby_psi_array {\l_hobby_npoints_int - 1})
* (\array_get:Nn \l_hobby_tension_in_array {\l_hobby_npoints_int - 1})^2
* (\array_get:Nn \l_hobby_tension_in_array {\l_hobby_npoints_int})
* (\array_get:Nn \l_hobby_distances_array {\l_hobby_npoints_int -1})
}
}
```

We also need to populate the u-vector

```
  \array_put:Nnn \l_hobby_vector_u_array {0} {1}
\array_put:Nnn \l_hobby_vector_u_array {\l_hobby_npoints_int - 1} {1}
  \int_step_inline:nnnn {1} {1} {\l_hobby_npoints_int - 2} {
  \array_put:Nnn \l_hobby_vector_u_array {##1} {0}
  }
```

And define the significant entry in the v-vector.

```
\fp_set:Nn \l_hobby_matrix_v_fp {
(\array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int -1})^2
* (\array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int -2})
* (\array_get:Nn \l_hobby_distances_array {\l_hobby_npoints_int -2})
}
}
{
```

Open path. First, we test to see if θ_0 has been specified.

```
\fp_compare:nTF { \l_hobby_out_angle_fp == \c_inf_fp }
{
  \array_put:Nnx \l_hobby_matrix_b_array {0}  {\fp_to_tl:n {
  (\array_get:Nn \l_hobby_tension_in_array {1})^3
* \l_hobby_in_curl_fp
+
(3 * (\array_get:Nn \l_hobby_tension_in_array {1}) - 1)
* (\array_get:Nn \l_hobby_tension_out_array {0})^3
}}

  \array_put:Nnx \l_hobby_matrix_c_array {0} {\fp_to_tl:n {
  (\array_get:Nn \l_hobby_tension_out_array {0})^3
+
(3 * (\array_get:Nn \l_hobby_tension_out_array {0}) - 1)
* (\array_get:Nn \l_hobby_tension_in_array {1})^3
* \l_hobby_in_curl_fp
}}

  \array_put:Nnx \l_hobby_matrix_d_array {0} {\fp_to_tl:n {
-(  (\array_get:Nn \l_hobby_tension_out_array {0})^3
+
(3 * (\array_get:Nn \l_hobby_tension_out_array {0}) - 1)
* (\array_get:Nn \l_hobby_tension_in_array {1})^3
* \l_hobby_in_curl_fp)
* (\array_get:Nn \l_hobby_psi_array {1})
}}

}
{
  \array_put:Nnn \l_hobby_matrix_b_array {0} {1}
  \array_put:Nnn \l_hobby_matrix_c_array {0} {0}
  \fp_set:Nn \l_hobby_tempa_fp { \l_hobby_out_angle_fp
    - \array_get:Nn \l_hobby_angles_array {0}}
```

We want to ensure that these angles lie in the range $(-\pi, \pi]$. So if the angle is bigger than π, we subtract 2π. (It shouldn't be that we can get bigger than 3π - check this)

```
    \fp_compare:nT {\l_hobby_tempa_fp > \c_pi_fp }
    {
      \fp_sub:Nn \l_hobby_tempa_fp {2 * \c_pi_fp}
    }
```

Similarly, we check to see if the angle is less than $-\pi$.

```
    \fp_compare:nT {\l_hobby_tempa_fp < -\c_pi_fp }
    {
      \fp_add:Nn \l_hobby_tempa_fp {2 * \c_pi_fp}
    }
  \array_put:Nnx \l_hobby_matrix_d_array {0} {\fp_to_tl:N \l_hobby_tempa_fp}
}
```

Next, if ϕ_n has been given.

```
\fp_compare:nTF { \l_hobby_in_angle_fp == \c_inf_fp }
{

 \array_put:Nnx \l_hobby_matrix_b_array {\l_hobby_npoints_int - 1} {\fp_to_tl:n {
\array_get:Nn \l_hobby_matrix_b_array {\l_hobby_npoints_int - 1}
- (\array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int - 1})^2
* (\array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int - 2})
* (\array_get:Nn \l_hobby_distances_array {\l_hobby_npoints_int - 2})
*
((3 * (\array_get:Nn \l_hobby_tension_in_array {\l_hobby_npoints_int} ) - 1)
* (\array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int - 1})^3 \l_tmpa_tl
* \l_hobby_out_curl_fp
+
(\array_get:Nn \l_hobby_tension_in_array {\l_hobby_npoints_int })^3)
/
((3 * (\array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int -2}) - 1)
* (\array_get:Nn \l_hobby_tension_in_array {\l_hobby_npoints_int})^3
+
( \array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int - 1})^3
* \l_hobby_out_curl_fp)
}}

 \array_put:Nnx \l_hobby_matrix_d_array {\l_hobby_npoints_int - 1} {\fp_to_tl:n {
- (3 * (\array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int - 2}) - 1)
* (\array_get:Nn \l_hobby_psi_array {\l_hobby_npoints_int - 1})
* (\array_get:Nn \l_hobby_tension_in_array {\l_hobby_npoints_int - 1})^2
* (\array_get:Nn \l_hobby_tension_in_array {\l_hobby_npoints_int})
* (\array_get:Nn \l_hobby_distances_array {\l_hobby_npoints_int - 1})
}}

}
{
  \fp_set:Nn \l_hobby_tempa_fp { - \l_hobby_in_angle_fp + \c_pi_fp
+ (\array_get:Nn \l_hobby_angles_array {\l_hobby_npoints_int - 1})}
  \fp_compare:nT {\l_hobby_tempa_fp > \c_pi_fp }
  {
    \fp_sub:Nn \l_hobby_tempa_fp {2 * \c_pi_fp}
  }
  \fp_compare:nT {\l_hobby_tempa_fp < -\c_pi_fp }
  {
    \fp_add:Nn \l_hobby_tempa_fp {2 * \c_pi_fp}
  }

  \array_put:Nnx \l_hobby_phi_array {\l_hobby_npoints_int}
  {\fp_to_tl:N \l_hobby_tempa_fp}

   \array_put:Nnx \l_hobby_matrix_d_array  {\l_hobby_npoints_int - 1} {\fp_to_tl:n {
 \l_hobby_tempa_fp
 * (\array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int - 1})^2
* (\array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int - 2})
* (\array_get:Nn \l_hobby_distances_array {\l_hobby_npoints_int - 2})
-
(3 * ( \array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int - 2}) - 1)
* (\array_get:Nn \l_hobby_psi_array {\l_hobby_npoints_int - 1})
* (\array_get:Nn \l_hobby_tension_in_array  {\l_hobby_npoints_int - 1})^2
* (\array_get:Nn \l_hobby_tension_in_array {\l_hobby_npoints_int})
* (\array_get:Nn \l_hobby_distances_array  {\l_hobby_npoints_int - 1}) }}
}
```

End of adjustments for open paths.

```
}
```

Now we have the tridiagonal matrix in place, we implement the solution. We start with the forward eliminations.

```
\int_step_inline:nnnn {1} {1} {\l_hobby_npoints_int - 1} {

  \array_put:Nnx \l_hobby_matrix_b_array {##1} {\fp_to_tl:n {
  (\array_get:Nn \l_hobby_matrix_b_array {##1 - 1})
* (\array_get:Nn \l_hobby_matrix_b_array {##1})
-
(\array_get:Nn \l_hobby_matrix_c_array {##1 - 1})
* (\array_get:Nn \l_hobby_matrix_a_array {##1})
}}
```

The last time, we don't touch the C-vector.

```
  \int_compare:nT {##1 < \l_hobby_npoints_int - 1} {

  \array_put:Nnx \l_hobby_matrix_c_array {##1} {\fp_to_tl:n {
(\array_get:Nn \l_hobby_matrix_b_array {##1 - 1})
    * (\array_get:Nn \l_hobby_matrix_c_array {##1})
}}
  }

  \array_put:Nnx \l_hobby_matrix_d_array {##1} {\fp_to_tl:n {
(\array_get:Nn \l_hobby_matrix_b_array {##1 - 1})
  * (\array_get:Nn \l_hobby_matrix_d_array {##1})
-
  (\array_get:Nn \l_hobby_matrix_d_array {##1 - 1})
  * (\array_get:Nn \l_hobby_matrix_a_array {##1})
}}
```

On a closed path, we also want to know $M^{-1}u$ so need to do the elimination steps on u as well.

```
  \bool_if:NT \l_hobby_closed_bool {
  \array_put:Nnx \l_hobby_vector_u_array {##1} {\fp_to_tl:n {
(\array_get:Nn \l_hobby_matrix_b_array {##1 - 1})
* (\array_get:Nn \l_hobby_vector_u_array {##1})
-
(\array_get:Nn \l_hobby_vector_u_array {##1 - 1})
* (\array_get:Nn \l_hobby_matrix_a_array {##1})
}}
}
}
```

Now we start the back substitution. The first step is slightly different to the general step.

```
 \array_put:Nnx \l_hobby_theta_array  {\l_hobby_npoints_int - 1} {\fp_to_tl:n {
(\array_get:Nn \l_hobby_matrix_d_array  {\l_hobby_npoints_int - 1})
/ (\array_get:Nn \l_hobby_matrix_b_array  {\l_hobby_npoints_int - 1})
}}
```

For a closed path, we need to work with u as well.

```
\bool_if:NT \l_hobby_closed_bool {
 \array_put:Nnx \l_hobby_vector_u_array  {\l_hobby_npoints_int - 1} {\fp_to_tl:n {
  (\array_get:Nn \l_hobby_vector_u_array  {\l_hobby_npoints_int - 1})
/ (\array_get:Nn \l_hobby_matrix_b_array  {\l_hobby_npoints_int - 1})
}}
}
```

Now we iterate over the vectors, doing the remaining back substitutions.

```
\int_step_inline:nnnn {\l_hobby_npoints_int - 2} {-1} {0} {

  \array_put:Nnx \l_hobby_theta_array {##1} {\fp_to_tl:n {
( (\array_get:Nn \l_hobby_matrix_d_array {##1})
  - (\array_get:Nn \l_hobby_theta_array  {##1 + 1})
  * (\array_get:Nn \l_hobby_matrix_c_array {##1})
) / (\array_get:Nn \l_hobby_matrix_b_array {##1})
}}
}
\bool_if:NT \l_hobby_closed_bool {
```

On a closed path, we also need to work out $M^{-1}u$.

```
\int_step_inline:nnnn {\l_hobby_npoints_int - 2} {-1} {0} {
  \array_put:Nnx \l_hobby_vector_u_array {##1} {\fp_to_tl:n
{
    ((\array_get:Nn \l_hobby_vector_u_array {##1})
    - (\array_get:Nn \l_hobby_vector_u_array  {##1 + 1})
    * (\array_get:Nn \l_hobby_matrix_c_array {##1})
    ) / (\array_get:Nn \l_hobby_matrix_b_array {##1})
}}
}
```

Then we compute $v^\top M^{-1}u$ and $v^\top M^{-1}\theta$. As v has a particularly simple form, these inner products are easy to compute.

```

\fp_set:Nn \l_hobby_tempb_fp {
((\array_get:Nn \l_hobby_theta_array {1})
* \l_hobby_matrix_v_fp
- (\array_get:Nn \l_hobby_theta_array  {\l_hobby_npoints_int - 1})
) / (
(\array_get:Nn \l_hobby_vector_u_array {1})
* \l_hobby_matrix_v_fp
- (\array_get:Nn \l_hobby_vector_u_array  {\l_hobby_npoints_int - 1})
+ 1
)}

\int_step_inline:nnnn {0} {1} {\l_hobby_npoints_int - 1} {

  \array_put:Nnx \l_hobby_theta_array {##1} {\fp_to_tl:n {
```

```
  (\array_get:Nn \l_hobby_theta_array {##1})
  - (\array_get:Nn \l_hobby_vector_u_array {##1})
  * \l_hobby_tempb_fp
}}
}
}
```

Now that we have computed the θ_is, we can quickly compute the ϕ_is.

```
\int_step_inline:nnnn {1} {1} {\l_hobby_npoints_int - 1} {

    \array_put:Nnx \l_hobby_phi_array {##1} {\fp_to_tl:n {
      - (\array_get:Nn \l_hobby_psi_array {##1})
      - (\array_get:Nn \l_hobby_theta_array {##1})
  }}
  }
```

If the path is open, this works for all except ϕ_n. If the path is closed, we can drop our added point. Cheaply, of course.

```
\bool_if:NTF \l_hobby_closed_bool {
  \int_gdecr:N \l_hobby_npoints_int
}{
```

If ϕ_n was not given, we compute it from θ_{n-1}.

```
\fp_compare:nT { \l_hobby_in_angle_fp == \c_inf_fp }
{
 \array_put:Nnx \l_hobby_phi_array {\l_hobby_npoints_int} {\fp_to_tl:n {
((3 * (\array_get:Nn \l_hobby_tension_in_array {\l_hobby_npoints_int}) - 1)
* (\array_get:Nn \l_hobby_tension_out_array {\l_hobby_npoints_int - 1})^3
* \l_hobby_out_curl_fp
+
(\array_get:Nn \l_hobby_tension_in_array {\l_hobby_npoints_int })^3)
/
((3 * (\array_get:Nn \l_hobby_tension_out_array  {\l_hobby_npoints_int -2}) - 1)
* (\array_get:Nn \l_hobby_tension_in_array {\l_hobby_npoints_int})^3 \l_tmpa_tl
+
(\array_get:Nn \l_hobby_tension_out_array  {\l_hobby_npoints_int - 1})^3
* \l_hobby_out_curl_fp)
*
(\array_get:Nn \l_hobby_theta_array  {\l_hobby_npoints_int -1})
}}
}
}
}
```

\hobby_build_path: Once we've computed the angles, we build the actual path.

```
\cs_new:Nn \hobby_build_path:
{
```

Next task is to compute the ρ_i and σ_i.

```
\int_step_inline:nnnn {0} {1} {\l_hobby_npoints_int - 1} {

  \fp_set:Nn \l_hobby_tempa_fp {\array_get:Nn \l_hobby_theta_array {##1}}

  \fp_set:Nn \l_hobby_tempb_fp {\array_get:Nn \l_hobby_phi_array  {##1 + 1}}

  \hobby_ctrllen:NVV \l_hobby_temps_fp \l_hobby_tempa_fp \l_hobby_tempb_fp

```

```
   \array_put:Nnx \l_hobby_sigma_array {##1 + 1} {\fp_to_tl:N \l_hobby_temps_fp}

  \hobby_ctrllen:NVV \l_hobby_temps_fp \l_hobby_tempb_fp \l_hobby_tempa_fp

   \array_put:Nnx \l_hobby_rho_array {##1} {\fp_to_tl:N \l_hobby_temps_fp}

  }
```

Lastly, we generate the coordinates of the control points.

```
\int_step_inline:nnnn {0} {1} {\l_hobby_npoints_int - 1} {
\array_gput:Nnx \l_hobby_controla_array  {##1 + 1} {x = \fp_eval:n  {
(\array_get:Nn \l_hobby_points_x_array {##1})
+
  (\array_get:Nn \l_hobby_distances_array {##1}) *
  (\array_get:Nn \l_hobby_rho_array {##1}) *
cos ( (\array_get:Nn \l_hobby_angles_array {##1})
+
  (\array_get:Nn \l_hobby_theta_array {##1}))
/3
}, y = \fp_eval:n {
( \array_get:Nn \l_hobby_points_y_array {##1}) +
  (\array_get:Nn \l_hobby_distances_array {##1}) *
  (\array_get:Nn \l_hobby_rho_array {##1}) *
sin ( (\array_get:Nn \l_hobby_angles_array {##1})
+
  (\array_get:Nn \l_hobby_theta_array {##1}))
/3
}
}
}
\int_step_inline:nnnn {1} {1} {\l_hobby_npoints_int} {
  \array_gput:Nnx \l_hobby_controlb_array {##1} {
    x = \fp_eval:n {\array_get:Nn \l_hobby_points_x_array {##1}
- (\array_get:Nn \l_hobby_distances_array  {##1 - 1})
* (\array_get:Nn \l_hobby_sigma_array {##1})
* cos((\array_get:Nn \l_hobby_angles_array  {##1 - 1})
- (\array_get:Nn \l_hobby_phi_array {##1}))/3
}, y = \fp_eval:n {
  (\array_get:Nn \l_hobby_points_y_array {##1})
- (\array_get:Nn \l_hobby_distances_array  {##1 - 1})
* (\array_get:Nn \l_hobby_sigma_array {##1})
* sin((\array_get:Nn \l_hobby_angles_array  {##1 - 1})
- (\array_get:Nn \l_hobby_phi_array {##1}))/3
} }
 }
}
```

\hobbyinit Initialise the settings for Hobby's algorithm

```
\NewDocumentCommand \hobbyinit {m m m} {
  \hobby_set_cmds:nnn#1#2#3
  \hobby_clear_path:
}
```

\hobbyaddpoint This adds a point, possibly with tensions, to the current stack.

```
\NewDocumentCommand \hobbyaddpoint { m } {
    \keys_set:nn { hobby/read in all }
    {
      tension~out,
      tension~in,
      excess~angle,
      blank,
      break,
      #1
    }
    \tl_if_eq:VnTF {\l_tmpa_tl} {true}
     {\tl_set:Nn \l_tmpa_tl {2}}
     {
       \tl_if_eq:VnTF {\l_tmpa_tl} {soft}
       {\tl_set:Nn \l_tmpa_tl {0}}
       {\tl_set:Nn \l_tmpa_tl {1}}
     }
    \tl_if_eq:VnTF {\l_tmpb_tl} {true}
     {\tl_put_right:Nn \l_tmpa_tl {1}}
     {\tl_put_right:Nn \l_tmpa_tl {0}}
    \array_gpush:Nx \l_hobby_actions_array {\l_tmpa_tl}
    \array_gpush:Nx \l_hobby_tension_out_array {\fp_to_tl:N \l_hobby_tempc_fp}
    \array_gpush:Nx \l_hobby_tension_in_array {\fp_to_tl:N \l_hobby_tempd_fp}
    \array_gpush:Nx \l_hobby_excess_angle_array {\fp_to_tl:N \l_hobby_temps_fp}
    \array_gpush:Nx \l_hobby_points_array {
      x = \fp_use:N \l_hobby_tempa_fp,
      y = \fp_use:N \l_hobby_tempb_fp }
    \array_gpush:Nx \l_hobby_points_x_array {\fp_to_tl:N \l_hobby_tempa_fp}
    \array_gpush:Nx \l_hobby_points_y_array {\fp_to_tl:N \l_hobby_tempb_fp}
}
```

\hobbysetparams This sets the parameters for the curve.

```
\NewDocumentCommand \hobbysetparams { m } {
  \keys_set:nn { hobby / read in params }
  {
    #1
  }
}
```

\hobby_set_cmds:nnn The path-generation code doesn't know what to actually do with the path so the initialisation code will set some macros to do that. This is an auxiliary command that sets these macros.

```
\cs_new:Npn \hobby_moveto:nnn #1#2#3 {}
\cs_new:Npn \hobby_curveto:nnn #1#2#3 {}
\cs_new:Npn \hobby_close:n #1 {}
\cs_generate_variant:Nn \hobby_moveto:nnn {VVV,nnV}
\cs_generate_variant:Nn \hobby_curveto:nnn {VVV}
\cs_generate_variant:Nn \hobby_close:n {V}
\cs_new:Nn \hobby_set_cmds:nnn {
  \cs_gset_eq:NN \hobby_moveto:nnn #1
  \cs_gset_eq:NN \hobby_curveto:nnn #2
  \cs_gset_eq:NN \hobby_close:n #3
}
```

\hobbygenpath This is the user (well, sort of) command that generates the curve.

```
\NewDocumentCommand \hobbygenpath { } {
  \array_if_empty:NF \l_hobby_points_array {
    \hobby_gen_path:
  }
}
```

\hobbygenifnecpath If the named path doesn't exist, it is generated and named. If it does exist, we restore it. Either way, we save it to the aux file.

```
\NewDocumentCommand \hobbygenifnecpath { m } {
  \tl_if_exist:cTF {g_hobby_#1_path}
  {
    \tl_use:c {g_hobby_#1_path}
  }
  {
    \hobby_gen_path:
  }
  \hobby_save_path:n {#1}
  \hobby_save_path_to_aux:x {#1}
}
```

\hobbygenifnecusepath If the named path doesn't exist, it is generated and named. If it does exist, we restore it. Either way, we save it to the aux file.

```
\NewDocumentCommand \hobbygenuseifnecpath { m } {
  \tl_if_exist:cTF {g_hobby_#1_path}
  {
    \tl_use:c {g_hobby_#1_path}
  }
  {
    \hobby_gen_path:
  }
  \hobby_save_path:n {#1}
  \hobby_save_path_to_aux:x {#1}
  \hobby_use_path:
}
```

\hobbyusepath This is the user (well, sort of) command that uses the last generated curve.

```
\NewDocumentCommand \hobbyusepath { m } {
  \hobbysetparams{#1}
  \hobby_use_path:
}
```

\hobbysavepath This is the user (well, sort of) command that uses the last generated curve.

```
\NewDocumentCommand \hobbysavepath { m } {
  \hobby_save_path:n {#1}
}
```

\hobbyrestorepath This is the user (well, sort of) command that uses the last generated curve.

```
\NewDocumentCommand \hobbyrestorepath { m } {
  \tl_if_exist:cT {g_hobby_#1_path} {
    \tl_use:c {g_hobby_#1_path}
  }
}
```

\hobbyshowpath This is the user (well, sort of) command that uses the last generated curve.

```
\NewDocumentCommand \hobbyshowpath { m } {
  \tl_if_exist:cT {g_hobby_#1_path} {
    \tl_show:c {g_hobby_#1_path}
  }
}
```

`\hobbygenusepath` This is the user (well, sort of) command that generates a curve and uses it.

```
\NewDocumentCommand \hobbygenusepath { } {
  \array_if_empty:NF \l_hobby_points_array {
    \hobby_gen_path:
    \hobby_use_path:
  }
}
```

`\hobbyclearpath` This is the user (well, sort of) command that generates a curve and uses it.

```
\NewDocumentCommand \hobbyclearpath { } {
  \hobby_clear_path:
}
```

`\hobby_use_path:` This is the command that uses the curve. As the curve data is stored globally, the same data can be reused by calling this function more than once without calling the generating function.

```
\tl_new:N \l_tmpc_tl
\cs_new:Nn \hobby_use_path: {
  \bool_if:NT \l_hobby_disjoint_bool {
    \array_get:NnN \l_hobby_points_array {0} \l_tmpa_tl
    \hobby_moveto:nnV {} {} \l_tmpa_tl
  }
  \int_step_inline:nnnn {1} {1} {\l_hobby_npoints_int} {
    \array_get:NnN \l_hobby_controla_array {##1} \l_tmpa_tl
    \array_get:NnN \l_hobby_controlb_array {##1} \l_tmpb_tl
    \array_get:NnN \l_hobby_points_array {##1} \l_tmpc_tl
    \array_get:NnN \l_hobby_actions_array {##1} \l_tmpd_tl
    \int_compare:nNnTF {\tl_item:Nn \l_tmpd_tl {1}} = {\l_hobby_draw_int} {
      \hobby_curveto:VVV \l_tmpa_tl \l_tmpb_tl \l_tmpc_tl
    }{
      \bool_gset_false:N \l_hobby_closed_bool
      \hobby_moveto:VVV \l_tmpa_tl \l_tmpb_tl \l_tmpc_tl
    }
    \tl_if_eq:xnTF {\tl_item:Nn \l_tmpd_tl {2}} {1} {
      \bool_gset_false:N \l_hobby_closed_bool
      \hobby_moveto:VVV \l_tmpa_tl \l_tmpb_tl \l_tmpc_tl
    }{}
  }
  \bool_if:NT \l_hobby_closed_bool {
    \array_get:NnN \l_hobby_points_array {0} \l_tmpa_tl
    \hobby_close:V \l_tmpa_tl
  }
}
```

`\hobby_save_path:n` This command saves all the data needed to reinvoke the curve in a global token list that can be used to restore it afterwards.

```
\cs_new:Nn \hobby_save_path:n {
  \tl_clear:N \l_tmpa_tl
  \tl_put_right:Nn \l_tmpa_tl {\int_gset:Nn \l_hobby_npoints_int}
  \tl_put_right:Nx \l_tmpa_tl {{\int_use:N \l_hobby_npoints_int}}
  \bool_if:NTF \l_hobby_disjoint_bool {
    \tl_put_right:Nn \l_tmpa_tl {\bool_gset_true:N}
  }{
    \tl_put_right:Nn \l_tmpa_tl {\bool_gset_false:N}
  }
  \tl_put_right:Nn \l_tmpa_tl {\l_hobby_disjoint_bool}
  \bool_if:NTF \l_hobby_closed_bool {
   \tl_put_right:Nn \l_tmpa_tl {\bool_gset_true:N}
  }{
   \tl_put_right:Nn \l_tmpa_tl {\bool_gset_false:N}
  }
  \tl_put_right:Nn \l_tmpa_tl {\l_hobby_closed_bool}
  \tl_put_right:Nn \l_tmpa_tl {\array_gclear:N \l_hobby_points_array}
  \array_map_inline:Nn \l_hobby_points_array {
   \tl_put_right:Nn \l_tmpa_tl {
    \array_gput:Nnn \l_hobby_points_array {##1} {##2}
   }
  }
  \tl_put_right:Nn \l_tmpa_tl {\array_gclear:N \l_hobby_actions_array}
  \array_map_inline:Nn \l_hobby_actions_array {
   \tl_put_right:Nn \l_tmpa_tl {
    \array_gput:Nnn \l_hobby_actions_array {##1} {##2}
   }
  }
  \tl_put_right:Nn \l_tmpa_tl {\array_gclear:N \l_hobby_controla_array}
  \array_map_inline:Nn \l_hobby_controla_array {
   \tl_put_right:Nn \l_tmpa_tl {
    \array_gput:Nnn \l_hobby_controla_array {##1} {##2}
   }
  }
  \tl_put_right:Nn \l_tmpa_tl {\array_gclear:N \l_hobby_controlb_array}
  \array_map_inline:Nn \l_hobby_controlb_array {
   \tl_put_right:Nn \l_tmpa_tl {
    \array_gput:Nnn \l_hobby_controlb_array {##1} {##2}
   }
  }
  \tl_gclear_new:c {g_hobby_#1_path}
  \tl_gset_eq:cN {g_hobby_#1_path} \l_tmpa_tl
}
```

bby_save_path_to_aux:n

```
\int_set:Nn \l_tmpa_int {\char_value_catcode:n {`@}}
\char_set_catcode_letter:N @
\cs_new:Npn \hobby_save_path_to_aux:n #1 {
  \bool_if:nT {
   \tl_if_exist_p:c {g_hobby_#1_path}
   &&
   ! \tl_if_exist_p:c {g_hobby_#1_path_saved}
   &&
   \l_hobby_save_aux_bool
  }
  {
   \tl_clear:N \l_tmpa_tl
   \tl_put_right:Nn \l_tmpa_tl {
    \ExplSyntaxOn
    \tl_gclear_new:c {g_hobby_#1_path}
    \tl_gput_right:cn {g_hobby_#1_path}
   }
   \tl_put_right:Nx \l_tmpa_tl {
    {\tl_to_str:c {g_hobby_#1_path}}
   }
   \tl_put_right:Nn \l_tmpa_tl {
    \ExplSyntaxOff
    }
    \protected@write\@auxout{}{
      \tl_to_str:N \l_tmpa_tl
    }
    \tl_new:c {g_hobby_#1_path_saved}
  }
}
\char_set_catcode:nn {`@} {\l_tmpa_int}
\cs_generate_variant:Nn \hobby_save_path_to_aux:n {x}
```

\hobby_clear_path:

```
\cs_new:Nn \hobby_clear_path:
{
\array_gclear:N \l_hobby_points_array
\array_gclear:N \l_hobby_points_x_array
\array_gclear:N \l_hobby_points_y_array
\array_gclear:N \l_hobby_angles_array
\array_gclear:N \l_hobby_actions_array
\array_gclear:N \l_hobby_distances_array
\array_gclear:N \l_hobby_tension_out_array
\array_gclear:N \l_hobby_tension_in_array
\array_gclear:N \l_hobby_excess_angle_array
\array_gclear:N \l_hobby_matrix_a_array
\array_gclear:N \l_hobby_matrix_b_array
\array_gclear:N \l_hobby_matrix_c_array
\array_gclear:N \l_hobby_matrix_d_array
\array_gclear:N \l_hobby_vector_u_array
\array_gclear:N \l_hobby_psi_array
\array_gclear:N \l_hobby_theta_array
\array_gclear:N \l_hobby_phi_array
\array_gclear:N \l_hobby_sigma_array
\array_gclear:N \l_hobby_rho_array
\array_gclear:N \l_hobby_controla_array
\array_gclear:N \l_hobby_controlb_array
\bool_gset_false:N \l_hobby_closed_bool
\bool_gset_false:N \l_hobby_disjoint_bool

  \int_gset:Nn \l_hobby_npoints_int {-1}
  \int_gset:Nn \l_hobby_draw_int {1}
  \fp_gset_eq:NN \l_hobby_in_angle_fp \c_inf_fp
  \fp_gset_eq:NN \l_hobby_out_angle_fp \c_inf_fp
  \fp_gset_eq:NN \l_hobby_in_curl_fp \c_one_fp
  \fp_gset_eq:NN \l_hobby_out_curl_fp \c_one_fp
}
```

```
\ExplSyntaxOff
```

1.2 PGF Library

The PGF level is very simple. All we do is set up the path-construction commands that get passed to the path-generation function.

```
\input{hobby.code.tex}
```

Points are communicated as key-pairs. These keys translate from the LaTeX3 style points to PGF points.

```
\pgfkeys{
  /pgf/hobby/.is family,
  /pgf/hobby/.cd,
  x/.code={\pgf@x=#1cm},
  y/.code={\pgf@y=#1cm}
}
```

`hobbyatan2` The original PGF version of atan2 had the arguments the wrong way around. This was fixed in the CVS version in July 2013, but as old versions are likely to be in use for some time, we define a wrapper function that ensures that the arguments are correct.

```
\pgfmathparse{atan2(0,1)}
\def\hobby@temp{0.0}
\ifx\pgfmathresult\hobby@temp
  \pgfmathdeclarefunction{hobbyatan2}{2}{%
    \pgfmathatantwo@{#1}{#2}%
  }
\else
  \pgfmathdeclarefunction{hobbyatan2}{2}{%
    \pgfmathatantwo@{#2}{#1}%
  }
\fi
```

`\hobby@curveto` This is passed to the path-generation code to translate the path into a PGF path.

```
\def\hobby@curveto#1#2#3{%
  \pgfpathcurveto{\hobby@topgf{#1}}{\hobby@topgf{#2}}{\hobby@topgf{#3}}%
}
```

`\hobby@moveto` This is passed to the path-generation code to translate the path into a PGF path.

```
\def\hobby@moveto#1#2#3{%
  \pgfpathmoveto{\hobby@topgf{#3}}%
}
```

`\hobby@topgf` Translates a LaTeX3 point to a PGF point.

```
\def\hobby@topgf#1{%
    \pgfqkeys{/pgf/hobby}{#1}%
}
```

`\hobby@close` Closes a path.

```
\def\hobby@close#1{%
  \pgfpathclose
}
```

`\pgfpathhobby` Low-level interface to the hobby construction. This sets up the commands and starts the iterator.

```
\def\pgfpathhobby{%
  \pgfutil@ifnextchar\bgroup{\pgfpath@hobby}{\pgfpath@hobby{}}}
\def\pgfpath@hobby#1{%
  \hobbyinit\hobby@moveto\hobby@curveto\hobby@close
  \hobbysetparams{#1}%
  \pgfmathsetmacro\hobby@x{\the\pgf@path@lastx/1cm}%
  \pgfmathsetmacro\hobby@y{\the\pgf@path@lasty/1cm}%
  \hobbyaddpoint{x = \hobby@x, y = \hobby@y}%
}
```

\pgfpathhobbypt Adds a point to the construction

```
\def\pgfpathhobbypt#1{%
  #1%
  \pgfmathsetmacro\hobby@x{\the\pgf@x/1cm}%
  \pgfmathsetmacro\hobby@y{\the\pgf@y/1cm}%
  \pgfutil@ifnextchar\bgroup{\pgfpathhobbyptparams}{\pgfpathhobbyptparams{}}%
}
```

\pgfpathhobbyptparams

```
\def\pgfpathhobbyptparams#1{%
  \hobbyaddpoint{#1,x = \hobby@x, y = \hobby@y}%
}
```

\pgfpathhobbyend

```
\def\pgfpathhobbyend{%
  \ifhobby@externalise
    \ifx\hobby@path@name\pgfutil@empty
      \hobbygenusepath
    \else
      \hobbygenuseifnecpath{\hobby@path@name}%
    \fi
  \else
    \hobbygenusepath
  \fi
  \ifx\hobby@path@name\pgfutil@empty
  \else
    \hobbysavepath{\hobby@path@name}%
  \fi
  \global\let\hobby@path@name=\pgfutil@empty
}
```

Plot handlers

\pgfplothanderhobby Basic plot handler; uses full algorithm but therefore expensive

```
\def\pgfplothandlerhobby{%
  \def\pgf@plotstreamstart{%
    \hobbyinit\hobby@moveto\hobby@curveto\hobby@close
    \global\let\pgf@plotstreampoint=\pgf@plot@hobby@firstpt
    \global\let\pgf@plotstreamspecial=\pgfutil@gobble
    \gdef\pgf@plotstreamend{%
      \ifhobby@externalise
       \ifx\hobby@path@name\pgfutil@empty
        \hobbygenusepath
       \else
        \hobbygenuseifnecpath{\hobby@path@name}%
       \fi
      \else
       \hobbygenusepath
      \fi
      \ifx\hobby@path@name\pgfutil@empty
      \else
       \hobbysavepath{\hobby@path@name}%
      \fi
      \global\let\hobby@path@name=\pgfutil@empty
    }%
    \let\tikz@scan@point@options=\pgfutil@empty
  }
}
```

plothandlerclosedhobby Same as above but produces a closed curve

```
\def\pgfplothandlerclosedhobby{%
  \def\pgf@plotstreamstart{%
    \hobbyinit\hobby@moveto\hobby@curveto\hobby@close
    \hobbysetparams{closed=true,disjoint=true}%
    \global\let\pgf@plotstreampoint=\pgf@plot@hobby@firstpt
    \global\let\pgf@plotstreamspecial=\pgfutil@gobble
    \gdef\pgf@plotstreamend{%
      \ifhobby@externalise
       \ifx\hobby@path@name\pgfutil@empty
        \hobbygenusepath
       \else
        \hobbygenuseifnecpath{\hobby@path@name}%
       \fi
      \else
       \hobbygenusepath
      \fi
      \ifx\hobby@path@name\pgfutil@empty
      \else
       \hobbysavepath{\hobby@path@name}%
      \fi
      \global\let\hobby@path@name=\pgfutil@empty
    }%
  }
}
```

pgf@plot@hobby@firstpt First point, move or line as appropriate and then start the algorithm.

```
\def\pgf@plot@hobby@firstpt#1{%
  \pgf@plot@first@action{#1}%
  \pgf@plot@hobby@handler{#1}%
  \global\let\pgf@plotstreampoint=\pgf@plot@hobby@handler
}
```

pgf@plot@hobby@handler Add points to the array for the algorithm to work on.

```
\def\pgf@plot@hobby@handler#1{%
    #1%
    \pgfmathsetmacro\hobby@x{\the\pgf@x/1cm}%
    \pgfmathsetmacro\hobby@y{\the\pgf@y/1cm}%
    \hobbyaddpoint{x = \hobby@x, y = \hobby@y}%
  }
```

gfplothandlerquickhobby Uses the “quick” algorithm.

```
\def\pgfplothandlerquickhobby{%
  \def\pgf@plotstreamstart{%
    \global\let\hobby@quick@curveto=\pgfpathcurveto
    \global\let\pgf@plotstreampoint=\pgf@plot@qhobby@firstpt
    \global\let\pgf@plotstreamspecial=\pgfutil@gobble
    \global\let\pgf@plotstreamend=\pgf@plot@qhobby@end
  }
}
```

gf@plot@qhobby@firstpt

Carry out first action (move or line) and save point.

```
\def\pgf@plot@qhobby@firstpt#1{%
  #1%
  \edef\hobby@temp{\noexpand\pgf@plot@first@action{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}}\hobby
  \xdef\hobby@qpoints{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
  \gdef\hobby@qpointa{}%
  \gdef\hobby@angle{}%
  \global\let\pgf@plotstreampoint=\pgf@plot@qhobby@secondpt
}
```

f@plot@qhobby@secondpt

Also need to save second point.

```
\def\pgf@plot@qhobby@secondpt#1{%
  #1%
  \xdef\hobby@qpointa{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
  \global\let\pgf@plotstreampoint=\pgf@plot@qhobby@handler
}
```

gf@plot@qhobby@handler

Wrapper around the computation macro that saves the variables globally.

```
\def\pgf@plot@qhobby@handler#1{%
  #1
  \edef\hobby@temp{\noexpand\hobby@quick@compute{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}}\hobby@t
  \global\let\hobby@qpointa=\hobby@qpointa
  \global\let\hobby@qpoints=\hobby@qpoints
  \global\let\hobby@angle=\hobby@angle
```

Also need to save some data for the last point

```
  \global\let\hobby@thetaone=\hobby@thetaone
  \global\let\hobby@phitwo=\hobby@phitwo
  \global\let\hobby@done=\hobby@done
  \global\let\hobby@omegaone=\hobby@omegaone
}
```

\pgf@plot@qhobby@end

Wrapper around the finalisation step.

```
\def\pgf@plot@qhobby@end{%
  \hobby@quick@computeend
}
```

\hobby@sf

Working with points leads to computations out of range so we scale to get them into the computable arena.

```
\pgfmathsetmacro\hobby@sf{10cm}
```

\hobby@quick@compute

This is the macro that does all the work of computing the control points. The argument is the current point, `\hobby@qpointa` is the middle point, and `\hobby@qpoints` is the first point.

```
\def\hobby@quick@compute#1{%
```

Save the current (second - counting from zero) point in `\pgf@xb` and `\pgf@yb`.

```
  #1%
  \pgf@xb=\pgf@x
  \pgf@yb=\pgf@y
```

Save the previous (first) point in `\pgf@xa` and `\pgf@ya`.

```
  \hobby@qpointa
  \pgf@xa=\pgf@x
  \pgf@ya=\pgf@y
```

Adjust so that (\pgf@xb,\pgf@yb) is the vector from second to third. Then compute and store the distance and angle of this vector. We view this as the vector *from* the midpoint and everything to do with that point has the suffix one. Note that we divide by the scale factor here.

```
  \advance\pgf@xb by -\pgf@xa
  \advance\pgf@yb by -\pgf@ya
  \pgfmathsetmacro\hobby@done{sqrt((\pgf@xb/\hobby@sf)^2 + (\pgf@yb/\hobby@sf)^2)}%
  \pgfmathsetmacro\hobby@omegaone{rad(hobbyatan2(\pgf@yb,\pgf@xb))}%
```

Now we do the same with the vector from the zeroth to the first point.

```
  \hobby@qpoints
  \advance\pgf@xa by -\pgf@x
  \advance\pgf@ya by -\pgf@y
  \pgfmathsetmacro\hobby@dzero{sqrt((\pgf@xa/\hobby@sf)^2 + (\pgf@ya/\hobby@sf)^2)}%
  \pgfmathsetmacro\hobby@omegazero{rad(hobbyatan2(\pgf@ya,\pgf@xa))}%
```

\hobby@psi is the angle subtended at the midpoint. We adjust to ensure that it is in the right range.

```
  \pgfmathsetmacro\hobby@psi{\hobby@omegaone - \hobby@omegazero}%
  \pgfmathsetmacro\hobby@psi{\hobby@psi > pi ? \hobby@psi - 2*pi : \hobby@psi}%
  \pgfmathsetmacro\hobby@psi{\hobby@psi < -pi ? \hobby@psi + 2*pi : \hobby@psi}%
```

Now we test to see if we're on the first run or not. If the first, we have no incoming angle.

```
  \ifx\hobby@angle\pgfutil@empty
```

First.

```
  \pgfmathsetmacro\hobby@thetaone{-\hobby@psi * \hobby@done%
/(\hobby@done + \hobby@dzero)}%
  \pgfmathsetmacro\hobby@thetazero{-\hobby@psi - \hobby@thetaone}%
  \let\hobby@phione=\hobby@thetazero
  \let\hobby@phitwo=\hobby@thetaone
  \else
```

Second or later.

```
  \let\hobby@thetazero=\hobby@angle
  \pgfmathsetmacro\hobby@thetaone{%
  -(2 * \hobby@psi + \hobby@thetazero) * \hobby@done%
  / (2 * \hobby@done + \hobby@dzero)}%
  \pgfmathsetmacro\hobby@phione{-\hobby@psi - \hobby@thetaone}%
  \let\hobby@phitwo=\hobby@thetaone
  \fi
```

Save the outgoing angle.

```
  \let\hobby@angle=\hobby@thetaone
```

Compute the control points from the angles.

```
  \hobby@quick@ctrlpts{\hobby@thetazero}{\hobby@phione}{\hobby@qpoints}{\hobby@qpointa}{\hobby@dzero}{
```

Now call the call-back function

```
  \edef\hobby@temp{\noexpand\hobby@quick@curveto{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}{\noexp
\hobby@temp
```

Cycle the points round for the next iteration.

```
  \global\let\hobby@qpoints=\hobby@qpointa
  #1
  \xdef\hobby@qpointa{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
```

Save needed values in global macros

```
  \global\let\hobby@angle=\hobby@angle
  \global\let\hobby@phitwo=\hobby@phitwo
  \global\let\hobby@thetaone=\hobby@thetaone
\global\let\hobby@done=\hobby@done
\global\let\hobby@omegaone=\hobby@omegaone
}
```

hobby@wuick@computeend

This is the additional code for the final run.

```
\def\hobby@quick@computeend{%
```

Compute the control points for the second part of the curve and add that to the path.

```
  \hobby@quick@ctrlpts{\hobby@thetaone}{\hobby@phitwo}{\hobby@qpoints}{\hobby@qpointa}{\hobby@done}{\h
```

Now call the call-back function

```
  \edef\hobby@temp{\noexpand\hobby@quick@curveto{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}{\noexp
\hobby@temp
}%
```

\hobby@quick@ctrlpts

Compute the control points from the angles and points given.

```
\def\hobby@quick@ctrlpts#1#2#3#4#5#6{%
  \pgfmathsetmacro\hobby@alpha{%
    sqrt(2) * (sin(#1 r) - 1/16 * sin(#2 r))%
* (sin(#2 r) - 1/16 * sin(#1 r))%
 * (cos(#1 r) - cos(#2 r))}%
  \pgfmathsetmacro\hobby@rho{%
    (2 + \hobby@alpha)/(1 + (1 - (3 - sqrt(5))/2)%
 * cos(#1 r) + (3 - sqrt(5))/2 * cos(#2 r))}%
  \pgfmathsetmacro\hobby@sigma{%
    (2 - \hobby@alpha)/(1 + (1 - (3 - sqrt(5))/2)%
  * cos(#2 r) +  (3 - sqrt(5))/2 * cos(#1 r))}%
  #3%
  \pgf@xa=\pgf@x
  \pgf@ya=\pgf@y
  \pgfmathsetlength\pgf@xa{%
    \pgf@xa + #5 * \hobby@rho%
  * cos((#1 + #6) r)/3*\hobby@sf}%
  \pgfmathsetlength\pgf@ya{%
    \pgf@ya + #5 * \hobby@rho%
  * sin((#1 + #6) r)/3*\hobby@sf}%
  #4%
  \pgf@xb=\pgf@x
  \pgf@yb=\pgf@y
  \pgfmathsetlength\pgf@xb{%
    \pgf@xb - #5 * \hobby@sigma%
  * cos((-#2 + #6) r)/3*\hobby@sf}%
  \pgfmathsetlength\pgf@yb{%
    \pgf@yb - #5 * \hobby@sigma%
  * sin((-#2 + #6) r)/3*\hobby@sf}%
  #4%
}
```

1.3 TikZ Library

```
\usepgflibrary{hobby}
\let\hobby@this@opts=\pgfutil@empty
\let\hobby@next@opts=\pgfutil@empty
\let\hobby@action=\pgfutil@empty
\let\hobby@path@name=\pgfutil@empty
\newif\ifhobby@externalise
```

We set various TikZ keys. These include the `to path` constructor and all the various parameters that will eventually get passed to the path-generation code.

```
\def\hobby@point@options{}%
\tikzset{
  curve through/.style={
    to path={
      \pgfextra{
        \expandafter\curvethrough\expandafter[\hobby@next@opts]{%
          (\tikztostart) .. #1 .. (\tikztotarget)%
        }
      }
    }
  },
  tension in/.code = {%
    \expandafter\gdef\expandafter\hobby@point@options\expandafter%
    {\hobby@point@options,tension in=#1}%
  },
  tension out/.code = {%
    \expandafter\gdef\expandafter\hobby@point@options\expandafter%
    {\hobby@point@options,tension out=#1}%
  },
  tension/.code = {%
    \expandafter\gdef\expandafter\hobby@point@options\expandafter%
    {\hobby@point@options,tension=#1}%
  },
  excess angle/.code = {%
    \expandafter\gdef\expandafter\hobby@point@options\expandafter%
    {\hobby@point@options,excess angle=#1}%
  },
  break/.code = {%
    \expandafter\gdef\expandafter\hobby@point@options\expandafter%
    {\hobby@point@options,break=#1}%
  },
  blank/.code = {%
    \expandafter\gdef\expandafter\hobby@point@options\expandafter%
    {\hobby@point@options,blank=#1}%
  },
  designated Hobby path/.initial={next},
  clear next Hobby path options/.code={%
    \gdef\hobby@next@opts{}%
  },
  clear this Hobby path options/.code={%
    \gdef\hobby@this@opts{}%
  },
  clear Hobby path options/.style={%
    clear \pgfkeysvalueof{/tikz/designated Hobby path} Hobby path options
  },
  add option to this Hobby path/.code={%
```

```
    \expandafter\gdef\expandafter\hobby@this@opts\expandafter{\hobby@this@opts#1,}%
  },
  add option to next Hobby path/.code={%
    \expandafter\gdef\expandafter\hobby@next@opts\expandafter{\hobby@next@opts#1,}%
  },
  add option to Hobby path/.style={%
    add option to \pgfkeysvalueof{/tikz/designated Hobby path} Hobby path={#1}%
  },
  closed/.style = {%
    add option to Hobby path={closed=#1,disjoint=#1}%
  },
  invert blank/.style = {%
    add option to Hobby path={invert blank=#1}%
  },
  closed/.default = true,
  blank/.default = true,
  break/.default = true,
  invert blank/.default = true,
  in angle/.code = {%
    \pgfmathparse{#1*pi/180}%
    \edef\@temp{in angle=\pgfmathresult,}%
    \pgfkeysalso{add option to Hobby path/.expand once=\@temp}%
  },
  out angle/.code = {%
    \pgfmathparse{#1*pi/180}%
    \edef\@temp{out angle=\pgfmathresult,}%
    \pgfkeysalso{add option to Hobby path/.expand once=\@temp}%
  },
  in curl/.style = {%
    add option to Hobby path={in curl=#1}%
  },
  out curl/.code = {%
    add option to Hobby path={out curl=#1}%
  },
  use Hobby shortcut/.code={%
    \let\tikz@curveto@auto=\hobby@curveto@override
    \global\let\hobby@curveto@delegate=\hobby@curveto@auto
  },
  use quick Hobby shortcut/.code={%
    \let\tikz@curveto@auto=\hobby@curveto@override
    \global\let\hobby@curveto@delegate=\hobby@qcurveto@auto
  },
  use previous Hobby path/.code={%
    \pgfextra{\hobbyusepath{#1}}
  },
  use previous Hobby path/.default={},%
  save Hobby path/.code={%
    \xdef\hobby@path@name{#1}%
  },
  restore Hobby path/.code={%
    \pgfextra{%
      \hobbyinit\hobby@tikz@moveto\hobby@tikz@curveto\hobby@tikz@close
      \global\let\hobby@collected@onpath\pgfutil@empty
      \hobbyrestorepath{#1}}
  },
  restore and use Hobby path/.code 2 args={%
```

```
    \pgfextra{%
      \hobbyinit\hobby@tikz@moveto\hobby@tikz@curveto\hobby@tikz@close
      \global\let\hobby@collected@onpath\pgfutil@empty
      \hobbyrestorepath{#1}%
      \hobbyusepath{#2}%
    }
  },
  show Hobby path/.code={%
    \pgfextra{\hobbyshowpath{#1}}
  },
  Hobby action/.code={%
    \expandafter\gdef\expandafter\hobby@action\expandafter{\hobby@action#1}%
  },
  Hobby finish/.style={%
    Hobby action=\hobby@finish%
  },
  Hobby externalise/.is if=hobby@externalise,
  Hobby externalize/.is if=hobby@externalise
}
```

\hobby@tikz@curveto This is passed to the path-generation code to translate the path into a PGF path.

```
\def\hobby@tikz@curveto#1#2#3{%
  \pgfutil@ifundefined{tikz@timer@start}{%
    \expandafter\hobby@topgf\expandafter{\hobby@initial@pt}%
    \edef\tikz@timer@start{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
  }{}%
  \hobby@topgf{#1}%
  \edef\tikz@timer@cont@one{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
  \hobby@topgf{#2}%
  \edef\tikz@timer@cont@two{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
  \hobby@topgf{#3}%
  \let\tikz@timer=\tikz@timer@curve
  \edef\tikz@timer@end{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
  \ifx\hobby@collected@onpath\pgfutil@empty
  \else
  \expandafter\hobby@nodes@onpath\hobby@collected@onpath\relax\relax
  \fi
  \pgfpathcurveto{\hobby@topgf{#1}}{\hobby@topgf{#2}}{\hobby@topgf{#3}}%
  \hobby@topgf{#3}%
  \edef\tikz@timer@start{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
}
```

\hobby@tikz@moveto This is passed to the path-generation code to translate the path into a PGF path.

```
\def\hobby@tikz@moveto#1#2#3{%
  \pgfutil@ifundefined{tikz@timer@start}{%
    \expandafter\hobby@topgf\expandafter{\hobby@initial@pt}%
    \edef\tikz@timer@start{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
  }{}%
  \hobby@topgf{#3}%
  \edef\tikz@timer@end{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
  \def\pgf@temp{#1}%
  \ifx\pgf@temp\pgfutil@empty
    \let\tikz@timer=\tikz@timer@line
  \else
    \hobby@topgf{#1}%
    \edef\tikz@timer@cont@one{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
    \hobby@topgf{#2}%
    \edef\tikz@timer@cont@two{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
    \let\tikz@timer=\tikz@timer@curve
  \fi
  \ifx\hobby@collected@onpath\pgfutil@empty
  \else
  \expandafter\hobby@nodes@onpath\hobby@collected@onpath\relax\relax
  \fi
  \pgfpathmoveto{\hobby@topgf{#3}}%
  \hobby@topgf{#3}%
  \edef\tikz@timer@start{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
}
```

\hobby@tikz@close Closes a path.

```
\def\hobby@tikz@close#1{%
  \hobby@topgf{#1}%
  \edef\tikz@timer@end{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
  \let\tikz@timer=\tikz@timer@line
  \ifx\hobby@collected@onpath\pgfutil@empty
  \else
  \expandafter\hobby@nodes@onpath\hobby@collected@onpath\relax\relax
  \fi
  \pgfpathclose
}
```

\hobby@nodes@onpath

```
\def\hobby@nodes@onpath#1#2\relax{%
  \gdef\hobby@collected@onpath{#2}%
  \def\pgf@temp{#1}%
  \ifx\pgf@temp\pgfutil@empty
  \else
  \def\@gtempa{\relax}
  \ifx\pgf@temp\@gtempa
  \else
  \tikz@node@is@a@labeltrue
  \tikz@scan@next@command#1\pgf@stop
  \tikz@node@is@a@labelfalse
  \fi
  \fi
}
```

\curvethrough This is the parent command. We initialise the path-generation code, set any parameters, and then hand over control to the point processing macro.

```
\newcommand\curvethrough[2][]{%
  \hobbyinit\hobby@tikz@moveto\hobby@tikz@curveto\hobby@tikz@close
  \global\let\hobby@collected@onpath\pgfutil@empty
  \let\hobby@initial@pt\pgfutil@empty
  \hobbysetparams{#1}%
  \tikzset{designated Hobby path=this}%
  \global\let\hobby@this@opts=\pgfutil@empty
  \global\let\hobby@next@opts=\pgfutil@empty
  \let\tikz@scan@point@options=\pgfutil@empty
  \def\hobby@point@options{}%
  \tikz@scan@one@point\hobby@processpt #2 \relax%
}
```

\hobby@processpt This processes a list of points in the format (0,0) [..] (1,1). Each point is scanned by TikZ and then added to the stack to be built into the path. If there are any remaining points, we call ourself again with them. Otherwise, we hand over control to the path-generation code.

```
\newcommand\hobby@processpt[1]{%
  #1%
  \pgfmathsetmacro\hobby@x{\the\pgf@x/1cm}%
  \pgfmathsetmacro\hobby@y{\the\pgf@y/1cm}%
  \ifx\hobby@initial@pt\pgfutil@empty
    \xdef\hobby@initial@pt{x = \hobby@x, y = \hobby@y}%
  \fi
  \expandafter\hobbyaddpoint\expandafter{\hobby@point@options,%
    x = \hobby@x, y = \hobby@y}%
  \def\hobby@point@options{}%
  \let\tikz@scan@point@options=\pgfutil@empty
  \pgfutil@ifnextchar\relax{%
    \expandafter\hobbysetparams\expandafter{\hobby@this@opts}%
  \ifhobby@externalise
    \ifx\hobby@path@name\pgfutil@empty
      \hobbygenusepath
    \else
      \hobbygenuseifnecpath{\hobby@path@name}%
    \fi
  \else
    \hobbygenusepath
  \fi
  \ifx\hobby@path@name\pgfutil@empty
  \else
    \hobbysavepath{\hobby@path@name}%
  \fi
  \global\let\hobby@path@name=\pgfutil@empty
  }{%
    \pgfutil@ifnextchar.{%
      \hobby@swallowdots}{%
      \tikz@scan@one@point\hobby@processpt}}}
```

\hobby@swallowdots Remove dots from the input stream.

```
\def\hobby@swallowdots.{%
  \pgfutil@ifnextchar.{%
    \hobby@swallowdots}{%
    \tikz@scan@one@point\hobby@processpt}}
```

There is a “spare hook” in the TikZ path processing code. If TikZ encounters a path of the form (0,0) .. (1,1) then it calls a macro \tikz@curveto@auto. However, that macro is not defined in the TikZ code. The following code provides a suitable definition. To play nice, we don’t install it by default but define a key (defined above) that installs it.

hobby@curveto@override

```
\def\hobby@curveto@override{%
  \hobby@curveto@delegate}
```

\hobby@curveto@auto When we’re called by TikZ, we initialise the path generation code and start adding points. To ensure that the generation code is called, we add a lot of hooks to lots of TikZ commands.

```
\def\hobby@curveto@auto{%
  \hobbyinit\hobby@tikz@moveto\hobby@tikz@curveto\hobby@tikz@close
  \expandafter\gdef\expandafter\hobby@collected@onpath\expandafter{\expandafter{\tikz@collected@onpath
  \let\tikz@collected@onpath=\pgfutil@empty
  \pgfmathsetmacro\hobby@x{\the\tikz@lastx/1cm}%
  \pgfmathsetmacro\hobby@y{\the\tikz@lasty/1cm}%
  \xdef\hobby@initial@pt{x = \hobby@x, y = \hobby@y}%
  \expandafter\hobbysetparams\expandafter{\hobby@next@opts}%
  \expandafter\hobbyaddpoint\expandafter{\hobby@point@options,%
      x = \hobby@x, y = \hobby@y}%
  \hobby@init@tikz@commands
  \tikzset{designated Hobby path=this}%
  \let\tikz@scan@point@options=\pgfutil@empty
  \global\let\hobby@action=\pgfutil@empty
  \global\let\hobby@this@opts=\pgfutil@empty
  \global\let\hobby@next@opts=\pgfutil@empty
  \global\let\hobby@point@options=\pgfutil@empty
  \tikz@scan@one@point\hobby@addfromtikz%
}
```

\hobby@addfromtikz

This adds our current point to the stack.

```
\def\hobby@addfromtikz#1{%
  #1%
  \tikz@make@last@position{#1}%
  \pgfmathsetmacro\hobby@x{\the\pgf@x/1cm}%
  \pgfmathsetmacro\hobby@y{\the\pgf@y/1cm}%
  \expandafter\hobbysetparams\expandafter{\hobby@this@opts}%
  \expandafter\hobbyaddpoint\expandafter{\hobby@point@options,%
    x = \hobby@x, y = \hobby@y}%
  \hobby@action
  \global\let\hobby@this@opts=\pgfutil@empty
  \global\let\hobby@action=\pgfutil@empty
  \global\let\hobby@point@options=\pgfutil@empty
  \tikz@scan@next@command%
}
```

bby@init@tikz@commands

```
\def\hobby@init@tikz@commands{%
  \hobby@init@tikz@modcmd\tikz@movetoabs
  \hobby@init@tikz@modcmd\tikz@movetorel
  \hobby@init@tikz@modcmd\tikz@lineto
  \hobby@init@tikz@modcmd\tikz@rect
  \hobby@init@tikz@modcmd\tikz@cchar
  \hobby@init@tikz@modcmd\tikz@finish
  \hobby@init@tikz@modcmd\tikz@arcA
  \hobby@init@tikz@modcmd\tikz@e@char
  \hobby@init@tikz@modcmd\tikz@g@char
  \hobby@init@tikz@modcmd\tikz@schar
  \hobby@init@tikz@modcmd\tikz@vh@lineto
  \hobby@init@tikz@modcmd\tikz@pchar
  \hobby@init@tikz@modcmd\tikz@to
  \hobby@init@tikz@modcmd\pgf@stop
  \hobby@init@tikz@modcmd\tikz@decoration
  \global\let\hobby@curveto@delegate=\hobby@midcurveto@auto
}
```

@restore@tikz@commands

```
\def\hobby@restore@tikz@commands{%
  \hobby@restore@tikz@modcmd\tikz@movetoabs
  \hobby@restore@tikz@modcmd\tikz@movetorel
  \hobby@restore@tikz@modcmd\tikz@lineto
  \hobby@restore@tikz@modcmd\tikz@rect
  \hobby@restore@tikz@modcmd\tikz@cchar
  \hobby@restore@tikz@modcmd\tikz@finish
  \hobby@restore@tikz@modcmd\tikz@arcA
  \hobby@restore@tikz@modcmd\tikz@e@char
  \hobby@restore@tikz@modcmd\tikz@g@char
  \hobby@restore@tikz@modcmd\tikz@schar
  \hobby@restore@tikz@modcmd\tikz@vh@lineto
  \hobby@restore@tikz@modcmd\tikz@pchar
  \hobby@restore@tikz@modcmd\tikz@to
  \hobby@restore@tikz@modcmd\pgf@stop
  \hobby@restore@tikz@modcmd\tikz@decoration
  \global\let\hobby@curveto@delegate=\hobby@curveto@auto
}
```

hobby@init@tikz@modcmd

```
\def\hobby@init@tikz@modcmd#1{%
    \expandafter\global\expandafter\let\csname hobby@orig@\string#1\endcsname=#1%
    \gdef#1{\hobby@finish#1}%
}
```

by@restore@tikz@modcmd

```
\def\hobby@restore@tikz@modcmd#1{%
    \expandafter\global\expandafter\let\expandafter#1\csname hobby@orig@\string#1\endcsname%
}
```

\hobby@midcurveto@auto

```
\def\hobby@midcurveto@auto{%
  \expandafter\expandafter\expandafter\gdef\expandafter\expandafter\expandafter\hobby@collected@onpath
  \let\tikz@collected@onpath=\pgfutil@empty
  \let\tikz@scan@point@options=\pgfutil@empty
  \global\let\hobby@action=\pgfutil@empty
  \global\let\hobby@this@opts=\pgfutil@empty
  \global\let\hobby@point@options=\pgfutil@empty
  \tikz@scan@one@point\hobby@addfromtikz%
}
```

\hobby@finish

```
\def\hobby@finish{%
  \ifhobby@externalise
    \ifx\hobby@path@name\pgfutil@empty
      \hobbygenusepath
    \else
      \hobbygenuseifnecpath{\hobby@path@name}%
    \fi
  \else
    \hobbygenusepath
  \fi
  \ifx\hobby@path@name\pgfutil@empty
  \else
    \hobbysavepath{\hobby@path@name}%
  \fi
  \global\let\hobby@path@name=\pgfutil@empty
  \hobby@restore@tikz@commands
}
```

quickcurvethrough The quick curve through is a to path which does the “quick” version of Hobby’s algorithm. The syntax is as with the curve through: to pass the midpoints as the argument to the style. We need to pass three points to the auxiliary macro. These are passed as \hobby@qpoints, \hobby@qpointa, and the current point. Then these get cycled round for the next triple. The path gets built up and stored as \hobby@quick@path. We also have to remember the angle computed for the next round.

```
\tikzset{
  quick curve through/.style={%
    to path={%
      \pgfextra{%
```

Scan the starting point and store the coordinates in \hobby@qpointa

```
        \let\hobby@next@qbreak=\relax
        \let\hobby@next@qblank=\relax
      \tikz@scan@one@point\pgfutil@firstofone(\tikztostart)%
        \tikz@make@last@position{\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
        \edef\hobby@qpoints{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
```

Blank the path and auxiliary macros.

```
        \def\hobby@qpointa{}%
        \def\hobby@quick@path{}%
        \def\hobby@angle{}%
        \let\hobby@quick@curveto=\hobby@quick@makepath
```

Now start parsing the rest of the coordinates.

```
        \tikz@scan@one@point\hobby@quickfirst #1 (\tikztotarget)\relax
      }
```

Invoke the path

```
      \hobby@quick@path
    }
  },
  quick hobby/blank curve/.is choice,
  quick hobby/blank curve/true/.code={%
    \gdef\hobby@next@qblank{%
      \qhobby@blanktrue
      \global\let\hobby@next@qblank=\relax
    }%
  },
  quick hobby/blank curve/false/.code={%
    \gdef\hobby@next@qblank{%
      \qhobby@blankfalse
      \global\let\hobby@next@qblank=\relax
    }%
  },
  quick hobby/blank curve/once/.code={%
    \gdef\hobby@next@qblank{%
      \qhobby@blanktrue
      \gdef\hobby@next@qblank{%
        \qhobby@blankfalse
        \global\let\hobby@next@qblank=\relax
      }%
    }%
  },
  quick hobby/blank curve/.default=true,
  quick hobby/break curve/.is choice,
  quick hobby/break curve/true/.code={%
    \gdef\hobby@next@qbreak{%
```

```
      \qhobby@breaktrue
      \global\let\hobby@next@qbreak=\relax
    }%
  },
  quick hobby/break curve/false/.code={%
    \gdef\hobby@next@qbreak{%
      \qhobby@breakfalse
      \global\let\hobby@next@qbreak=\relax
    }%
  },
  quick hobby/break curve/once/.code={%
    \gdef\hobby@next@qbreak{%
      \qhobby@breaktrue
      \gdef\hobby@next@qbreak{%
        \qhobby@breakfalse
        \global\let\hobby@next@qbreak=\relax
      }%
    }%
  },
  quick hobby/break curve/.default=true,
}
\newif\ifqhobby@break
\newif\ifqhobby@blank
```

Add plot handlers

```
\tikzoption{hobby}[]{\let\tikz@plot@handler=\pgfplothandlerhobby}
\tikzoption{quick hobby}[]{\let\tikz@plot@handler=\pgfplothandlerquickhobby}
\tikzoption{closed hobby}[]{\let\tikz@plot@handler=\pgfplothandlerclosedhobby}
```

`\hobby@quickfirst` The first time around we just set the next point.

```
\def\hobby@quickfirst#1{%
  #1%
  \xdef\hobby@qpointa{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
  \tikz@make@last@position{\hobby@qpointa}%
```

Now a check to ensure that we have more points.

```
  \pgfutil@ifnextchar\relax{%
```

Ooops, no more points. That's not good. Bail-out.

```
    \xdef\hobby@quick@path{ -- (\the\pgf@x,\the\pgf@y)}%
  }{%
```

Okay, have more points. Phew. Call the next round. If we have dots, swallow them.

```
    \pgfutil@ifnextchar.{%
      \hobby@qswallowdots}{%
    \tikz@scan@one@point\hobby@quick}}}
```

`\hobby@qswallowdots` Remove dots from the input stream.

```
\def\hobby@qswallowdots.{%
  \pgfutil@ifnextchar.{%
    \hobby@qswallowdots}{%
    \tikz@scan@one@point\hobby@quick}}
```

`\hobby@quick` This is our wrapper function that handles the loop.

```
\def\hobby@quick#1{%
  \hobby@quick@compute{#1}%
  \tikz@make@last@position{\hobby@qpointa}%
  \pgfutil@ifnextchar\relax{%
```

End of loop

```
    \hobby@quick@computeend%
  }{%
```

More to go, scan in the next coordinate and off we go again.

```
    \pgfutil@ifnextchar.{%
      \hobby@qswallowdots}{%
      \tikz@scan@one@point\hobby@quick}}}
```

\hobby@quick@makepath

Path constructor for `to path` use.

```
\def\hobby@quick@makepath#1#2#3{%
  #1%
  \pgf@xa=\pgf@x\relax
  \pgf@ya=\pgf@y\relax
  #2%
  \pgf@xb=\pgf@x\relax
  \pgf@yb=\pgf@y\relax
  #3%
  \ifqhobby@blank
  \xdef\hobby@quick@path{\hobby@quick@path (\the\pgf@x,\the\pgf@y)}%
  \else
  \xdef\hobby@quick@path{\hobby@quick@path .. controls%
  (\the\pgf@xa,\the\pgf@ya) and (\the\pgf@xb,\the\pgf@yb) .. (\the\pgf@x,\the\pgf@y) }%
  \fi
  \ifqhobby@break
  \xdef\hobby@quick@path{\hobby@quick@path +(0,0)}%
  \fi
  \hobby@next@qbreak
  \hobby@next@qblank
}
```

\hobby@qcurveto@auto

Uses the “quick” method for the shortcut syntax.

```
\def\hobby@qcurveto@auto{%
  \global\let\hobby@next@qbreak=\relax
  \global\let\hobby@next@qblank=\relax
  \xdef\hobby@qpoints{\noexpand\pgfqpoint{\the\tikz@lastx}{\the\tikz@lasty}}%
  \gdef\hobby@qpointa{}%
  \gdef\hobby@quick@path{}%
  \gdef\hobby@angle{}%
  \global\let\hobby@quick@curveto=\hobby@quick@makepathauto
  \hobby@qinit@tikz@commands
  \let\tikz@scan@point@options=\pgfutil@empty
  \global\let\hobby@action=\pgfutil@empty
  \global\let\hobby@point@options=\pgfutil@empty
  \tikz@scan@one@point\hobby@qfirst@auto}
```

hobby@qmidcurveto@auto

```
\def\hobby@qmidcurveto@auto{%
  \let\tikz@scan@point@options=\pgfutil@empty
  \global\let\hobby@action=\pgfutil@empty
  \global\let\hobby@point@options=\pgfutil@empty
  \tikz@scan@one@point\hobby@qaddfromtikz}
```

\hobby@qfirst@auto

```
\def\hobby@qfirst@auto#1{%
  #1%
  \xdef\hobby@qpointa{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
  \tikz@make@last@position{\hobby@qpointa}%
  \tikz@scan@next@command%
}
```

bby@quick@makepathauto

Path constructor for shortcut method to use.

```
\def\hobby@quick@makepathauto#1#2#3{%
  #1%
  \pgf@xa=\pgf@x\relax
  \pgf@ya=\pgf@y\relax
  #2%
  \pgf@xb=\pgf@x\relax
  \pgf@yb=\pgf@y\relax
  #3%
  \ifqhobby@blank
  \edef\hobby@temp{%
    \noexpand\pgfpathmoveto{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
  }%
  \hobby@temp
  \else
  \edef\hobby@temp{%
    \noexpand\pgfpathcurveto{\noexpand\pgfqpoint{\the\pgf@xa}{\the\pgf@ya}}%
    {\noexpand\pgfqpoint{\the\pgf@xb}{\the\pgf@yb}}%
    {\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
  }%
  \hobby@temp
  \fi
  \ifqhobby@break
  #3%
  \edef\hobby@temp{%
    \noexpand\pgfpathmoveto{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}%
  }%
  \hobby@temp
  \fi
  \hobby@next@qbreak
  \hobby@next@qblank
}
```

\hobby@qaddfromtikz

This adds our current point to the stack.

```
\def\hobby@qaddfromtikz#1{%
  \hobby@quick@compute{#1}%
  \tikz@make@last@position{\hobby@qpointa}%
  \tikz@scan@next@command%
}
```

by@qinit@tikz@commands

```
\def\hobby@qinit@tikz@commands{%
  \hobby@qinit@tikz@modcmd\tikz@movetoabs
  \hobby@qinit@tikz@modcmd\tikz@movetorel
  \hobby@qinit@tikz@modcmd\tikz@lineto
  \hobby@qinit@tikz@modcmd\tikz@rect
  \hobby@qinit@tikz@modcmd\tikz@cchar
  \hobby@qinit@tikz@modcmd\tikz@finish
  \hobby@qinit@tikz@modcmd\tikz@arcA
  \hobby@qinit@tikz@modcmd\tikz@e@char
  \hobby@qinit@tikz@modcmd\tikz@g@char
  \hobby@qinit@tikz@modcmd\tikz@schar
  \hobby@qinit@tikz@modcmd\tikz@vh@lineto
  \hobby@qinit@tikz@modcmd\tikz@pchar
  \hobby@qinit@tikz@modcmd\tikz@to
  \hobby@qinit@tikz@modcmd\pgf@stop
  \hobby@qinit@tikz@modcmd\tikz@decoration
  \hobby@qinit@tikz@modcmd\tikz@@close
  \global\let\hobby@curveto@delegate=\hobby@qmidcurveto@auto
}
```

\hobby@qrestore@tikz@commands

```
\def\hobby@qrestore@tikz@commands{%
  \hobby@restore@tikz@modcmd\tikz@movetoabs
  \hobby@restore@tikz@modcmd\tikz@movetorel
  \hobby@restore@tikz@modcmd\tikz@lineto
  \hobby@restore@tikz@modcmd\tikz@rect
  \hobby@restore@tikz@modcmd\tikz@cchar
  \hobby@restore@tikz@modcmd\tikz@finish
  \hobby@restore@tikz@modcmd\tikz@arcA
  \hobby@restore@tikz@modcmd\tikz@e@char
  \hobby@restore@tikz@modcmd\tikz@g@char
  \hobby@restore@tikz@modcmd\tikz@schar
  \hobby@restore@tikz@modcmd\tikz@vh@lineto
  \hobby@restore@tikz@modcmd\tikz@pchar
  \hobby@restore@tikz@modcmd\tikz@to
  \hobby@restore@tikz@modcmd\pgf@stop
  \hobby@restore@tikz@modcmd\tikz@decoration
  \hobby@restore@tikz@modcmd\tikz@@close
  \global\let\hobby@curveto@delegate=\hobby@qcurveto@auto
}
```

obby@qinit@tikz@modcmd

```
\def\hobby@qinit@tikz@modcmd#1{%
    \expandafter\global\expandafter\let\csname hobby@orig@\string#1\endcsname=#1%
    \gdef#1{\hobby@qfinish#1}%
}
```

\hobby@qfinish

```
\def\hobby@qfinish{%
  \hobby@quick@computeend
  \hobby@qrestore@tikz@commands
}
```

1.4 Arrays

A lot of our data structures are really arrays. These are implemented as LaTeX3 "property lists". For ease of use, an array is a property list with numeric entries together with entries "base" and "top" which hold the lowest and highest indices that have been set.

```
\RequirePackage{expl3}
\ExplSyntaxOn
```

Some auxiliary variables.

```
\tl_new:N \l_array_tmp_tl
\tl_new:N \l_array_show_tl
\int_new:N \l_array_base_int
```

```
\int_new:N \l_array_top_int
\int_new:N \l_array_tmp_int
```

The global variable `\g_array_base_int` says what index a blank array should start with when pushed or unshifted.

```
\int_new:N \g_array_base_int
\int_set:Nn \g_array_base_int {0}
```

`\array_adjust_ends:Nn` This ensures that the “base” and “top” are big enough to include the given index.

```
\cs_new:Npn \array_adjust_ends:Nn #1#2 {
  \prop_get:NnNTF #1 {base} \l_tmpa_tl
  {
    \int_compare:nNnTF {\l_tmpa_tl} > {#2}
    {
      \prop_put:Nnx #1 {base} {\int_eval:n {#2}}
    }
    {}
  }
  {
    \prop_put:Nnx #1 {base} {\int_eval:n {#2}}
  }
  \prop_get:NnNTF #1 {top} \l_tmpa_tl
  {
    \int_compare:nNnTF {\l_tmpa_tl} < {#2}
    {
      \prop_put:Nnx #1 {top} {\int_eval:n {#2}}
    }
    {}
  }
  {
    \prop_put:Nnx #1 {top} {\int_eval:n {#2}}
  }
}
```

`\array_gadjust_ends:Nn` This ensures that the “base” and “top” are big enough to include the given index. (Global version)

```
\cs_new:Npn \array_gadjust_ends:Nn #1#2 {
  \prop_get:NnNTF #1 {base} \l_tmpa_tl
  {
    \int_compare:nNnTF {\l_tmpa_tl} > {#2}
    {
      \prop_gput:Nnx #1 {base} {\int_eval:n {#2}}
    }
    {}
  }
  {
    \prop_gput:Nnx #1 {base} {\int_eval:n {#2}}
  }
  \prop_get:NnNTF #1 {top} \l_tmpa_tl
  {
    \int_compare:nNnTF {\l_tmpa_tl} < {#2}
    {
      \prop_gput:Nnx #1 {top} {\int_eval:n {#2}}
    }
    {}
  }
  {
    \prop_gput:Nnx #1 {top} {\int_eval:n {#2}}
  }
}
```

\array_put:Nnn When adding a value to an array we have to adjust the ends.

```
\cs_new:Npn \array_put:Nnn #1#2#3 {
  \exp_args:NNx \prop_put:Nnn #1 {\int_eval:n {#2}} {#3}
  \array_adjust_ends:Nn #1{#2}
}
\cs_generate_variant:Nn \array_put:Nnn {Nnx}
```

\array_gput:Nnn When adding a value to an array we have to adjust the ends. (Global version)

```
\cs_new:Npn \array_gput:Nnn #1#2#3 {
  \exp_args:NNx \prop_gput:Nnn #1 {\int_eval:n {#2}} {#3}
  \array_gadjust_ends:Nn #1{#2}
}
\cs_generate_variant:Nn \array_gput:Nnn {Nnx}
```

\array_get:NnN

```
\cs_new:Npn \array_get:NnN #1#2#3 {
  \exp_args:NNx \prop_get:NnN #1 {\int_eval:n {#2}} #3
}
```

\array_get:Nn

```
\cs_new:Npn \array_get:Nn #1#2 {
  \exp_args:NNf \prop_get:Nn #1 { \int_eval:n {#2} }
}
```

\array_get:NnNTF

```
\cs_new:Npn \array_get:NnNTF #1#2#3#4#5 {
  \exp_args:NNx \prop_get:NnNTF #1 {\int_eval:n {#2}} #3 {#4}{#5}
}
```

\array_if_empty:NTF

```
\prg_new_conditional:Npnn \array_if_empty:N #1 { p, T, F, TF }
{
  \if_meaning:w #1 \c_empty_prop
    \prg_return_true:
  \else:
    \prg_return_false:
  \fi:
}
```

\array_new:N

```
\cs_new_eq:NN \array_new:N \prop_new:N
```

\array_clear:N

```
\cs_new_eq:NN \array_clear:N \prop_clear:N
```

\array_gclear:N

```
\cs_new_eq:NN \array_gclear:N \prop_gclear:N
```

\array_map_function When stepping through an array, we want to iterate in order so a simple wrapper to \prop_map_function is not enough. This maps through every value from the base to the top so the function should be prepared to deal with a \q_no_value.

```
\cs_new:Npn \array_map_function:NN #1#2
{
  \array_if_empty:NTF #1 {} {
    \prop_get:NnNTF #1 {base} \l_array_tmp_tl {
      \int_set:Nn \l_array_base_int {\l_array_tmp_tl}
    }{
      \int_set:Nn \l_array_base_int {0}
    }
    \prop_get:NnNTF #1 {top} \l_array_tmp_tl {
      \int_set:Nn \l_array_top_int {\l_array_tmp_tl}
    }{
      \int_set:Nn \l_array_top_int {0}
    }
    \int_step_inline:nnnn {\l_array_base_int} {1} {\l_array_top_int} {
  \array_get:NnN #1 {##1} \l_array_tmp_tl
  \exp_args:NnV #2 {##1} \l_array_tmp_tl
}
} {}
}
\cs_generate_variant:Nn \array_map_function:NN {     Nc }
\cs_generate_variant:Nn \array_map_function:NN { c , cc }
```

y_reverse_map_function This steps through the array in reverse order.

```
\cs_new:Npn \array_reverse_map_function:NN #1#2
{
  \array_if_empty:NTF #1 {} {
    \prop_get:NnNTF #1 {base} \l_array_tmp_tl {
      \int_set:Nn \l_array_base_int {\l_array_tmp_tl}
    }{
      \int_set:Nn \l_array_base_int {0}
    }
    \prop_get:NnNTF #1 {top} \l_array_tmp_tl {
      \int_set:Nn \l_array_top_int {\l_array_tmp_tl}
    }{
      \int_set:Nn \l_array_top_int {0}
    }
    \int_step_inline:nnnn {\l_array_top_int} {-1} {\l_array_base_int} {
  \array_get:NnN #1 {##1} \l_array_tmp_tl
  \exp_args:Nno #2 {##1} \l_array_tmp_tl
}
} {}
}
\cs_generate_variant:Nn \array_reverse_map_function:NN {     Nc }
\cs_generate_variant:Nn \array_reverse_map_function:NN { c , cc }
```

\array_map_inline:Nn Inline version of the above.

```
\cs_new_protected:Npn \array_map_inline:Nn #1#2
  {
    \int_gincr:N \g__prg_map_int
    \cs_gset:cpn { array_map_inline_ \int_use:N \g__prg_map_int :nn }
      ##1##2 {#2}
    \exp_args:NNc \array_map_function:NN #1
      { array_map_inline_ \int_use:N \g__prg_map_int :nn }
    \__prg_break_point:Nn \array_map_break: { \int_gdecr:N \g__prg_map_int }
  }
\cs_generate_variant:Nn \array_map_inline:Nn { c }
```

_reverse_map_inline:Nn Inline version of the above.

```
\cs_new_protected:Npn \array_reverse_map_inline:Nn #1#2
  {
    \int_gincr:N \g__prg_map_int
    \cs_gset:cpn { array_map_inline_ \int_use:N \g__prg_map_int :nn }
      ##1##2 {#2}
    \exp_args:NNc \array_reverse_map_function:NN #1
      { array_map_inline_ \int_use:N \g__prg_map_int :nn }
    \__prg_break_point:Nn \array_map_break: { \int_gdecr:N \g__prg_map_int }
  }
\cs_generate_variant:Nn \array_reverse_map_inline:Nn { c }
```

\array_map_break:

```
\cs_new_nopar:Npn \array_map_break:
  { \__prg_map_break:Nn \array_map_break: { } }
\cs_new_nopar:Npn \array_map_break:n
  { \__prg_map_break:Nn \array_map_break: }
```

For displaying arrays, we need some messages.

```
\__msg_kernel_new:nnn { kernel } { show-array }
  {
    The~array~\token_to_str:N #1~
    \array_if_empty:NTF #1
      { is~empty }
      { contains~the~items~(without~outer~braces): }
  }
```

\array_show:N Mapping through an array isn't expandable so we have to set a token list to its contents first before passing it to the message handler.

```
\cs_new_protected:Npn \array_show:N #1
  {
    \tl_clear:N \l_array_show_tl
    \array_map_function:NN #1 \array_show_aux:nn
    \__msg_show_variable:Nno
      #1
      { array }
    { \l_array_show_tl }
  }
\cs_generate_variant:Nn \__msg_show_variable:Nnn { Nno }

\cs_new_protected:Npn \array_show_aux:nn #1#2
{
  \tl_if_eq:nnTF {#2} {\q_no_value} {}
  {
  \tl_put_right:No \l_array_show_tl {\__msg_show_item:nn {#1}{#2}}
  }
}
\cs_generate_variant:Nn \array_show:N { c }
```

\array_push:Nn

```
\cs_new_protected:Npn \array_push:Nn #1#2
{
\prop_get:NnNTF #1 {top} \l_array_tmp_tl
{
  \int_set:Nn \l_array_tmp_int {\l_array_tmp_tl}
  \int_incr:N \l_array_tmp_int
  \array_put:Nnn #1 {\l_array_tmp_int} {#2}
}
{
  \array_put:Nnn #1 {\g_array_base_int} {#2}
}
}
\cs_generate_variant:Nn \array_push:Nn {Nx}
```

\array_gpush:Nn b

```
\cs_new_protected:Npn \array_gpush:Nn #1#2
{
  \prop_get:NnNTF #1 {top} \l_array_tmp_tl
  {
    \int_set:Nn \l_array_tmp_int {\l_array_tmp_tl}
    \int_incr:N \l_array_tmp_int
    \array_gput:Nnn #1 {\l_array_tmp_int} {#2}
  }
  {
    \array_gput:Nnn #1 {\g_array_base_int} {#2}
  }
}
\cs_generate_variant:Nn \array_gpush:Nn {Nx}
```

\array_unshift:Nn

```
\cs_new_protected:Npn \array_unshift:Nn #1#2
{
  \prop_get:NnNTF #1 {base} \l_array_tmp_tl
  {
    \int_set:Nn \l_array_tmp_int {\l_array_tmp_tl}
    \int_decr:N \l_array_tmp_int
    \array_put:Nnn #1 {\l_array_tmp_int} {#2}
  }
  {
    \array_put:Nnn #1 {\g_array_base_int} {#2}
  }
}
\cs_generate_variant:Nn \array_unshift:Nn {Nx}
```

\array_gunshift:Nn

```
\cs_new_protected:Npn \array_gunshift:Nn #1#2
{
  \prop_get:NnNTF #1 {base} \l_array_tmp_tl
  {
    \int_set:Nn \l_array_tmp_int {\l_array_tmp_tl}
    \int_decr:N \l_array_tmp_int
    \array_gput:Nnn #1 {\l_array_tmp_int} {#2}
  }
  {
    \array_gput:Nnn #1 {\g_array_base_int} {#2}
  }
}
\cs_generate_variant:Nn \array_gunshift:Nn {Nx}
```

\array_pop:NN

```
\cs_new_protected:Npn \array_pop:NN #1#2
{
  \prop_get:NnN #1 {top} \l_array_tmp_tl
  \array_get:NnN #1 {\l_array_tmp_tl} #2
  \array_del:Nn #1 {\l_array_tmp_tl}
}
```

\array_gpop:NN

```
\cs_new_protected:Npn \array_gpop:NN #1#2
{
  \prop_get:NnN #1 {top} \l_array_tmp_tl
  \array_get:NnN #1 {\l_array_tmp_tl} #2
  \array_gdel:Nn #1 {\l_array_tmp_tl}
}
```

\array_shift:NN

```
\cs_new_protected:Npn \array_shift:NN #1#2
{
  \prop_get:NnN #1 {base} \l_array_tmp_tl
  \array_get:NnN #1 {\l_array_tmp_tl} #2
  \array_del:Nn #1 {\l_array_tmp_tl}
}
```

\array_gshift:NN

```
\cs_new_protected:Npn \array_gshift:NN #1#2
{
  \prop_get:NnN #1 {base} \l_array_tmp_tl
  \array_get:NnN #1 {\l_array_tmp_tl} #2
  \array_gdel:Nn #1 {\l_array_tmp_tl}
}
```

\array_top:NN

```
\cs_new_protected:Npn \array_top:NN #1#2
{
  \prop_get:NnN #1 {top} \l_array_tmp_tl
  \array_get:NnN #1 {\l_array_tmp_tl} #2
}
```

\array_base:NN

```
\cs_new_protected:Npn \array_base:NN #1#2
{
  \prop_get:NnN #1 {base} \l_array_tmp_tl
  \array_get:NnN #1 {\l_array_tmp_tl} #2
}
```

\array_top:N

```
\cs_new:Npn \array_top:N #1
{
  \array_get:Nn #1 {\prop_get:Nn #1 {top}}
}
```

\array_base:N

```
\cs_new:Npn \array_base:N #1
{
  \array_get:Nn #1 {\prop_get:Nn #1 {base}}
}
```

\array_del:Nn

```
\cs_new_protected:Npn \array_del:Nn #1#2
{
  \exp_args:NNx \prop_pop:Nn #1 {\int_eval:n {#2}}
  \int_set:Nn \l_array_tmp_int {0}
  \array_map_inline:Nn #1 {
    \tl_if_eq:NNTF {##2} {\q_no_value} {}
    {
      \int_incr:N \l_array_tmp_int
    }
  }
  \int_compare:nNnTF {\l_array_tmp_int} = {0}
  {
    \prop_clear:N #1
  }
  {
  \prop_get:NnN #1 {top} \l_array_tmp_tl
  \int_compare:nNnTF {#2} = {\l_array_tmp_tl} {
    \prop_get:NnN #1 {base} \l_array_tmp_tl
    \int_set:Nn \l_array_tmp_int {\l_array_tmp_tl}
    \array_map_inline:Nn #1 {
    \tl_if_eq:NNTF {##2} {\q_no_value} {}
    {
      \int_compare:nNnTF {\l_array_tmp_int} < {##1} {
        \int_set:Nn \l_array_tmp_int {##1}
      }{}
    }
      }
    \prop_put:Nnx #1 {top} {\int_use:N \l_array_tmp_int}
  }{}
  \prop_get:NnN #1 {base} \l_array_tmp_tl
  \int_compare:nNnTF {#2} = {\l_array_tmp_tl} {
    \prop_get:NnN #1 {top} \l_array_tmp_tl
    \int_set:Nn \l_array_tmp_int {\l_array_tmp_tl}
    \array_map_inline:Nn #1 {
    \tl_if_eq:NNTF {##2} {\q_no_value} {}
    {
      \int_compare:nNnTF {\l_array_tmp_int} > {##1} {
        \int_set:Nn \l_array_tmp_int {##1}
      }{}
    }
      }
    \prop_put:Nnx #1 {base} {\int_use:N \l_array_tmp_int}
  }{}
  }
}
```

\array_gdel:Nn

```
\cs_new_protected:Npn \array_gdel:Nn #1#2
{
  \exp_args:NNx \prop_gpop:Nn #1 {\int_eval:n {#2}}
  \int_set:Nn \l_array_tmp_int {0}
  \array_map_inline:Nn #1 {
    \tl_if_eq:NNTF {##2} {\q_no_value} {}
    {
      \int_incr:N \l_array_tmp_int
    }
  }
  \int_compare:nNnTF {\l_array_tmp_int} = {0}
  {
    \prop_gclear:N #1
  }
  {
  \prop_get:NnN #1 {top} \l_array_tmp_tl
  \int_compare:nNnTF {#2} = {\l_array_tmp_tl} {
    \prop_get:NnN #1 {base} \l_array_tmp_tl
    \int_set:Nn \l_array_tmp_int {\l_array_tmp_tl}
    \array_map_inline:Nn #1 {
    \tl_if_eq:NNTF {##2} {\q_no_value} {}
    {
      \int_compare:nNnTF {\l_array_tmp_int} < {##1} {
        \int_set:Nn \l_array_tmp_int {##1}
      }{}
    }
      }
    \prop_gput:Nnx #1 {top} {\int_use:N \l_array_tmp_int}
  }{}
  \prop_get:NnN #1 {base} \l_array_tmp_tl
  \int_compare:nNnTF {#2} = {\l_array_tmp_tl} {
    \prop_get:NnN #1 {top} \l_array_tmp_tl
    \int_set:Nn \l_array_tmp_int {\l_array_tmp_tl}
    \array_map_inline:Nn #1 {
    \tl_if_eq:NNTF {##2} {\q_no_value} {}
    {
      \int_compare:nNnTF {\l_array_tmp_int} > {##1} {
        \int_set:Nn \l_array_tmp_int {##1}
      }{}
    }
      }
    \prop_gput:Nnx #1 {base} {\int_use:N \l_array_tmp_int}
  }{}
  }
}
```

\array_length:N

```
\cs_new_protected:Npn \array_length:N #1
{
  \int_eval:n {\prop_get:Nn #1 {top} - \prop_get:Nn #1 {base}}
}
```

```
\ExplSyntaxOff
```