The randomwalk package: customizable random walks using TikZ[*]

Bruno Le Floch

July 10, 2012

Contents

Abstract

The randomwalk package draws random walks using TikZ. The following parameters can be customized:

- The number of steps, of course.
- The length of the steps, either a fixed length, or a length taken at random from a given set.
- The angle of each step, either taken at random from a given set, or uniformly distributed.

1 How to use it

The randomwalk package has exactly one user command: `\RandomWalk`, which takes a list of key-value pairs as its argument. A few examples:

```
\RandomWalk {number = 100, length = {4pt, 10pt}}
\RandomWalk {number = 100, angles = {0,60,120,180,240,300}, degree}
\RandomWalk {number = 100, length = 2em,
  angles = {0,10,20,-10,-20}, degree, angles-relative}
```

[*]This file has version number 0.2, last revised 2012-07-10.

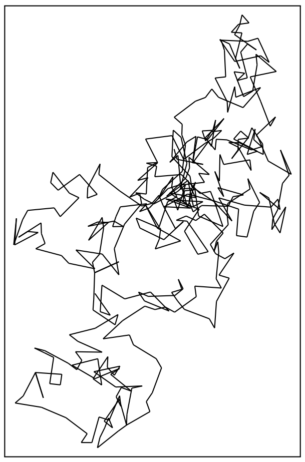

Figure 1: The result of `RandomWalk{number = 400, length = {4pt, 10pt}}`: a 400 steps long walk, where each step has one of two lengths.

The simplest is to give a list of all the keys, and their meaning:

- `number`: the number of steps (default 10)
- `length`: the length of each step: either one dimension (*e.g.*, `1em`), or a comma-separated list of dimensions (*e.g.*, `{2pt, 5pt}`), by default `10pt`. The length of each step is a random element in this set of possible dimensions.
- `angles`: the polar angle for each step: a comma-separated list of angles, and each step takes a random angle among the list. If this is not specified, then the angle is uniformly distributed along the circle.
- `degree` or `degrees`: specify that the angles are given in degrees.
- `angles-relative`: instead of being absolute, the angles are relative to the direction of the previous step.

2 randomwalk implementation

2.1 Packages

The whole expl3 bundle is loaded first.

<*package>

```
⟨@@=randomwalk⟩
```

```
\ProvidesExplPackage
  {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
\RequirePackage{expl3}
\RequirePackage{xparse}
```

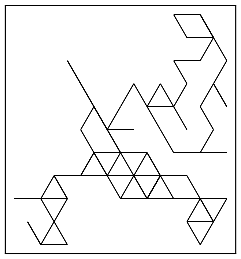

Figure 2: The result of `\RandomWalk{number = 100, angles = {0,60,120,180,240,300}, degrees}`: angles are constrained.

Figure 3: A last example: `\RandomWalk {number = 100, length = 2em, angles = {0,10,20,-10,-20}, degree, angles-relative}`

I use some LaTeX 2_ε packages: TikZ, for figures, and lcg for random numbers.

```
\RequirePackage{tikz}
```

lcg needs to know the smallest and biggest random numbers that it should produce, which we take to be 0 and `\c__randomwalk_lcg_last_int` $= 2^{31} - 2$. It will then store them in `\c@lcg@rand`: the `\c@` is there because of how LaTeX 2_ε defines counters. To make it clear that `\c` has a very special meaning here, I do not follow LaTeX3 naming conventions.

```
\int_const:Nn \c__randomwalk_lcg_last_int { \c_max_int - \c_one }
\RequirePackage
  [
    first= \c_zero ,
    last = \c__randomwalk_lcg_last_int ,
    counter = lcg@rand
  ]
  { lcg }
\rand % This \rand avoids some very odd bug.
```

2.2 Variables

`\l__randomwalk_step_number_int` The number of steps requested by the caller.

```
\int_new:N \l__randomwalk_step_number_int
```

(*End definition for* `\l__randomwalk_step_number_int` *This variable is documented on page* **??**.)

`\l__randomwalk_relative_angles_bool` Booleans for whether angles are relative (keyval option).

```
\bool_new:N \l__randomwalk_relative_angles_bool
```

(*End definition for* `\l__randomwalk_relative_angles_bool` *This variable is documented on page* **??**.)

\l__randomwalk_revert_random_bool

Booleans for whether to revert the random seed to its original value or keep the last value reached at the end of a random path.

```
\bool_new:N \l__randomwalk_revert_random_bool
```

(*End definition for* `\l__randomwalk_revert_random_bool` *This variable is documented on page* **??**.)

__randomwalk_rand_angle:
__randomwalk_rand_length:

Set the `\l__randomwalk_angle_fp` and `\l__randomwalk_length_fp` of the next step, most often randomly.

```
\cs_new_protected_nopar:Npn \__randomwalk_rand_angle: { }
\cs_new_protected_nopar:Npn \__randomwalk_rand_length: { }
```

(*End definition for* `\__randomwalk_rand_angle:` *and* `\__randomwalk_rand_length:` *These functions are documented on page* **??**.)

\l__randomwalk_angle_fp
\l__randomwalk_length_fp

Angle and length of the next step.

```
\fp_new:N \l__randomwalk_angle_fp
\fp_new:N \l__randomwalk_length_fp
```

(*End definition for* `\l__randomwalk_angle_fp` *and* `\l__randomwalk_length_fp` *These variables are documented on page* **??**.)

\l__randomwalk_old_x_fp
\l__randomwalk_old_y_fp
\l__randomwalk_new_x_fp
\l__randomwalk_new_y_fp

Coordinates of the two ends of each step: each `\draw` statement goes from the `_old` point to the `_new` point. See `\__randomwalk_step_draw:`.

```
\fp_new:N \l__randomwalk_old_x_fp
\fp_new:N \l__randomwalk_old_y_fp
\fp_new:N \l__randomwalk_new_x_fp
\fp_new:N \l__randomwalk_new_y_fp
```

(*End definition for* `\l__randomwalk_old_x_fp` *and* `\l__randomwalk_old_y_fp` *These functions are documented on page* **??**.)

\l__randomwalk_angles_seq
\l__randomwalk_lengths_seq

Sequences containing all allowed angles and lengths.

```
\seq_new:N \l__randomwalk_angles_seq
\seq_new:N \l__randomwalk_lengths_seq
```

(*End definition for* `\l__randomwalk_angles_seq` *and* `\l__randomwalk_lengths_seq` *These variables are documented on page* **??**.)

2.3 How the key-value list is treated

\RandomWalk

The only user command is `\RandomWalk`: it simply does the setup, and calls the internal macro `\__randomwalk_walk:`.

```
\DeclareDocumentCommand \RandomWalk { m }
  {
    \__randomwalk_set_defaults:
    \keys_set:nn { randomwalk } { #1 }
    \__randomwalk_walk:
  }
```

(*End definition for* `\RandomWalk` *This function is documented on page* **??**.)

`\__randomwalk_set_defaults:` Currently, the package treats the length of steps, and the angle, completely independently. The function `\__randomwalk_rand_length:` contains the action that decides the length of the next step, while the function `\__randomwalk_rand_angle:` pertains to the angle.

`\__randomwalk_set_defaults:` sets the default values before processing the user's key-value input.

```
\cs_new:Npn \__randomwalk_set_defaults:
  {
    \int_set:Nn \l__randomwalk_step_number_int {10}
    \cs_gset_protected_nopar:Npn \__randomwalk_rand_angle:
      { \__randomwalk_fp_set_rand:Nnn \l__randomwalk_angle_fp { - pi } { pi } }
    \cs_gset_protected_nopar:Npn \__randomwalk_rand_length:
      { \fp_set:Nn \l__randomwalk_length_fp {10} }
    \bool_set_false:N \l__randomwalk_revert_random_bool
    \bool_set_false:N \l__randomwalk_relative_angles_bool
  }
```

(*End definition for* `\__randomwalk_set_defaults:` *This function is documented on page* **??**.)

`\keys_define:nn` We introduce the keys for the package.

```
\keys_define:nn { randomwalk }
  {
    number .value_required: ,
    length .value_required: ,
    angles .value_required: ,
    number .int_set:N = \l__randomwalk_step_number_int ,
    length .code:n =
      {
        \seq_set_split:Nnn \l__randomwalk_lengths_seq { , } {#1}
        \seq_set_map:NNn \l__randomwalk_lengths_seq
          \l__randomwalk_lengths_seq { \dim_to_fp:n {##1} }
        \int_compare:nNnTF { \seq_length:N \l__randomwalk_lengths_seq } = {1}
          {
            \cs_gset_protected_nopar:Npn \__randomwalk_rand_length:
              { \fp_set:Nn \l__randomwalk_length_fp {#1} }
          }
          {
            \cs_gset_protected_nopar:Npn \__randomwalk_rand_length:
              {
                \__randomwalk_fp_set_rand_seq_item:NN
                  \l__randomwalk_length_fp \l__randomwalk_lengths_seq
              }
          }
      } ,
    angles .code:n  =
      {
        \seq_set_split:Nnn \l__randomwalk_angles_seq { , } {#1}
        \cs_gset_protected_nopar:Npn \__randomwalk_rand_angle:
          {
            \bool_if:NTF \l__randomwalk_relative_angles_bool
              { \__randomwalk_fp_add_rand_seq_item:NN }
            { \__randomwalk_fp_set_rand_seq_item:NN }
            \l__randomwalk_angle_fp \l__randomwalk_angles_seq
        }
      } ,
    degree .code:n  =
      { \__randomwalk_radians_from_degrees:N \l__randomwalk_angles_seq } ,
    degrees .code:n =
      { \__randomwalk_radians_from_degrees:N \l__randomwalk_angles_seq } ,
    angles-relative .code:n =
      { \bool_set_true:N \l__randomwalk_relative_angles_bool } ,
    revert-random .bool_set:N = \l__randomwalk_revert_random_bool ,
  }
```

(*End definition for* `\keys_define:nn` *This function is documented on page* **??**.)

`\__randomwalk_radians_from_degrees:N`

Helper macro to convert all items in `#1` to degrees.

```
\cs_new:Npn \__randomwalk_radians_from_degrees:N #1
  { \seq_set_map:NNn #1 #1 { \fp_eval:n { ##1 deg } } }
```

(*End definition for* `\__randomwalk_radians_from_degrees:N` *This function is documented on page* **??**.)

2.4 Drawing

`\__randomwalk_walk:`

We are ready to define `\__randomwalk_walk:`, which draws a TikZ picture of a random walk with the parameters set up by the `keys`. We reset all the coordinates to zero originally. Then we draw the relevant TikZ picture by repeatedly calling `\__randomwalk_step_draw:`.

```
\cs_new:Npn \__randomwalk_walk:
  {
    \begin{tikzpicture}
      \fp_zero:N \l__randomwalk_old_x_fp
      \fp_zero:N \l__randomwalk_old_y_fp
      \fp_zero:N \l__randomwalk_new_x_fp
      \fp_zero:N \l__randomwalk_new_y_fp
      \prg_replicate:nn { \l__randomwalk_step_number_int } { \__randomwalk_step_draw: }
      \bool_if:NF \l__randomwalk_revert_random_bool
        { \int_gset_eq:NN \cr@nd \cr@nd }
    \end{tikzpicture}
  }
```

`\cr@nd` is internal to the lcg package.
(*End definition for* `\__randomwalk_walk:` *This function is documented on page* **??**.)

`\__randomwalk_step_draw:`

`\__randomwalk_step_draw:` calls `\__randomwalk_rand_length:` and `\__randomwalk_rand_angle:` to determine the length and angle of the new step. This is then converted to cartesian coordinates and added to the previous end-point. Finally, we call TikZ's `\draw` to produce a line from the `_old` to the `_new` point.

```
\cs_new:Npn \__randomwalk_step_draw:
  {
    \__randomwalk_rand_length:
    \__randomwalk_rand_angle:
```

```
    \fp_set_eq:NN \l__randomwalk_old_x_fp \l__randomwalk_new_x_fp
    \fp_set_eq:NN \l__randomwalk_old_y_fp \l__randomwalk_new_y_fp
    \fp_add:Nn \l__randomwalk_new_x_fp { \l__randomwalk_length_fp * cos \l__randomwalk_angle_f
    \fp_add:Nn \l__randomwalk_new_y_fp { \l__randomwalk_length_fp * sin \l__randomwalk_angle_f
    \draw ( \fp_to_dim:N \l__randomwalk_old_x_fp, \fp_to_dim:N \l__randomwalk_old_y_fp )
       -- ( \fp_to_dim:N \l__randomwalk_new_x_fp, \fp_to_dim:N \l__randomwalk_new_y_fp );
  }
```

(*End definition for* `\__randomwalk_step_draw`*: This function is documented on page* **??**.)

2.5 On random numbers and items

For random numbers, the interface of lcg is not quite enough, so we provide our own LaTeX3-y functions. Also, this will allow us to change quite easily our source of random numbers.

__randomwalk_int_set_rand:Nnn

Sets the integer register #1 equal to a random integer between #2 and #3 inclusive.

```
\cs_new:Npn \__randomwalk_int_set_rand:Nnn #1#2#3
  {
    \rand
    \int_set:Nn #1 { #2 + \int_mod:nn {\c@lcg@rand} { #3 + 1 - (#2) } }
  }
```

(*End definition for* `\__randomwalk_int_set_rand:Nnn`)

__randomwalk_fp_set_rand:Nnn
__randomwalk_fp_add_rand:Nnn
__randomwalk_fp_set_rand_aux:NNnn

We also need floating point random numbers, both assigned and added to the variable #1 (well, #2 of the auxiliary).

```
\cs_new_nopar:Npn \__randomwalk_fp_set_rand:Nnn
  { \__randomwalk_fp_set_rand_aux:NNnn \fp_set:Nn }
\cs_new_nopar:Npn \__randomwalk_fp_add_rand:Nnn
  { \__randomwalk_fp_set_rand_aux:NNnn \fp_add:Nn }
\cs_new:Npn \__randomwalk_fp_set_rand_aux:NNnn #1#2#3#4
  {
    \rand
    #1 #2 { #3 + (#4 - (#3)) * \c@lcg@rand / \c__randomwalk_lcg_last_int }
  }
```

(*End definition for* `\__randomwalk_fp_set_rand:Nnn` *and* `\__randomwalk_fp_add_rand:Nnn` *These functions are documented on page* **??**.)

__randomwalk_fp_set_rand_seq_item:NN
__randomwalk_fp_add_rand_seq_item:NN
__randomwalk_fp_set_rand_item_aux:NNNNN

We can now pick an element at random from a sequence, and either assign it or add it to the fp variable #4. The same auxiliary could be used for picking random items from other types of lists.

```
\cs_new_protected_nopar:Npn \__randomwalk_fp_set_rand_seq_item:NN
  { \__randomwalk_fp_set_rand_item_aux:NNNNN \fp_set:Nn \seq_item:Nn \seq_length:N }
\cs_new_protected_nopar:Npn \__randomwalk_fp_add_rand_seq_item:NN
  { \__randomwalk_fp_set_rand_item_aux:NNNNN \fp_add:Nn \seq_item:Nn \seq_length:N }
\cs_new_protected:Npn \__randomwalk_fp_set_rand_item_aux:NNNNN #1#2#3#4#5
  {
    \rand
    #1 #4 { #2 #5 { 1 + \int_mod:nn { \c@lcg@rand } { #3 #5 } } }
  }
```

(*End definition for* `\__randomwalk_fp_set_rand_seq_item:NN` *and* `\__randomwalk_fp_add_rand_seq_item:NN` *These functions are documented on page* **??**.)

```
⟨/package⟩
```