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

Bruno Le Floch[†]

Released on 2015/03/03

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.

[*]This file describes version v0.3, last revised 2015/03/03.
[†]E-mail blflatex@gmail.com

1 How to use randomwalk

`\RandomWalk` 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 = 200, length = {4pt, 10pt}}
\RandomWalk {number = 100, angles = {0,60,120,180,240,300}, degree}
\RandomWalk {number = 100, length = 1ex,
  angles = {0,24,48,-24,-48}, degree, angles-relative}
```

Here is 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.*, `1ex`), or a comma-separated list of dimensions (*e.g.*, `{2pt, 5pt}`), by default `10pt`. The length of each step is a (uniformly distributed) 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 in 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 (by default, they are in radians).
- `angles-relative`: instead of being absolute, the angles are relative to the direction of the previous step.
- `revert-random` (boolean, false by default): revert the seed of the random number generator to its original value after the random walk.

2 randomwalk implementation

2.1 Packages

The expl3 bundle is loaded first.
<*package>

```
⟨@@=randomwalk⟩
```

```
\RequirePackage{expl3}[2012/08/14]
\ProvidesExplPackage
  {randomwalk.sty} {2015/03/03} {0.3} {Customizable random walks using TikZ}
\RequirePackage{xparse}[2012/08/14]
```

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

```
\RequirePackage{tikz}
```

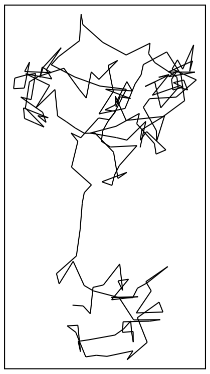

Figure 1: The result of `RandomWalk {number = 200, length = {4pt, 10pt}}`: a 200 steps long walk, where each step has one of two lengths.

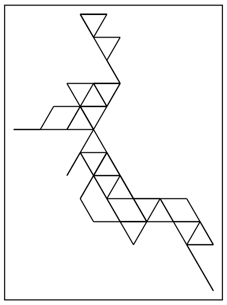

Figure 2: The result of `\RandomWalk {number = 100, angles = {0, 60, 120, 180, 240, 300}, degrees}`: angles are constrained.

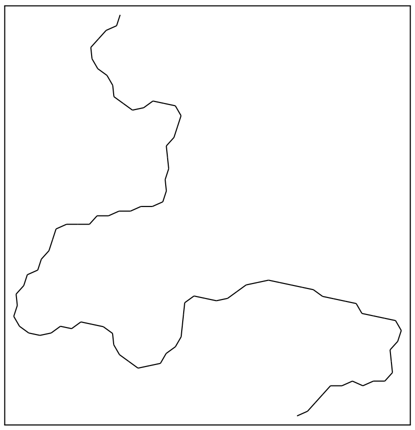

Figure 3: A last example: `\RandomWalk {number = 100, length = 1ex, angles = {0, 24, 48, -24, -48}, degree, angles-relative}`

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 2ε defines counters. To make it clear that `\c` has a very special meaning here, I do not follow LaTeX3 naming conventions. Also of note is that I use `\cr@nd` in `\__randomwalk_walk:`.

It seems that the lcg package has to be loaded after the document class, hence we do it `\AtBeginDocument`.

```
\int_const:Nn \c__randomwalk_lcg_last_int { \c_max_int - \c_one }
\AtBeginDocument
  {
    \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_internal_tl`
`\l__randomwalk_internal_int`

Used for scratch assignments.

```
\tl_new:N \l__randomwalk_internal_tl
\int_new:N \l__randomwalk_internal_int
```

(*End definition for* `\l__randomwalk_internal_tl` *and* `\l__randomwalk_internal_int`.)

\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`*.*)

\l__randomwalk_relative_angles_bool
\l__randomwalk_degrees_bool

Booleans for whether angles are relative (keyval option), and whether they are in degrees.

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

(*End definition for* `\l__randomwalk_relative_angles_bool` *and* `\l__randomwalk_degrees_bool`*.*)

\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`*.*)

__randomwalk_next_angle:
__randomwalk_next_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_next_angle: { }
\cs_new_protected_nopar:Npn \__randomwalk_next_length: { }
```

(*End definition for* `\__randomwalk_next_angle:` *and* `\__randomwalk_next_length:`*.*)

\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`*.*)

\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`*.*)

\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`*.*)

2.3 How the key-value list is treated

\RandomWalk
\randomwalk:n

The user command `\RandomWalk`, based on the code-level command `\randomwalk:n`, which simply does the setup and calls the internal macro `\__randomwalk_walk:`.

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

(*End definition for* `\RandomWalk` *and* `\randomwalk:n`. *These functions are documented on page 2.*)

__randomwalk_set_defaults:

Currently, the package treats the length of steps, and the angle, completely independently. The function `\__randomwalk_next_length:` contains the action that decides the length of the next step, while the function `\__randomwalk_next_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_next_angle:
      { \__randomwalk_fp_set_rand:Nnn \l__randomwalk_angle_fp { 0 } { 360 } }
    \cs_gset_protected_nopar:Npn \__randomwalk_next_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:`.)

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} }
        \cs_gset_protected_nopar:Npn \__randomwalk_next_length:
          {
            \__randomwalk_get_rand_seq_item:NN
              \l__randomwalk_lengths_seq \l__randomwalk_internal_tl
            \fp_set:Nn \l__randomwalk_length_fp { \l__randomwalk_internal_tl }
          }
      } ,
    angles .code:n  =
      {
        \seq_set_split:Nnn \l__randomwalk_angles_seq { , } {#1}
        \seq_set_map:NNn \l__randomwalk_angles_seq
          \l__randomwalk_angles_seq { \fp_to_tl:n {##1} }
        \cs_gset_protected_nopar:Npn \__randomwalk_next_angle:
          {
            \__randomwalk_get_rand_seq_item:NN
              \l__randomwalk_angles_seq \l__randomwalk_internal_tl
            \bool_if:NF \l__randomwalk_degrees_bool
              { \tl_put_right:Nn \l__randomwalk_internal_tl { rad } }
            \bool_if:NTF \l__randomwalk_relative_angles_bool
              { \fp_add:Nn } { \fp_set:Nn }
              \l__randomwalk_angle_fp { \l__randomwalk_internal_tl }
          }
      } ,
    degree .bool_set:N = \l__randomwalk_degrees_bool ,
    degrees .bool_set:N = \l__randomwalk_degrees_bool ,
    angles-relative .bool_set:N = \l__randomwalk_relative_angles_bool ,
    revert-random .bool_set:N = \l__randomwalk_revert_random_bool ,
  }
```

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 draw the relevant TikZ picture by repeatedly calling `\__randomwalk_step_draw:`.

```
\cs_new:Npn \__randomwalk_walk:
  {
    \begin{tikzpicture}
      \fp_zero:N \l__randomwalk_angle_fp
      \fp_zero:N \l__randomwalk_length_fp
      \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:`.)

__randomwalk_step_draw:

`\__randomwalk_step_draw:` calls `\__randomwalk_next_length:` and `\__randomwalk_next_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_next_length:
    \__randomwalk_next_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 * cosd \l__randomwalk_angle_fp }
    \fp_add:Nn \l__randomwalk_new_y_fp
      { \l__randomwalk_length_fp * sind \l__randomwalk_angle_fp }
    \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:`.)

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_fp_set_rand:Nnn

We also need floating point random numbers, assigned to the variable `#1`.

```
\cs_new_protected:Npn \__randomwalk_fp_set_rand:Nnn #1#2#3
  {
    \rand
    \fp_set:Nn #1
      { #2 + (#3 - (#2)) * \c@lcg@rand / \c__randomwalk_lcg_last_int }
  }
```

(*End definition for* `\__randomwalk_fp_set_rand:Nnn`.)

__randomwalk_get_rand_seq_item:NN

We can now pick an element at random from a sequence. If the sequence has a single element, no need for randomness.

```
\cs_new_protected:Npn \__randomwalk_get_rand_seq_item:NN #1#2
  {
    \int_set:Nn \l__randomwalk_internal_int { \seq_count:N #1 }
    \int_compare:nTF { \l__randomwalk_internal_int = 1 }
      { \tl_set:Nx #2 { \seq_item:Nn #1 { 1 } } }
      {
        \rand
        \tl_set:Nx #2
          {
            \seq_item:Nn #1
              {
                1 +
                \int_mod:nn { \c@lcg@rand } { \l__randomwalk_internal_int }
              }
          }
      }
  }
```

(*End definition for* `\__randomwalk_get_rand_seq_item:NN`*.*)

```
⟨/package⟩
```