

The `randomwalk` package: customizable random walks using TikZ*

Bruno Le Floch

August 24, 2012

Contents

1	How to use it	1
2	randomwalk implementation	3
2.1	Packages	3
2.2	Variables	4
2.3	How the key-value list is treated	5
2.4	Drawing	6
2.5	On random numbers and items	7

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 0.2b, last revised 2012-08-24.



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.

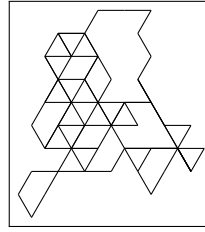


Figure 2: The result of `\RandomWalk{number = 100, angles = {0,60,120,180,240,300}, degrees}`: angles are constrained.



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

2 randomwalk implementation

2.1 Packages

The whole `expl3` bundle is loaded first.

```
<*package>
1 <@@=randomwalk>
2 \RequirePackage {expl3} [2012/08/14]
3 \ProvidesExplPackage
4   {randomwalk.sty} {2012/08/24} {0.2b} {Customizable random walks using TikZ}
5 \RequirePackage {xparse} [2012/08/14]
```

I use some $\text{\LaTeX} 2_{\epsilon}$ packages: `TikZ`, for figures, and `lcg` for random numbers.

```
6 \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 $\text{\LaTeX} 2_{\epsilon}$ defines counters. To make it clear that `\c` has a very special meaning here, I do not follow $\text{\LaTeX} 3$ naming conventions.

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

```
7 \int_const:Nn \c__randomwalk_lcg_last_int { \c_max_int - \c_one }
8 \AtBeginDocument
9   {
10     \RequirePackage
11       [
12         first= \c_zero ,
13         last = \c__randomwalk_lcg_last_int ,
14         counter = lcg@rand
```

```

15     ]
16     { lcg }
17     \rand % This \rand avoids some very odd bug.
18 }

```

2.2 Variables

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

```

19 \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).

```

20 \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.

```

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

```

22 \cs_new_protected_nopar:Npn \__randomwalk_rand_angle: { }
23 \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` Angle and length of the next step.
`\l__randomwalk_length_fp`

```

24 \fp_new:N \l__randomwalk_angle_fp
25 \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` Coordinates of the two ends of each step: each `\draw` statement goes from the `_old` point
`\l__randomwalk_old_y_fp` to the `_new` point. See `__randomwalk_step_draw:.`
`\l__randomwalk_new_x_fp`
`\l__randomwalk_new_y_fp`

```

26 \fp_new:N \l__randomwalk_old_x_fp
27 \fp_new:N \l__randomwalk_old_y_fp
28 \fp_new:N \l__randomwalk_new_x_fp
29 \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` Sequences containing all allowed angles and lengths.
`\l__randomwalk_lengths_seq`

```

30 \seq_new:N \l__randomwalk_angles_seq
31 \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:`.

```

32 \DeclareDocumentCommand \RandomWalk { m }
33 {
34   \__randomwalk_set_defaults:
35   \keys_set:nn { randomwalk } { #1 }
36   \__randomwalk_walk:
37 }

```

(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.

```

38 \cs_new:Npn \__randomwalk_set_defaults:
39 {
40   \int_set:Nn \l__randomwalk_step_number_int {10}
41   \cs_gset_protected_nopar:Npn \__randomwalk_rand_angle:
42     { \__randomwalk_fp_set_rand:Nnn \l__randomwalk_angle_fp { - pi } { pi } }
43   \cs_gset_protected_nopar:Npn \__randomwalk_rand_length:
44     { \fp_set:Nn \l__randomwalk_length_fp {10} }
45   \bool_set_false:N \l__randomwalk_revert_random_bool
46   \bool_set_false:N \l__randomwalk_relative_angles_bool
47 }

```

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

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

```

48 \keys_define:nn { randomwalk }
49 {
50   number .value_required: ,
51   length .value_required: ,
52   angles .value_required: ,
53   number .int_set:N = \l__randomwalk_step_number_int ,
54   length .code:n =
55     {
56       \seq_set_split:Nnn \l__randomwalk_lengths_seq { , } {#1}
57       \seq_set_map:Nnn \l__randomwalk_lengths_seq
58         \l__randomwalk_lengths_seq { \dim_to_fp:n {##1} }
59       \int_compare:nNnTF { \seq_length:N \l__randomwalk_lengths_seq } = {1}
60       {
61         \cs_gset_protected_nopar:Npn \__randomwalk_rand_length:
62           { \fp_set:Nn \l__randomwalk_length_fp {#1} }
63       }
64       {
65         \cs_gset_protected_nopar:Npn \__randomwalk_rand_length:
66           {

```

```

67         \l__randomwalk_fp_set_rand_seq_item:NN
68         \l__randomwalk_length_fp \l__randomwalk_lengths_seq
69     }
70 }
71 },
72 angles .code:n =
73 {
74     \seq_set_split:Nnn \l__randomwalk_angles_seq { , } {#1}
75     \cs_gset_protected_nopar:Npn \l__randomwalk_rand_angle:
76     {
77         \bool_if:NTF \l__randomwalk_relative_angles_bool
78         { \l__randomwalk_fp_add_rand_seq_item:NN }
79         { \l__randomwalk_fp_set_rand_seq_item:NN }
80         \l__randomwalk_angle_fp \l__randomwalk_angles_seq
81     }
82 },
83 degree .code:n =
84 { \l__randomwalk_radians_from_degrees:N \l__randomwalk_angles_seq } ,
85 degrees .code:n =
86 { \l__randomwalk_radians_from_degrees:N \l__randomwalk_angles_seq } ,
87 angles-relative .code:n =
88 { \bool_set_true:N \l__randomwalk_relative_angles_bool } ,
89 revert-random .bool_set:N = \l__randomwalk_revert_random_bool ,
90 }

```

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

\l__randomwalk_radians_from_degrees:N Helper macro to convert all items in #1 to degrees.

```

91 \cs_new:Npn \l__randomwalk_radians_from_degrees:N #1
92 { \seq_set_map:Nnn #1 #1 { \fp_eval:n { ##1 deg } } }

```

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

2.4 Drawing

\l__randomwalk_walk: We are ready to define \l__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 \l__randomwalk_step_draw:.

```

93 \cs_new:Npn \l__randomwalk_walk:
94 {
95     \begin{tikzpicture}
96         \fp_zero:N \l__randomwalk_old_x_fp
97         \fp_zero:N \l__randomwalk_old_y_fp
98         \fp_zero:N \l__randomwalk_new_x_fp
99         \fp_zero:N \l__randomwalk_new_y_fp
100         \prg_replicate:nn { \l__randomwalk_step_number_int } { \l__randomwalk_step_draw: }
101         \bool_if:NF \l__randomwalk_revert_random_bool
102         { \int_gset_eq:NN \cr@end \cr@end }
103     \end{tikzpicture}
104 }

```

`\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.

```

105 \cs_new:Npn \__randomwalk_step_draw:
106 {
107   \__randomwalk_rand_length:
108   \__randomwalk_rand_angle:
109   \fp_set_eq:NN \l__randomwalk_old_x_fp \l__randomwalk_new_x_fp
110   \fp_set_eq:NN \l__randomwalk_old_y_fp \l__randomwalk_new_y_fp
111   \fp_add:Nn \l__randomwalk_new_x_fp { \l__randomwalk_length_fp * cos \l__randomwalk_angle_fp }
112   \fp_add:Nn \l__randomwalk_new_y_fp { \l__randomwalk_length_fp * sin \l__randomwalk_angle_fp }
113   \draw ( \fp_to_dim:N \l__randomwalk_old_x_fp, \fp_to_dim:N \l__randomwalk_old_y_fp )
114         -- ( \fp_to_dim:N \l__randomwalk_new_x_fp, \fp_to_dim:N \l__randomwalk_new_y_fp );
115 }
```

(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 \LaTeX -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.

```

116 \cs_new:Npn \__randomwalk_int_set_rand:Nnn #1#2#3
117 {
118   \rand
119   \int_set:Nn #1 { #2 + \int_mod:nn {\c@lcg@rand} { #3 + 1 - (#2) } }
120 }
```

(End definition for `__randomwalk_int_set_rand:Nnn`)

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

```

121 \cs_new_nopar:Npn \__randomwalk_fp_set_rand:Nnn
122 { \__randomwalk_fp_set_rand_aux:NNnn \fp_set:Nn }
123 \cs_new_nopar:Npn \__randomwalk_fp_add_rand:Nnn
124 { \__randomwalk_fp_set_rand_aux:NNnn \fp_add:Nn }
125 \cs_new:Npn \__randomwalk_fp_set_rand_aux:NNnn #1#2#3#4
126 {
127   \rand
128   #1 #2 { #3 + (#4 - (#3)) * \c@lcg@rand / \c__randomwalk_lcg_last_int }
129 }
```

(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.

```

130 \cs_new_protected_nopar:Npn \__randomwalk_fp_set_rand_seq_item:NN
131   { \__randomwalk_fp_set_rand_item_aux:NNNNN \fp_set:Nn \seq_item:Nn \seq_length:N }
132 \cs_new_protected_nopar:Npn \__randomwalk_fp_add_rand_seq_item:NN
133   { \__randomwalk_fp_set_rand_item_aux:NNNNN \fp_add:Nn \seq_item:Nn \seq_length:N }
134 \cs_new_protected:Npn \__randomwalk_fp_set_rand_item_aux:NNNNN #1#2#3#4#5
135   {
136     \rand
137     #1 #4 { #2 #5 { 1 + \int_mod:nn { \c@lcg@rand } { #3 #5 } } }
138   }

```

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

```

139 </package>

```