

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

Bruno Le Floch

2011-09-09

Contents

1	How to use it	1
2	randomwalk implementation	2
2.1	Packages	2
2.2	How the key-value list is treated	4
2.3	Drawing	5
2.4	On random numbers etc.	7
2.5	Other comma list operations	8
2.6	Variables	9

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 2011-09-09.

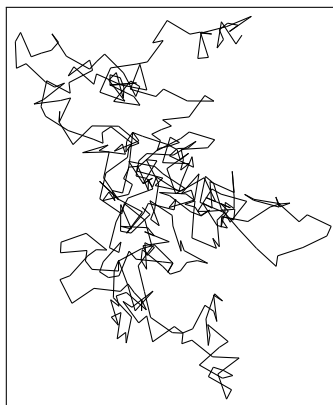


Figure 1: The result of `RandomWalk{number = 400, length = {4pt, 10pt}}`: a 400 steps long walk, where each step has one of two lengths.

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>
1 \ProvidesExplPackage
2   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
3 \RequirePackage{expl3}
4 \RequirePackage{xparse}
   I use some LATEX 2ε packages: TikZ, for figures, and lcg for random numbers.
5 \RequirePackage{tikz}
```

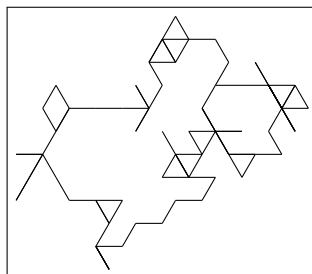


Figure 2: The result of `\RandomWalk{number = 100, angles = {0,60,120,180,240,300}, degrees}`: angles are constrained.

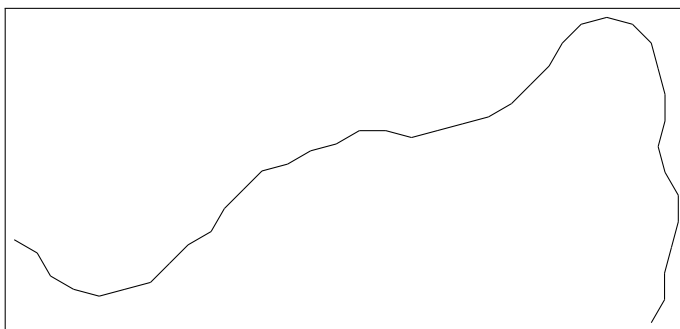


Figure 3: A last example: `\RandomWalk {number = 100, length = 2em, angles = {0,10,20,-10,-20}, degree, angles-relative}`

lcg needs to know the smallest and biggest random numbers that it should produce, `\c_rw_lcg_first` and `_last`. It will then store them in `\c@lcg@rand`: the `\c@` is there because of how L^AT_EX 2_ε defines counters. To make it clear that `\c` has a very special meaning here, I do not follow L^AT_EX 3 naming conventions.

The lcg package would support a range of $2^{31} - 1$, but l3fp constrains us to 9 digit numbers, so we take the closest available power of 2, namely $536870911 = 2^{29} - 1$.

```

6 \int_const:Nn \c_rw_lcg_first_int {0}
7 \int_const:Nn \c_rw_lcg_last_int  {536870911}
8 \int_const:Nn \c_rw_lcg_range_int
9   { \c_rw_lcg_last_int - \c_rw_lcg_first_int }
10 \RequirePackage
11   [
12     first= \c_rw_lcg_first_int,
13     last = \c_rw_lcg_last_int,
14     counter = lcg@rand
15   ]
16   { lcg }
17 \rand % This \rand avoids some very odd bug.
```

We need this constant for fast conversion from degrees to radians later.

```

18 \fp_const:Nn \c_rw_one_degree_fp {+1.74532925e-2}
```

2.2 How the key-value list is treated

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

```

19 \DeclareDocumentCommand \RandomWalk { m }
20   {
21     \rw_set_defaults:
22     \keys_set:nn { randomwalk } { #1 }
23     \rw_walk:
24   }
```

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

`\g_rw_Ado_tl` Currently, the package treats the length of steps, and the angle, completely independently.
`\g_rw_Ldo_tl` The token list `\g_rw_Ldo_tl` contains the action that should be done to decide the length
`\rw_set_defaults:` of the next step, while the token list `\g_rw_Ado_tl` pertains to the angle.

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

```

25 \tl_new:N \g_rw_Ado_tl
26 \tl_new:N \g_rw_Ldo_tl
27 \bool_new:N \l_rw_A_relative_bool
28 \bool_new:N \l_rw_revert_random_bool
29 \cs_new:Npn \rw_set_defaults:
30   {
31     \fp_set:Nn \l_rw_step_length_fp {10}
32     \int_set:Nn \l_rw_step_number_int {10}
33     \tl_gset:Nn \g_rw_Ado_tl { \rw_Ainterval:nn {-\c_pi_fp} {\c_pi_fp} }
```

```

34 \tl_gset:Nn \g_rw_Ldo_tl { \rw_Lfixed:n \l_rw_step_length_fp } %^^A bug?
35 \bool_set_false:N \l_rw_revert_random_bool
36 \bool_set_false:N \l_rw_A_relative_bool
37 }

```

(End definition for \g_rw_Ado_tl This function is documented on page ??.)

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

```

38 \keys_define:nn { randomwalk }
39 {
40   number .value_required:,
41   length .value_required:,
42   angles .value_required:,
43   number .code:n = { \int_set:Nn \l_rw_step_number_int {#1}},
44   length .code:n =
45   {
46     \clist_set:Nn \l_rw_lengths_clist {#1}
47     \rw_clist_fp_from_dim:N \l_rw_lengths_clist
48     \int_compare:nNnTF { \clist_length:N \l_rw_lengths_clist } = {1}
49     { \tl_gset:Nn \g_rw_Ldo_tl { \rw_Lfixed:n \l_rw_lengths_clist } }
50     { \tl_gset:Nn \g_rw_Ldo_tl { \rw_Llist:N \l_rw_lengths_clist } }
51   },
52   angles .code:n =
53   {
54     \clist_set:Nn \l_rw_angles_clist {#1}
55     \tl_gset:Nn \g_rw_Ado_tl { \rw_Alist:N \l_rw_angles_clist }
56   },
57   degree .code:n = { \rw_radians_from_degrees:N \l_rw_angles_clist },
58   degrees .code:n = { \rw_radians_from_degrees:N \l_rw_angles_clist },
59   angles-relative .code:n = { \bool_set_true:N \l_rw_A_relative_bool },
60   revert-random .bool_set:N = \l_rw_revert_random_bool,
61 }

```

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

2.3 Drawing

\rw_walk: We are ready to define \rw_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 \rw_step_draw:.

```

62 \cs_new:Npn \rw_walk:
63 {
64   \fp_zero:N \l_rw_old_x_fp
65   \fp_zero:N \l_rw_old_y_fp
66   \fp_zero:N \l_rw_new_x_fp
67   \fp_zero:N \l_rw_new_y_fp
68   \begin{tikzpicture}
69     \prg_replicate:nn { \l_rw_step_number_int } { \rw_step_draw: }
70     \bool_if:NF \l_rw_revert_random_bool
71     { \int_gset_eq:NN \cr@nd \cr@nd }
72   \end{tikzpicture}

```

```
73 }
```

`\cr@nd` is internal to the `lcg` package.

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

`\rw_step_draw`: `\rw_step_draw`: passes its second argument *with one level of braces removed* to its first argument, responsible for making a random step. Then, `\rw_step_draw`: draws the random step.

```
74 \cs_new:Npn \rw_step_draw:
75 {
76   \g_rw_Ldo_tl
77   \g_rw_Ado_tl
78   \rw_cartesian_from_polar:NNNN
79   \l_rw_step_x_fp \l_rw_step_y_fp
80   \l_rw_radius_fp \l_rw_angle_fp
81   \fp_add:Nn \l_rw_new_x_fp { \l_rw_step_x_fp }
82   \fp_add:Nn \l_rw_new_y_fp { \l_rw_step_y_fp }
83   \draw ( \fp_to_dim:N \l_rw_old_x_fp, \fp_to_dim:N \l_rw_old_y_fp )
84   -- ( \fp_to_dim:N \l_rw_new_x_fp, \fp_to_dim:N \l_rw_new_y_fp );
85   \fp_set_eq:NN \l_rw_old_x_fp \l_rw_new_x_fp
86   \fp_set_eq:NN \l_rw_old_y_fp \l_rw_new_y_fp
87 }
```

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

The next couple of macros store a random floating point in `\l_rw_length_fp` or `\l_rw_angle_fp`.

`\rw_L...`: First for the length of steps.

```
88 \cs_new:Npn \rw_Lfixed:n #1
89 { \fp_set:Nn \l_rw_radius_fp {#1} }
90 \cs_new:Npn \rw_Llist:N #1
91 { \rw_set_to_random_clist_element:NN \l_rw_radius_fp #1 }
92 \cs_new:Npn \rw_Linterval:nn #1#2
93 { \rw_set_to_random_fp:Nnn \l_rw_radius_fp {#1} {#2} }
```

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

`\rw_A...`: Then for angles.

```
94 \cs_new:Npn \rw_Ainterval:nn #1#2
95 {
96   \bool_if:NTF \l_rw_A_relative_bool
97   { \rw_add_to_random_fp:Nnn }
98   { \rw_set_to_random_fp:Nnn }
99   \l_rw_angle_fp {#1} {#2}
100 }
101 \cs_new:Npn \rw_Alist:N #1
102 {
103   \bool_if:NTF \l_rw_A_relative_bool
104   { \rw_add_to_random_clist_element:NN }
105   { \rw_set_to_random_clist_element:NN }
106   \l_rw_angle_fp #1
107 }
```

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

`\rw_cartesian_from_polar:NNNN` The four arguments of `\rw_cartesian_from_polar:NNNN` are (x, y, r, θ) : it sets (x, y) equal to the cartesian coordinates corresponding to a radius r and an angle θ . We also give a version with global assignments.

```

108 \cs_new_protected:Npn \rw_cartesian_from_polar:NNNN #1#2#3#4
109   {
110     \fp_cos:Nn #1 {\fp_use:N #4}
111     \fp_sin:Nn #2 {\fp_use:N #4}
112     \fp_mul:Nn #1 {\fp_use:N #3}
113     \fp_mul:Nn #2 {\fp_use:N #3}
114   }
115 \cs_new_protected:Npn \rw_gcartesian_from_polar:NNNN #1#2#3#4
116   {
117     \fp_gcos:Nn #1 {\fp_use:N #4}
118     \fp_gsin:Nn #2 {\fp_use:N #4}
119     \fp_gmul:Nn #1 {\fp_use:N #3}
120     \fp_gmul:Nn #2 {\fp_use:N #3}
121   }

```

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

We cannot yet do the conversion in the other direction: `l3fp.dtx` does not yet provide inverse trigonometric functions. But in fact, we do not need this conversion, so let's stop worrying.

2.4 On random numbers etc.

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.

```

122 \cs_new:Npn \rw_set_to_random_int:Nnn #1#2#3
123   {
124     \rand
125     \int_set:Nn #1 { #2 + \int_mod:nn {\c@lcg@rand} { #3 + 1 - (#2) } }
126   }

```

We also need floating point random numbers.

```

127 \cs_new:Npn \rw_set_to_random_fp:Nnn #1#2#3
128   {
129     \fp_set:Nn \l_rw_tmpa_fp {#3}
130     \fp_sub:Nn \l_rw_tmpa_fp {#2}
131     \rand
132     \fp_set:Nn \l_rw_tmpb_fp { \int_use:N \c@lcg@rand }
133     \fp_div:Nn \l_rw_tmpb_fp { \int_use:N \c_rw_lcg_range_int }
134     \fp_mul:Nn \l_rw_tmpa_fp { \l_rw_tmpb_fp }
135     \fp_add:Nn \l_rw_tmpa_fp {#2}
136     \fp_set:Nn #1 { \l_rw_tmpa_fp }
137   }
138 \cs_new:Npn \rw_add_to_random_fp:Nnn #1#2#3

```

```

139 {
140   \fp_set:Nn \l_rw_tmpa_fp {#3}
141   \fp_sub:Nn \l_rw_tmpa_fp {#2}
142   \rand
143   \fp_set:Nn \l_rw_tmpb_fp { \int_use:N \c@lcg@rand }
144   \fp_div:Nn \l_rw_tmpb_fp { \int_use:N \c_rw_lcg_range_int }
145   \fp_mul:Nn \l_rw_tmpa_fp { \l_rw_tmpb_fp }
146   \fp_add:Nn \l_rw_tmpa_fp {#2}
147   \fp_add:Nn #1 { \l_rw_tmpa_fp } %here: mod?
148 }

```

We can now pick an element at random from a comma-separated list

```

149 \cs_new:Npn \rw_set_to_random_clist_element:NN #1#2
150 {
151   \rw_set_to_random_int:Nnn \l_rw_tmpb_int {1} { \clist_length:N #2 }
152   \fp_set:Nn #1 { \clist_item:Nn #2 { \l_rw_tmpb_int } }
153 }
154 \cs_new:Npn \rw_add_to_random_clist_element:NN #1#2
155 {
156   \rw_set_to_random_int:Nnn \l_rw_tmpb_int {1} { \clist_length:N #2 }
157   \fp_add:Nn #1 { \clist_item:Nn #2 { \l_rw_tmpb_int } }
158 }

```

2.5 Other comma list operations

More stuff on clists.

```

159 \cs_new:Npn \rw_radians_from_degrees:N #1
160 {
161   \clist_clear:N \l_rw_tmpa_clist
162   \clist_map_inline:Nn #1
163   {
164     \fp_set:Nn \l_rw_tmpa_fp {##1}
165     \fp_mul:Nn \l_rw_tmpa_fp { \c_rw_one_degree_fp }
166     \clist_push:NV \l_rw_tmpa_clist \l_rw_tmpa_fp
167   }
168   \clist_set_eq:NN #1 \l_rw_tmpa_clist
169 }
170 \cs_new:Npn \rw_clist_fp_from_dim:N #1
171 {
172   \clist_clear:N \l_rw_tmpa_clist
173   \clist_map_inline:Nn #1
174   {
175     \fp_set_from_dim:Nn \l_rw_tmpa_fp {##1}
176     \clist_push:NV \l_rw_tmpa_clist \l_rw_tmpa_fp
177   }
178   \clist_set_eq:NN #1 \l_rw_tmpa_clist
179 }

```


2.6 Variables

We need a bunch of floating point numbers: each step line goes from the `_old` point to the `_new` point. The coordinates `_add` are those of the vector from one to the next, so that `_new = _old + _add`.

```
180 \fp_new:N \l_rw_old_x_fp
181 \fp_new:N \l_rw_old_y_fp
182 \fp_new:N \l_rw_step_x_fp
183 \fp_new:N \l_rw_step_y_fp
184 \fp_new:N \l_rw_new_x_fp
185 \fp_new:N \l_rw_new_y_fp
186 \fp_new:N \l_rw_angle_fp
187 \fp_new:N \l_rw_step_length_fp
188 \fp_new:N \l_rw_radius_fp
189 \int_new:N \l_rw_step_number_int
190 \clist_new:N \l_rw_angles_clist
191 \clist_new:N \l_rw_lengths_clist
192 \fp_new:N \l_rw_tmpa_fp
193 \fp_new:N \l_rw_tmpb_fp
194 \clist_new:N \l_rw_tmpa_clist
195 \int_new:N \l_rw_tmpb_int

</package>
```