The randomwalk package: customizable random walks using TikZ*

Bruno Le Floch

2011/01/09

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:

*This file has version number 1, last revised 2011/01/09.

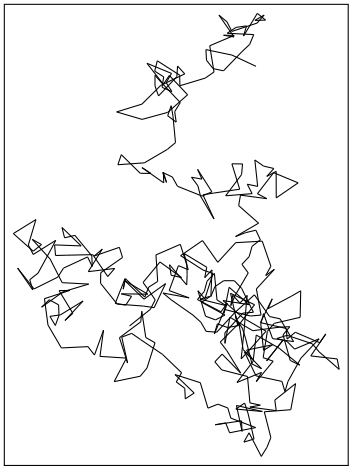

Figure 1: The result of `RandomWalk{number = 400, length = {4pt, 10pt}}`: a 400 steps long walk, where each step has one of two lengths.

```
\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}
```

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`(`s`): specifies that the angles are given in degrees.
- `angles-relative`: instead of being absolute, the angles are relative to the direction of the previous step.

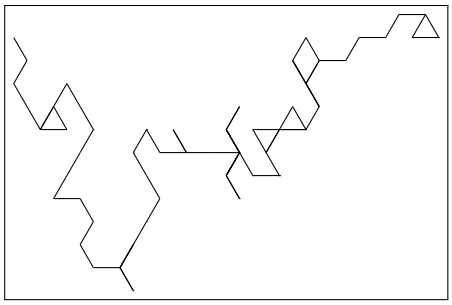

Figure 2: The result of `\RandomWalk{number = 100, angles = {0,60,120,180,240,300}, degrees}`: angles are constrained.

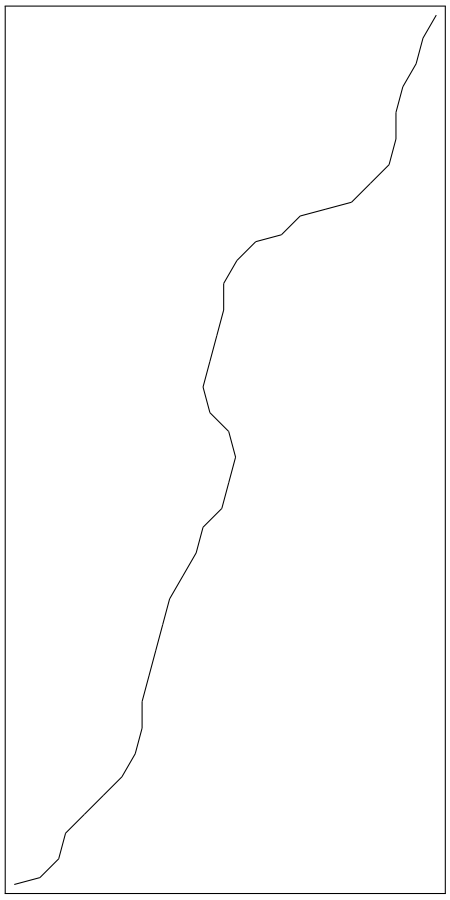

Figure 3: A last example: `\RandomWalk {number = 100, length = 2em, angles = {0,10,20,-10,-20}, degree, angles-relative}`

2 randomwalk implementation

2.1 Packages

The whole `expl3` bundle is loaded first, including Joseph Wright's very useful package `l3fp.sty` for floating point calculations.

<*package>

```
\ProvidesExplPackage
  {\filename}{\filedate}{\fileversion}{\filedescription}
\RequirePackage{expl3}
\RequirePackage{xparse}
```

I use some LaTeX2e 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, `\c_rw_lcg_first` and `_last`. 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.

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 $536870911 = 2^{29} - 1$.

```
\int_const:Nn \c_rw_lcg_first_int {0}
\int_const:Nn \c_rw_lcg_last_int  {536870911}
\int_const:Nn \c_rw_lcg_range_int {
  \c_rw_lcg_last_int - \c_rw_lcg_first_int
}
\RequirePackage [
  first= \c_rw_lcg_first_int,
  last = \c_rw_lcg_last_int,
  counter = lcg@rand ]
  { lcg }
\rand % This \rand avoids some very odd bug.
```

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

```
\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:`.

```
\DeclareDocumentCommand \RandomWalk { m } {
  \rw_set_defaults:
```

```
  \keys_set:nn { randomwalk } { #1 }
  \rw_walk:
}
```

\rw_Atype
\rw_Ltype
\l_rw_Aargs_tl
\l_rw_Largs_tl
\rw_set_defaults:

Currently, the package treats the length of steps, and the angle, completely independently. Later, we build a control sequence from some constant text and the content of the token list `\rw_Atype:`, and apply it to `\l_rw_Aargs_tl`. Same for `\rw_Ltype:`, applied to `\l_rw_Largs_tl` (why are `\rw_Atype:` and `\rw_Ltype:` implemented as control sequences and not token lists?).

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

```
\cs_new:Nn \rw_Atype: {}
\cs_new:Nn \rw_Ltype: {}
\tl_new:Nn \l_rw_Aargs_tl {}
\tl_new:Nn \l_rw_Largs_tl {}
\bool_new:N \l_rw_revert_random_bool

\cs_new:Nn \rw_set_defaults:
{
  \fp_set:Nn \l_rw_step_length_fp {10}
  \int_set:Nn \l_rw_step_number_int {10}
  \cs_set:Nn \rw_Atype: {interval:nn}
  \tl_set:Nn \l_rw_Aargs_tl { {-\c_pi_fp} {\c_pi_fp} }
  \cs_set:Nn \rw_Ltype: {fixed:n}
  \tl_set:Nn \l_rw_Largs_tl {\l_rw_step_length_fp}
  \bool_set_false:N \l_rw_revert_random_bool
}
```

\keys_define:nn

We introduce the keys for our package.

```
\keys_define:nn { randomwalk } {
  number .value_required:,
  length .value_required:,
  angles .value_required:,
  number .code:n = {\int_set:Nn \l_rw_step_number_int {#1}},
  length .code:n = {
    \clist_clear:N \l_rw_lengths_clist
    \clist_put_right:Nn \l_rw_lengths_clist {#1}
    \tl_set:Nn \l_rw_Largs_tl {\l_rw_lengths_clist}
    \rw_clist_fp_from_dim:N \l_rw_lengths_clist
    \rw_clist_count:NN \l_rw_tmpa_int \l_rw_lengths_clist
    \int_compare:nNnTF {\l_rw_tmpa_int}={1}
    {
      \cs_gset:Nn \rw_Ltype: {fixed:n}
    }
    {
      \cs_gset:Nn \rw_Ltype: {list:N}
    }
```

```
  },
  angles .code:n  = {
    \clist_clear:N \l_rw_angles_clist
    \clist_put_right:Nn \l_rw_angles_clist {#1}
    \cs_gset:Nn \rw_Atype: {list:N}
    \tl_set:Nn \l_rw_Aargs_tl {\l_rw_angles_clist}
  },
  degree .code:n = {\rw_radians_from_degrees:N \l_rw_angles_clist},
  degrees .code:n = {\rw_radians_from_degrees:N \l_rw_angles_clist},
  angles-relative .code:n = {\cs_gset:Nx \rw_Atype: {rel_\rw_Atype:}},
  revert-random .bool_set:N = \l_rw_revert_random_bool,
}
```

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 0 originally. Then we draw the relevant TikZ picture by repeatedly calling `\rw_draw_step:`.

```
\cs_new:Nn \rw_walk:
{
  \fp_set:Nn \l_rw_old_x_fp {0}
  \fp_set:Nn \l_rw_old_y_fp {0}
  \fp_set:Nn \l_rw_new_x_fp {0}
  \fp_set:Nn \l_rw_new_y_fp {0}
  \begin{tikzpicture}
    \prg_stepwise_inline:nnnn {1}{1}{\l_rw_step_number_int}
    {
      \rw_step_draw:
    }
    \bool_if:NF \l_rw_revert_random_bool {
      \global \cr@nd \cr@nd
    }
  \end{tikzpicture}
}
```

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

`\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.

```
\cs_new:Nn \rw_step_draw:
{
  \rw_step_random_generic:VV \l_rw_Largs_tl \l_rw_Aargs_tl
  \fp_add:Nn \l_rw_new_x_fp {\l_rw_step_x_fp}
```

```
  \fp_add:Nn \l_rw_new_y_fp {\l_rw_step_y_fp}
  \draw (\fp_to_dim:N \l_rw_old_x_fp, \fp_to_dim:N \l_rw_old_y_fp)
  -- (\fp_to_dim:N \l_rw_new_x_fp, \fp_to_dim:N \l_rw_new_y_fp);
  \fp_set:Nn \l_rw_old_x_fp {\l_rw_new_x_fp}
  \fp_set:Nn \l_rw_old_y_fp {\l_rw_new_y_fp}
}
```

`\rw_step_random_generic:nn` It is better to write a function that produces one random step.

```
\cs_new:Nn \rw_step_random_generic:nn
{
  \cs:w rw_L \rw_Ltype: \cs_end: #1
  \cs:w rw_A \rw_Atype: \cs_end: #2
  \rw_step_build:
}
\cs_generate_variant:Nn \rw_step_random_generic:nn {VV}
```

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.

```
\cs_new:Nn \rw_Lfixed:n {
  \fp_set:Nn \l_rw_radius_fp {#1} }
\cs_new:Nn \rw_Llist:N {
  \rw_set_to_random_clist_element:NN \l_rw_radius_fp #1 }
\cs_new:Nn \rw_Linterval:nn {
  \rw_set_to_random_fp:Nnn \l_rw_radius_fp {#1} {#2} }
```

`\rw_L..:.` Then for angles.

```
\cs_new:Nn \rw_Ainterval:nn {
  \rw_set_to_random_fp:Nnn \l_rw_angle_fp {#1} {#2} }
\cs_new:Nn \rw_Alist:N {
  \rw_set_to_random_clist_element:NN \l_rw_angle_fp #1 }
\cs_new:Nn \rw_Arel_interval:nn {
  \rw_add_to_random_fp:Nnn \l_rw_angle_fp {#1} {#2} }
\cs_new:Nn \rw_Arel_list:N {
  \rw_add_to_random_clist_element:NN \l_rw_angle_fp #1 }
```

`\rw_step_build:` And the operation to build the step from the random polar coordinates (these, we obtain via the `\rw_A...` and `\rw_L...` commands):

```
\cs_new:Nn \rw_step_build:
{
  \rw_cartesian_from_polar:NNNN  \l_rw_step_x_fp \l_rw_step_y_fp
                                 \l_rw_radius_fp \l_rw_angle_fp
}
```

`\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.

```
\cs_new_protected:Nn \rw_cartesian_from_polar:NNNN
{
  \fp_cos:Nn #1 {\fp_use:N #4}
  \fp_sin:Nn #2 {\fp_use:N #4}
  \fp_mul:Nn #1 {\fp_use:N #3}
  \fp_mul:Nn #2 {\fp_use:N #3}
}
\cs_new_protected:Nn \rw_gcartesian_from_polar:NNNN
{
  \fp_gcos:Nn #1 {\fp_use:N #4}
  \fp_gsin:Nn #2 {\fp_use:N #4}
  \fp_gmul:Nn #1 {\fp_use:N #3}
  \fp_gmul:Nn #2 {\fp_use:N #3}
}
```

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 LaTeX3y functions. Also, this will allow us to change quite easily our source of random numbers.

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

We also need floating point random numbers.

```
\cs_new:Nn \rw_set_to_random_fp:Nnn
{
  \fp_set:Nn \l_rw_tmpa_fp {#3}
  \fp_sub:Nn \l_rw_tmpa_fp {#2}
  \rand
  \fp_set:Nn \l_rw_tmpb_fp {\int_use:N \c@lcg@rand}
  \fp_div:Nn \l_rw_tmpb_fp {\int_use:N \c_rw_lcg_range_int}
  \fp_mul:Nn \l_rw_tmpa_fp {\l_rw_tmpb_fp}
```

```
  \fp_add:Nn \l_rw_tmpa_fp {#2}
  \fp_set:Nn #1 { \l_rw_tmpa_fp }
}
\cs_new:Nn \rw_add_to_random_fp:Nnn
{
  \fp_set:Nn \l_rw_tmpa_fp {#3}
  \fp_sub:Nn \l_rw_tmpa_fp {#2}
  \rand
  \fp_set:Nn \l_rw_tmpb_fp {\int_use:N \c@lcg@rand}
  \fp_div:Nn \l_rw_tmpb_fp {\int_use:N \c_rw_lcg_range_int}
  \fp_mul:Nn \l_rw_tmpa_fp {\l_rw_tmpb_fp}
  \fp_add:Nn \l_rw_tmpa_fp {#2}
  \fp_add:Nn #1 { \l_rw_tmpa_fp } %here: mod?
}
```

There does not seem to be any `clist`-counting implemented in LaTeX3, so we do it ourselves.

```
\cs_new:Nn \rw_clist_count:NN
{
  \int_set:Nn \l_rw_tmpa_int {0}
  \clist_set_eq:NN \l_rw_tmpa_clist #2
  \bool_until_do:nn
  {
    \clist_if_empty_p:N \l_rw_tmpa_clist
  }{
    \clist_pop:NN \l_rw_tmpa_clist \l_rw_tmpa_toks
    \int_add:Nn \l_rw_tmpa_int {1}
  }
  \int_set_eq:NN #1 \l_rw_tmpa_int
}
```

We also pick the `n`-th element of a `clist`.[1]

```
\cs_new:Nn \rw_clist_nth:NNn {
  \int_set:Nn \l_rw_tmpa_int {#3}
  \clist_set_eq:NN \l_rw_tmpa_clist #2
  \bool_until_do:nn
  {
    \int_compare_p:nNn {\l_rw_tmpa_int}<{0}
  }{
    \clist_pop:NN \l_rw_tmpa_clist \l_rw_tmpc_fp
    \int_add:Nn \l_rw_tmpa_int {-1}
  }
  \fp_set:Nn #1 {\l_rw_tmpc_fp}
}
```

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

[1] Is `\l_rw_tmpa_toks` a complete misnomer?

```
\cs_new:Nn \rw_set_to_random_clist_element:NN
{
  \rw_clist_count:NN \l_rw_tmpa_int #2
  \rw_set_to_random_int:Nnn \l_rw_tmpb_int {0} {\l_rw_tmpa_int}
  \rw_clist_nth:NNn #1 #2 {\l_rw_tmpb_int}
}
\cs_new:Nn \rw_add_to_random_clist_element:NN
{
  \rw_clist_count:NN \l_rw_tmpa_int #2
  \rw_set_to_random_int:Nnn \l_rw_tmpb_int {0} {\l_rw_tmpa_int}
  \rw_clist_nth:NNn \l_rw_tmpb_fp #2 {\l_rw_tmpb_int}
  \fp_add:Nn #1 {\l_rw_tmpb_fp}
}
```

More stuff on `clists`.

```
\cs_new:Nn \rw_radians_from_degrees:N
{
  \clist_clear:N \l_rw_tmpa_clist
  \bool_until_do:nn
  {
    \clist_if_empty_p:N #1
  }{
    \clist_pop:NN #1 \l_rw_tmpa_toks
    \fp_set:Nn \l_rw_tmpa_fp {\l_rw_tmpa_toks}
    \fp_mul:Nn \l_rw_tmpa_fp {\c_rw_one_degree_fp}
    \clist_push:NV \l_rw_tmpa_clist \l_rw_tmpa_fp
  }
  \clist_put_right:NV #1 \l_rw_tmpa_clist
}

\cs_new:Nn \rw_clist_fp_from_dim:N
{
  \clist_clear:N \l_rw_tmpa_clist
  \bool_until_do:nn
  {
    \clist_if_empty_p:N #1
  }{
    \clist_pop:NN #1 \l_rw_tmpa_toks
    \fp_set_from_dim:Nn \l_rw_tmpa_fp {\l_rw_tmpa_toks}
    \clist_push:NV \l_rw_tmpa_clist \l_rw_tmpa_fp
  }
  \clist_put_right:NV #1 \l_rw_tmpa_clist
}
```

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`.

```
\fp_new:N \l_rw_old_x_fp
```

```
\fp_new:N \l_rw_old_y_fp
\fp_new:N \l_rw_step_x_fp
\fp_new:N \l_rw_step_y_fp
\fp_new:N \l_rw_new_x_fp
\fp_new:N \l_rw_new_y_fp
\fp_new:N \l_rw_angle_fp
\int_new:N \l_rw_step_number_int
\clist_new:N \l_rw_angles_clist
\clist_new:N \l_rw_lengths_clist

\fp_new:N \l_rw_tmpa_fp
\fp_new:N \l_rw_tmpb_fp
\fp_new:N \l_rw_tmpc_fp
\clist_new:N \l_rw_tmpa_clist
\clist_new:N \l_rw_tmpb_clist
\int_new:N \l_rw_tmpa_int
\int_new:N \l_rw_tmpb_int
\int_new:N \l_rw_tmpc_int
```

</package>