# A users guide to Twister

Mark Bell

October 20, 2013

Twister is a program by myself, Tracy Hall and Saul Schleimer for constructing triangulations of surface bundles and Heegaard splittings from a description of a mapping class of a surface. This users guide will be based on Twister 2.4.0.

## 1 Getting Twister

Twister can be compiled as a stand-alone program, or as a Python 2 or Python 3 extension. However, it is also included in SnapPy[1] and we recommend using this. See the instructions for getting SnapPy at `http://www.math.uic.edu/t3m/SnapPy/installing.html`

To compile Twister yourself, you can get the source code for Twister from `https://bitbucket.org/Mark_Bell/twister/` or straight from the Mercurial repository with the command:

```
> hg clone https://bitbucket.org/Mark_Bell/twister
```

To install Twister as a site-package in Python use the command:

```
> python setup.py install
```

Or, to install Twister into your user directory, use the command:

```
> python setup.py install --user
```

To test your installation of Twister use the command:

```
> python setup.py test
```

This will use Twister to construct fibred knot complements in various different ways and check that they are isometric to those in SnapPy. It should finish with the last line being:

```
Overall Result: PASS
```

[1]Version 1.3.10 and later.

## 2 Working Examples

We begin by looking at a basic example of using Twister from within SnapPy.

### 2.1 Example 1

```
>>> S = twister.Surface('S_1_1')
>>> S.info()
A Twister surface of genus 1 with 1 boundary component(s).
Loops: a, b
Arcs: x
>>> S.info(verbose=True)
A Twister surface of genus 1 with 1 boundary component(s).
Loops: a, b
Inverse names: A, B
Arcs: x
Inverse names: X
Macros:
['', 'a', 'b', 'x']
['a', 0, 1, 1]
['b', 1, 0, 0]
['x', 1, 0, 0]
>>> M = S.bundle(monodromy='a*B')
>>> type(M)
<type 'snappy.SnapPy.Manifold'>
>>> M.volume()
2.0298832128
>>> M.is_isometric_to(Manifold('4_1'))
True
```

We start by creating an instance of a twister surface class. In this case we have passed in the string `S_1_1`. As this is the name of one of the surface files in Twister's surface database, that file is loaded and the corresponding surface created.

As the name suggests, this file describes a torus with one boundary component. However, we check this by asking for a description of the underlying surface. From this we can also see that this surface has several curves drawn on it. There are two closed loops, named `a` and `b`, and an arc, named `x` which connects the boundary to itself. By requesting more verbose information we can find out that $|a \cap b| = |a \cap x| = 1$ and $|b \cap x| = 0$.

Next we construct a surface bundle over the circle with fibre `S`. By providing the parameter `a*B`, we specify that the monodromy of this bundle should be made by composing a right Dehn twist about `a` with a left Dehn twist about `b`.

This process produces a SnapPy Manifold, which we see has hyperbolic volume 2.0298832128. Finally we check that the manifold that we have produce is actually the figure-eight knot complement by loading the complement from SnapPy's database and checking that they are isometric.

## 2.2 Example 2

```
>>> S = twister.Surface('heeg_fig8')
>>> S.info()
A Twister surface of genus 2 with 0 boundary component(s).
Loops:a,b,c
Arcs:
>>> S.info(True)
A Twister surface of genus 2 with 0 boundary component(s).
Loops: a, b, c
Inverse names: A, B, C
Arcs:
Inverse names:
Macros: h:a*b*C
['', 'a', 'b', 'c']
['a', 0, 0, 5]
['b', 0, 0, 5]
['c', 5, 5, 0]
>>> M = S.splitting(gluing='', handles='a*b*C')
>>> M.volume()
2.02988321282
>>> M.is_isometric_to(Manifold('4_1'))
True
```

Again we start by creating an instance of a twister surface class. Again, `heeg_fig8` is the name of one of the surface files in Twister's surface database and so that file is loaded and the corresponding surface created.

Checking the information of this surface we see that it is the genus two surface and it has three closed loops on it, named `a`, `b` and `c`. By requesting more verbose information we discover that these curves intersect in a complicated way.

Next we construct a Heegaard splitting over `S`. We do this by taking two copies of $S \times [0, 1]$ and first identifying their upper boundaries via a mapping class and then attaching 2–handles along certain curves in their lower boundaries. and

The mapping class is determined by the gluing parameter. In this case, as we provided `b*A`, we specify that the gluing between the two upper boundaries should be a compostion of a left Dehn twist about `b` and a right Dehn twist about `a`. The 2–handles to attach are determined by the handles parameter. As we provided `a*b*C`, 2-handles will be attached along `a` and `b` in the lower boundary of the first compression body and a 2–handle will also be attached along `c` in the lower boundary of second compression body.

Again, this produces a SnapPy Manifold which has hyperbolic volume 2.0298832128 and is isometric to the the figure-eight knot complement.

In fact in this case, the curves on the surface we specially chosen so that this processes would create the genus two handlebody / compression body decomposition of the figure eight knot complement.

# 3 Twister Methods

The Twister module provides: `Surface` (a class), `DT_drilling_surface` and `DT_handles_surface` (functions) and `surface_database` and `version` (variables). In this section we discuss each in turn.

## 3.1 Surfaces

Twister is centered around the `Surface` class. This represents a finite square complex whose underlying space $S$ is a 2–manifold (possibly with boundary) along with a special set of named curves on the surface. The *mapping class group* of the surface

$$\mathrm{Mod}^+(S) := {}^{\mathrm{Homeo}^+(S)}\big/_{\mathrm{isotopy}}$$

is the group of self-homeomorphisms of $S$ up to isotopy and will be key to construct 3–manifolds later.

### 3.1.1 Initialisation

A `Surface` can be initialised in several different ways. We can create a new surface by using `Surface(surface)` where `surface` is:

- a string containing the name of a surface file in Twisters surface file database,
- a string containing the path to a surface file,
- a string containing the path to a plink virtual link projection file,
- a string containing the contents of a surface file, or
- a pair of integers (genus, boundary).

In the first three cases, Twister will create the surface specified by the file. In the fourth case, a surface with the required genus and number of boundary components will be created with the curves of the Humphries generating set as the special set of curves, see [1, Figure 13].

The special set of curves that are specified for a `Surface` are the *core curves* of the square complex, that is maximal curves that can be made by connecting one side of a square to the opposite. We require that core curves these are always simple. A core curve is either a closed loop or an arc with endpoints in the boundary of the surface.

The surface provides each curve with a name and an inverse name. In the case of a loop, we identify its name with a right Dehn twist about it and its inverse name with a left Dehn twister about it. In the case of an arc between two distinct boundary components, we identify its name with a right half twist about it and its inverse name with a left half twist about it. See Figure 1 and Figure 2. We will also use the notation $T_a$ for a right Dehn twist (half twist) about a loop (arc) named `a`.

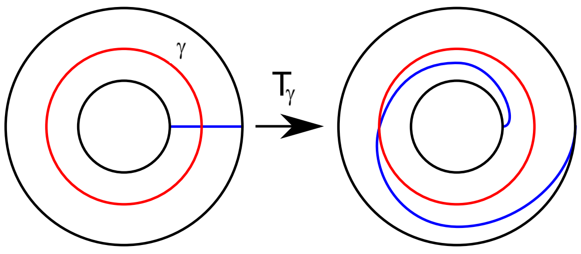


Figure 1: A left Dehn twist about the loop $\gamma$.

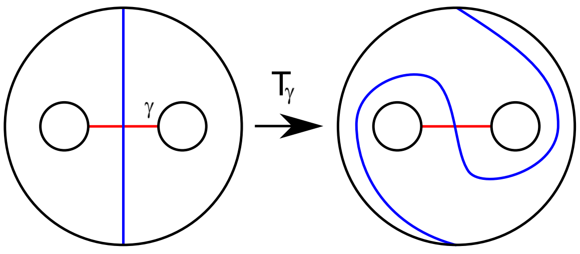


Figure 2: A left half-twist about the arc $\gamma$.

We use a string of curve names and inverse names, each separated by a `*`, to denote the composition of these Dehn twists and half twists. The composition is performed from left to right, so for example `a*b` represents $T_a \circ T_b$. This allows us to specify elements of $\mathrm{Mod}^+(S)$ by strings.

### 3.1.2 Information

Information about a `Surface` can be obtained by using the `Surface.info([verbose])` method. This prints out various pieces of information about the `Surface` such as the genus, number of boundary components and as well as the curves on the surface. If `verbose` is set to `True` then additional information, such as the cuves inverse names, macros and a matrix of the number of intersections between each of the pairs of curves is included.

This is a useful method to use to check that the surface is indeed the one that you think it is.

### 3.1.3 Bundles

One of the main methods of a `Surface` is `Surface.bundle(monodromy)` which constructs a surface bundle over the circle with fibre this surface.

More precisely, suppose that `S` is a surface class and `monodromy` is a string describing the mapping class $f \in \mathrm{Mod}^+(S)$. Then `S.bundle(monodromy)` returns the 3–manifold:

$$S \times [0,1] \Big/ (x,1) \sim (\phi(x),0)$$

where $\phi$ is any representative of $f$.

### 3.1.4 Splittings

The other main method of a `Surface` is `Surface.splitting(gluing, handles)` which constructs a Heegaard splitting over this surface.

More precisely, suppose that `S` is a surface class, `gluing` is a string describing the mapping class $f \in \mathrm{Mod}^+(S)$ and `handles` is a string containing loop names and inverse names, each separated by a `*`. Then `S.splitting(gluing, handles)` returns the 3–manifold obtained by the following process. Take two copies of $S \times [0,1]$, called $M_1$ and $M_2$: For each loop name (inverse name) listed in `handles`, attach a 2–handle via its cylinder boundary along a copy of that loop in $\partial_- M_1$ ($\partial_- M_2$). Finally identify $\partial_+ M_1$ and $\partial_+ M_2$ via $f$.

### 3.1.5 Optional arguments

Both `Surface.bundle(mondromy)` and `Surface.splitting(gluing, handles)` also accept several optional arguements:

- `name=None` This sets a custom name for the manifold produced. If none is specified then `monodromy` and `gluing + ' ' + handles` are used for `Surface.bundle()` and `Surface.splitting()` respectively.
- `warnings=True` This specifies if any warnings that are produced duing the construction should be shown. For example if a boundary component with genus greater than 1 is found. Setting this to False suppresses all warnings, which may be useful when running Twister as part of a batch command. However we recommend the reader to consider all warnings as errors.
- `optimize=True` This specifies whether or not after building a manifold Twister should attempt to reduce the number of tetrahedra used. This is done by a folding off processes in which adjacent boundary faces are glued together where it will not change the manifold (topologically). See the function 'foldoff' in twister.cpp and close_cusps.c in the SnapPea kernel for further information about this process. This makes the triangulation smaller and quicker to load in SnapPy.
- `debugging_level=0` This specifies the debugging level which Twister is run at. This determines how much information Twister will display about the processes that it performs. Debugging level 1 can be used to check that commands are being parsed correctly.
- `return_type='manifold'` This specifies how the result should be returned. The possible options are 'manifold', 'triangulation' and 'string'. In the 'manifold' case a SnapPy Manifold is returned. In the 'triangulation' case a SnapPy Triangulation is returned. In the 'string' case a string containing a snappea file is returned.

### 3.1.6 Surface methods

As well as `Surface.info(...)`, `Surface.bundle(...)` and `Surface.splitting(...)`, a `Surface` object also stores various properties of the underlying surface. Below are listed the `Surface` methods:

- `Surface.surface_contents`
- `Surface.num_vertices`
- `Surface.num_edges`
- `Surface.num_squares`
- `Surface.Euler_characteristic`
- `Surface.genus`
- `Surface.num_boundary`
- `Surface.curves`
- `Surface.intersection_matrix`

These should satisfy identities such as:

$$\begin{aligned}\texttt{Euler\_characteristic} &= \texttt{num\_vertices} - \texttt{num\_edges} + \texttt{num\_squares} \\ &= 2 - 2\texttt{genus} - \texttt{num\_boundary}\end{aligned}$$

## 3.2 DT surface functions

Twister includes a pair of functions for producing surfaces from Dowker–Thistlethwaite codes. These are used as part of the test suite.

`twister.DT_drilling_surface(DT_code)`

This function takes a Dowker–Thistlethwaite code of a knot, given as a list of integers, and returns the a `Surface` that can be used to produce that knot complement by performing drillings.

`twister.DT_handles_surface(DT_code)` A function which takes a Dowker–Thistlethwaite code of a knot, given as a list of integers, and returns the a Surface that can be used to produce that knot complement by attaching handles. This is used as part of the test suite.

## 3.3 Surface database

Twister maintains a set `twister.surface_database` which stores the name of each surface file in Twister's surfacfe database. If at initialisation `Surface` is passed a string in this set then the corresponding surface is loaded.

## 3.4 Version

Finally `twister.version` is a string storing the current version number of the twister kernel.

# 4 Parsing

A `Surface` may also define several macros. These listed by `Surface.info(verbose=True)`. The primary use for macros is to create shortcuts for frequently used commands. In particular, scripts for producing surface files can also include macros to allow particular surface bundles or Heegaard splittings to be produced with little input.

Macros have a `name` and a `command`. When creating a bundle or splitting any marco `name` that appear in the `monodromy`, `gluing` or `handles` parameters are replaced by their `command`.

For example, consider Example 2.2 again. We can see from the advanced information printed that there is a macro which replaces `h` with `a*b*C`. Because of this we can produce the genus two handlebody / compression body decomposition of the figure eight knot complement with two different commands.

```
>>> S = twister.Surface('heeg_fig8')
>>> M = S.splitting(gluing='', handles='a*b*C')
>>> N = S.splitting(gluing='', handles='h')
>>> M.is_isometric_to(N)
True
```

## 5 Figure-eight examples

We now give several different examples for how to build the figure-eight knot complement using Twister. Note that in all of these examples the surface files used are prefix unique and so all '*' separators can be omitted.

```
>>> M1 = twister.Surface(surface='S_0_4').splitting(gluing='z*z*Y*Y', handles='a*B')
>>> M2 = twister.Surface('S_0_4').splitting('z*z*A', 'a*B')
>>> M3 = twister.Surface('S_0_6').splitting('z*Y*z*Y', 'a*A*b*B')
>>> M4 = twister.Surface('S_1_1').bundle(monodromy='A*b')
>>> M5 = twister.Surface('S_2_heeg').splitting(gluing='!e', handles='a*b*C*D')
>>> M6 = twister.Surface('heeg_fig8').splitting('', 'a*b*C')
>>> M7 = twister.Surface('S_0_5').splitting('!v*!y*!w*!z*!x','a*b*c*d*e*A*B*C*D*E')
```

- `M1`: This command builds a two-bridge presentation of the figure-eight knot. We start with the surface $S_{0,4}$, attach two-handles above the loop `a` and below the loop `b` and finally perform four half twists on neighbourhoods of the rectangles `z` and `y`.
- `M2`: Performs the same actions as before except this time we do a pair of half twists on a neighbourhood of the rectangle `z` and one Dehn twist about the annulus `a`. As `a` is isotopic to the boundary of a regular neighbourhood of `y`, performing a Dehn twist about `a` is isotopic to performing two half twists about `y`.
- `M3`: Using $S_{0,6}$ as the Heegaard surface, this gives the figure-eight knot as the closure of a three-strand braid.
- `M4`: Build a surface bundle over the circle with fibre $S_{1,1}$ and then perform two Dehn twists, a right one about `b` and then a left one about `a`.
- `M5`: Build a Heegaard splitting over the surface $S_2$, attach two handles above `a` and `b` and below `c` and `d` and finally drill out a neighbourhood of the loop `e`. Here the loop `e` is the figure-eight knot lying on the standard genus two splitting of $S^3$.
- `M6`: Build a Heegaard splitting of the figure-eight knot complement. The file describes the handle structure coming from the handlebody / compression body decomposition of the manifold. This is derived from the upper tunnel for the two-bridge presentation.
- `M7`: Build a Heegaard splitting over the surface $S_{0,5}$ with a carefully chosen generating set such that drilling the rectangles (in this order) after capping each cusp above and below is the complement of the figure-eight knot in the three-sphere.

It is worth noting that, with an appropriate gluing, any two-bridge link can be built using the command:

```
>>> M = twister.Surface(surface='S_0_4').splitting(gluing=GLUING, handles='a*B')
```

and all possible three-braid closures can be built using the commmand:

```
>>> M = twister.Surface(surface='S_0_6').splitting(gluing=GLUING, handles='a*A*b*B')
```

Consider, for example, the complement of the $8_{15}$ knot which is the closure of a three-braid but not a two-bridge knot. Hence it can be built using last command with the gluing `V*V*z*y*x*w*v*x*W*v*x*w*v*w*z` but cannot be built using the penultimate command.

## 6 Surface Files

We now describe the specification for a Twister surface file. An example of a valid surface file is given in Figure 4. To avoid conflicting with any of the surfaces in Twister's surface database we suggest that any user made surface files are given the '.sur' extension.

The first line of a surface file must be

```
# A Twister surface file
```

Each following line must either start with a `#`, in which case this line is a comment, or be of one of the following forms:

```
annulus,<name>,<inverse_name>,<sequence>#
rectangle,<name>,<inverse_name>,<sequence>#
macro,<name>,<command>#
```

We require that `<name>` and `<inverse_name>` are strings matching the Perl compatible regular expression `[a-zA-Z]\w+`.

We also require that `<sequence>` is a comma separated list of signed integers. The required sequence is given by first numbering the squares from 0 to $n$ and orienting the curves. The sequence assigned to a curve are the indices of the squares it passes through and the sign used for each is index is `+` if this curve and the other curve passing through the square form a positive basis and `-` otherwise. For example, see Figure 3.

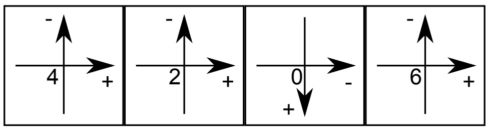
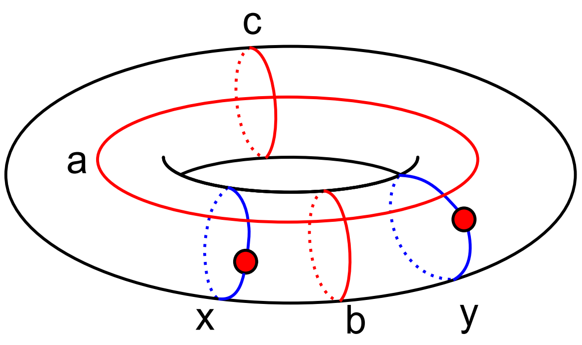


Figure 3: Curve sequence '+4, +2, −0, +6'.

We require that `<command>` be a valid Twister command.



```
# A Twister surface file
annulus,a,A,+0,+1,+2,+3#
annulus,b,B,-1#
annulus,c,C,-4#
rectangle,x,X,-0#
rectangle,y,Y,-2#
```

Figure 4: A surface file encoding the twice-punctured torus.

## References

[1] Catherine Labruère and Luis Paris. Presentations for the punctured mapping class groups in terms of Artin groups. *Algebr. Geom. Topol.*, 1:73–114 (electronic), 2001. [4]