# SIMPLIFYING CURVES ON SURFACES

SHREYAS CASTURI, JONATHAN CHEN, VIGNESH RAMAN, KYLE XIAO
*MENTOR: BALÁZS STRENNER*

## Contents

## 1. Project overview

This document is the write-up of an undergraduate project that took place in Fall 2017. The group met weekly on Fridays, for roughly four hours per meeting.

During this project, the group worked on Macaw, a Python module that does computations with curves on surfaces and homeomorphisms (deformation) of surfaces. General references on the topic include [CM17] and [FM12]. Macaw is being designed as an implementation of a quadratic-time algorithm of Margalit-Strenner-Yurttas [MSY17] for the Nielsen–Thurston Classification Problem. This project was a contribution to this implementation.

## 2. Motivation

There are many real-world examples that use applications of curves on surfaces. Take, for example, a taffy puller that can stretch and fold a piece of taffy. This piece of taffy can be modeled by a surface, and each fold can be modeled by a curve on this surface. Pulling and folding the taffy can be thought of, then, as mixing a surface, which creates more and more complicated curves on the surface. Observing the amount of mixing being done on the taffy gives the stretch factor, which quantifies how complicated the curve gets with increasing folds.

*Date*: December 7, 2017.

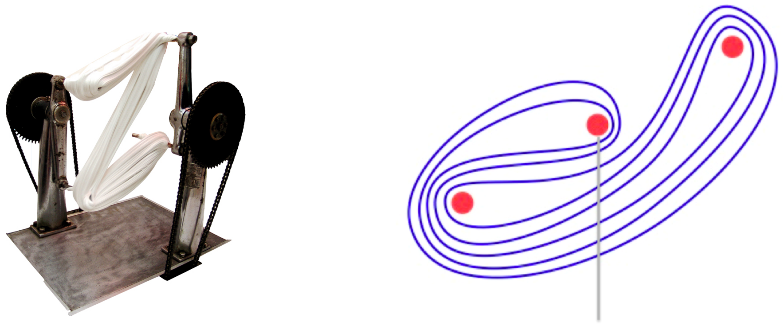


Figure 1. Taffy Puller and its Cross Section

Cross-section of the taffy is a curve and as rods are moving around, the curve is getting more complex.

More specifically, this project was motivated by two questions:

(1) Determine if a mapping class (”deformation”) of a surface is finite order, reducible or pseudo-Anosov. Informally, whether the deformation mixes up the whole surface or just part of it.
(2) Determine if a curve on surface is trivial or not. That is, whether a curve can be simplified down to a point on a surface by a continuous motion (isotopy).

The first question is a large topic which we will not elaborate on here. We address the second question by trying to ”tighten” the curve on its respective surface, and see if it can be tightened down to a point. This is done by replacing all ”illegal” paths with their ”legal” counterparts which are tighter/shorter.

## 3. Describing surfaces and curves

### 3.1. **Pants Surfaces and their Decomposition - An Overview.**

- A decomposition of a topological surface to smaller pieces is important in order to perform operations of the object. On possible decomposition is known as ”pants decomposition.”
- In a pants decomposition, surfaces are decomposed into *pairs of pants* along curve called *pants curves.* Pairs of pants are surfaces that can be obtained from the sphere by removing three disks.
- A curve on a surface can be described as a path that moves from pants curves to pants curves. However, there are paths that are not optimal (not tight).
- One can simplify a path by defining ”illegal subpaths”, which are not tight, and replacing them with their equivalent ”legal paths” which are tight. (The simplest example of this idea is that instead of going from A to B, then from B to C (an illegal path), we can just go from A to C (a legal path).)

3.2. **Formal Encoding of Curves.**

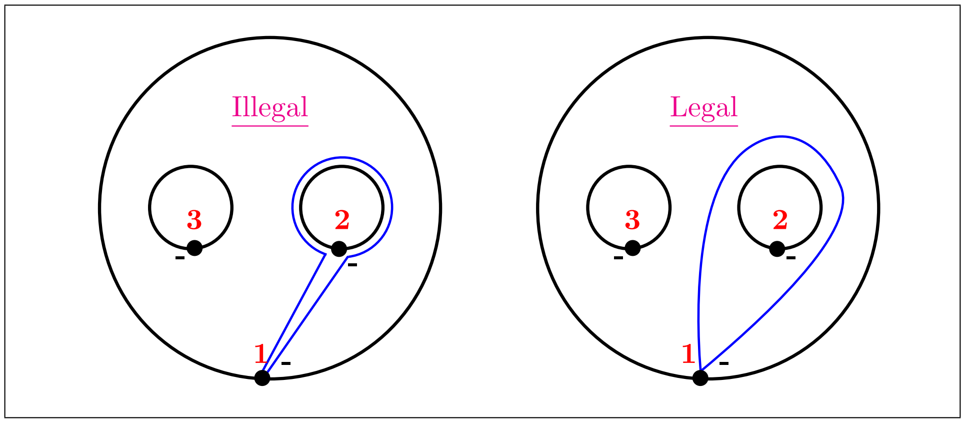
- Given that our paths are paths going from one pants curve to another, path encoding in the Macaw implementation is as follows:
  (1) A pair of pants has three boundary curves. Each boundary curve has 2 gates arbitrarily associated with it - one being positive and the other, negative.
  (2) Additionally, when a curve on a pair of pants starting from one of the boundaries goes around another boundary and returns, it forms a teardrop, and is encoded as a string denoting: (i) the direction it leaves the initial boundary; (ii) the curve it goes around; (iii) the direction it comes back to the original boundary.
- A path starts at an initial boundary curve, which we call 1 without loss of generality. This curve leaves the boundary 1 either through a negative or positive gate. The path then travels to boundary curve 2, reaching 2 via 2's negative or positive gate. At 2, the path can travel around the boundary curve, either by traveling "right" or "left". Once the path has returned to its original location at 2, it can either go to a different boundary curve, doing the aforementioned processes (now with different numbers to denote different boundary curves, such as 3) again and again, or return to the original boundary curve 1.
- The path that describes a curve on a pants surface is encoded by a list. Here is a simple example of an illegal curve and its encoding (from Test Case 1):

**Example 1.** $[1, -, -, 2, L, 2, -, -, 1]$ *is an illegal path.*

*What does this list mean? We see the path starts at boundary curve 1, leaves 1 through the negative gate, reaches 2 through 2's negative gate, travels leftwards along 2's boundary curve, leaves 2 through 2's negative gate, and reaches 1 through 1's negative gate.*

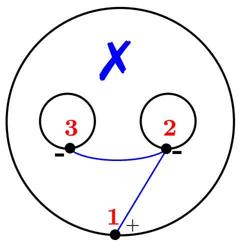
**Example 2.** $[1, L2R, 1]$ *is the legal path.*

*This list is a simplification of the illegal path above. L2R refers to the path traveling around 2 from the "left" and coming back to 2 from the right, and then leaving 2 to come back to 1. See the diagram below for a comparison.*



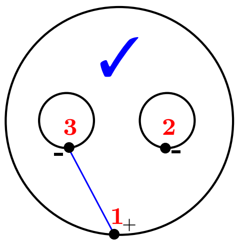
## 4. Illegal paths

- For a given pair of pants, it is possible to give multiple descriptions of a curve lying on it. However, when faced with complex curves, this often implies a massive amount of redundancy in description.
- Curves which have a redundancy in description, and which can generally be represented in a simpler form, are henceforth termed to be **illegal** *paths* (or **illegal** *curves*). They can simply be thought of as unnecessarily stretched strings (a.k.a not taut) across a surface, according to certain predefined rules, with some being:
  (1) For a pants curve decomposition, a curve is illegal if it starts from one pants boundary, traverses to a second pants boundary, and then comes back to the first.
  (2) For a pants curve decomposition, a curve is illegal if it starts from one pants boundary, traverses to, goes around, and then comes back to the first boundary from a second pants boundary.
- Curves which don't violate the aforementioned rules (mentioned in the statements) are said to be tight/taut paths/curves and these are the curves we wish to ultimately simplify to, given an input of an illegal curve. For instance,



The figure to the left is essentially a curve from boundary 1 to boundary 3, with a redundant attachment (i.e. traversal via) boundary point 2.

The figure to the right is the correct simplification - no inherent properties of the curve (i.e. without tearing or cutting) have changed while removing the redundancy.



- To summarize, it is in our interest to develop a succinct algorithm to simplify an inputted, illegal curve associated with a pair of pants, into legal paths (i.e. tight curves).

This section lays out the fundamental algorithm (by going through test cases, which individually deal with a differing type of illegal curve) used for replacing illegal paths with legal paths. Below is the list of associated 'Replacement Rules' with labeled diagrams giving an example of the same.

**Case 1.**

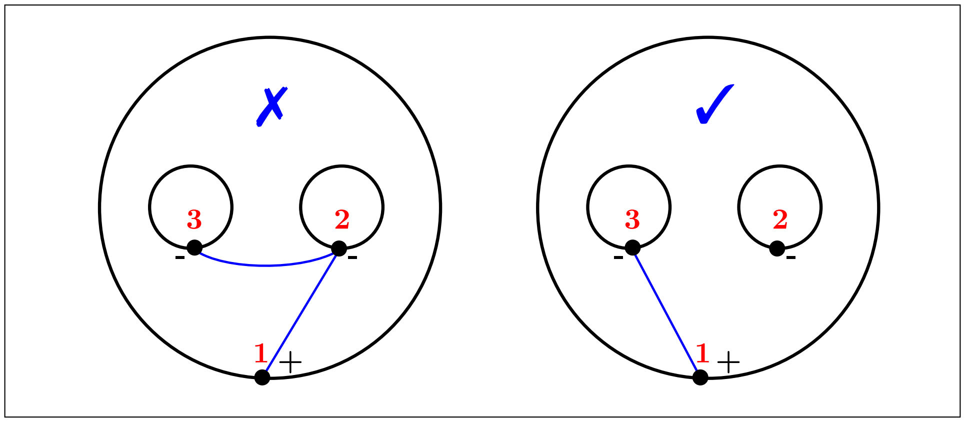
Input:$[c_1, g_1, g_2, c_2, g_2, g_x, c_x]$

**If**$(x == 3)$

Output: $[c_1, g_1, g_3, c_3]$

**Else**

Output: [] i.e. empty array



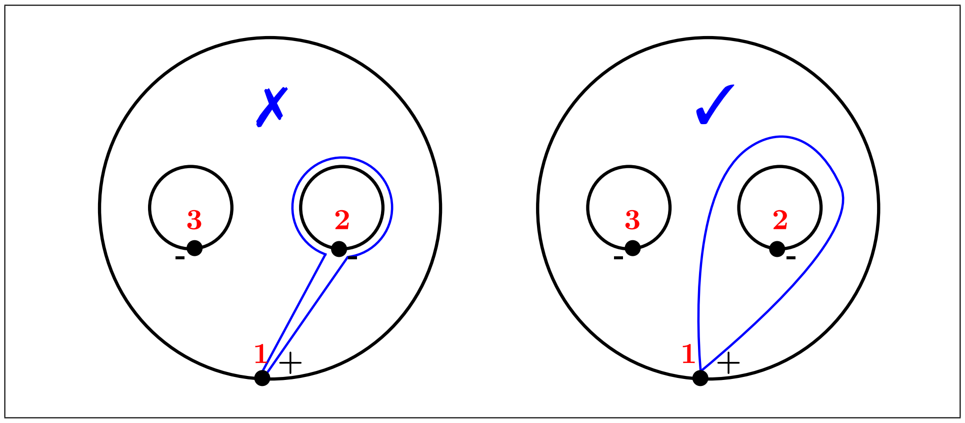
**Case 2.**

Input: $[c_1, g_1, g_2, c_2, t_2, c_2, g_2, g_1, c_1]$

Output: $[c_1, T_1, c_1]$

$T_1 = (t_2 == R)\ ?\ Rc_2L : Lc_2R$

Note: $c_2$ is the cyclic **successor** of $c_1$.



**Case 3.**

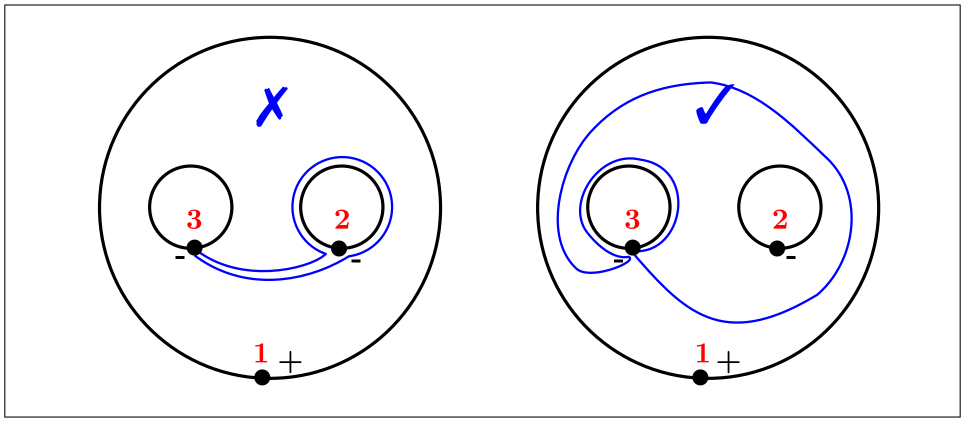
Input: $[c_1, g_1, g_2, c_2, t_2, c_2, g_2, g_1, c_1]$

Output: $[c_1, t_1, c_1, T_1, c_1]$

$t_1 = (t_2 == R)\ ?\ L : R$

$T_1 = (t_2 == R)\ ?\ Lc_3R : Rc_3L$

Note: $c_2$ is the cyclic **predecessor** of $c_1$.



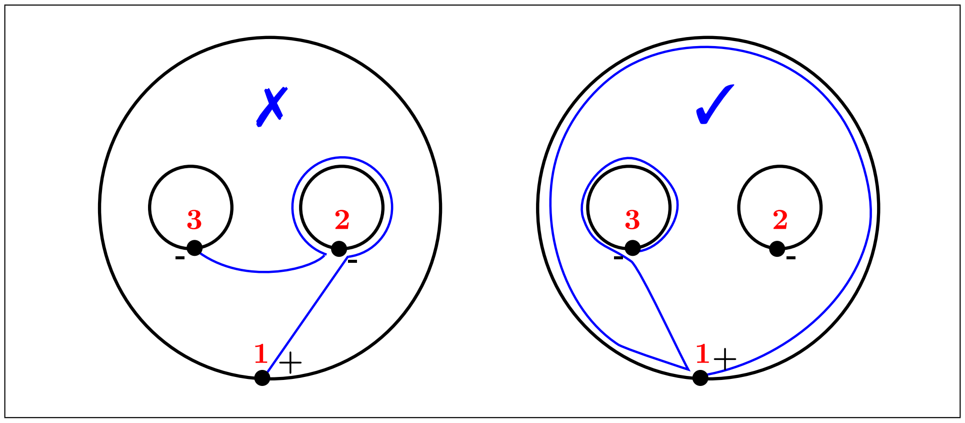
**Case 4.**

Input: $[c_1, g_1, g_2, c_2, t_2, c_2, g_2, g_3, c_3]$

Output: $[c_1, t_1, c_1, g_1, g_3, c_3, t_3, c_3]$

$t_1 = (t_2 == R)\ ?\ L : R$

$t_3 = (t_2 == R)\ ?\ L : R$



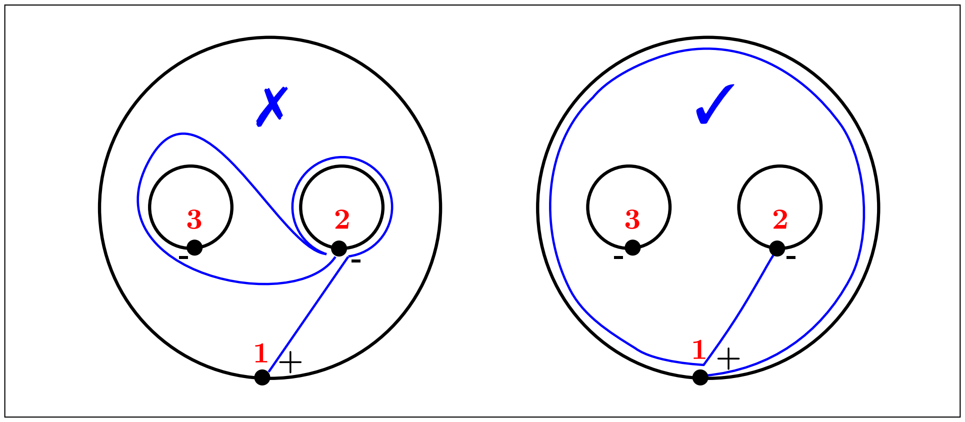
**Case 5.**

Input: $[c_1, g_1, g_2, c_2, t_2, c_2, T_2, c_2]$

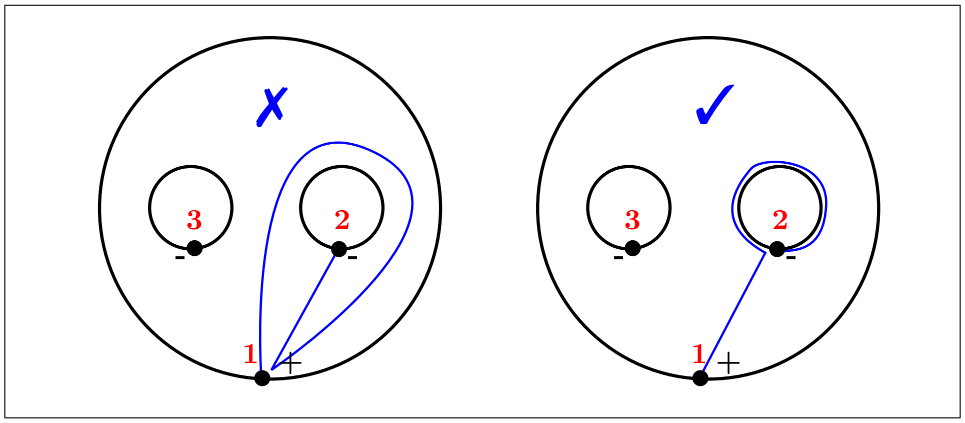
Output: $[c_1, t_1, c_1, g_1, g_2, c_2]$

$t_1 = (t_2 == R)\ ?\ L : R$

Note: $t_1$ is not dependent on $T_2$ because $T_2$ has to possess the same turning direction as $t_2$ - otherwise curves will intersect.



| **Case 6** | | |
|---|---|---|
| ***Input***: $[c_1, g_1, g_2, c_2, T_2, c_2]$ | | |
| ***If*** $(T_2$ contains $c_1)$ | ***Elseif*** $(T_2$ contains $c_3)$ | |
| | If($c_2$ is cyc. **successor** of $c_1$) | Else(i.e.$c_2$ is cyc. **pred.** of $c_1$) |
| Output: $[c_1, t_1, c_1, g_1, g_2, c_2]$ $t_1 = (T_2 == Rc_1L)\ ?\ R : L$ | Output: $[c_1, t_1, c_1, g_1, g_2, c_2, t_2, c_2]$ $t_1 = t_2 = R$ | Output: $[c_1, t_1, c_1, g_1, g_2, c_2, t_2, c_2]$ $t_1 = t_2 = L$ |



## 5. Acknowledgements

We would like to thank our mentor, Dr Balázs Strenner for his guidance, patience and advice during the entire course of our research project. We would also like to thank Dr Dan Margalit for offering advice as well as insight into how our solution fit into the big picture of the Nielson Thurston Classification Problem. Additionally, we would like to thank Dr Doron Lubinsky and the NSF foundation for financial support.

## References

[CM17] Matt Clay and Dan Margalit. *Office Hours with a Geometric Group Theorist.* Princeton University Press, 2017.

[FM12] Benson Farb and Dan Margalit. *A primer on mapping class groups*, volume 49 of *Princeton Mathematical Series.* Princeton University Press, Princeton, NJ, 2012.

[MSY17] Dan Margalit, Balázs Strenner, and Öykü Yurttaş. Fast Nielsen-Thurston classification. *In preparation*, 2017.