A software code to manipulate apparent contours

Oberwolfach, january 2007

Maurizio Paolini

Catholic University, Brescia

joint work with G. Bellettini, V. Beorchia, F. Pasquarelli

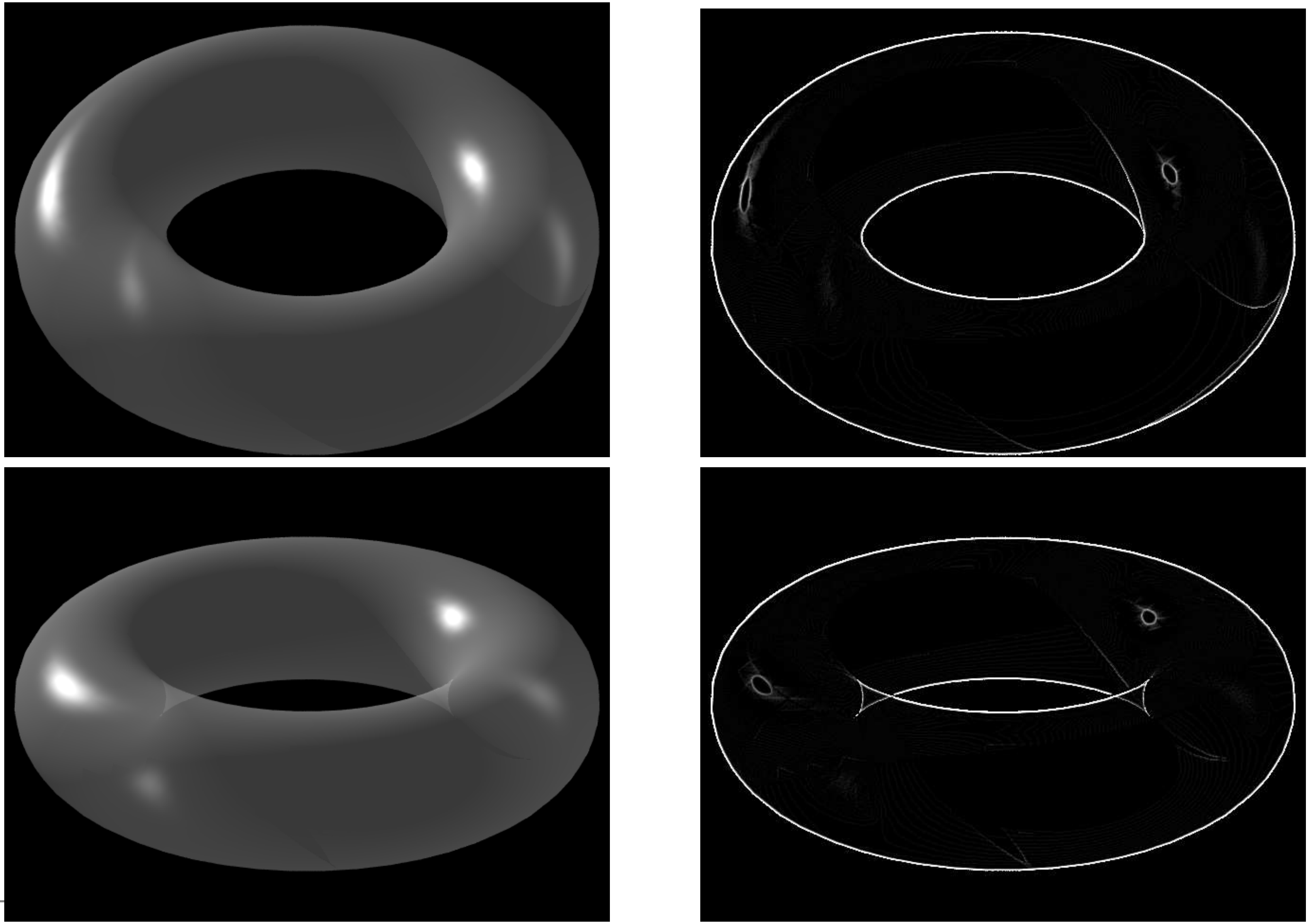
Outline

topological point of view

- Motivation [show animation ./anim1 (ctrl-F3)]
- Isotopic surfaces
- Cut and glue elementary rules
- Describing a contour
- The **“appcontour”** software
- Displaying a contour
- Conclusions

[Cerf, Huffman, Karpenko-Hughes, Bellettini-Beorchia-P., Luminati, Pignoni,...]

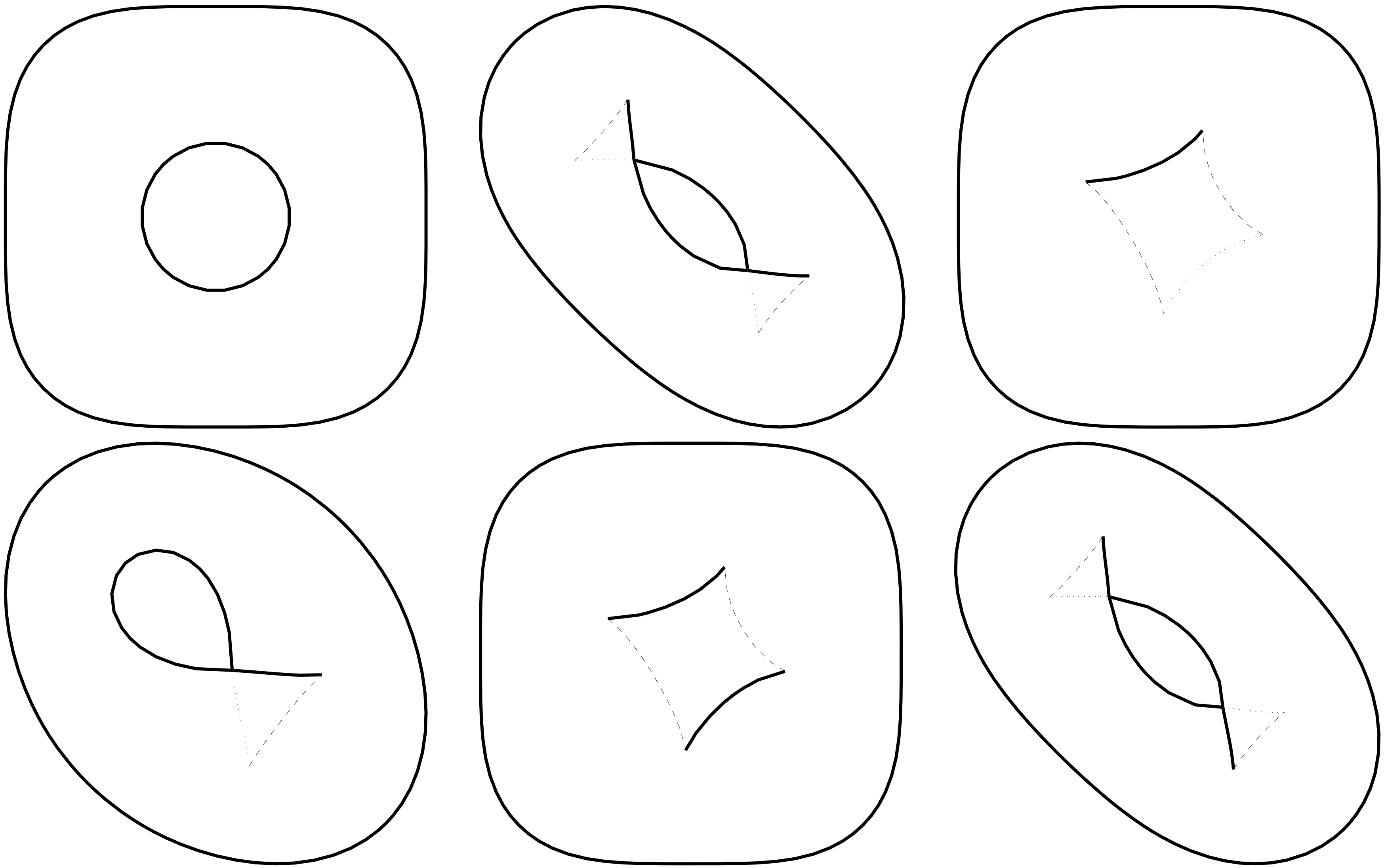
Motivation: isotopic surfaces



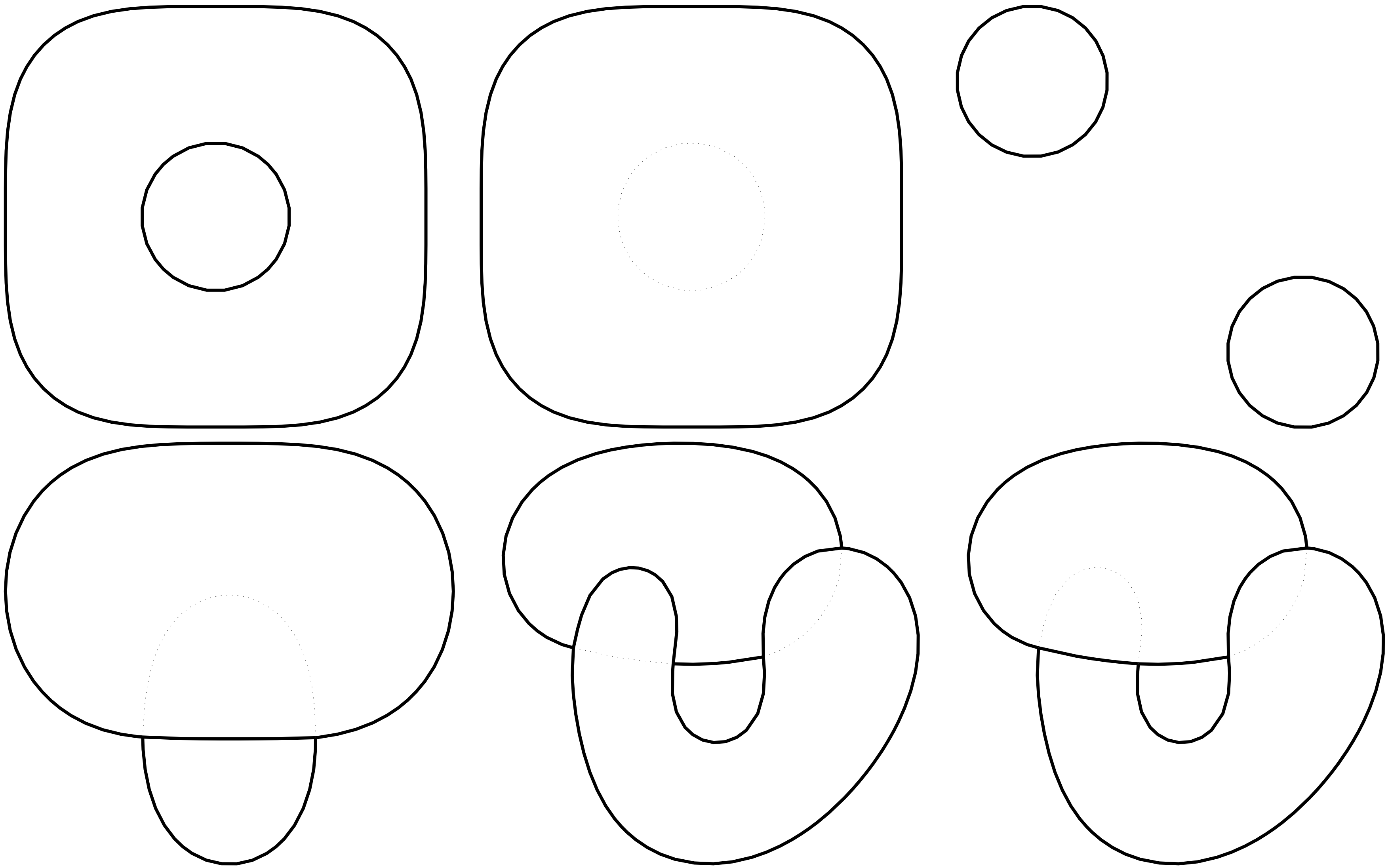
Isotopic surfaces

Two surfaces are isotopic
if
I can smoothly deform one onto the other
(without selfintersections)

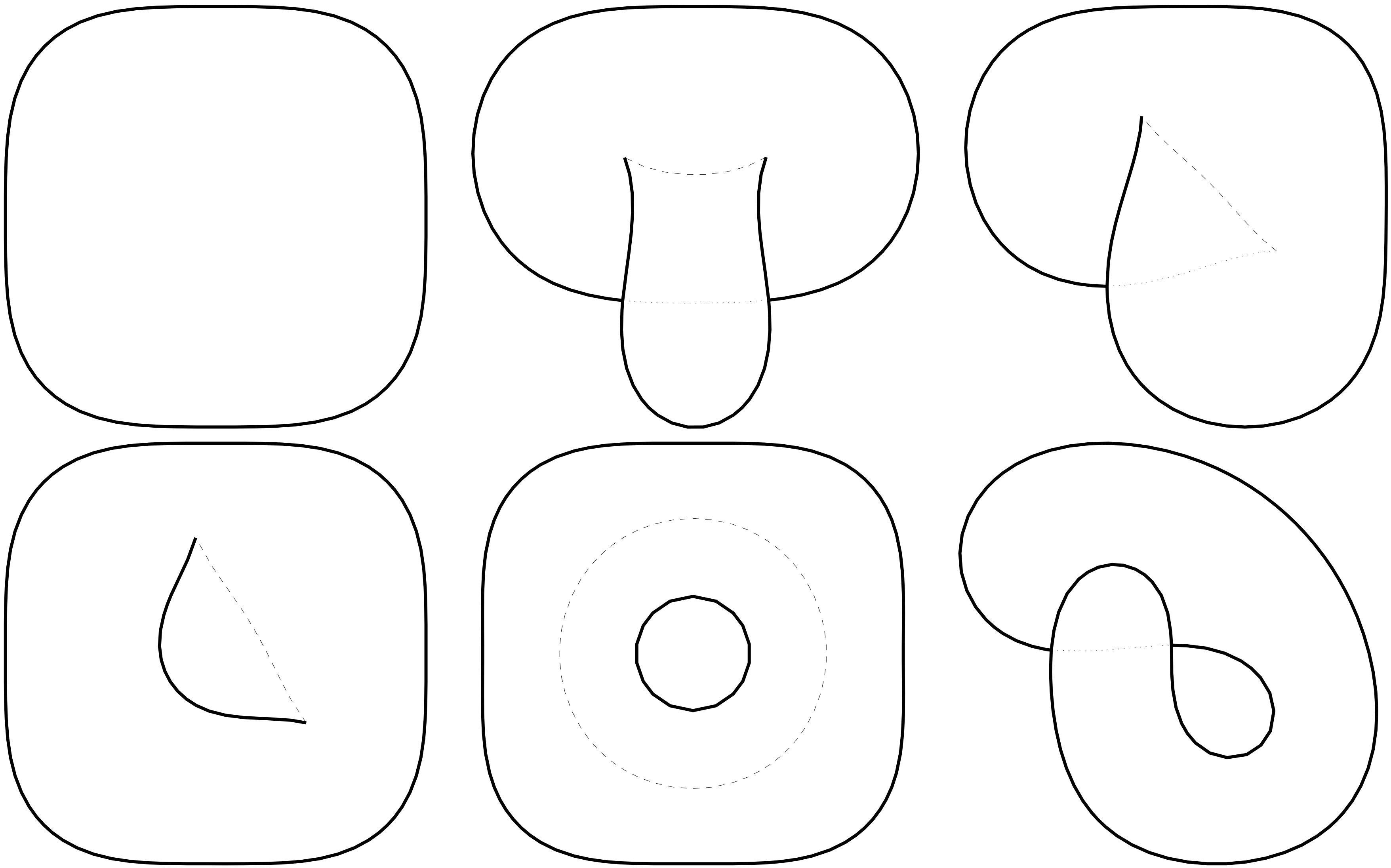
3D-isotopic equivalence



3D-isotopic equivalence (2)



3D-isotopic equivalence (3)



Manipulating apparent contours

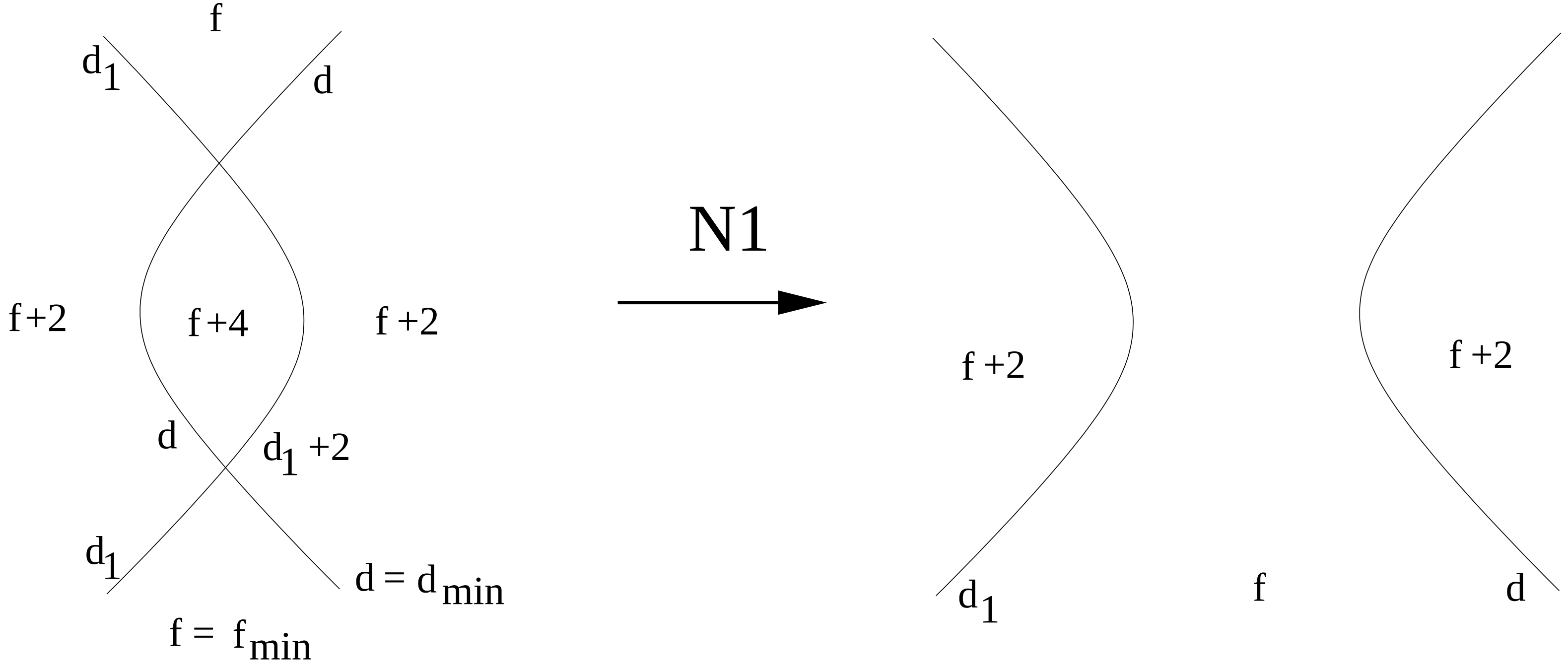
Aim: devise a (small) set of elementary rules that change an apparent contour keeping the 3D surface isotopic (No rigorous proof of this, yet). Is this set complete?
Similar to the Reidemeister rules for knotted links.

Manipulating apparent contours

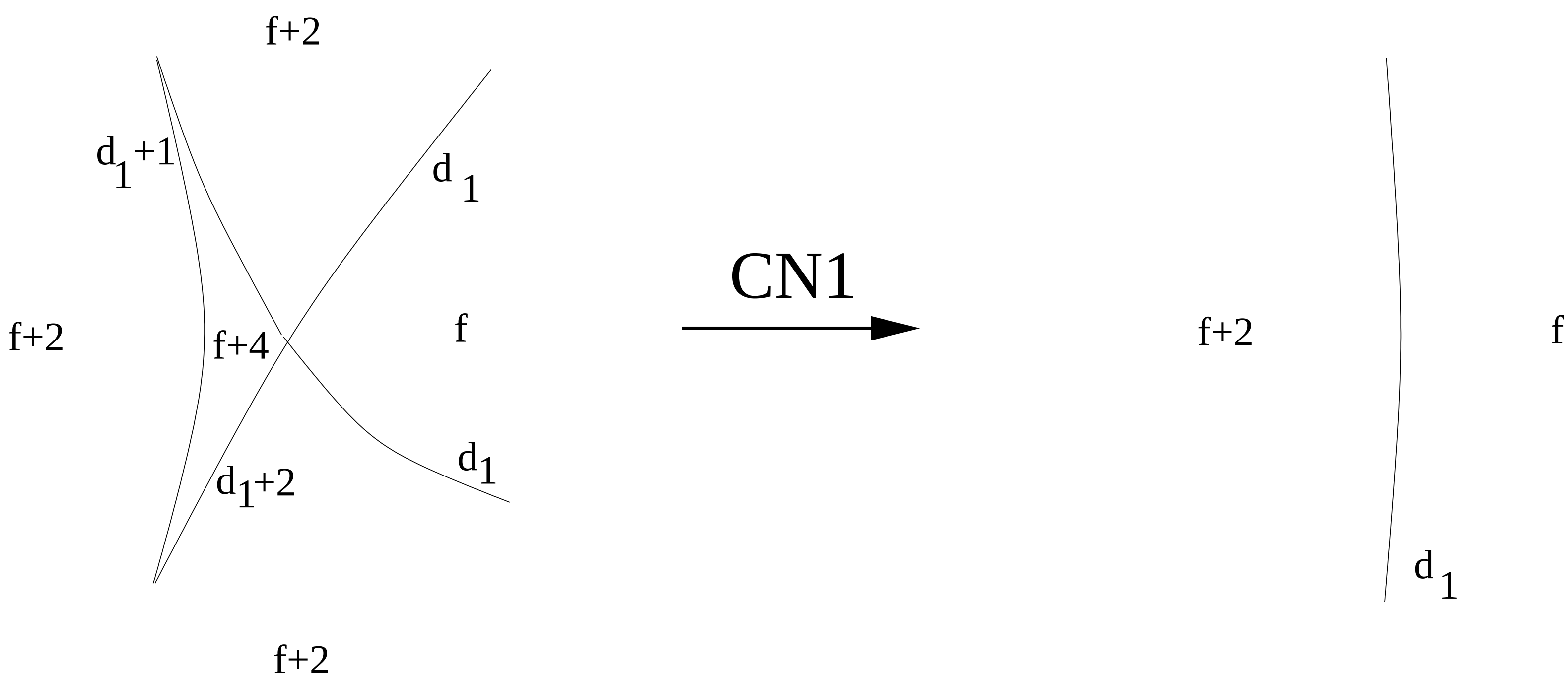
Aim: devise a (small) set of elementary rules that change an apparent contour keeping the 3D surface isotopic (No rigorous proof of this, yet). Is this set complete?

Similar to the Reidemeister rules for knotted links.

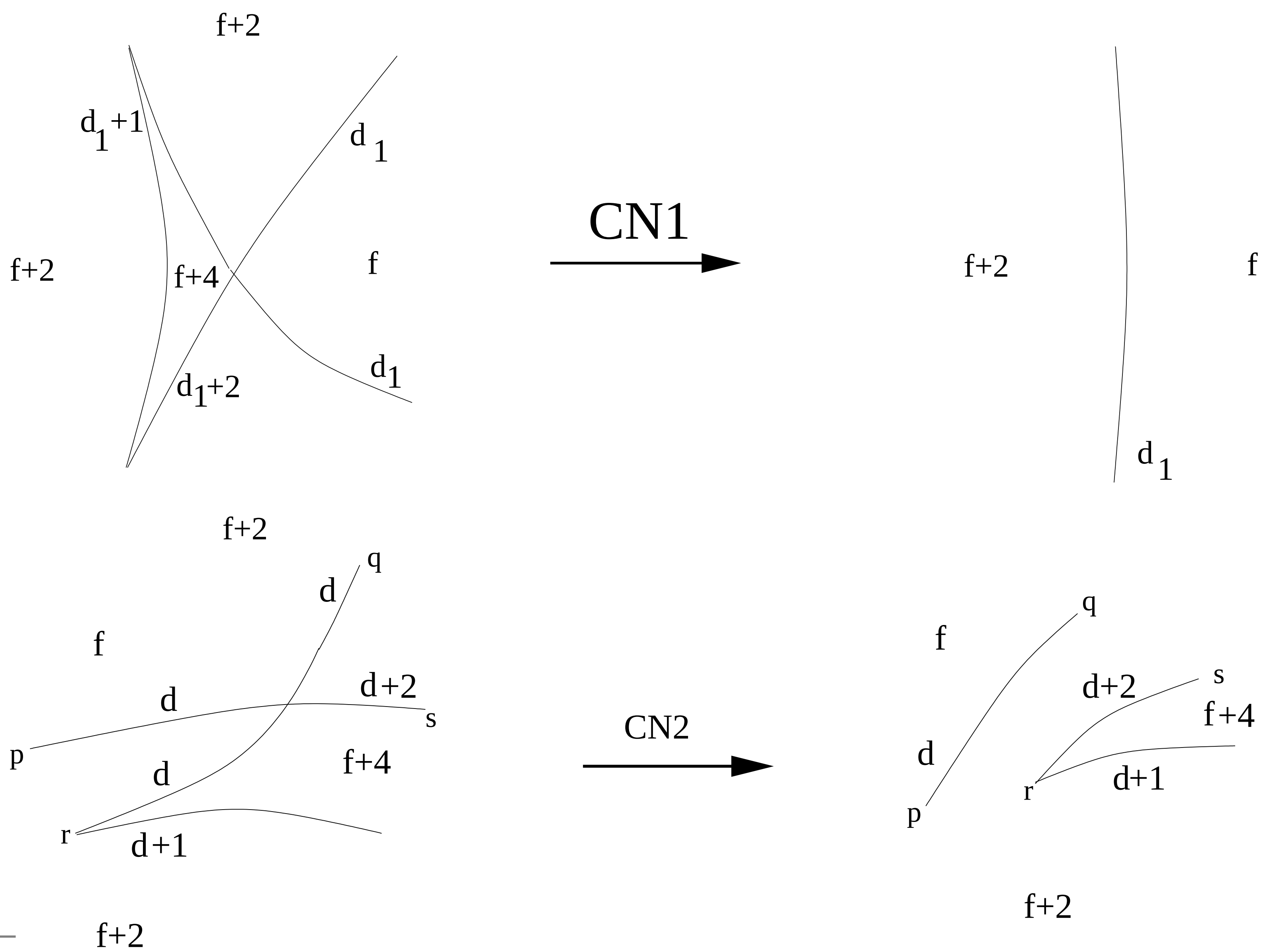
Some of our rules:



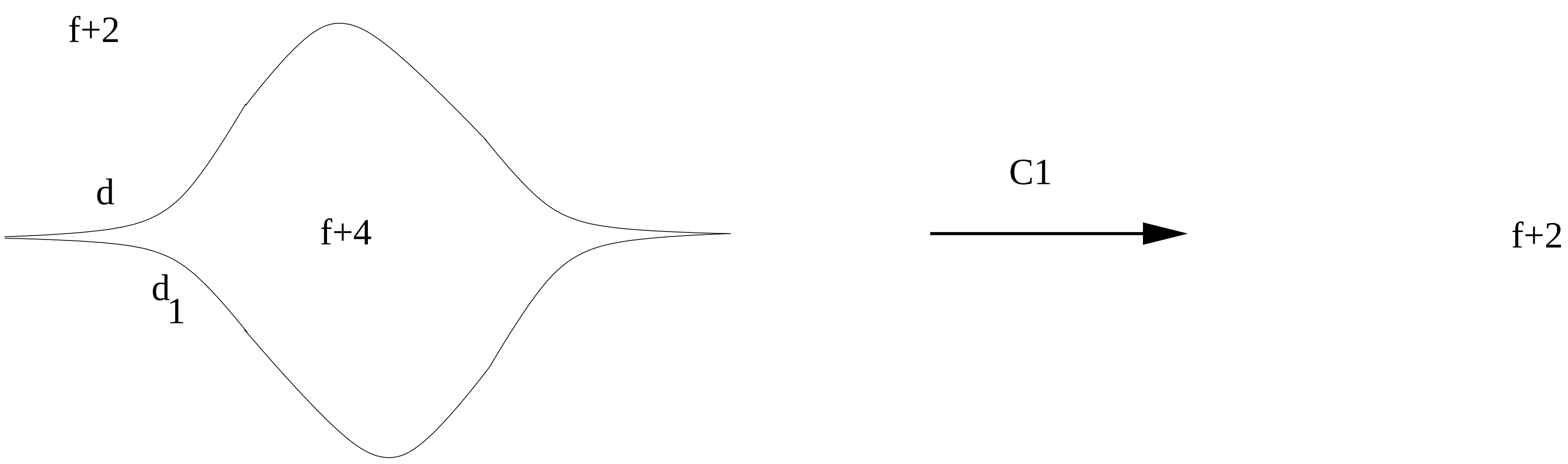
More rules...



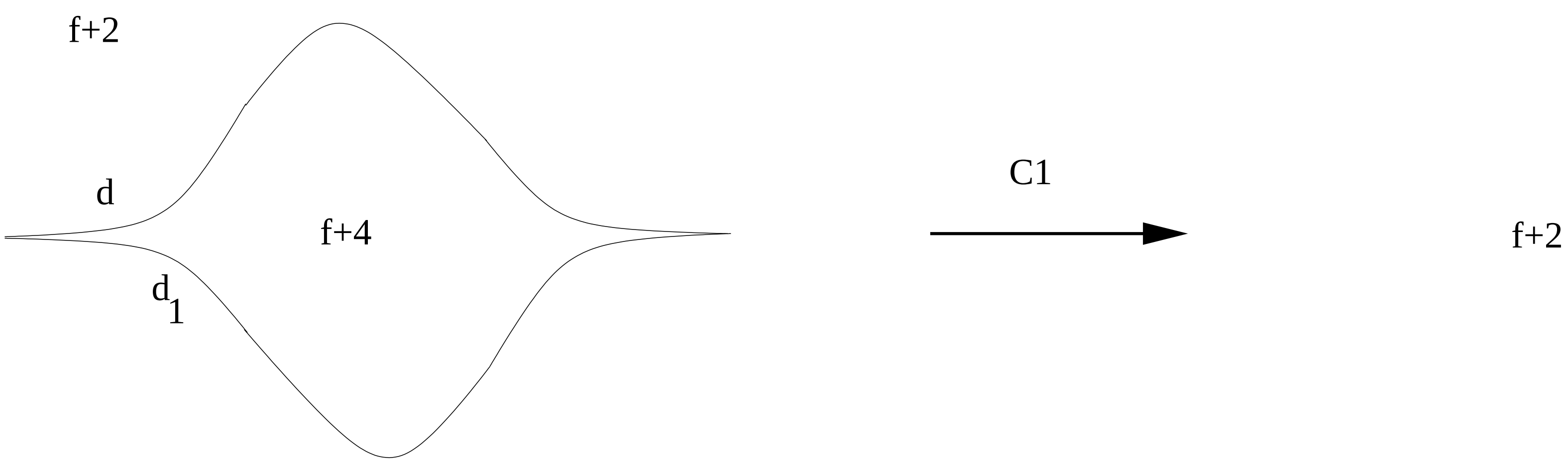
More rules...

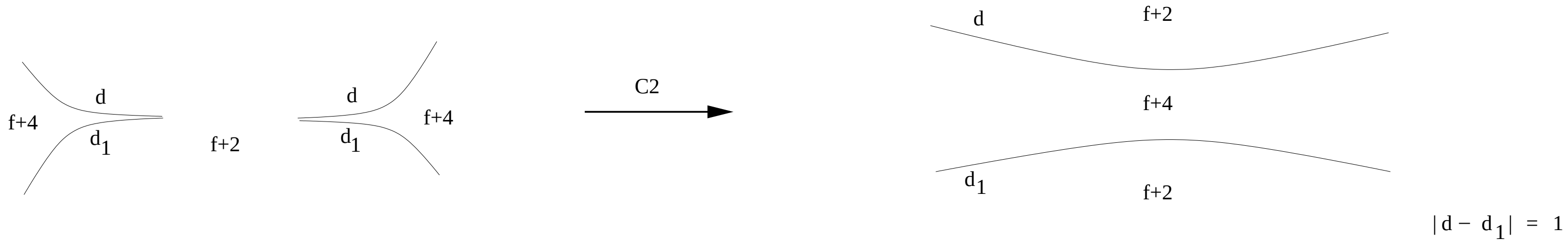

f+2
d1+1
d 1
CN1
f+2
f+4
f
f+2
f
d1
d1+2
d 1
f+2
q
d
f
f
q
d
d+2
d+2
s
s
p
d
f+4
f+4
CN2
d
d
r
p
r
d+1
d+1
f+2
f+2

More rules... (2)

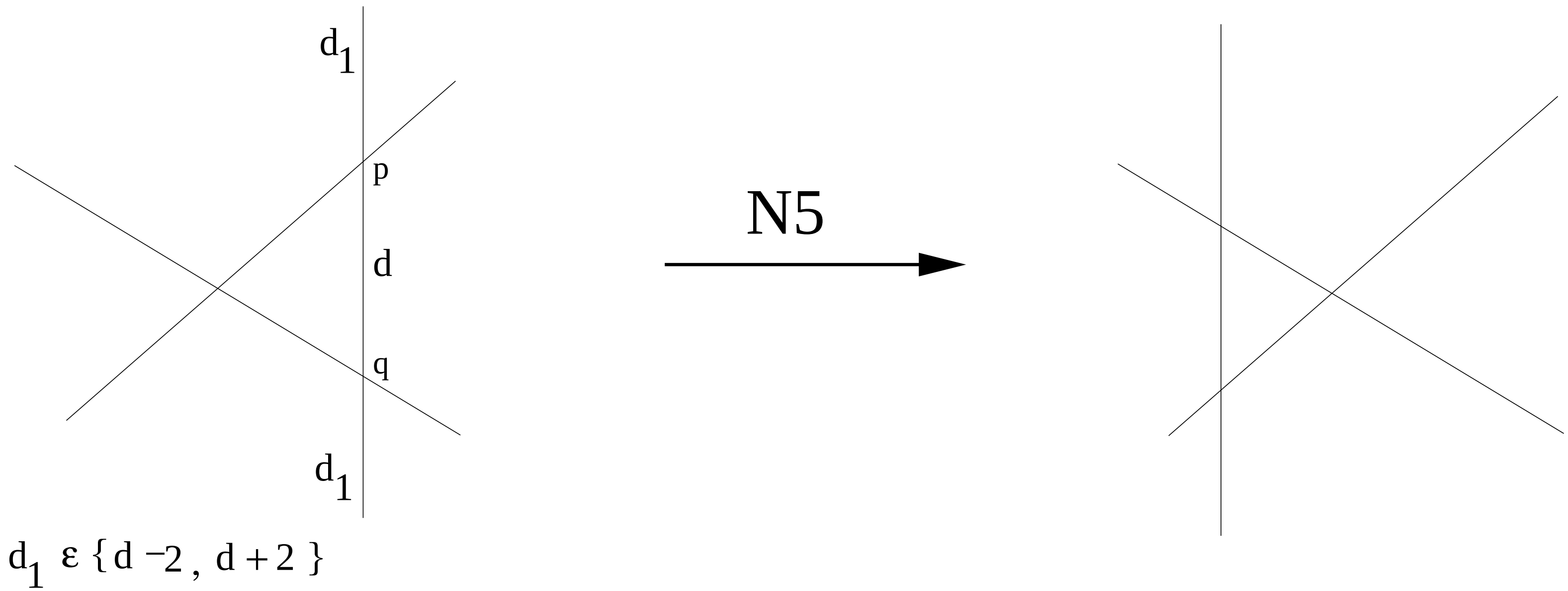


More rules... (2)

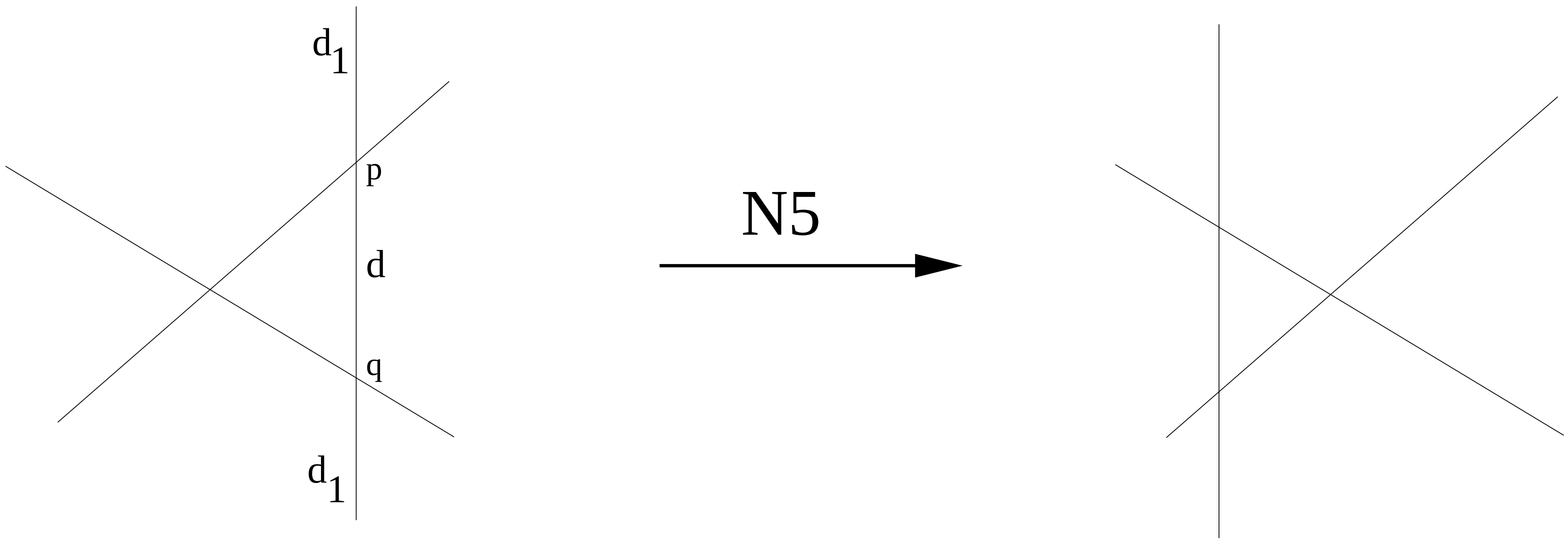




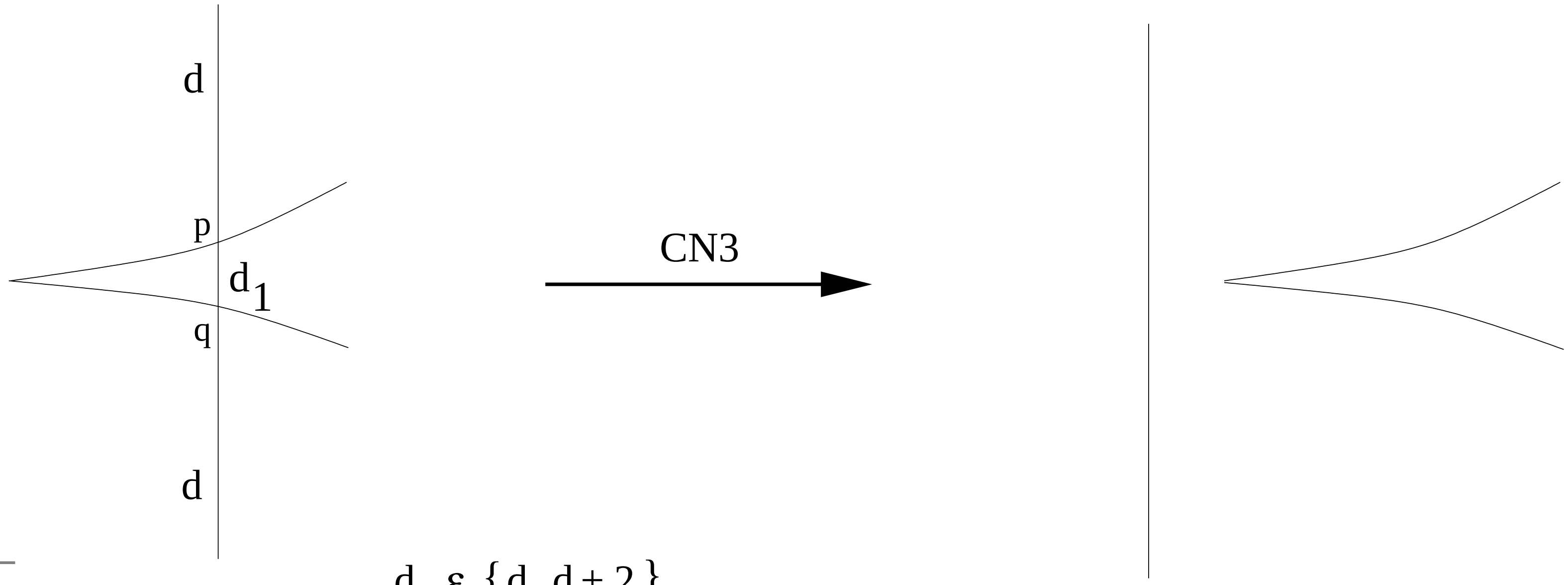
More rules... (3)



More rules... (3)



$d_1 \in \{d-2, d+2\}$



$d_1 \in \{d, d+2\}$

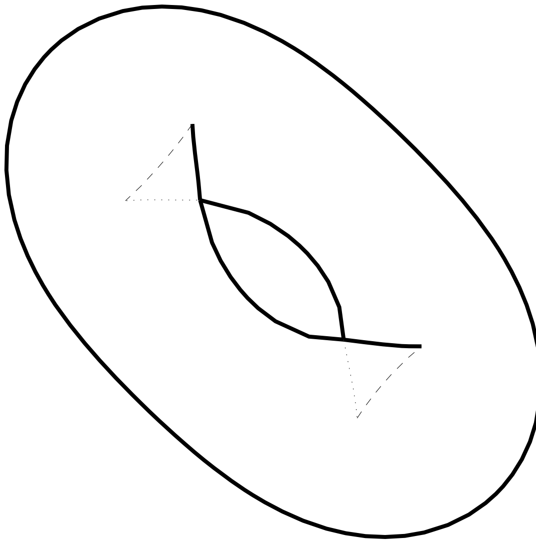
The software code

Let’s forget math for a moment... these rules seem “mechanical", can we develop a software program capable of authomatically applying them?
Software project appcontour, hosted on SourceForge.

- How to feed contours to the software
- Internal representation of contours
- Manipulating contours
- Presentation of the results

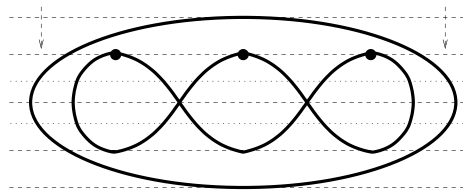
Region description

```
sketch {
Arc 1: (0);
Arc 2: [0 1 2);
Arc 3: [2 1 0);
Arc 4: [0];
Arc 5: [0];
Region 0 (f = 0): () (-a1);
Region 1 (f = 4): (+a2);
Region 2 (f = 4): (+a3);
Region 3 (f = 0): (-a4 -a5);
Region 4 (f = 2): (+a1) (-a2 +a4 -a3 +a5);
}
```



Morse description

```
morse {
                ^ l,0              ;
   /   ^        ^ r,0   ^      \   ;
  /  /   \   /   \   /  |       \  ;
 |  (d0++  X       X    |u2--    );
  \  \   /   \   /   \  |       /  ;
   \   U        U l,0   U      /   ;
                U                  ;
}
```



The **morse line** traverses the domain and crosses the contour at some critical times generating “critical events”, indicated by the letters ‘ˆ’, ‘|\/)(’, ‘U’, ‘X’.

Dealing with the standard torus

- The main engine: `contour`

Typical usage: `contour` *action*; it reads a description from *standard input* and outputs the result on *standard output*.

The command `contour testallrules torus.morse` [show] searches for all rules that apply on this contour:

```
$ contour testallrules torus.morse
Rules that apply:
N4 C2 C2:2 CN1 CN1:2 CN2L CN2R CN2LB CN2RB
```

Since there are two *sparrowtails* we apply rule **CN1** (or **CN1:2**):

```
$ contour applyrule cn1 torus.morse |
     contour printmorse | showcontour
```

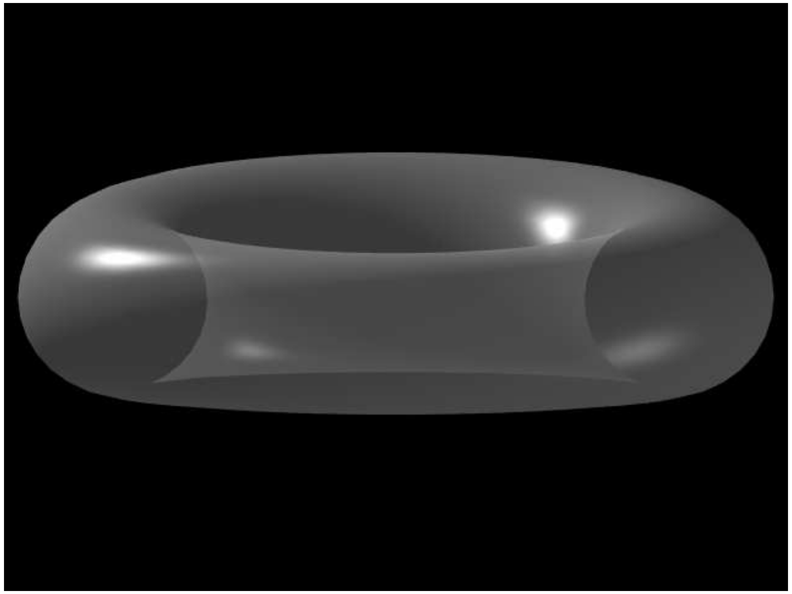
`contour_interact.sh`

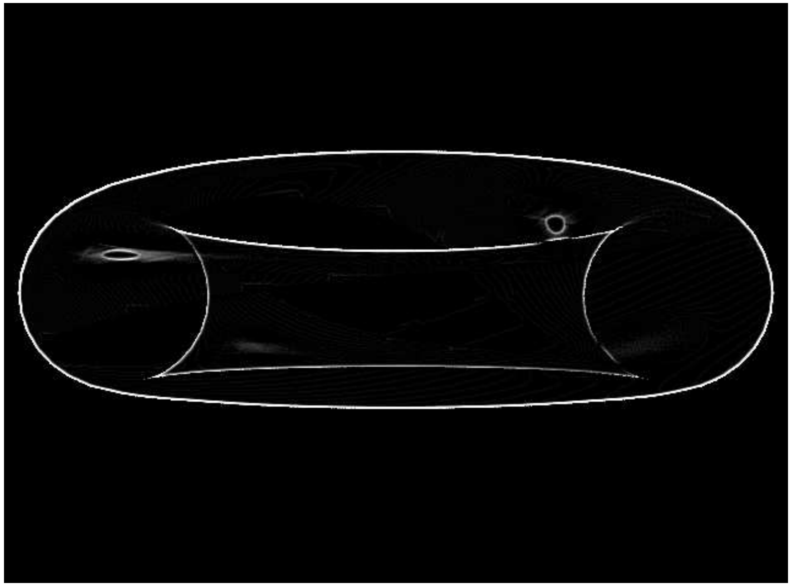
- Easier interaction with the engine
- Graphical presentation of the result (`showcontour`)
- Usage:

```
$ contour_interact.sh torus.morse
```

- Aim of `showcontour`: Produce a *nice* graphical representation of a contour starting from a *region* (*morse*) description.
- Future: produce a corresponding 3D surface

Second example: torus with bottleneck





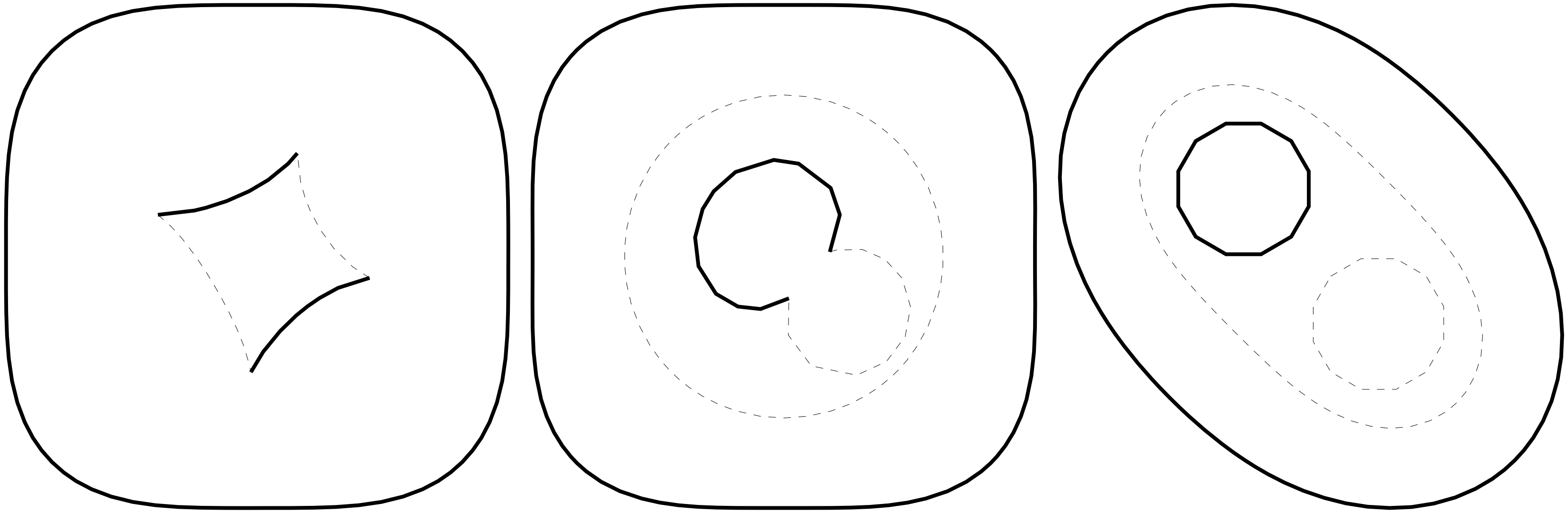
[animation ./anim2]

We apply rule "C2" twice... What do we obtain?

```
contour_interact.sh torus2.morse
```

Ask for the Euler characteristic. [Contour> info]

Is this a torus?



Can you imagine what is the shape of the surfaces? [animation: ./anim3]

Knot simulation with thin tubes

Knot description

These files describe two simple knots:

```
knot {
  ^ ^ ;
 ( x );
  X X ;
 ( U );
  \ / ;
   U  ;
}
```

```
knot {
  ^ ^ ;
 ( X );
  X X ;
 ( U );
  \ / ;
   U  ;
}
```

Try to unlink with

```
$ contour_interact.sh simpleknot.knot
$ contour_interact.sh simpleknotfake.knot
```

`transform.sh` utility

There is an automated tool that applies all possible chains of rules to a given contour.
This process always terminates because... although it usually produces a very big number of equivalent contours.
There are 128 different contour obtainable from "torus.morse"!

```
./transform.sh torus.morse
```

[try it]

Conclusions

- Make things rigorous!
- Does the region description uniquely identify apparent contours (up to isotopies)?
- And the morse description?
- An *elementary rule* really corresponds to some surface isotopy?
- **Can any surface isotopy be split into sequences of elementary rules?** (completeness)