MFLua

- MFLua is a new implementation of METAFONT which embeds a Lua interpreter
- it's completely compatible with METAFONT: a METAFONT source can be used with MFLua without any modification
- a MFLua source can be used with METAFONT without any modification

So. . .where is the news ?

- METAFONT uses cubic Bézier curve
- METAFONT has a bitmap model for a glyph (the *edges structure*)
- Lua is used to collect and manage informations of curves and pixel:
- we can compute the outlines of a glyph without using a tracing program (`Potrace`, `Autotrace`)

- Embedding a Lua interpreter in METAFONT is not a new idea: LuaTEX did the same
- LuaTEX uses *callbacks*, MFLua uses *sensors*
- A sensor cannot modify the state: it is read-only
- basically, it's equivalent to postprocessing the log

- A quick way to place a sensor is modify `mf.web` by adding the procedure and/or the function needed, and eventually register it in `texmf.defines`
- write the relative `C` function in `mflua.h` and `mflua.c`
- eventually implement the sensor in Lua

Lua sensors

```
@p begin @!{|start_here|}
mflua_begin_program;
:
initialize; {set global variables to their starting
values}
:
ready_already:=314159;
mfluaPRE_start_of_MF;
start_of_MF: @<Initialize the output routines@>;
:
mflua_initialize;
if start_sym>0 then {insert the `\&{everyjob}' symbol}
  begin cur_sym:=start_sym; back_input;
  end;
mfluaPRE_main_control;
main_control; {come to life}
mfluaPOST_main_control;
final_cleanup; {prepare for death}
mfluaPOST_final_cleanup;
end_of_MF: close_files_and_terminate;
final_end: ready_already:=0;
end.
```

Example: mflua_begin_program

In mflua.h:

```
extern int mfluabeginprogram();
```

In mflua.c:

```
lua_State *Luas[];
int mfluabeginprogram()
{
  lua_State *L = luaL_newstate();
  luaL_openlibs(L);
  Luas[0] = L;
/* execute Lua external "begin_program.lua" */
  const char* file = "begin_program.lua";
  int res = luaL_loadfile(L, file);
  if ( res==0 ) {
      res = lua_pcall(L, 0, 0, 0);
    }
  priv_lua_reporterrors(L, res);
  return 0;
}
```

The script
`begin_program.lua`
is quite simple, just the “greetings” message

```
print(”··· mflua_begin_program says: ’Hello world!’ ···”)
```

but usually the functions are more complexes, as in
`PRE_fill_envelope_rhs(rhs)`

Lua sensors

```
function PRE_fill_envelope_rhs(rhs)
   print("PRE_fill_envelope_rhs")
   local knots ,knots_list
   local index,char
   local chartable = mflua.chartable
   knots = _print_spec(rhs)
   index = (0+print_int(LUAGLOBALGET_char_code()))
          +(0+print_int(LUAGLOBALGET_char_ext()))*256
   char = chartable[index] or {}
   knots_list = char['knots'] or {}
   knots_list[#knots_list+1] = knots
   char['knots'] = knots_list
   chartable[index] = char
   return 0
end
```

(it stores the knots of an envelope)

Lua sensors

Within a sensor we may need to read the state of METAFONT or executes some METAFONT procedures or functions. Of course we must be careful not to change the data.

- It's possible to use WEB2C to export a PascalWEB symbol (macro, procedure, function, variable etc.) to Lua: for example the "WEB2C" code for info field

```
/* @d info(#) == mem[#].hh.lh */
/*{the |info| field of a memory word} */
static int priv_mfweb_info(lua_State *L)
{
  halfword p,q;
  p = (halfword) lua_tonumber(L,1);
  q = mem [p ].hhfield.v.LH ;
  lua_pushnumber(L,q);
  return 1;
}
```

becomes available to Lua as `info` after initialisation:

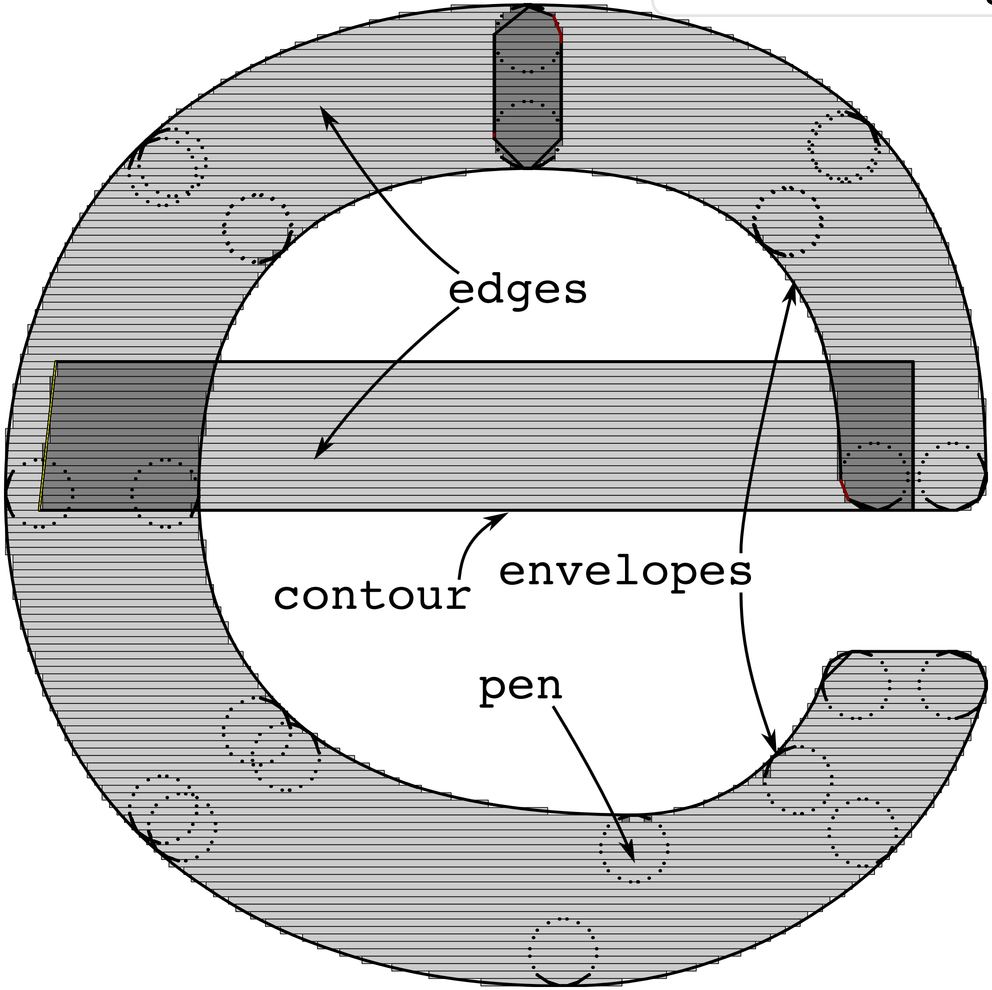
```
int mfluainitialize()
{
  lua_State *L = Luas[0];
  /* register lua functions */
  :
  lua_pushcfunction(L, priv_mfweb_info);
  lua_setglobal(L, "info");
  :
  return 0;
}
```

- It's also possible to translate a `PascalWEB` procedure directly to Lua because they are not so different. This is easy only if the procedure or function involves primitive types (integers and strings).
- the goal is minimize the numbers of sensors, not to misure every part of METAFONT

To properly draw the outline of a glyph we need the following information:

- the edge structures, i.e. the pixels of the picture
- the paths from the filling of a contour
- the paths from the drawing of an envelope with a pen
- the pen used in drawing an envelope.

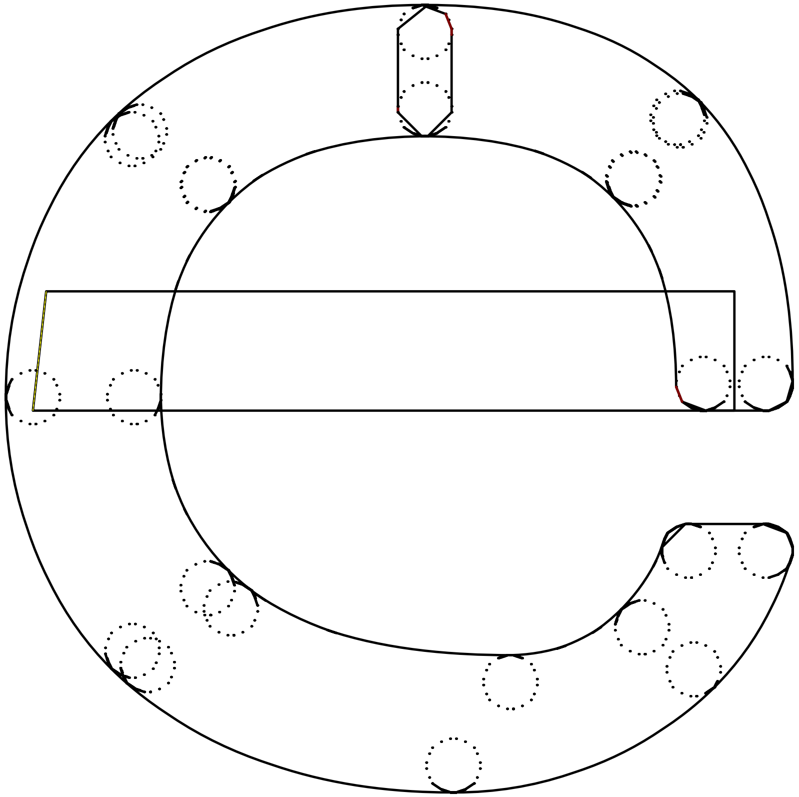
Collecting data



The place where to put a sensor is given initially by trial and error. It's mandatory to have the METAFONT*: The program* and the METAFONT*book* at hands, but after a while the number of the sensor has stabilised at around twenty sensors (more or less).

- For each character the sensors collect the data and fill the `char['edges']`, `char['contour']` and `char['envelope']` tables
- the sensor `mflua_end_program` (that is just before the end of METAFONT) processes the tables and store the result into `envelope.tex` as METAPOST paths

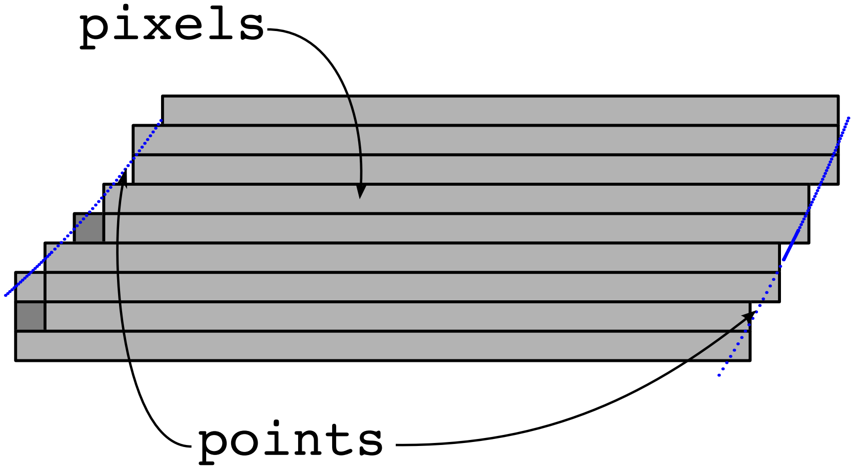
When METAFONT ends we have these data (edges omitted):



To remove all the unnecessary paths we follow 3 steps:

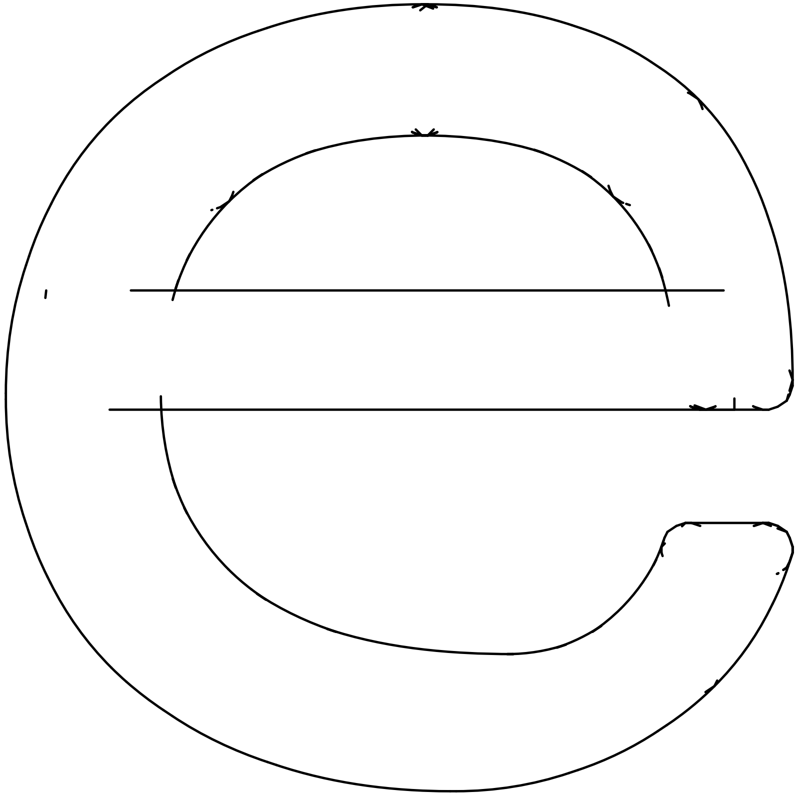
- Preparation
- Compute the intersections
- Remove unwanted paths

Preparation: for each curve we check, with the de Casteljau algorithm, if a point is *internal* or not.



It’s easy to implement, but a Bézier curve is not linear (maybe implicitization ?)

We then remove all the subpath made with internal points only — always with de Casteljau algorithm.



We calculate the intersections with a trick.

For each pair of paths p_1 and p_2 `end_program()` appends a METAFONT snippet to `intersect.mf` and then executes MFLua on it; the log `intersect.log` is then parsed to extract the intersections.

This is a typical METAFONT snippet:

```
batchmode;
message "BEGIN i=2,j=1";
path p[];
p1:=(133.22758,62) ..
   controls (133.22758,62.6250003125)
    and (133.22758,63.250000800781) ..
     (133.22758,63.875001431885);
p2:=(28.4097126027З,62) ..
  controls (63.349007932129,62)
   and (98.28829,62) ..
    (133.22758,62);
numeric t,u; (t,u) = p1 intersectiontimes p2;
show t,u;
message "" ;
```

and this is a fragment of `intersec.log`:

```
This is METAFONT, Version 2.718281
(Web2C 7.5.7) (base=mf 2011.1.16)
10 APR 2011 08:35
**intersec.mf
(intersec.mf
BEGIN i=2,j=1
>> 0
>> 1
```

It’s simple and fast — fast enough that is not necessary to reimplement the intersection algorithm in Lua

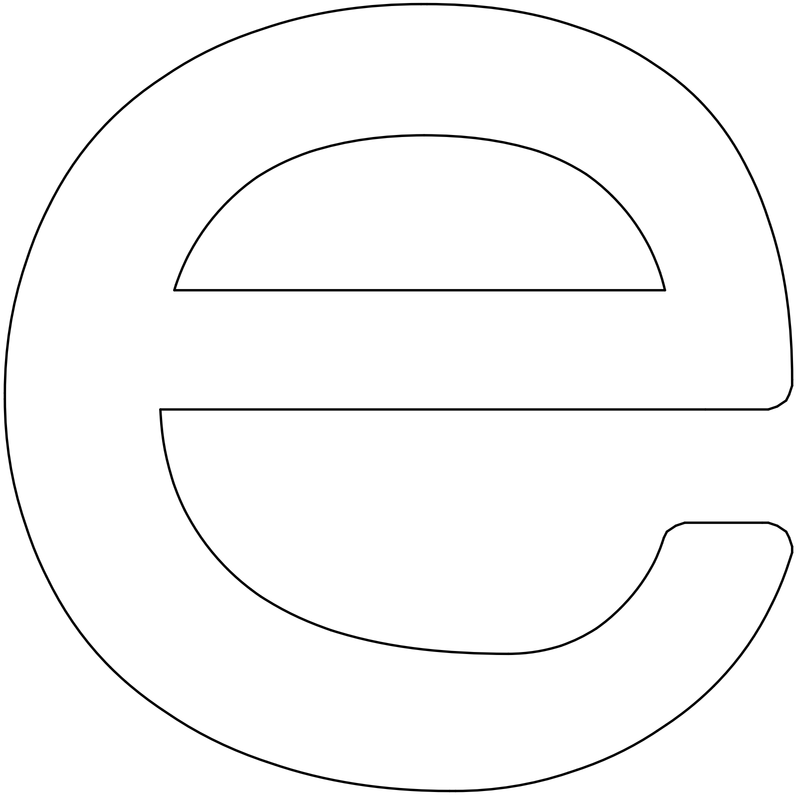
The last step is the more euristic one. The strategy is to gradually clean up the outlines by identifying a pattern and implementing a filter for it with a Lua function.

```
--
-- remove isolate paths
--
valid_curves, matrix_inters =
  _remove_isolate_path(valid_curves,
                       matrix_inters)
--
-- remove duplicate paths
--
valid_curves, matrix_inters =
  _remove_duplicate_path_I(valid_curves,
                           matrix_inters)
```

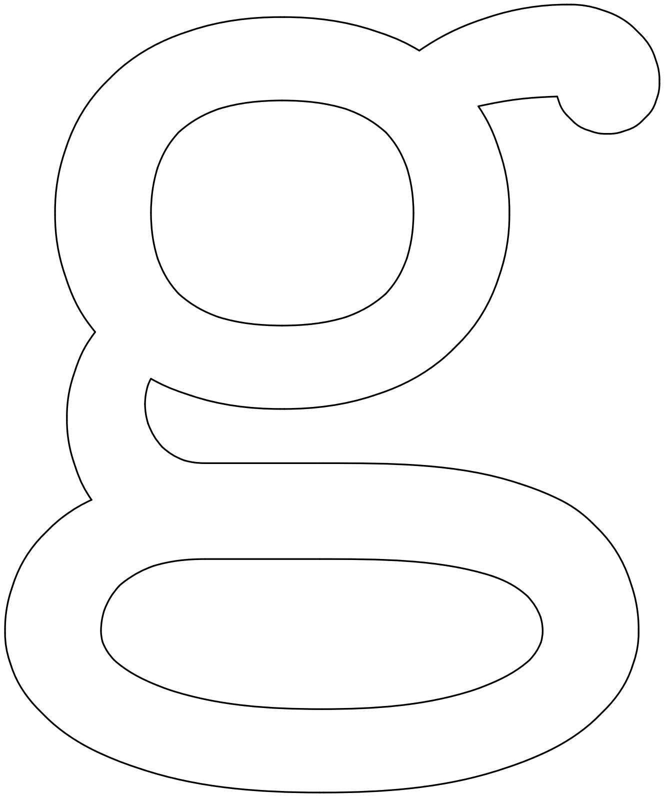
Remove paths

There are about twenty rules: some are very specific for a char, and maybe some are partially redundant. These rules are (almost) valid for the lower case letter of Concrete Roman 5 point.

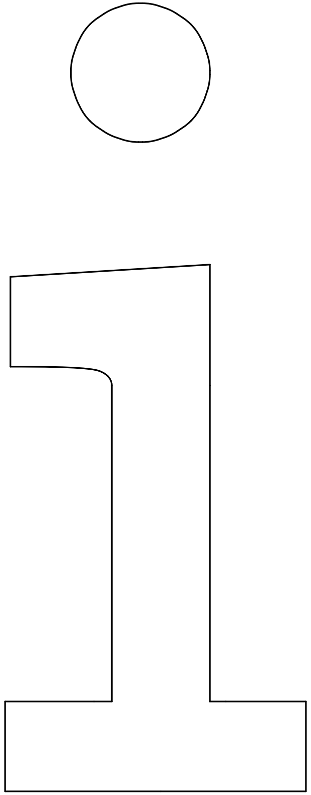
Remove paths



Remove paths



Remove paths



MFLua is still a proof-of-concept: there are too much details to check and fix before to start to consider as a tool to produce vectorial fonts. For example, it should be built at least for MicroSoft Windows; the sensors should be located on a change file `mflua.ch` and not in `mf.web`; the rules should be general as much as possible.

But the choice of external Lua files is not bad: if it is too much difficult to find a general algorithm at least they can be used as auxiliaries files for a specific METAFONT source.

MFLua is at `https://github.com/luigiScarso/mflua`

That’s all

Thank you !