MetaPost
path resolution isolated

EuroTEX 2012 / 6CM

Breskens, 9 October 2012

Metapost path solving

- Metapost is pretty good at finding pleasing control points for paths.
- Besides drawing on a picture, it can also display the found path in the log file.

Metapost path input

```
tracingchoices := 1;
path p;
p := (0,0) ..(10,10) ..(10,-5) ..cycle;
```

```
p := (0,0)..(2,20){curl1}..{curl1}(10, 5)..controls (2,2) and (9,4.5)..
  (3,10)..tension 3 and atleast 4 .. (1, 14){2,0} .. {0,1}(5,-4) ;

end.
```

Metapost log output

Path at line 5, before choices:
(0,0)
 ..(10,10)
 ..(10,-5)
 ..cycle

Path at line 5, after choices:
(0,0)..controls (-1.8685,6.35925) and (4.02429,12.14362)
 ..(10,10)..controls (16.85191,7.54208) and (16.9642,-2.22969)
 ..(10,-5)..controls (5.87875,-6.6394) and (1.26079,-4.29094)
 ..cycle

Metapost log output (cont.)

Path at line 7, before choices:
(0,0){curl 1}
..{curl 1}(2,20){curl 1}
..{curl 1}(10,5)..controls (2,2) and (9,4.5)
..(3,10)..tension 3 and atleast4.5
..{4096,0}(1,14){4096,0}
..{0,4096}(5,-4)

Path at line 7, after choices:
(0,0)..controls (0.66667,6.66667) and (1.33333,13.33333)
..(2,20)..controls (4.66667,15) and (7.33333,10)
..(10,5)..controls (2,2) and (9,4.5)
..(3,10)..controls (2.34547,10.59998) and (0.48712,14)
..(1,14)..controls (13.40117,14) and (5,-35.58354)
..(5,-4)

But what if ...

you want that functionality in a C program?

- You have to include MPLib in your program;
- create a Metapost language input string;
- execute it;
- and parse the log result.

Not very appealing!

Much better would be ...

- If you could include MPLib in your program;
- create a path programmatically;
- and then run the Metapost path solver directly;
- automatically updating the original path.

And that is what the new Metapost will allow!

Input source code

```
#include "mplib.h"

int main (int argc, char ** argv)  {
        MP mp ;
        MP_options * opt = mp_options () ;
        opt -> command_line = NULL;
        opt -> noninteractive = 1 ;
        mp = mp_initialize ( opt ) ;
        my_try_path(mp);
        mp_finish ( mp ) ;
        free(opt);
        return 0;
}
```

Input source code

```
void my_try_path(MP mp) {
      mp_knot first, p, q;
      first = p = mp_append_knot(mp,NULL,0,0);
      q = mp_append_knot(mp,p,10,10);
      p = mp_append_knot(mp,q,10,-5);
      mp_close_path_cycle(mp, p, first);
      if (mp_solve_path(mp, first)) {
         mp_dump_solved_path(mp, first);
      }
      mp_free_path(mp, first);
}
```

Used library functions

- `mp_append_knot()` creates and appends a new knot in the path.
- `mp_close_path_cycle()` mimics `cycle` in the Metapost language.
- `mp_solve_path()` finds the control points of the path.
- `mp_dump_solved_path()` *see next slide . . .*
- `mp_free_path()` releases the used memory.

mp_dump_solved_path

```
#define SHOW(a,b) mp_number_as_double(mp,mp_knot_##b(mp,a))
void mp_dump_solved_path (MP mp, mp_knot h) {
    mp_knot p, q;
    p = h;
    do {
        q = mp_knot_next(mp,p);
        printf ("(%g,%g)..controls (%g,%g) and (%g,%g)",
                SHOW(p,x_coord), SHOW(p,y_coord), SHOW(p,right_x),
                SHOW(p,right_y), SHOW(q,left_x), SHOW(q,left_y));
        p = q;
        if (p != h || mp_knot_left_type(mp,h) != mp_endpoint)
            printf ("\n ..");
    } while (p != h);
    if (mp_knot_left_type(mp,h) != mp_endpoint)
        printf("cycle");
    printf ("\n");
}
```

Used library functions

- `mp_knot_next()` go to the next knot in the path
- `mp_knot_x_coord()`, `mp_knot_y_coord()`, `mp_knot_right_x()`, `mp_knot_right_y()`, `mp_knot_left_x()`, `mp_knot_left_y()` return the value of a knot field, as a `mp_number` object
- `mp_knot_left_type()` returns the type of a knot, normally either `mp_endpoint` or `mp_open`.
- `mp_number_as_double()` converts a `mp_number` to `double`.

Program output

(0,0)..controls (-1.8685,6.35925) and (4.02429,12.1436)
..(10,10)..controls (16.8519,7.54208) and (16.9642,-2.22969)
..(10,-5)..controls (5.87875,-6.6394) and (1.26079,-4.29094)
..cycle

. . . almost exactly the same as in the log file:

(0,0)..controls (-1.8685,6.35925) and (4.02429,12.14362)
..(10,10)..controls (16.85191,7.54208) and (16.9642,-2.22969)
..(10,-5)..controls (5.87875,-6.6394) and (1.26079,-4.29094)
..cycle

More complex paths

Look for example like this:

```
first = p = mp_append_knot(mp,NULL,0,0);
q = mp_append_knot(mp,p,2,20);
p = mp_append_knot(mp,q,10,5);
if (!mp_set_knotpair_curls(mp, q,p, 1.0, 1.0)) exit ( EXIT_FAILURE ) ;
q = mp_append_knot(mp,p,3,10);
if (!mp_set_knotpair_controls(mp, p,q, 2.0, 2.0, 9.0, 4.5))
    exit ( EXIT_FAILURE ) ;
p = mp_append_knot(mp,q,1,14);
if (!mp_set_knotpair_tensions(mp,q,p, 3.0, -4.0)) exit ( EXIT_FAILURE ) ;
q = mp_append_knot(mp,p,5,-4);
if (!mp_set_knotpair_directions(mp, p,q, 2.0, 0.0, 0.0, 1.0))
   exit ( EXIT_FAILURE ) ;
mp_close_path(mp, q, first);
```

Important to remember

- Metaposts rules still apply, for example, tensions should be more than 0.75.
- All paths have to be closed with either `mp_close_path_cycle` or `mp_close_path`, because Metaposts internals are always cyclic pointer lists.
- Do not forget to check for memory allocation errors.
- Elaborate documentation is in `mplibapi.tex` in the Metapost distribution.
- Lua bindings for LuaTEX are coming.