MetaPost 1.750: Numerical engines

Taco Hoekwater

After two years of talks about future plans for MetaPost 2.0, there is finally real progress being made. This paper will introduce a pre-release of MetaPost 2 that can optionally use IEEE floating points for its internal calculations instead of the traditional 32-bit integers.

1. Introduction

I am sure some of you are curious to know why it is taking so long before MetaPost 2 comes out, considering that I have been giving talks on the subject for multiple years now. Just a quick recap to get you started, from the initial project proposal dating back to May 2009:

> In the original MetaPost library proposal we wrote in May 2007, one of the big user-side problem points that was mentioned was this:
>
> - All number handling is based on fractions of a 32-bit integer. User input often hits one of the many boundaries that are a result of that. For instance, all numbers must be smaller than 16384, and there is a noticeable lack of precision in the intersection point calculations.
>
> The current proposal aims to resolve that issue once and for all. The goal is to replace the MetaPost internal 32-bit numeric values with something more useful and, to achieve that goal, the plan is to incorporate one of these libraries:
>
> GNU MPFR library `http://www.mpfr.org/`
> IBM decNumber `http://www.alphaworks.ibm.com/tech/decnumber`
>
> We have not decided yet which one. MPFR will likely be faster and has a larger development base, but decNumber is more interesting from a user interface point of view because decimal calculus is generally more intuitive. For both libraries the same internal steps need to be taken, so that decision can be safely postponed until a little later in the project. The final decision will be based on a discussion to be held on the MetaPost mailing list.

Since then, there has been a small change to that statement: MetaPost 2 will in fact contain four different calculus engines at the same time:

- scaled 32-bit (a.k.a. compatibility mode)
- IEEE floating point (a.k.a. double)
- MPFR (arbitrary precision, binary)
- decNumber (arbitrary precision, decimal)

The internal structure of the program will also allow further engines to be added in the future.
The traditional scaled 32-bit engine is the default, thus maintaining backward compatibility with older versions of MetaPost. The other engines will be selected using a command line switch.
Working backwards from that final goal, some sub-projects could be formulated.

- Because values in any numerical calculation library are always expressed as C pointers, it is necessary to move away from the current array-based memory structure with overloaded members to a system using dynamic allocation (using `malloc()`) and named structure components everywhere, so that all internal MetaPost values can be expressed as C pointers internally.
  As a bonus, this removes the last bits of static allocation code from MetaPost so that it will finally be able to use all of the available RAM.

This first sub-project was a major undertaking in itself, and was finally completed when MetaPost 1.5 was released in July 2010.
The current 1.750 release of MetaPost implements most of the following two other sub-project goals (in fact so far only the PostScript backend has been updated):

- An internal application programming interface layer will need to be added for all the internal calculation functions and the numeric parsing and serialization routines. All such functions will have to be stored in an array of function pointers, thus allowing a start-up switch between 32-bit backward-compatible calculation and the arbitrary precision library.
  As a bonus, this will make it possible to add even more additional numerical engines in the future.
- The SVG and PostScript back-ends need to be updated to use double precision float values for exported points instead of the current 32-bit scaled integers.
  In the picture export API, doubles are considered to be the best common denominator because there is an acceptable range and precision and they are simple to manipulate in C code. This way the actual SVG and PostScript backend implementations and the Lua bindings can remain small and simple.

So, not accounting for hunting for bugs and fixing documentation, there is only one large step that remains to be taken before MetaPost 2 can be released, namely the actual integration of the two arbitrary precision libraries. That is why the version is set at 1.750 at the moment.

2. Some internal stuff

One thing that is not immediately obvious from the project goals as written above is that moving all the core arithmetic operations into functions that must be swappable instead of resolved at executable compilation time meant a whole lot of editing work,

and almost none of that could automated. This is the main reason why everything took so long. Let me illustrate that with an example.

2.1 An example: a simple procedure

Let's look at the `trans` procedure, that applies a transform to a pair of coordinates. It calculates the following formula:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} txx & tyx \\ txy & tyy \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} tx \\ ty \end{pmatrix}$$

First, here is the original Pascal implementation of that function:

```
procedure trans(p,q:pointer);
var v:scaled; {the new |x| value}
begin
  v:=take_scaled(mem[p].sc,txx)+take_scaled(mem[q].sc,txy)+tx;
  mem[q].sc:=take_scaled(mem[p].sc,tyx)+take_scaled(mem[q].sc,tyy)+ty;
  mem[p].sc:=v;
end;
```

The meaning of all those variables:

p,q	The variables for the x and y coordinates that have to be transformed
txx, txy, tyx, tyy, tx, ty	The six components of the transformation matrix, stored in global variables
v	An intermediate value that is needed because p cannot be updated immediately: its old value is used in the calculation of the new q
mem[]	The statically allocated memory table where Pascal MetaPost stored all its variables
mem[].sc	The structure object that holds the `scaled` value of a variable
take_scaled(a,b)	This function calculates $p = \lfloor (a \cdot b)/2^{16} + \frac{1}{2} \rfloor$

In the conversion of MetaPost from Pascal WEB to C WEB (in version 1.2), not all that much has changed:

```
static void mp_trans (MP mp,pointer p, pointer q) {
  scaled v; /* the new |x| value */
  v=mp_take_scaled(mp, mp->mem[p].sc,mp->txx)+
    mp_take_scaled(mp, mp->mem[q].sc,mp->txy)+mp->tx;
  mp->mem[q].sc=mp_take_scaled(mp, mp->mem[p].sc,mp->tyx)+
    mp_take_scaled(mp, mp->mem[q].sc,mp->tyy)+mp->ty;
  mp->mem[p].sc=v;
}
```

The only big difference here is the use of a global mp object instead of global variables. MetaPost 1.5 uses dynamic allocation instead of the mem array, and that makes the function a lot easier to understand.

```
static void mp_trans (MP mp, scaled * p, scaled * q) {
  scaled v;     /* the new |x| value */
  v = mp_take_scaled (mp, *p, mp->txx) +
    mp_take_scaled (mp, *q, mp->txy) + mp->tx;
  *q = mp_take_scaled (mp, *p, mp->tyx) +
    mp_take_scaled (mp, *q, mp->tyy) + mp->ty;
  *p = v;
}
```

It would be great if that could stay, but unfortunately, when numerical variables become objects, it is no longer possible to use simple + for addition. In turn, that means that more local variables are needed to store intermediate results. To make matters even worse, these local variables have to be allocated and released.
The end result is that the same function looks like this in MetaPost 1.750:

```
static void mp_number_trans (MP mp, mp_number p, mp_number q) {
  mp_number pp, qq, r1, r2;
  new_number (pp); new_number (qq); new_number (r1); new_number (r2);
  take_scaled (r1, p, mp->txx); take_scaled (r2, q, mp->txy);
  number_add (r1, r2);
  set_number_from_addition(pp, r1, mp->tx);
  take_scaled (r1, p, mp->tyx); take_scaled (r2, q, mp->tyy);
  number_add (r1, r2);
  set_number_from_addition(qq, r1, mp->ty);
  number_clone(p,pp); number_clone(q,qq);
  free_number(pp); free_number(qq); free_number(r1); free_number(r2);
}
```

The variables r1, r2, pp and qq exist only for storing intermediate results. To be honest, qq is not really needed, but it adds a nice bit of symmetry and the overhead is negligible. The new arithmetic functions do not return a value since that would force the introduction of even more new_number | free_number calls. Instead, they adjust their first argument. Stripped down to only the actual actions, the function looks like this:

```
r1 = p * mp->txx; r2 = q * mp->txy;
r1 = r1 + r2; pp = r1 + mp->tx;
r1 = p * mp->tyx; r2 = q * mp->tyy;
r1 = r1 + r2; qq = r1 + mp->ty;
p = pp; q = qq;
```

Where the first two lines match the first statement in the previous versions of the function, the next two lines the second statement, and the last line does the final assignments.
In the listing above, all those identifiers like `new_number` and `take_scaled` are not really functions. Instead, they are C preprocessor macros with definitions like this:

```
#define take_scaled(R,A,B)  (mp->math->take_scaled)(mp,R,A,B)
```

Here the right-hand-side `take_scaled` is one of the function fields in the structure `mp->math`. Each of the arithmetic engines defines a few dozen such functions each for its own type of `mp_number`. With this new internal structure in place, adding a new arithmetic engine is not much more work than defining a few dozen – mostly very simple – functions.

3. Using 1.750

As said, there are currently only two engines: scaled 32-bit and IEEE double. Switching to IEEE double is done on the command-line by using

```
mpost --math=double mpman
```

3.1 Warning checks

In MetaPost 1, the parameter `warningcheck` can be set to a positive value. This will downgrade the limit on numerical ranges from 16384 to 4096, but it has the advantage that it guards against various internal cases of overflow.
With the `double` numerical engine, numerical values can range up to 1.0E+307. The warning check could be set at something like 2.5E+306, but that is actually not the most important point for a warning to take place.
Because of the way double values are stored internally in the hardware, it is possible to store a certain range of integers *exactly*. However, when an integer value gets above a threshold (it has to fit in 52bits), precision is lost. For this reason, `warningcheck` now kicks in a little below this limit, and that is currently set at 4.5E+15.

3.2 An example

```
beginfig(1);
warningcheck:=0;
path p;
p = fullcircle scaled 23.45678888E-200;
p := p scaled 1E201;
draw p;
currentpicture := currentpicture scaled .5;
endfig;
end.
```