METAPOST: terminally ill or just playing dead?

Taco Hoekwater
Dordrecht, The Netherlands
taco@elvenkind.com

Abstract

In recent years, there is evidence of a renewed interest in the use of METAPOST for various drawing tasks. Simultaneously, it seems that just about every METAPOST user runs into some kind of limitation that makes the use of METAPOST far from ideal for the proposed task.

The diagnosis we have to make is whether these symptoms indicate a fatal disease in the program, or if they are only idiosyncracies and scratches that can be cured with some therapy and a few band-aids.

1 Introduction

When a software program is not used very often, or when the users complain a lot, there are a few potential causes.

The first of those causes is lack of knowledge: if you do not know that a certain program exists nor what it does, you obviously do not know how to use it. Since the focus of this paper is to assess the qualities of metapost itself, I will not go deeper into ways to make metapost more known.

The second cause is lack of abilities: if you know that a program exists and what it does, but that list of features does not actually fit your needs, then you will not use the program. Requested features are the subject of the immediately following section.

The third cause is lack of quality: if you know that a program exists and what it supposedly does fits your needs, then you might still not want to use it, because it may not be of sufficient quality to perform the tasks you believe it should be able to do. Software quality is the subject of the second, fairly long section.

For completeness I want to mention a fourth cause: lack of willingness. You know that a program exists and what it does actually does fit you needs, but you are unwilling to use it. This can be a matter of taste or even threshold fever. Just like lack of knowledge, this is also outside of the scope of this examination. Metapost is the patient here, not the user.

2 Abilities

There is of course lots and lots of stuff that metapost cannot do. But lets start by stating that metapost is a programmable and therefore batch-oriented general-purpose drawing tool. That way, we can forget about interactive wireframe modelling and fractal visualization.

What follows is fairly small list of desires that are expressed regularly by (ex-)users of metapost. It lists most often heard requests only.

2.1 Dynamic data limits

In this day and age, it is weird that we use a program that still depends on static allocation of memory and therefore stops with an 'out of memory' error when only a fraction of the available memory is actually used. Ideally, all memory should be allocated dynamically, thereby making the best use of the hardware at hand.

Even more restrictive is that the maximum integer value is 16384, and the fractional precision is only six digits. In a drawing program, this is a severe limitation, especially when external data has to be plotted. Points in a data file can come in many units, most of which are not easily rearranged to fit in metapost's data model. While there are some workarounds avaiable as macro packages, this would be better solved in the actual program code.

2.2 3-D support

Quite possibly this is the most often asked for extension. But it is also by far the most vaguely formulated one. As of now, I am not aware of anybody who has written down a clear list of demands stating: 'this is what I need for the drawing I want to make, please do it like so'.

Instead it goes like: 'Metapost should support 3D, because that would make it a better program'. Until someone is found who knows how to formulate a functional design, it seems unlikely that we will see a metapost with extra 3d support.

2.3 Font creation

It is strange that this program that is a direct descendant of metafont is not capable of creating any kind of font directly. Not very valuable perhaps, but PostScript Type 3 should be easily doable, and PostScript Type 1 with fairly small amounts of work.

2.4 Font embedding

When fonts are used for labelling point in a figure, metaposts output is no longer self-contained: it relies on external postprocessing to provide the actual font files instead of embedding them in the output.

Current metapost also does not understand reencoding of fonts, nor the use of TeX's virtual font mechanism.

Font embeddding, subsetting en reencoding are fairly simple additions that will make it into the next release of te program. There are currently no plans for the supprot virtual fonts, as far as I know.

2.5 Erasing and transparancy

Metapost does not have an eraser operator like metafonts unfill. If you want something to have 'no color' than you can choose between not drawing on that area at all (usually possible but often very cumbersome) or drawing that area twice, using the color of the background in the second stage.

It would not be trivial to add this functionality, but it is a very much desired one. Related requests have been made for calculating the intersection, difference, merge etc. of two overlapping paths.

2.6 Output formats

Requests have been made to output other file formats instead of EPS. Most often requested are a bitmap format like PNG, and Scalable Vector Graphics. Especially that last format would improve the re-usability of Metapost graphics outside of the TEX environment.

Sometimes PDF is requested as well, but the conversion from EPS to PDF is a rather simple one that has been solved already outside of metapost, so there seems little point in putting much effort in that.

2.7 Alternative color models

Sometimes requests are made for alternate color models, esp. CMYk. This would aid the adoption of metapost in the traditional printing workflow.

2.8 Tagged paths

For postprocessing it would be handy to have 'special' information that tags along with a path, just like it's color and transformation matrix. Normally all specials end up at the top of the output file, and that makes it hard to target a specific part of a figure for postprocessing tricks.

3 Quality

In my search to come up with a list of possible maladies that metapost may be suffering from, I came upon a very handy page on Wikipedia: `http://en.wikipedia.org/wiki/Software_quality`. It lists a number of factors that determine the quality of a piece of software, along with a set of diagnostic questions that can be asked for each of the factors.

The italic text below is straight from that wikipedia page on the day I wrote this (april 7, 2006). The intermittent (upright) text is my assessment of the state of metapost.

A scheme which could be used for evaluating software quality factors is given below. For every characteristic, there are a set of questions which are relevant to that characteristic. Some type of scoring formula could be developed based on the answers to these questions, from which a measure of the characteristic may be obtained.

3.1 Understandability

Are variable names descriptive of the physical or functional property represented? Do uniquely recognisable functions contain adequate comments so that their purpose is clear? Are deviations from forward logical flow adequately commented? Are all elements of an array functionally related?

In general, I believe metapost scores fairly well in this regard, but it could be better. Most of the builtin (or predefined) variables and functions have acceptable names (except 'ditto' perhaps). It is easy to guess what the symbols for operators are (except '++', that stands for pythagorean addition, not the unary incrementation)

A little bit daunting for a first-time user may be the expression syntax. There are quite a lot of named operators, of which a number like 'intersectiontimes' and 'transformed' are infix forms where most people would have expected a function call instead. Also, flow control operations can occur in the middle of expressions. While this makes it harder for new users to understand 'wizard' code, it

is not an immediate problem: it is still possible to write expressions that are much more traditional.

A bigger problem are variables. In particular, the square brackets in 'a[]' do not stand for 'array contruction', but instead simply mean 'any numeric value following 'a". Variable names like 'a[1.5]' are perfectly acceptable metapost syntax. There is not even an implied array logic: 'a[4]' may be a numeric value, while simultaneously 'a[3]' may be a path or a boolean or even completely undefined.

Advanced macro use is often confusing and its definition syntax quite arcane. The situation is not unlike that in TeX, but metaposts macros are more versatile and therefore even trickier to comprehend fully. But, just like for expression syntax, write simple macros is indeed simple.

3.2 Completeness

Does the program contain all referenced subprograms not available in the usual systems library? Are all parameters required by the program available? Are all inputs required by the program available?

Technically, the font metrics to be used to typeset labels are missing from the metapost distribution, but normally metapost is shipped alongside TEX, with a wealth of usable fonts.

3.3 Conciseness

Is all code reachable? Is any code redundant? How many statements within loops could be placed outside the loop, thus reducing computation time? Are branch decisions too complex?

It is safe to say that metaposts actual source code is concise, almost in the extreme. In part this is thanks to the use of the web programming language that provides macros and reusable code modules. Another part is because it is already a rather old program and it reached the limits of the computers that were available at the time it was first developed.

3.4 Portability

Does the program depend upon system or library routines unique to a particular installation? Have machine-dependent statements been flagged and commented? Has dependency on internal bit representation of alphanumeric or special characters been avoided?

Metapost is very portable indeed, because it does almost all of the work one would expect to be done by the system libraries itself. There are only a few, really minor, bits that need changing for any new platform (not just the 'posix-like' small systems that are targeted by the GNU build tools, but even systems with different byte sizes or character sets).

3.5 Consistency

Is one variable name used to represent difficult physical entities in the program? Does the program contain only one representation for physical or mathematical constants? Are functionally similar arithmetic expressions similarly constructed? Is a consistent scheme for indentation used?

When a program is essentially created by one single person over a fairly short period of time, one expects it to be quite consistent internally. Metapost is no exception.

3.6 Maintainability

Has some memory capacity been reserved for future expansion? Is the design cohesive, i.e., each module has recognisable functionality? Does the software allow for a change in data structures (object-oriented designs are more likely to allow for this)? If a functionally-based design (rather than object-oriented), is a change likely to require restructuring the main-program, or just a module?

This is a definate weak spot. The functionality of the separate modules is pretty well contained, and most access to the data structures is through accessor macros, so that is OK.

But: a lot of the communication between the modules takes place through global variables, lots of flow control actively uses the fall-through / default branch, and the used data structures are stuffed full with information to the very last bit.

It is not at all easy to extend metapost.

3.7 Testability

Are complex structures employed in the code? Does the detailed design contain clear pseudo-code? Is the pseudo-code at a higher level of abstraction than the code? If tasking is used in concurrent designs, are schemes available for providing adequate test cases?

In a way, the actual code of metapost simutaneously *is* pseudo-code, an intended side-effect of the use of the web system of literate programming.

Also, we have a marvellous unit-test suite (trap test). Metapost (like its siblings metafont and tex) scores very well on this point.

3.8 Usability

Is a GUI used? Is there adequate on-line help? Is a user manual provided? Are meaningful error messages provided?

There is a user manual, but it is not as good as it could be. The manual is not very large and makes quite some references to the metafont book. While this makes sense from the standpoint of a documentation writer (more economical and less chance of errors), it is very unpractical for a user. Especially since the metafont book has to be bought and is therefore not readily available to everybody.

I personally believe the error messages are nice and easy to understand, but I have been told by other people that this is very much a matter of opinion. It may be that metapost sometimes provides too much information, instead of not enough.

3.9 Reliability

Are loop indexes range tested? Is input data checked for range errors? Is divide-by-zero avoided? Is exception handling provided?

The compiled source code of metapost has lots of tests to make sure that variables are within ranges etcetera. Even more tests are in place to make sure that user-supplied data cannot contain invalid values. Strict bounds checking is performed on every array.

About the only thing that metapost does not guard you against is the incorrect use of recursive macros. Runaways are not detected until it runs out of memory. But since it does all of its memory maintenance internally, that does not pose a threat to the rest of your system.

But metapost is unreliable in a completely different field: the possible input range for numerics is very limited, and the precision of its calculations is far from perfect. The hope is that this situation can be cured by applying a set of patches that improves the allowed range as well as precision.

3.10 Structuredness

Is a block-structured programming language used? Are modules limited in size? Have the rules for transfer of control between modules been established and followed?

The descriptive text parts of the web source normally set out to define the rules for control transfer and the programming interface for the module at hand. While those rules are not always adhered to rigidly, the exceptions are always documented in the text parts as well.

Metapost is quick a well-structured program, but it is so by choice, not by force.

3.11 Efficiency

Have functions been optimized for speed? Have repeatedly used blocks of code been formed into subroutines?

No problems here: the efficiency of the compiled metapost code is near optimal. Whether user-supplied macros are equally efficient is up to user, of course.

3.12 Security

Does the software protect itself and its data against unauthorized access and use? Does it allow its operator to enforce security policies? Are appropriate security mechanisms in place? Are those security mechanisms implemented correctly? Can the software withstand attacks that must be expected in its intended environment? Is the software free of errors that would make it possible to circumvent its security mechanisms? Does the architecture limit the impact of yet unknown errors?

Metapost is as secure as it should be for the tasks it has, I think. There is limited support in some environments to execute other system commands, but that ability can in all cases be turned off by the supervisor. Metapost has the ability to write and read files, but it is not normal for it to be started automatically by the operating system, so while the chance of malware being written for it is negligible.

4 Conclusion

The title of this paper raised the question 'Terminally ill or just playing dead?' Neither seems to be the case. Metapost does not appear to be doing all that bad after considation of all the points on the checklist regarding quality. But, on the other hand, there are some definate problems, and not all is cured by simple a kiss on the head.