minim-mp

version 2021/1.0

author Esger Renkema
contact minim@elrenkema.nl

This package offers low-level mplib integration for plain luatex. The use of multiple simultaneous metapost instances is supported, as well as running tex or lua code from within metapost. In order to use it, simply say `\input minim-mp.tex`.

After this, `\directmetapost [ options ] { mp code }` will result in a series of images corresponding to the `beginfig ... endfig` statements in your `mp code`. Every image will be in a box of its own.

Every call to `\directmetapost` opens and closes a separate metapost instance. If you want your second call to remember the first, you will have to define a persistent metapost instance. This will also give you more control over image extraction. See below under „Metapost instances”. The `options` will also be explained there (for simple cases, you will not need them).

The logging of the metapost run will be included in the regular log file. If an error occurs, the logging will also be shown on the terminal.

This package can also be used as a stand-alone metapost compiler. Saying

```
luatex  --fmt=minim-mp  your_file.mp
```

will create a pdf file of all images in `your_file.mp`, in order, with page sizes adjusted to image dimensions.

Metapost instances

For more complicated uses, you can define your own instances by saying `\newmetapostinstance [ options ] \id`. An instance can be closed with `\closemetapostinstance \id`. These are the options you can use:

Option	Default	Description
`jobname`	`':metapost:'`	Used in error messages.
`format`	`'plain.mp'`	Format to initialise the instance with.
`math`	`'scaled'`	One of `scaled`, `decimal` or `double`.
`seed`	`nil`	Random seed for this instance.
`catcodes`	`0`	Catcode table for `btex ... etex`.
`env`	copy of `_G`	Lua environment; see below.

Now that you have your own instance, you can run chunks of metapost code in it with `\runmetapost \id { code }`. Any images that your code may have contained will have to be extracted explicitly. This is possible in a number of ways, although each image can be retrieved only once.

`\getnextmpimage \id` – Writes the first unretrieved image to the current node list. There, the image will be contained in a single box node.

`\getnamedmpimage \id {name}` – Retrieves an image by name regardless of its position, and writes it to the current node list.

`\boxnextmpimage \id box-nr` – Puts the next unretrieved image in box `box-nr`. The number may be anything tex can parse as a number.

`\boxnamedmpimage \id box-nr {name}` – Puts the image named `name` in box `box-nr`.

Say `\remainingmpimages \id` for the number of images not yet retrieved. Finally, as a shorthand, `\runmetapostimage \id { code }` will add `beginfig ... endfig` to your `code` and write the resulting image immediately to the current list.

Running tex from within metapost

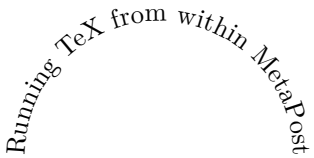
You can include tex snippets with either `maketext "tex text"` or `btex ... etex` statements. The tex code will be executed in the current environment without an extra grouping level. The result of either statement at the place where it is invoked is an image object of the proper dimensions that can be moved, scaled, rotated and mirrored. You can even specify a colour. Its contents, however, will only be added afterwards and are invisible to metapost.

Arbitrary tex statements may be included in `verbatimtex ... etex`, which may occur anywhere. These `btex` and `verbatimtex` statements are executed in the order they are given.

You can also use metapost's `infont` operator, which restricts the text to-be-typeset to a single font, but returns an `picture` containing a `picture` for each character. The right-hand argument of `infont` should either be a (numerical) font id or the (cs)name of a font.

One possible use of the `infont` operator is setting text along curves:

```
beginfig(1)
    save t, w, r, a; picture t;
    t = "Running TeX from within MetaPost" infont "tenrm";
    w = xpart lrcorner t = 3.141593 r;
    for c within t :
        x := xpart (llcorner c + lrcorner c)/2;
        a := 90 - 180 x/w;
        draw c rotatedaround((x,0), a)
               shifted (-r*sind(a)-x, r*cosd(a));
    endfor
endfig;
```



Running lua from within metapost

You can call out to lua with `runscript "lua code"`. For this purpose, each metapost instance carries around its own lua environment so that assignments you make are local to the instance. (You can of course order the global environment to be used by giving `env = _G` as option to `\newmetapostinstance`.)

If your lua snippet returns nothing, the `runscript` call will be invisible to metapost. If on the other hand it does return a value, that value will have to be translated to metapost. Numbers and strings will be returned as they are (so make sure the string is surrounded by quotes if you want to return a metapost string). You can return a point or colour by returning an array of two to four elements. For other return values, `tostring()` will be called.

Do keep in mind that metapost and lua represent numbers in different ways and that rounding errors may occur. For instance, metapost's `decimal epsilon` returns `0.00002`, which metapost understands as `1/65536`, but lua as `1/50000`. Use the metapost macro `hexadecimal` instead of `decimal` for passing unambiguous numbers to lua.

Additionally, you should be aware that metapost uses slightly bigger points than tex, so that `epsilon` when taken as a dimension is not quite equal to `1sp`. Use the metapost macro `scaledpoints` for passing to lua a metapost dimension as an integral number of scaled points.

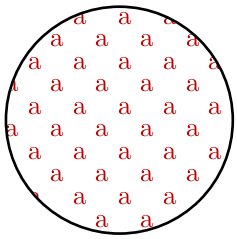
Tiling patterns

The `withpattern(<name>)` added to a `fill` statement will fill the path with a pattern instead of a solid colour. If the patterns contains no colour information of itself, it will have the colour given by `withcolor`. Stroking operations (the `draw` part) will not be affected. Patterns will always look the same, irrespective of any transformations you apply to the picture.

To define a pattern, sketch it between `beginpattern(<name>) ... endpattern(xstep, ystep);` where `<name>` is a suffix and `(xstep, ystep)` are the horizontal and vertical distances between applications of the pattern. Inside the definition, you can draw the pattern using whatever coordinates you like; assign a value to the `matrix` transformation to specify how the pattern should be projected onto the page. This `matrix` will also be applied to `xstep` and `ystep`.

You can also change the internal variable `tilingtype` and the normal variable `painttype`, although the latter will be set to 1 automatically if you use any colour inside the pattern definition. Consult the pdf specification for more information on these parameters.

You can use text inside patterns, as in this example:



```
% define the pattern
picture letter; letter = maketext("a");
beginpattern(a)
    draw letter rotated 45;
    matrix = identity rotated 45;
endpattern(12pt,12pt);
% use the pattern
beginfig(1)
    fill fullcircle scaled 3cm withpattern(a) withcolor 3/4red;
    draw fullcircle scaled 3cm withpen pencircle scaled 1;
endfig;
```

A small pattern library is available in the `minim-hatching.mp` file; see the accompanying documentation sheet for an overview of patterns. Tiling patterns cannot be used together with tikz/pgf; see below under 'Resource management'.

Other metapost extensions

There is currently no support for the `glyph of` operator.

You can set the baseline of an image with `baseline(p)`. There, `p` must either be a point through which the baseline should pass, or a number (where an x coordinate of zero will be added). Transformations will be taken into account, hence the specification of two coordinates. The last given baseline will be used.

Previously-defined box resources can be included with `boxresource(nr)`. The result will be an image object with the proper dimensions. This image can be transformed in any way you like, but you cannot inspect the contents of the resource within metapost.

You can write to tex's log directly with `texmessage "hello"`.

You can write direct pdf statements with `special "pdf: statements"` and you can add comments to the pdf file with `special "pdfcomment: comments"`. Say `special "latelua: lua code"` to insert a `late_lua` whatsit. All three specials can also be given as pre- or postscripts to another object. In that case, they will be added before or after the object they are attached to.

Lua interface

In what follows, you should assume `M` to be the result of

```
M = require('minim-mp')
```

as this package does not claim a table in the global environment for itself.

You can open a new instance with `nr = M.open {options}`. This returns an index in the `M.instances` table. Run code with `M.run (nr, code)` and close the instance with `M.close (nr)`. Images can be retrieved only with `box_node = M.get_image(nr, [name])`; omit the `name` to select the first image. Say `nr_remaining = M.left(nr)` for the number of remaining images.

Each metapost instance is a table containing the following entries:

`jobname`	The jobname.
`instance`	The primitive metapost instance.
`results`	A linked list of unretrieved images.
`status`	The last exit status (will never decrease).
`catcodes`	Number of the catcode table used with `btex ... etex`.
`env`	The lua environment for `runscript`.

PDF resource management

This package can add `/Pattern` and `/ColorSpace` entries to all page and xform resource dictionaries. Both refer to a single, global dictionary shared by all pages. Support for other keys may be added in the future.

At the moment, this implementation only serves tiling pattern support; the mechanism will be enabled automatically at the first use of a tiling pattern (merely defining a pattern will not enable it) and is of little use for anything else. The relevant tables, should you want to expand on it yourself, are `M.colourspaces` and `M.patterns`; see the source file for additional instructions.

Since pdf resource management must be done exactly once, this package may clash with other graphics packages doing the same. In particular, minim's resource management will cause double (and thus invalid) entries in pages'

attribute dictionaries when used together with tikz or pgf. They can be used together, however, if you do not use minim's tiling patterns.

Debugging

You can enable (global) debugging by saying `debug_pdf` to metapost or `M.enable_debugging()` to lua. This will write out a summary of metapost object information to the pdf file, just above the pdf instructions that object was translated into. For this purpose, the pdf will be generated uncompressed. Additionally, a small summary of every generated image will be written to log and terminal.

Extending metapost

You can extend this package by adding new metapost specials. Specials should have the form `"identifier: instructions"` and can be added as pre- or postscript to metapost objects. A single object can carry multiple specials and a `special "..."` statement is equivalent to an empty object with a single prefix.

Handling of specials is specified in three lua tables: `M.specials`, `M.prescripts` and `M.postscripts`. The `identifier` above should equal the key of an entry in the relevant table, while the value of an entry in one of these tables should be a function with three parameters: the internal image processor state, the `instructions` from above and the metapost object itself.

If the `identifier` of a prescript is present in the first table, the corresponding function will replace normal object processing. Only one prescript may match with this table. Functions in the the other two tables will run before or after normal processing.

Specials can store information in the `user` table of the picture that is being processed; this information is still available inside the `finish_mpfigure` callback that is executed just before the processed image is surrounded by properly-dimensioned boxes.

The `M.init_files` table contains the list of metapost files that new instances are initialised with. The actual format will be loaded after the files in this table.

Licence

This package may be distributed under the terms of the European Union Public Licence (EUPL) version 1.2 or later. An english version of this licence has been included as an attachment to this file; copies in other languages can be obtained at

https://joinup.ec.europa.eu/collection/eupl/eupl-text-eupl-12