MetaPost pictures

(with colors)

Taco Hoekwater

September 16, 2022

- MetaPost outputs graphics using picture objects
- But let us discuss colors first

MetaPost internally has four color models:

- No model (1)
- Greyscale (3)
- RGB (5, initial default)
- CMYK (7)

Each of the models has an associated data type:

- No model: *boolean*
- Greyscale: *numeric*
- RGB: *rgbcolor*
- CMYK: *cmykcolor*

none of these have an alpha/opacity component

There is an internal variable **defaultcolormodel**:

```
defaultcolormodel := 5; % RGB
```

To use a default ‘black’ when drawing a path:

```
draw p;
```

This uses a suitable ‘black’ definition for the current **defaultcolormodel**.

Equivalent to that is this:

```
draw p withcolor true;
```

To skip black initialization when drawing a path:

```
draw p withcolor false;
```

or its alias:

```
draw p withoutcolor;
```

To use an explicit black greyscale when drawing a path:

```
draw p withcolor 0;
```

or its alias:

```
draw p withgreyscale 0;
```

To use an explicit black RGB when drawing a path:

```
draw p withcolor (0,0,0);
```

or its alias:

```
draw p withrgbcolor (0,0,0);
```

To use an explicit black CMYK when drawing a path:

```
draw p withcolor (0,0,0,1);
```

or its alias:

```
draw p withcmykcolor (0,0,0,1);
```

Variables of all types can be created using:

```
boolean mynocolor;
numeric mygreycolor;
rgbcolor myrgbcolor;
cmykcolor mycmykcolor;
```

The *withcolor* command will then automatically find the right color model.

You can access the parts of a color variable.

For RGB:

```
redpart myrgbcolor;
greenpart myrgbcolor;
bluepart myrgbcolor;
```

For CMYK:

```
cyanpart mycmykcolor;
magentapart mycmykcolor;
yellowpart mycmykcolor;
blackpart mycmykcolor;
```

Even works for greyscale
(even though that is not quite a ‘color’ variable):

```
greypart mygreycolor;
```

You can divide or multiply color variables by a numeric:

```
rgbcolor myrgb;
myrgb = (0.5,0.5,0.5) * 1.5;
% (0.75,0.75,0.75)
```

Or you can add a color to another color of the same type:

```
rgbcolor myrgb;
myrgb = (0.5,0.5,0.5) + (0.25,0.25,0.25);
% also (0.75,0.75,0.75)
```

You can also negate a color:

```
rgbcolor myrgb;
myrgb = -(0.5,0.5,0.5);
% (-0.5,-0.5,-0.5)
```

Finally, you can compare colors of the same type with each other:

```
rgbcolor myrgb, myrgba;
myrgb = (0.5,0.5,0.5);
myrgba = (0.25,0.25,0.25);
if myrgb > myrgba:
  message "true";
fi
```

tests process each component in order.

- Color components outside of the 0,1 range are clipped when an item is added to a picture.

- MetaPost uses *picture* **variables** to output items.
- There is one predefined such variable: *nullpicture*.

You create a new pictures using *picture*:

```
picture A;
```

Writing an output file for a picture:

```
shipout A;
```

(this uses `outputtemplate` to set the filename)

Add another picture to a picture:

```
picture A,B;
...
addto A also B;
```

Add a stroked path to a picture:

```
picture A;
path p;
...
addto A doublepath p;
```

Add a filled path to a picture:

```
picture A;
path p;
...
addto A contour p;
```

(path must be cyclic)

Add a text label to a picture:

```
picture A,B;
B = "a" infont "cmr10";
addto A also B;
```

Clip a picture to a path:

```
picture A;
path p;
...
clip A to p;
```

Set the bounding box of a picture to a path:

```
picture A;
path p;
...
setbounds A to p;
```

the path must be cyclic (results in a rectangle)

The *addto* command forms accept options. Color options:

withcolor⟨*color* **expression**⟩
withrgbcolor⟨*rgbcolor* **expression**⟩
withcmykcolor⟨*cmykcolor* **expression**⟩
withgreyscale⟨*numeric* **expression**⟩
withoutcolor

Specifying a pen:

```
withpen⟨pen expression⟩
```

This also works for the `contour` case, where it then does ‘filldraw’

Specifying pre- or postscripts:

```
withprescript ⟨string expression⟩
withpostscript ⟨string expression⟩
```

these are like `special` in TEX.

Specifying a dash pattern:

```
dashed⟨picture expression⟩
```

the picture expression can be lots of things, but is typically a pattern of dots or dashes separated by some whitespace

All the drawing options can be repeated multiple times

At the expression level, a picture can be transformed:

```
picture A,B;
...
A := B scaled 20;
```

or

```
addto A also B scaled 20;
```

You can ask for the corners of a picture:

```
picture A;
pair t;
...
t = llcorner A;
% also lrcorner, urcorner, ulcorner
```

- If you use *for* ... *within*, than you can ask for lots of parts of drawing items.
- For the sake of simplicity, each subitem in a picture is presented as a picture itself with a single item in it.
- For the next tests, you may have to check the type *filled*, *stroked*, *clipped*, *bounded*, *textual* first, because not all types have all parts.
- Each of the tests pick the value of the first item in the picture.

Pre- and postscripts:

```
for v within A:
 prescriptpart v;
% postscriptpart
endfor
```

Transformer parts:

```
for v within A:
  xpart v;
% ypart xxpart yypart xypart yxpart
endfor
```

Color model and/or color part

```
for v within A:
  colormodel v;
  colorpart v;
endfor
```

Color parts (RGB)

```
for v within A:
  redpart v;
  % bluepart greenpart
endfor
```

Color parts (grey)

```
for v within A:
  greypart v;
endfor
```

The dash part:

```
for v within A:
  dashpart v;
endfor
```

The pen part:

```
for v within A:
  penpart v;
endfor
```

The path part:

```
for v within A:
  pathpart v;
endfor
```

The text part of a label:

```
for v within A:
  textpart v;
endfor
```

The font part of a label:

```
for v within A:
  fontpart v;
endfor
```

That's all!