Entry-level MetaPost 2: Move it!

Mari Voipio

This installment introduces some of the basic commands for moving a line or an object — i.e. a *path* — to a different position: shifting, rotating, reflecting, repeating. In programs with a graphical user interface, these operations are typically done by clicking and dragging or clicking and selecting a command on a toolbar.

MetaPost has a slightly different approach to e.g. rotation and this can be confusing at first, although it is completely logical on its own terms. It adds to the confusion that some commonly used commands like flip and duplicate do not exist in MetaPost (nor MetaPost manuals), although the operations are doable once you know what to look for.

For basic information on running MetaPost, either standalone or within a ConTEXt document, see http://tug.org/metapost/runningmp.html.

1   Store it first

Before we start to manipulate a path, we typically store it for further access by defining a *path variable*. In many MetaPost tutorials you see paths named p, q and r, but I prefer slightly longer and more descriptive names, even though that means more typing. Below we first define and then draw a diamond that is used for many examples in this tutorial.
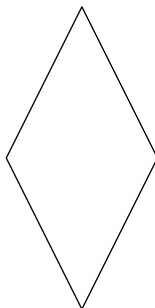
```
beginfig (1) ;
numeric u ; u := 1cm ; % define the unit

% define path variable "dmnd" (diamond shape,
% intentionally asymmetrical)
path dmnd ;
dmnd := (1u,0u) -- (0u,2u) -- (1u,4u) -- (2u,2u)
        -- cycle ;

% drawing diamond (outline)
draw dmnd ;

endfig ;
end .
```

Here is the output:

Mari Voipio

Troubleshooting: If your file compiles but the graphic is empty, you probably forgot to draw at least one path, i.e. the output "paper" is still empty. No draw/fill/filldraw command at all leads to an empty file.
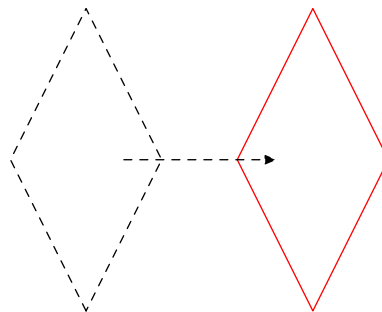
When multiple path variables are defined, they can all go at the top of the file to make sure that we define each variable before trying to use it. However, we can use the variables in any order after that and as many times as is needed. Personally I like to list my variables in alphabetic order so I can find one quickly if I need to check or change the path definition.
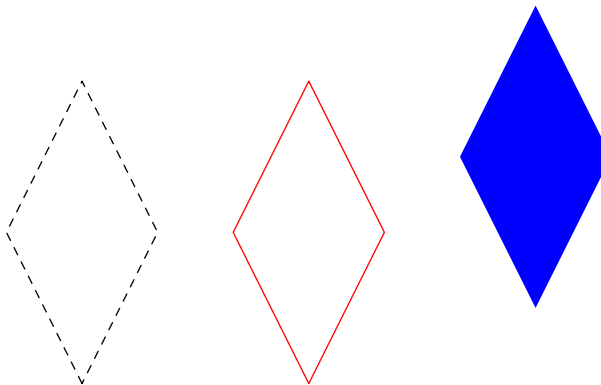
2   Shift (copy, duplicate)

To shift means to move, and that is exactly what the command does. It works in a fairly intuitive way:

```
draw dmnd shifted (3u,0u) withcolor red;
```

That line can be read as "take a diamond, draw it to (3u,0u) using a red pen". Visually:

If we draw the original diamond as well as the shifted one, we now have two diamonds, i.e. we have copied an object. We can change the attributes of the second (shifted) diamond, e.g. to make a coloured diamond by using the `fill` command. For example:

Here's the MetaPost code. The beginning is what we saw in the first section.

```
beginfig (1) ;
numeric u ; u := 1cm ; % define the unit

% define path variable "dmnd" (diamond shape)
path dmnd;
```

```
dmnd := (1u,0u) -- (0u,2u) -- (1u,4u)--(2u,2u)
        -- cycle;

% draw dashed diamond at original location
draw dmnd dashed evenly ;

% draw second diamond to the right, in red
draw dmnd shifted (3u,0u) withcolor red ;

% draw third diamond, filled with blue,
%   to the right and up from original
fill dmnd shifted (6u,1u) withcolor blue ;

endfig ;
end .
```
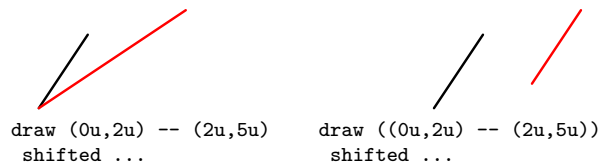
The `shift` command only applies to the element preceding it; we need to use parentheses if we intend otherwise. Thus `draw (0u,2u) -- (2u,5u) shifted (4u,1u)` and `draw ((0u,2u) -- (2u,5u)) shifted (4u,1u)` produce very different results:

```
draw (0u,2u) -- (2u,5u)        draw ((0u,2u) -- (2u,5u))
 shifted ...                       shifted ...
```

The black is the original `(0u,2u)--(2u,5u)` line, while the red is the result of the whole expression, including the shift. Here is the code (combined for exposition):

```
...
% set the penwidth (see previous article)
drawoptions (withpen pencircle scaled 1/10u) ;

% draw original line in black
draw (0u,2u) -- (2u,5u) ;

% draw red line with shift of endpoint only:
% (first example)
draw (0u,2u) -- (2u,5u) shifted (4u,1u)
     withcolor red ;
% ... or ...
% draw red line with whole line being shifted:
% (second example)
draw ((0u,2u) -- (2u,5u)) shifted (4u,1u)
     withcolor red ;
...
```

3   Rotate

Rotation is another basic graphical transformation. In MetaPost, the basic operation is done with the keyword `rotated`. We also always need to specify the angle of rotation. However, if one is used to a graphical program (say, Inkscape), the results of `rotated` can be a bit of a surprise at first. Let's look at an example.

```
% define a path variable "tetris"
path tetris ;
tetris := (3u,2u) -- (4u,2u) -- (4u,5u)--(2u,5u)
        -- (2u,4u) -- (3u,4u) -- cycle ;

% draw solid blue tetris block
fill tetris withcolor blue ;

% draw rotated tetris block in red
fill tetris rotated 90 withcolor red ;
```

And the output (scaled down, here and in the following, from `1cm` for *TUGboat*'s narrow columns):

Say what?

The logic becomes more apparent if we add a small dot at origin (0,0):

```
% mark origin with a black dot
fill fullcircle scaled 1/10u ;
```
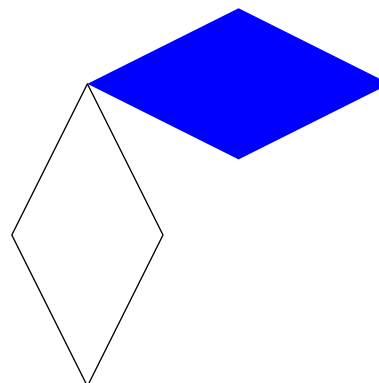
yielding:

The lesson: **The `rotated` command rotates the path's bounding box around the origin (0,0), and rotation direction is counterclockwise.**

If we want to rotate the path around any other point, we have to use the command `rotatedaround`, for which we need to specify both the location of the rotation point and the angle of rotation. Example:

```
% draw a diamond rotated around its top point,
% at (1u,4u)
fill dmnd rotatedaround ((1u,4u),90)
        withcolor blue;
```
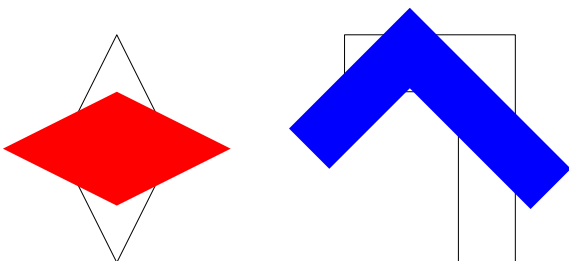
Entry-level MetaPost 2: Move it!

To rotate around the "midpoint" of the object (that is, the center of bounding box, the default rotation point in many programs), we don't need to painfully compute the coordinates for the center. MetaPost has a handy keyword `center` for that:

```
% draw diamond outline
draw dmnd ;
% draw red diamond rotated around its center
fill dmnd rotatedaround (center dmnd,45)
         withcolor red;

% draw tetris outline
draw tetris;
% draw tetris block rotated around its center
fill tetris rotatedaround (center tetris, 90)
           withcolor blue ;

% these commands would make the centers visible:
%fill fullcircle scaled 1/10u shifted
%                 (center dmnd) ;
%fill fullcircle scaled 1/10u shifted
%                 (center tetris) ;
```
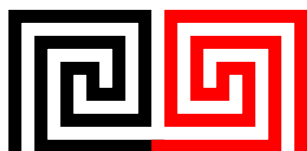

4   Reflect (flip, mirror)

Another command that may seem to be missing in MetaPost is horizontal or vertical mirroring (also known as flipping). The functionality does exist, invoked with the keyword `reflectedabout`, although a bit of practice is needed to get used to it — but on the other hand we can specify any straight line to be the reflection axis, it doesn't have to be horizontal or vertical. Consequently, to use `reflectedabout`, we must specify *two* points for the reflection axis. If you find this hard, think of a mirror and where you'd place its edge to get the reflection needed.

Here I'm playing around with a Greek key pattern and its reflection (yes, they do overlap in the middle) around a vertical line.

And the code to produce it:

Mari Voipio

```
beginfig (2) ;

% design source:
%  http://gwydir.demon.co.uk/jo/greekkey/turns.htm
numeric u ; u := 3.8mm ; % define the unit

% creating sharp squared joins
linecap := squared ;
linejoin := mitered ;

% set the penwidth
drawoptions (withpen pencircle scaled 1/2u) ;

% define the path for the greek key
path gkey;
gkey :=   (origin) -- (0u,5u) -- (5u,5u) -- (5u,1u)
        -- (2u,1u) -- (2u,3u) -- (3u,3u) -- (3u,2u)
        -- (4u,2u) -- (4u,4u) -- (1u,4u) -- (1u,0u)
        -- (5.5u,0u);

% draw it
draw gkey ;

% flip it and draw it in red
draw gkey reflectedabout ((5.5u,0u),(5.5u,5u))
        withcolor red ;
endfig ;
end .
```
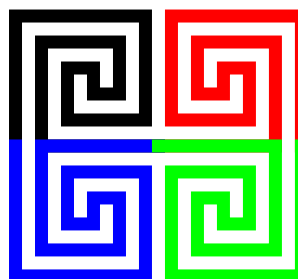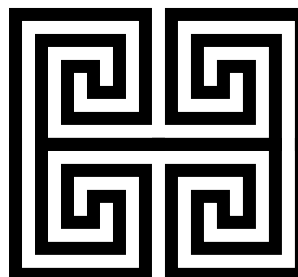
By adding another, horizontally flipped, key and then a key with rotation we can create a square Greek key pattern variation:

```
draw gkey reflectedabout ((origin),(5.5u,0u))
        withcolor blue ;
draw gkey rotatedaround (lrcorner gkey,180)
        withcolor green ;
```


The pattern is more apparent entirely in black:

If you want to flip an object along a side of the bounding box, the MetaFun package provides a set of handy shortcuts: the corners of the bounding box are called `llcorner`, `lrcorner`, `ulcorner` and `urcorner`. Thus, to flip an object along the right edge of the bounding box, the lower right and upper right corner are called for:

```
% drawing flipped tetris block
fill tetris reflectedabout (lrcorner tetris,
                            urcorner tetris)
         withcolor green ;
```

Yielding:

5   ... and repeat

If you need to repeat the same pattern at regular intervals, a combination of shift and loop is possible. Besides the angular variety above, I've also designed a rounded version of the Greek key that could e.g. make a nice header for a book. To alter the size of the "frieze" I can either change the number of repetitions or the final size, depending on what shape is desired.

```
% define one spiral
path spiral;
spiral := (0,7/4) .. (2,4) .. (5,2) .. (3,0)
          .. (2,2) .. (4,2) .. (3,1) ;

% repeat to get 10 spirals in a row
for i = 0 step 5 until 45 :
  draw spiral shifted (i,0) ;
endfor ;

% add a bit of white around the pattern
setbounds currentpicture
         to boundingbox currentpicture
         enlarged 1/4 ;

% resize the whole thing
currentpicture := currentpicture xsized 7cm ;
```

And the output:

6   MetaFun

The `xsized` command used in the last line, like the `corner` shortcuts mentioned above, is part of the MetaFun package, not MetaPost proper. MetaFun is loaded in ConTEXt by default, but needs to be explicitly loaded when using standalone plain MetaPost documents, like this:

```
mpost --mem=metafun yourfile.mp
```

See http://wiki.contextgarden.net/MetaFun for more.

⋄ Mari Voipio
  mari dot voipio (at) lucet dot fi
  http://www.lucet.fi