The `shapes` Macros, v1.0

Donald P. Goodman III

July 18, 2015

Abstract

The `shapes` macros for METAPOST provide regular polygons, their corresponding reentrant stars, and images demonstrating fractions. These macros are quite configurable.

Contents

1 Introduction

The `shapes` macros are not revolutionary, and in fact are quite simple; however, I spent some time generalizing them for a text I'm currently working on, and so I thought they might be useful for the general populace. They are divided into two main groups: regular polygons and their corresponding reentrant star shapes; and fractionals, circles divided into a certain number of parts with the desired fraction filled in.

This document was typeset in accordance with the `docstrip` utility, which allows the automatic extraction of code and documentation from the same document.

2 Prerequisites and Conventions

Some prerequesites for using this package are METAPOST itself (obviously). If you're using the package with LaTeX, the `gmp` package would probably be help-

ful; be sure to use the `latex` package option. These should be packaged in any reasonably modern LaTeX system, such as TeXLive or MikTeX.

This documentation assumes nothing about your personal TeX or METAPOST environment. ConTeXt and the various forms of LuaTeX have METAPOST built-in; with pdfLaTeX, the author's choice, one can use the `gmp` package to include the source directly in one's document (that's what's been done in this documentation) or develop a simple script to compile them afterwards and include them in the source via `\includegraphics` (probably the quickest option, since compilation is done in advance). Here, we simply post the plain vanilla METAPOST code, and let you work out those details however you prefer.

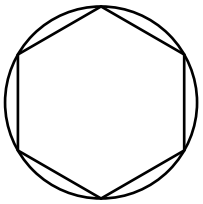
3 The Shapes Macros

We begin with the simple shapes macros, which are about as basic as they can be.

All of these shapes default to circles with a one inch diameter, so you can scaled them with that kept in mind.

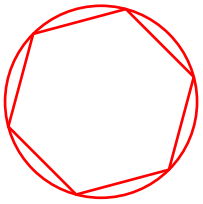
The simplest case is with a regular polygon, like so:

```
modfig(6,0);
draw modfigure;
```



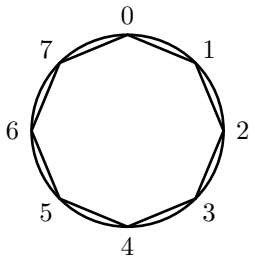
Of course, these drawn figures can be manipulated in the usual ways:

```
modfig(6,0);
draw modfigure rotated (45) withcolor red;
```



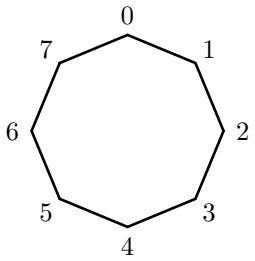
The two arguments to `modfig` are simple: the first tells the number of sides desired, while the second means *no numbers are printed* if 0, and *print numbers* if 1:

```
modfig(8,1);
draw modfigure;
```



If you don't want the circle to be printed, simply tell METAPOST with `modcircle := false;` and it will not print it:

```
modcircle := false;
modfig(8,1);
draw modfigure;
```

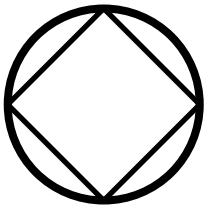


`modcircle` defaults to `true`.

Notice that `modfig` doesn't care if there's a circle or not; if you want numbers on the vertices, it will print them there.

You can adjust the width of the lines by specifying `modcirclepen` and `modshapepen`:

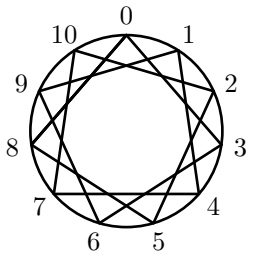
```
modcirclepen := pencircle scaled 3;
modshapepen := pencircle scaled 2;
modfig(4,0);
draw modfigure;
```



Both `modcirclepen` and `modshapepen` default to `pencircle scaled 1`.

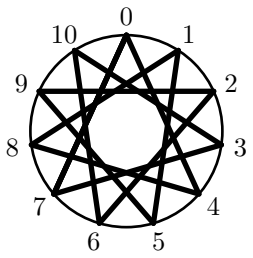
We can do essentially the same thing with reentrant star figures with `modstar`. Unlike `modfig`, `modstar` takes *three* arguments: the number of vertices, whether or not you want those vertices numbered, and how many points you want to skip as you go around the circle.

```
modstar(11,1,3);
draw modfigure;
```



Note that the third argument actually skips $n-1$ points, not n points. But all the same parameters we saw when looking at `modfig` will still work in the same way:

```
modcirclepen := pencircle scaled 1;
modshapepen := pencircle scaled 2;
modstar(11,1,4);
draw modfigure;
```

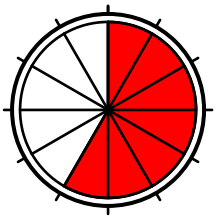


And that's about all there is to it.

4 Fraction Images

The following macros are useful for demonstrating the nature and size of fractions in a visible way. The name of the game here is `fraccirc`, which takes two arguments: the number of parts to be filled, and the number of parts in the whole:

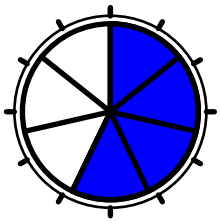
```
fraccirc(7)(12);
draw thefrac;
```



Note that, due to the internal implementation, these two arguments must each be enclosed in their own parentheses.

We can adjust these things as appropriate. For example, we can fill with blue rather than red, and use thicker lines:

```
fracfillcolor := blue;
fraccirclepen := pencircle scaled 1;
fractionpen := pencircle scaled 2;
fraccirc(4)(7);
draw thefrac;
```



As the above example suggests, `fracfillcolor` gives the color with which the portion of the fraction should be filled; it defaults to `red`. `fraccirclepen` is the pen used to draw the circle around the fraction; it defaults to `pencircle scaled 1.5`. Finally, `fractionpen` is the pen used to draw the partition and the circle immediately surrounding them; it defaults to `pencircle scaled 1`.

5 Implementation

```
color fracfillcolor; fracfillcolor := red;
pen fraccirclepen; fraccirclepen := pencircle scaled 1.5;
pen fractionpen; fractionpen := pencircle scaled 1;
def fraccirc(suffix x)(expr y) =
radius := 1in;
ticklen := radius/24;
path circ; circ := fullcircle scaled radius;
pair p[]; pair q[]; pair r[]; pair s[];
p[0] := (0,0) shifted (0,radius/2);
q[0] := p[0] shifted (0,ticklen);
r[0] := p[0] shifted (0,-ticklen);
s[0] := r[0];
picture thefrac;
picture addition;
thefrac := image(pickup fraccirclepen; draw circ;);
addition := image(pickup fractionpen; draw p[0]--q[0];);
addto thefrac also addition;
for i=1 upto 12:
p[i] := p[i-1] rotatedaround ((0,0),-30);
q[i] := q[i-1] rotatedaround ((0,0),-30);
addition := image(pickup fractionpen; draw p[i]--q[i];);
addto thefrac also addition;
endfor;
addition := image(pickup fractionpen; draw (0,0)--r[0];);
addto thefrac also addition;
pair t; pair q;
addition := image(
t = r[0] rotatedaround ((0,0),-((360/y)*x)/2);
q = r[0] rotatedaround ((0,0),-((360/y)*x));
fill r[0]--(0,0)--q..t..cycle withcolor fracfillcolor;
```

```
draw r[0]--(0,0)--q..t..cycle;
);
addto thefrac also addition;
for i=1 upto y:
r[i] := r[i-1] rotatedaround ((0,0),-(360/y));
s[i] := r[i-1] rotatedaround ((0,0),-(360/y)/2);
addition := image(%
pickup fractionpen;%
draw (0,0)--r[i];
draw r[i-1]..s[i]..r[i];
);
addto thefrac also addition;
endfor;
enddef;
% put in the pens for the modular shapes
boolean modcircle; modcircle := true;
pen modcirclepen; modcirclepen := pencircle scaled 1;
pen modshapepen; modshapepen := pencircle scaled 1;
def modstar(expr numpoints,numbers,numstar) =
picture modfigure;
if (modcircle = true):
modfigure := image(draw fullcircle scaled 1in withpen modcirclepen;);
else:
modfigure := image();
fi;
pickup modshapepen;
pair p[]; pair q[];
picture addition;
p[0] = (0,0.5in);
q[0] = (0,0.6in);
if (numbers = 1):
addition := image(label("0",q[0]));
addto modfigure also addition;
fi
for i=1 upto numpoints:
p[i] = p[i-1] rotatedaround ((0,0),-(360/numpoints));
q[i] = q[i-1] rotatedaround ((0,0),-(360/numpoints));
if (numbers = 1):
if (i <> numpoints):
addition := image(label(decimal i,q[i]));
addto modfigure also addition;
fi
fi
endfor
for i=0 upto numpoints:
if (i < numstar):
addition := image(%
draw p[i]--p[i+numstar];
draw p[i]--p[numpoints - numstar + i];
);
```

```
addto modfigure also addition;
elseif (i >= (numstar*2)):
addition := image(draw p[i-numstar]--p[i]);
addto modfigure also addition;
fi
endfor
addto modfigure also addition;
enddef;
def modfig(expr numpoints,numbers) =
picture modfigure;
if (modcircle = true):
modfigure := image(draw fullcircle scaled 1in withpen modcirclepen);
else:
modfigure := image();
fi
pair p[]; pair q[];
picture addition;
pickup modshapepen;
p[0] = (0,0.5in);
q[0] = (0,0.6in);
if (numbers = 1):
addition := image(label("0",q[0]));
addto modfigure also addition;
fi
for i=1 upto numpoints:
p[i] = p[i-1] rotatedaround ((0,0),-(360/numpoints));
q[i] = q[i-1] rotatedaround ((0,0),-(360/numpoints));
addition := image(draw p[i-1]--p[i]);
addto modfigure also addition;
if (numbers = 1):
if (i <> numpoints):
addition := image(label(decimal i,q[i]));
addto modfigure also addition;
fi
fi
endfor;
enddef;
```