

# The `shapes` Macros, v1.0

Donald P. Goodman III

August 1, 2015

## Abstract

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

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Prerequisites and Conventions</b>	<b>1</b>
<b>3</b>	<b>The Shapes Macros</b>	<b>2</b>
<b>4</b>	<b>Fraction Images</b>	<b>4</b>
<b>5</b>	<b>Implementation</b>	<b>5</b>

## 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 prerequisites for using this package are METAPOST itself (obviously). If you're using the package with L<sup>A</sup>T<sub>E</sub>X, the `gmp` package would probably be help-

ful; be sure to use the `latex` package option. These should be packaged in any reasonably modern L<sup>A</sup>T<sub>E</sub>X system, such as T<sub>E</sub>XLive or MikT<sub>E</sub>X.

This documentation assumes nothing about your personal T<sub>E</sub>X or METAPOST environment. ConT<sub>E</sub>Xt and the various forms of LuaT<sub>E</sub>X have METAPOST built-in; with pdfL<sup>A</sup>T<sub>E</sub>X, 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

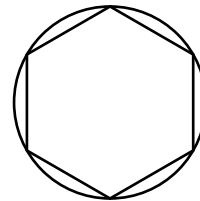
We begin with the simple shapes macros, which are about as basic as they can be.

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;
```

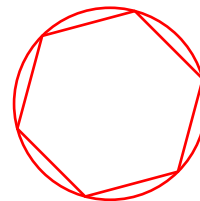


---

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

---

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

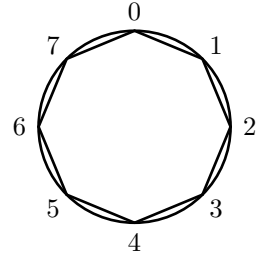


---

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:

---

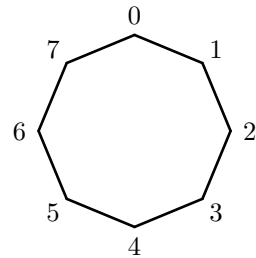
```
modfig(8,1);  
draw modfigure;
```



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;
```



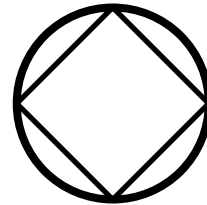
`modcircle` defaults to `true`.

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`.

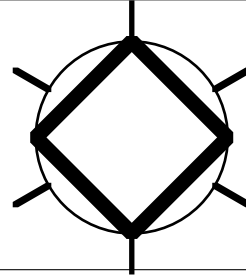
We can also put ticks around the outside of the circle, at whatever intervals we want. The parameters here are `modticks`, which is a boolean set to `false` by default. Set it to `true` if you want ticks. You can control the number of ticks with `nummodticks`, which defaults to twelve. You can control the length of those ticks with `ticklen`, which defaults to `0.1in`. Finally, you can control the pen that is used to draw the ticks, which is `tickpen`, which defaults to `pencircle scaled 1`:

---

```

nummodticks := 6;
ticklen := 0.2in;
modticks := true;
tickpen := pensquare scaled 2;
modshapepen := pensquare scaled 4;
modfig(4,0);
draw modfigure;

```




---

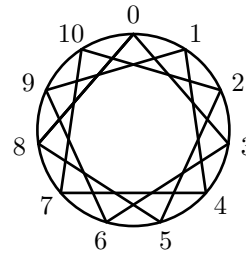
We can do essentially the same thing with reentrant star figures with `modstar`. Unlike `modfig`, `modstar` takes *three* arguments: the number of vertices, whether or not you want those vertices numbered, and how many points you want to skip as you go around the circle.

---

```

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

```




---

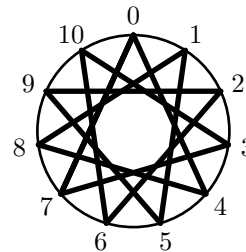
Note that the third argument actually skips  $n - 1$  points, not  $n$  points. But all the same parameters we saw when looking at `modfig` will still work in the same way:

---

```

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

```




---

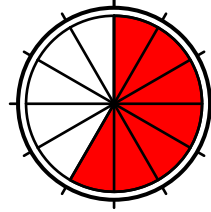
And that's about all there is to it.

## 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;
```



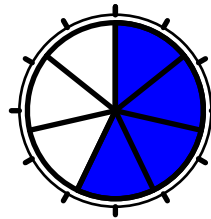
---

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

```
1 color fracfillcolor; fracfillcolor := red;  
2 pen fraccirclepen; fraccirclepen := pencircle scaled 1.5;  
3 pen fractionpen; fractionpen := pencircle scaled 1;  
4 def fraccirc(suffix x)(expr y) =  
5 radius := 1in;  
6 ticklen := radius/24;  
7 path circ; circ := fullcircle scaled radius;  
8 pair p[]; pair q[]; pair r[]; pair s[];  
9 p[0] := (0,0) shifted (0,radius/2);  
10 q[0] := p[0] shifted (0,ticklen);  
11 r[0] := p[0] shifted (0,-ticklen);  
12 s[0] := r[0];  
13 picture thefrac;  
14 picture addition;
```

```

15 thefrac := image(pickup fraccirclepen; draw circ;);
16 addition := image(pickup fractionpen; draw p[0]--q[0]);
17 addto thefrac also addition;
18 for i=1 upto 12:
19 p[i] := p[i-1] rotatedaround ((0,0),-30);
20 q[i] := q[i-1] rotatedaround ((0,0),-30);
21 addition := image(pickup fractionpen; draw p[i]--q[i]);
22 addto thefrac also addition;
23 endfor;
24 addition := image(pickup fractionpen; draw (0,0)--r[0]);
25 addto thefrac also addition;
26 pair t; pair q;
27 addition := image(
28 t = r[0] rotatedaround ((0,0),-((360/y)*x)/2);
29 q = r[0] rotatedaround ((0,0),-((360/y)*x));
30 fill r[0]--(0,0)--q..t..cycle withcolor fracfillcolor;
31 draw r[0]--(0,0)--q..t..cycle;
32 );
33 addto thefrac also addition;
34 for i=1 upto y:
35 r[i] := r[i-1] rotatedaround ((0,0),-(360/y));
36 s[i] := r[i-1] rotatedaround ((0,0),-(360/y)/2);
37 addition := image(%
38 pickup fractionpen;%
39 draw (0,0)--r[i];
40 draw r[i-1]..s[i]..r[i];
41 );
42 addto thefrac also addition;
43 endfor;
44 enddef;
45 % put in the pens for the modular shapes
46 boolean modcircle; modcircle := true;
47 pen modcirclepen; modcirclepen := pencircle scaled 1;
48 pen modshapepen; modshapepen := pencircle scaled 1;
49 boolean modticks; modticks := false;
50 numeric nummodticks; nummodticks := 12;
51 numeric ticklen; ticklen := 0.1in;
52 pen tickpen; tickpen := pencircle scaled 1;
53 def modstar(expr numpoints,numbers,numstar) =
54 picture modfigure;
55 if (modcircle = true):
56 modfigure := image(draw fullcircle scaled 1in withpen modcirclepen);
57 else:
58 modfigure := image();
59 fi;
60 picture modtickpic;
61 if (modticks = true):
62 modtickpic := image(
63 pair tickstart[]; pair tickend[];
64 tickstart[0] = (0,0) shifted (0,0.5in);

```

```

65 tickend[0] = tickstart[0] shifted (0,ticklen);
66 draw tickstart[0]--tickend[0] withpen tickpen;
67 for i=1 upto (nummodticks-1):
68 tickstart[i] = tickstart[i-1] rotatedaround
69 ((0,0), (360/nummodticks));
70 tickend[i] = tickend[i-1] rotatedaround
71 ((0,0), (360/nummodticks));
72 draw tickstart[i]--tickend[i] withpen tickpen;
73 endfor;
74 );
75 addto modfigure also modtickpic;
76 fi
77 pickup modshapepen;
78 pair p[]; pair q[];
79 picture addition;
80 p[0] = (0,0.5in);
81 q[0] = (0,0.6in);
82 if (numbers = 1):
83 addition := image(label("0",q[0]));
84 addto modfigure also addition;
85 fi
86 for i=1 upto numpoints:
87 p[i] = p[i-1] rotatedaround ((0,0),-(360/numpoints));
88 q[i] = q[i-1] rotatedaround ((0,0),-(360/numpoints));
89 if (numbers = 1):
90 if (i <> numpoints):
91 addition := image(label(decimal i,q[i]));
92 addto modfigure also addition;
93 fi
94 fi
95 endfor
96 for i=0 upto numpoints:
97 if (i < numstar):
98 addition := image(%
99 draw p[i]--p[i+numstar];
100 draw p[i]--p[numpoints - numstar + i];
101 );
102 addto modfigure also addition;
103 elseif (i >= (numstar*2)):
104 addition := image(draw p[i-numstar]--p[i]);
105 addto modfigure also addition;
106 fi
107 endfor
108 addto modfigure also addition;
109 enddef;
110 def modfig(expr numpoints,numbers) =
111 picture modfigure;
112 if (modcircle = true):
113 modfigure := image(draw fullcircle scaled 1in withpen modcirclepen);
114 else:

```

```

115 modfigure := image();
116 fi
117 picture modtickpic;
118 if (modticks = true):
119 modtickpic := image(
120 pair tickstart[]; pair tickend[];
121 tickstart[0] = (0,0) shifted (0,0.5in);
122 tickend[0] = tickstart[0] shifted (0,ticklen);
123 draw tickstart[0]--tickend[0] withpen tickpen;
124 for i=1 upto (nummodticks-1):
125 tickstart[i] = tickstart[i-1] rotatedaround
126 ((0,0), (360/nummodticks));
127 tickend[i] = tickend[i-1] rotatedaround
128 ((0,0), (360/nummodticks));
129 draw tickstart[i]--tickend[i] withpen tickpen;
130 endfor;
131 );
132 addto modfigure also modtickpic;
133 fi
134 pair p[]; pair q[];
135 picture addition;
136 pickup modshapepen;
137 p[0] = (0,0.5in);
138 q[0] = (0,0.6in);
139 if (numbers = 1):
140 addition := image(label("0",q[0]));
141 addto modfigure also addition;
142 fi
143 for i=1 upto numpoints:
144 p[i] = p[i-1] rotatedaround ((0,0),-(360/numpoints));
145 q[i] = q[i-1] rotatedaround ((0,0),-(360/numpoints));
146 addition := image(draw p[i-1]--p[i]);
147 addto modfigure also addition;
148 if (numbers = 1):
149 if (i <> numpoints):
150 addition := image(label(decimal i,q[i]));
151 addto modfigure also addition;
152 fi
153 fi
154 endfor;
155 enddef;

```