比較対照 METAPOST & Asymptote 〜 第6回

1 前回のクイズの解答

picture 値は .fit() すると **frame** 値に変換され，その後貼り付け先の図を拡大縮小したりしても影響を受けなくなります[1]。また,「固定サイズの貼り付け」は拡大率 1 倍の .fit() であること，currentpicture は出力時に .fit() されること，などを考慮して，前回の解答は次のようになります。

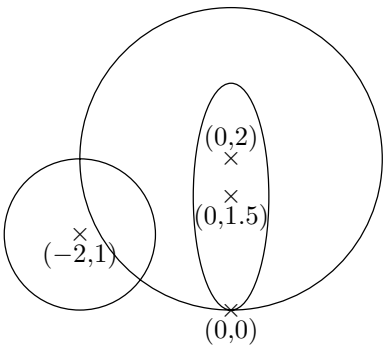
pica→picb	picb→currentpicture	中心	半径
add(picb,pica);	add(picb);	(0,2cm)	2cm
	add(picb.fit());	(0,3cm)	3cm
	add(picb,(w,0));	(2cm,1cm)	1cm
add(picb,pica.fit());	add(picb);	(0,1.5cm)	0.5cm×1.5cm
	add(picb.fit());	(0,1.5cm)	0.5cm×1.5cm
	add(picb,(w,0));	(2cm,1.5cm)	0.5cm×1.5cm
add(picb,pica,(-w,0));	add(picb);	(-2cm,1cm)	1cm
	add(picb.fit());	(-3cm,1cm)	1cm
	add(picb,(w,0));	(1cm,1cm)	1cm

半径に関しては上の説明でおわかりでしょう。中心も「固定サイズでの貼り付け」以外は簡単です。「固定サイズでの貼り付け」の場合，原点をずらせるのですが，そのズレの累積の計算がちょっとわかりにくいかも知れません。たとえば，最後の例では

[もとの中心 (0,1cm)] × 1 + [picb 上でのズレ (-1cm,0)] × 1 + [currentpicture 上でのズレ (1cm,0)] × 2

です。下にソースと実行例を挙げておきますので，参考にして下さい。

```
\begin{ASYpic}*<1cm>(0,0)(2,-1)\astRenew
  \sendASY{picture pica; unitsize(pica,0.5,1.5);  picture picb; unitsize(picb,3);  unitsize(2);
    draw(pica,circle((0,1cm),1cm));
    add(picb,pica);
    add(picb,pica.fit());
    add(picb,pica,(-w,0));
    add(picb);
%   add(picb,(w,0));
%   add(picb.fit());
  }
  \astput(0,0)  {$\times$} \astput(0,0)  [t]<0mm,-1mm>{(0,0)}
  \astput(-2,1) {$\times$} \astput(-2,1) [t]<0mm,-1mm>{($-2$,1)}
  \astput(0,1.5){$\times$} \astput(0,1.5)[t]<0mm,-1mm>{(0,1.5)}
  \astput(0,2)  {$\times$} \astput(0,2)  [b]<0mm,1mm> {(0,2)}
\end{ASYpic}
```



なお，今回の話も踏まえて add の引数は以下のようになります。

add(**picture** 値引数 (貼り付け先)，**picture** 値または **frame** 値引数 (貼り付ける図：必須)，**pair** 値引数)[2]

貼り付け先は，省略すると currentpicture です[3]。

[1] 平行移動などは影響を受けているように見えますが，これは「貼り付けられる図の原点」に対応する**貼り付け先の座標**が影響を受けているだけであって，貼り付けられた図自体が影響を受けているわけではありません。

[2] 最後の **pair** 値は「固定サイズで貼り付け」の貼り付け位置です。貼り付け先のこの座標に，貼り付ける図の原点を合わせます。

[3] add にはまだまだオプション引数がありますし，貼り付ける図が **picture** 値か **frame** 値かで，オプションの種類やそれらのデフォルト値が違ってくることもあるのですが，そういった細かいことについてはまた今度って事で。あと，貼り付け先が **frame** 値の add もあるのですが，引数の種類や使い道がちょっと違うので，これもまた機会があればって事で。

2　パターン塗り

Asymptote では〈巡回パス〉の内部領域を，市松模様やタイル模様，ブロック模様や斜線などで塗りつぶすことができます[4]。

ちなみにこの機能は patterns.asy で定義された命令を使うので，この機能を使う前に

```
import patterns;
```

で patterns.asy を読み込んでおく必要があります。

さて，パターン塗りの仕方ですが，これは内部的には **frame** 値変数の currentpatterns に **picture** 値でパターンの繰り返し単位を与えて（add して），PostScript の命令としてパターン塗りを指示するのですが，ユーザーの側からはパターンに名前を付けて，fill や filldraw の際にパターン名を指定する形になります。

まず，パターンに名前を付ける命令。

add(**string** 値 (必須) , **picture** 値 (必須) , **pair** 値 , **pair** 値)

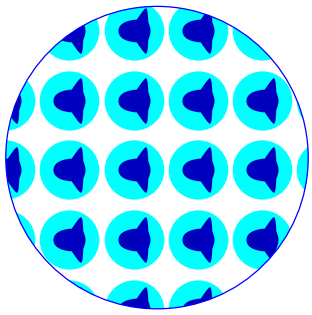
です。**string** 値はパターンの名前，**picture** 値はくり返し単位の図，2 つの **pair** 値はくり返し単位のまわりにつくマージンで，順に左下マージン，右上マージンです。省略すると 0 です。ここで，add の書式が前ページで紹介したものと全く違うことに違和感がある人も多いでしょう。実はこれは Asymptote の特徴の 1 つで,「たとえ同名であっても，引数の並びが違う命令（function）は異なる命令と見なされる[5]」という仕様によります。これにより，ある命令から派生した命令や，機能的に同じようなことをする命令を同じ名前で定義できるようになります。たとえば **picture** 値に貼り付けるのと **frame** 値に貼り付けるのと，ユーザーからすればどちらも似たようなことをするのに，内部的には異なる処理が必要なので異なる命令なのですが，これを同名にすればユーザーは内部的なことを意識しないで済むようになります。これは,「図形を描く」という直感的な要素の強いものを扱うプログラムとしてはある意味合理的な仕様と言えるでしょう。

ちょっと脱線しましたが，次に塗りつぶし。これは fill や filldraw の **pen** 値引数として，

pattern(先程決めたパターン名)

を指定すれば OK です。1 つ例を挙げましょう[6]。

```
\begin{ASYpic}<1cm>(0,0)(2,-3)
  \sendASY{pair[] z = new pair[8]; real u=3mm; picture mizutama;
    import patterns;
    z[0]=(0.7,0); z[1]=(0.6,0.4); z[2]=(0.5,1); z[3]=(0.1,0.5); z[4]=-z[0];
    for(int i=1; i<4; ++i){z[8-i] = yscale(-1)*z[i];}
    path mambo=z[1]{up}..tension 3 ..{left}z[2]..tension 3 ..{curl 0}z[3]
               ..z[4]..z[5]{curl 0}..tension 3 ..z[6]{right}
               ..tension 3 ..{up}z[7]{curl 0}..z[0]..{curl 0}cycle;
    fill(mizutama,circle(0,1.3u),cyan);
    fill(mizutama,scale(u)*mambo,heavyblue);
    add("suizokukan",mizutama,(1bp,2bp),(1bp,2bp));
    filldraw(circle(0,2),pattern("suizokukan"),blue);
  }
\end{ASYpic}
```



なお，くり返し単位の **picture** 値ですが，有名どころは以下の命令で簡単に生成できます[7]。

市松模様の図を含む **picture** 値	checker(**real** 値 , **real** 値 , **pen** 値)	引数は幅，高さ，色
ブロックの図を含む **picture** 値	brick(**real** 値 , **real** 値 , **pen** 値)	引数は幅，高さ，色
タイル型の図を含む **picture** 値	tile(**real** 値 , **real** 値 , **pen** 値 ,**filltype** 値)	幅，高さ，色，脚注参照
斜線模様の図を含む **picture** 値	hatch(**real** 値 , **pair** 値 , **pen** 値)	引数は間隔，向き，色
網目模様の図を含む **picture** 値	crosshatch(**real** 値 , **pen** 値)	引数は間隔，色

[4] METAPOST でもやろうと思えばできます。やりかたは **picture** 値変数を用意し，塗りつぶしたい領域よりちょっと大きめの矩形をパターンで埋め尽くします。これを目的の〈巡回パス〉で clip して，currentpicture に貼り付ければ OK です。実際，MEPOTEX の斜線はこの方法で実現されています。

[5] 同様に，命令（function）と同名の変数（variable）も許されます。（function には引数があり，variable には引数がないため区別できる。）

[6] METAPOST と違い，Asymptote では tension 3 .. や curl 0 で，tension と数値，数値と .. , curl と数値の間の空白は必須です。

[7] 表で tile の引数にある **filltype** 値とは，塗りつぶし方法を指定する変数タイプで，ここではとりあえず Fill（塗りつぶす）と NoFill（枠線のみ）を覚えておけば十分でしょう。なお，表の命令の「仕上がり見本」は Asymptote のマニュアルの再録（p.41）になりますので省略します。