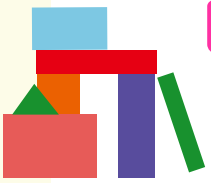


04 論文の図をプログラミングで描こう — MetaPost

基専

松崎公紀（高知工科大学）

プログラムで図を描こう

本会の会員の中には，論文をLaTeXで書いている人も多くいるだろう．LaTeXの利点は複数あるが，数式が美しく書けること，テキストファイルであり文章と書式とが分かれていることが重要な利点である．

論文そのものをLaTeXを用いて書いている場合でも，図については少し面倒が残る．簡易な図であれば，xypic や picture 環境を利用して直接LaTeXのソースとして書くことも可能である．しかし，少し複雑な図になると，ほかのソフトウェアで作成したものを取り込んでいることが多いであろう．実際筆者の周辺だと，Microsoft PowerPoint で作成した図を PDF 形式でエクスポートして利用している人が多い．

近年の PowerPoint や Adobe Illustrator では，スマートガイド（オブジェクトの操作時に一時的に表示されるガイド）などの機能があり，ある程度は正確な図を描くことがやりやすくなった．しかし，規則性のある図を描く場合には，本稿で説明するプログラミングにより図を描く（生成する）ことが便利である．たとえば，円周を 5 等分する位置（正 5 角形の頂点）に文字 A-E を配置した図や二分木の図（図 -1）を描きたいとしよう．PowerPoint を用いてこの図を正確に描くのはかなり困難もしくは手間がかかる．ここでプログラミングの出番である．たとえば図 -1 左の例だと，プログラミングであれば，正 5 角形の頂点の位置を計算させて文字を置くだけである．本稿で紹介する MetaPost を使えば，次のプログラムで図 -1（左）を描くことができる．

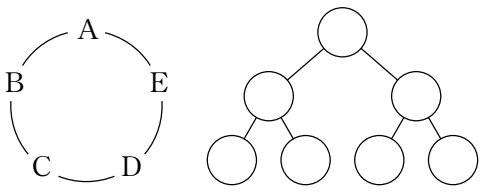


図 -1　例題：これらの図をどう描きますか？

```
input TEX;
beginfig(0)
  pair p[];
  u := 5mm; r := 2u;
  draw fullcircle scaled 2r;     % 円周
  for i = 0 upto 4:              % 5つの文字
    p[i] = dir(72i + 90) scaled r;
    unfill fullcircle scaled u shifted p[i];
    label(TEX(char(ASCII("A")+i)), p[i]);
  endfor;
endfig;
end.
```

多くの場合，論文の図は線画に文字列を配置したものであり，凝った装飾よりも正確性が求められる．MetaPost によるプログラミングで図を描けば，ほかの方法よりも正確に，美しく，規則的な部分は簡単に描くことができる．さぁ，図を描くプログラミングを始めよう！

MetaPost 準備

MataPost は，D. E. Knuth がフォント生成のために作った METAFONT をもとに，1990 年に John D. Hobby によって作られたプログラム言語および処理系である．METAFONT がビットマップフォント（とフォントの情報）を生成するのに対して，MetaPost はベクター形式の画像を生成することが大きく異なる．

近年のLaTeX環境では MetaPost が同梱される

（TEX Live では，LATEX を動かすための最小限のスキームである scheme-small 以上のスキームでインストールすれば MetaPost がインストールされる）．したがって，LATEX を利用されている人のコンピュータのほとんどには MataPost が入っているはずである．ターミナルにて

```
mpost --help
```

と入力してヘルプが出力されれば問題ない．

日本語を利用するには，pmpost というコマンドを利用するが，詳細については本稿の最後に述べる．

MetaPost の初歩

Hello World!

MataPost プログラムの拡張子は標準的には .mp とする．ここでは，sample.mp というファイルにプログラムを書いていくこととしよう．MetaPost において，beginfig(*n*) から endfig; で指定される範囲が 1 つの図に対応する．ここで *n* は，画像の番号を指定する 4095 以下の非負整数であるが，連番である必要はない．では，最も簡単な図として，4 つの直線をつないで正方形を書く例から始めよう．エディタを用いて次のプログラムを写し，それを sample.mp とする．

```
beginfig(1)
  draw (0cm,0cm)--(1cm,0cm)--(1cm,1cm)
       --(0cm,1cm)--cycle;
endfig;

end.
```

上記のプログラムから画像を生成するには，以下の 2 つの方法がある（日本語などを含む場合については後述する）．

- コマンド mpost sample.mp を実行すると sample.1 というファイルが生成される．このファイルは EPS 形式となっており，LATEX の文書に取り込むことができる．後述する文字列ラベルを用いていなければ，Ghostview などのアプリケーションで開くこともできる．文字列ラベルを用いている場合には，それらのフォント情報が入っていないため，LATEX 文書に埋め込まないと表示ができない．
- コマンド mptopdf sample.mp を実行すると，sample-1.pdf というファイルが生成される．こちらのコマンドでは PDF 形式のファイルが出力されるため，特に pdatex で処理する文書の場合にはこちらのほうが便利であろう．

上記のいずれかの方法で画像を生成すると，次の画像が得られる．

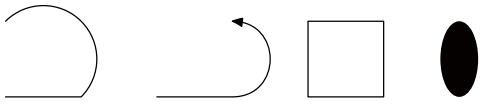
プログラムにおいて座標は 2 次元座標 (x, y) で与える．x は右方向，y は上方向に値が増加する．座標の単位は，ポストスクリプトポイント（= 0.352777mm）である．上のプログラムのように，単位を付けて 1cm などと指定することもできる．実際には，後に拡大縮小がやりやすいように，基準となる大きさを u := 1cm; などと定義して，それを単位としてプログラムを作るとよい（次のプログラムを見よ）．

描画の基本となる命令は，draw（線），drawarrow（矢印），drawdblarrow（両矢印），fill（塗りつぶし）である．次のプログラムの例では，これらの基本命令を用いて，直線，曲線，正方形，円を描く方法を一度に示そう．

```
beginfig(2);
  u := 1cm;

  draw (0,0)--(1u, 0)..(1u, 1u)..(0, 1u);
  drawarrow (2u,0)--(3u, 0){dir(0)}
          ..{dir(180)}(3u,1u);
  draw unitsquare scaled 1u shifted (4u, 0);
  fill fullcircle xscaled .5u yscaled 1u
          shifted (6u, .5u);
endfig;
```

（実行結果）

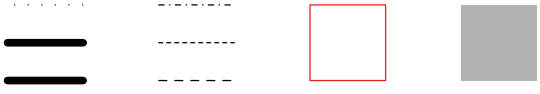


最初の draw 命令では，(0, 0) から (1u, 0) まで直線を，続けて (1u, 0) から (1u, 1u) を通って (0, 1u) まで曲線を引いている．次の drawarrow の行に示すように，曲線がある点から出る／入る際にその方向を指定することもできる（ここで，dir(d) は角度 d° の方向の単位ベクトルである．MetaPost ではほとんどの場合で sin，cos を使う必要がない）．3 つ目の unitsquare は (0, 0)，(1, 0)，(1, 1)，(0, 1) を辺とする正方形である．中心の位置が (0, 0) でないことに注意がいる．4 つ目の fullcircle は中心 (0, 0) 直径 1 の円である．これらの例のように，図形を拡大縮小 (scaled m，xscaled m，yscaled m)，回転 (rotated d)，平行移動 (shifted (x, y)) することもできる．

実線だけでなく，さまざまな線（太さ，点線）を用いることができる．また（論文ではモノクロの図が多いかもしれないが）色を付けることもできる．

```
beginfig(3)
  u := 1cm;
  draw ((0,0)--(1u,0)) shifted (0,0)
         withpen pencircle scaled 3;
  draw ((0,0)--(1u,0)) shifted (0,.5u)
         withpen pencircle scaled 3;
  draw ((0,0)--(1u,0)) shifted (0,1u)
         dashed withdots;
  draw ((0,0)--(1u,0)) shifted (2u,0)
         dashed evenly;
  draw ((0,0)--(1u,0)) shifted (2u,.5u)
         dashed evenly scaled .5;
  draw ((0,0)--(1u,0)) shifted (2u,1u)
         dashed dashpattern(
                  on 2 off 2 on 0.2 off 2);
  draw unitsquare scaled 1u shifted (4u, 0)
         withcolor (1.0, 0, 0);
  fill unitsquare scaled 1u shifted (6u, 0)
         withcolor .7white;
endfig;
```

（実行結果）



これだけ知っていれば，最低限の図は書ける．これから MataPost のプログラムらしいところに入っていこう．

値と型，代入と等式

MetaPost では単なる数だけでなく，数の 2 つ組（対）や 3 つ組，draw コマンドで描くことができるパス，描画された図などを値として扱うことができる．MetaPost では，数は浮動小数点数ではなく，固定小数点数で表されており，numeric 型である．数の対は pair 型，パスは path 型，図は picutre 型である．MetaPost では，numeric 型を除いて，変数宣言が必要である．次の例は，path 型の変数 mypath，および，pair 型の配列変数 pairs を宣言して使うものである．

```
beginfig(4);
  path mypath; pair pairs[];
  u := 1cm;
  pairs[0] := (0,0); pairs[1] := (u,0);
  pairs[2] := (u,u);
  mypath := pairs[0]--pairs[1]--pairs[2];
  drawarrow mypath;
endfig;
```

（実行結果）

代入のみが行えるほかのプログラミング言語と異なり，MetaPost では等式もある．代入が := であるのに対し，等式は = である．等式では，MetaPost 内部のソルバにより変数の値が決定される．たとえば，(a, 1) = (2, b); のような式を書くと，a = 2, b = 1 となる．式が解を持たない場合には，エラーとなる．

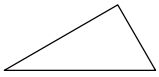
等式をうまく利用すると，プログラミングの前に値を計算する必要がなく，MetaPost に計算をさせることができる．そのような例を 1 つ示そう．次のプログラムは，30° の直線と 150° の直線の交点を求めている．ここで，a[v, w] という表記は，v と w を結ぶ直線上で a:(1−a) に内分（外分）する点を表す．

```
beginfig(5);
  pair pairs[];
```

```
  pairs[0] := (0,0); pairs[1] := (2,0);
  pairs[2] = s [(0,0), (0,0)+dir(30)]
           = t [(2,0), (2,0)+dir(120)];
  show "s=" & decimal(s);
  show "t=" & decimal(t);

  draw (pairs[0]--pairs[1]--pairs[2]--cycle)
       scaled 1cm;
endfig;
```

（実行結果）



show は，引数で与えられた値をターミナルに出力する．この例では，

```
>> "s=1.73206"
>> "t=1"
```

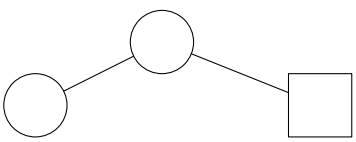
と出力される．

MetaPost に計算させる例をもう 1 つ示そう．次のプログラムは，2 つの円と四角を配置して，その間を矢印で結ぶものである．2 つの図形を線や矢印で結ぶ際には中心どうしを結びたいが，図形の中まで引きたくはない．図形の中心を求めるには center を用いる．図形の中まで線を引かないためには，cutbefore や cutafter を用いる．これらの計算を MetaPost に行わせることで，正確な図形を手計算することなく描ける．

```
beginfig(6);
  u := 1cm; path p[];

  p[0] := fullcircle scaled u shifted (0,0.5u);
  p[1] := fullcircle scaled u shifted (2u, 1.5u);
  p[2] := unitsquare scaled u shifted (4u, 0);
  draw p[0]; draw p[1]; draw p[2];
  draw center(p[0])--center(p[1])
       cutbefore p[0] cutafter p[1];
  draw center(p[1])--center(p[2])
       cutbefore p[1] cutafter p[2];
endfig;
```

（実行結果）



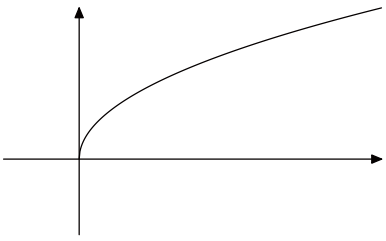
パスなどを扱う計算のためのマクロには，これらのほかにも bbox（BoundingBox），buildcycle，intersectionpoint など多数ある．

制御構文とマクロ

さらに MetaPost プログラミングの深部へと進もう．まずは for による繰り返しだ．MetaPost らしい for による繰り返しの例を示そう．

```
beginfig(7);
  u := 1cm;
  drawarrow (-1u, 0)--(4u, 0);
  drawarrow (0, -1u)--(0, 2u);
  draw (0,0)
    for i = 0 step 0.01 until 4:
      --(i * u, sqrt(i) * u)
    endfor
  ;
endfig;
```

（実行結果）



繰り返しは出現した位置で展開され，展開された後のプログラムが評価される．上のプログラムでは，繰り返しの展開によって

```
  draw (0,0)
       --(0 * u, sqrt(0) * u)
       --(0.01 * u, sqrt(0.01) * u)
       --(0.02 * u, sqrt(0.02) * u)
        中略
       --(4 * u, sqrt(4) * u)
  ;
```

となり，生成された 400 個の点が結ばれ $y = \sqrt{x}$ のきれいな曲線が描かれる．

for による繰り返しは，おおよそ任意の場所に書くことができる．この例のように，1 つの draw 命

令の中に含めることもできるし，逆にbeginfigとendfigを含むような繰り返しを書くこともできる．後者の場合は，展開によって複数の画像が生成されることになる．

繰り返しの基本的な構文は上記で示した

```
for 変数 = a step b until c :
  繰り返しの内容
endfor
```

であるが，増分値 b が 1 または –1 であれば，それぞれ 1 行目を

```
for 変数 = a upto c :
for 変数 = a downto c :
```

と書くこともできる．

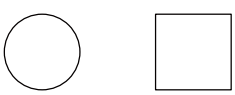
規則的な図を描こうとすると，ほとんど同じプログラム断片を繰り返し書くことがある．繰り返しによってそれらの重複を避けることができれば，プログラミングで図を描くメリットが発揮される．繰り返しが使えない（使いにくい）場合であっても，次に説明するマクロによって簡潔なプログラムにできる．

マクロの最初の例として，引数としたパス（または画像）の中心を (0, 0) に平行移動させるマクロcenteringを示す．

```
vardef centering(expr p) =
  p shifted -center (p)
enddef;

beginfig(8);
  u := 1cm;
  draw centering(fullcircle scaled u)
         shifted (-u, 0);
  draw centering(unitsquare scaled u)
         shifted ( u, 0);
endfig;
```

（実行結果）



図を描くにあたっては条件分岐を用いることはほとんどないが，マクロを作る際には使う．条件分岐の文法は，

```
if 条件式 : 真のときの内容
else: 偽のときの内容
fi
```

である．これも，繰り返しと同様に展開された後で評価される．条件式の部分には，数式のほかにも，変数の型を判定するpath aやpicture bなどを使うことができる．

次のマクロconnectsは 2 つの点もしくはパスを受け取り，それら（の中心）を直線でつなぐものである．これを用いると，以前の例はとても簡潔にかける．筆者が実際に使っているマクロでは，引数が画像となっている場合への対処も含めて利便性を向上させている（条件が長くなるが本質は変わらない）．

```
vardef connects(expr pp, qq) =
  (if (pair pp): pp else: center(pp) fi
   --
   if (pair qq): qq else: center(qq) fi
   if (path pp): cutbefore pp fi
   if (path qq): cutafter qq fi
  )
enddef;

beginfig(9);
  u := 1cm; path p[];

  p[0] := fullcircle scaled u shifted (0,0.5u);
  p[1] := fullcircle scaled u shifted (2u, 1.5u);
  p[2] := unitsquare scaled u shifted (4u, 0);
  draw p[0]; draw p[1]; draw p[2];
  draw connects(p[0], p[1]);
  draw connects(p[1], p[2]);
endfig;
```

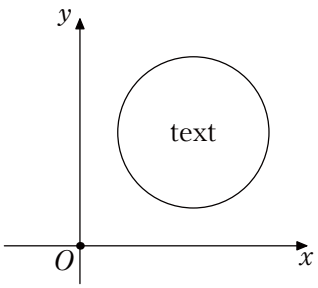
TEX形式のラベル

（特に論文のための）図を描くのにMetaPostを用いる利点の 1 つは，TEX形式のラベル（文字列）を図に書き込むことができることである．図にラベルを書き込む代表的な方法は，次のプログラムのようにlabelとdotlabelを用いることである．

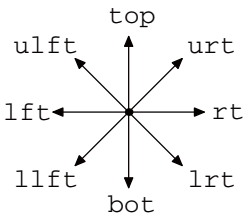
```
beginfig(10);
  u := 1cm;

  drawarrow (-u, 0)--(3u, 0);
  drawarrow (0, -.5u)--(0, 3u);
  draw fullcircle scaled 2u shifted (1.5u, 1.5u);
  label(btex text etex, (1.5u, 1.5u));
  dotlabel.llft(btex $O$ etex, (0,0));
  label.bot(btex $x$ etex, (3u,0));
  label.lft(btex $y$ etex, (0,3u));
endfig;
```

（実行結果）



label がテキストを書き込むのに対し，dotlabel はテキストと点を書き込む．それぞれ，第 1 引数が書き込みたいテキスト，第 2 引数が配置する位置である．btex で始まり etex で終わる部分には，TEX 形式でテキストを書くことができる．また，label.*pos* という文法で，テキストを配置する位置を指定できる．ここで，*pos* として指定できるものは次の 8 通りである．



btex で始まり etex で終わる部分は，リテラル（定数）である．したがって，

```
String text; text := "12345";
label(btex text etex, (0,0));
```

としても，text としか表示されない．プログラムで動的に生成した文字列をもとにテキストを書き込むには，次の例のように標準で入っている TEX マクロを利用する．まずプログラムの先頭で

```
input TEX;
```

としてマクロを取り込む．その上で，

```
String text; text := "12345";
label(TEX(text), (0,0));
```

とすればよい．TEX マクロは，コンパイル時に動的に btex で始まり etex で終わる文字列を生成し，それを当該個所に読み込む．動的に TEX 形式のラベルを書き込むことができるが，コンパイルが相当に遅くなってしまうことに注意が必要である．

さらなる利用のために

日本語と LATEX コマンドの利用

「MetaPost の初歩」において説明した 2 つのコマンドでは，上記で説明したラベルにおいて多バイト文字を含まない TEX 形式のテキストのみしか扱えない．実際の利用にあたっては，日本語の利用と LATEX コマンドの利用が望まれる．

日本語を含む MetaPost プログラムをコンパイルするには，pmpost というコマンドを利用する．日本語を扱う場合には文字コードの問題が発生し得るので，各環境でどの文字コードを使えばよいか試行錯誤が必要かもしれない☆1．

次に LATEX コマンドの利用であるが，これは少し面倒である．btex で始まり etex で終わる文字列を処理するプログラム（ここでは platex とする）と，その文字列の前に置く必要のある LATEX ソース断片を指定するため，MetaPost プログラムの先頭に次のように書く．

```
verbatimtex
%&platex
\documentclass[10pt]{article}
\begin{document}
etex
```

文字を共通してサンセリフ体としたい場合には，上の \begin{docuemnt} の後に \sf と書いてお

☆1 筆者は Windows 上で pMetaPost 1.999-0.04（TeX Live2016/W32TeX/dev）を用いているが，簡単なテストプログラムでは Shift-JIS，iso-2022-7bit，UTF-8 のいずれでも動いた．

けばよい．%&platex 上記の指定により，コンパイルの際に platex が利用されるはずである．もし，

```
fatal: Command failed: eptex --parse-first
  -line --interaction=nonstopmode mpa04552
  .tex; see mpxerr.log
```

のように別のコマンドが利用されている場合には，次のように環境変数 TEX を指定して実行する必要がある．

```
TEX=platex pmpost sample.mp
```

TEX マクロを用いる場合には，テキストごとに LATEX のヘッダ部分が必要なので，input TEX; の後に次のように（実際には 1 行で）書いておく．TEXPRE は TEX.mp において定義されているマクロである．

```
TEXPRE "\documentclass[10pt]{jarticle}
\begin{document}";
```

Gnuplot の出力の加工

Gnuplot において

```
set terminal mp
```

とすると，Gnuplot の出力を MetaPost のソースとして得ることができる．たとえば，1 セットのデータを直線と点でプロットした場合，176 行の MetaPost プログラムが出力される．ここでは，よくある編集についてその方法を示す．

1 つ目の編集はラベルの置き換えである．これは，MetaPost プログラムから置き換えたい文字列を探して書き換えればよい．たとえば，縦軸に対して—x-label—を x-label としたい場合には該当する文字列を探して

```
put_text(btex $y$-label etex, 27.1a,
          400.8b, -270, 2);
```

とする．put_text は，テキスト，x 座標，y 座標，配置位置を表す整数を引数にとる．この例で分かるように，x 座標と y 座標はそれぞれ変数 a と b を単位としており，それらの変数は

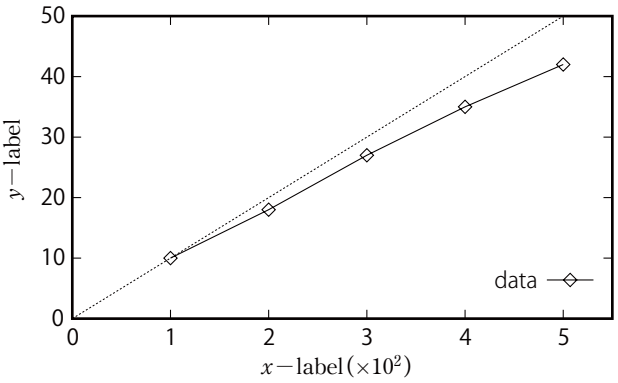
```
w:=5.000in;h:=3.000in;
a:=w/1200.0;b:=h/720.0;
```

と定義されている．これらを書き換えることで，グラフの大きさを変更することができる．

2 つ目の編集は線の変更である．出力された MetaPost プログラムでは，まず外枠とその目盛が描かれた後で，グラフの各線や点が描かれる．したがって，プログラムの下のほうから探せば該当する draw または gpdraw を探すことができる．線種・点種は，linetype もしくは gpdraw の第 1 引数によって指定される．これらを変更することにより，線種や色などを変更することができる．線種や色をより細かく変更したい場合には，lt[] または col[] の定義を書き換えればよい．

もちろん，MetaPost のプログラムとして書くことにより，これら以外にも任意の図やテキストを書き込むことができる．

次の図は，Gnuplot から生成された MetaPost ソースを少し修正して生成したグラフである．ラベルに LATEX の文字列を使うことに加え，フォントサイズの変更，線の太さの調整，斜めの線の追加などを行ってみた．



PowerPoint などでの利用

論文は LATEX で書くがスライドには PowerPoint を使うという人（筆者もその 1 人である）は，論文で書いた図をどうやって PowerPoint で用いるか悩んだこともあるだろう．MetaPost では，EPS 形式または PDF 形式で図が出力され，これらはベクター形式（拡大縮小しても品質が劣化しない）である．そのベクター形式の利点を残したまま PowerPoint で利用する方法について紹介する．

PowerPoint で扱うことができるベクター形式に，WMF（Windows Meta File）形式がある．MetaPost

で生成した EPS 形式の図を WMF 形式に変換できれば，ベクター形式で図を取り込むことができ，さらに，PowerPoint 上で編集することもできる．問題はその変換を行うためのツールである．その 1 つは LaTeX 環境に含まれる pstoedit である．もう 1 つは，オンラインのサービスを利用することである．たとえば，CloudConvert[☆2] では，EPS 形式のファイルをアップロードし，WMF 形式にてダウンロードすることができる．筆者が試した範囲では，曲線を含むような図について，CloudConvert のほうが美しい画像を生成するようである．

さらなる図を描くプログラミング

本稿で紹介した MetaPost は，LaTeX で論文を書いている人にお薦めの図を描くプログラミング言語である．論文のための図を描くのに十分な機能を持っており，正確できれいな図を出力することができる．プログラミング言語として見ると，その文法は（繰り返しやマクロなど）多少癖があるが，等式を自動で解くソルバの利用などはほかのプログラミング言語とは大きく異なる面白さもある．

☆2 https://cloudconvert.com/eps-to-wmf

さらに MetaPost プログラミングをやりたい人向けの情報を 2 つ示そう．1 つ目は Hobby によって書かれた『METAPOST A User's Manual』[1] である．MetaPost において扱うことができるすべての演算子やマクロのほか，マクロの定義のやり方の細かなところまで書かれている．2 つ目は，AndréHeck によるチュートリアル[2] である．これには，多数の演習問題が含まれており，それを解くことで MetaPost を学べるようになっている．

プログラムで生成されたデータを可視化するのに MetaPost を用いる場合には，各言語の MetaPost ライブラリを利用すると便利かもしれない．たとえば，筆者が調べた範囲では metapost-erb (Ruby), funcmp (Haskell)，mlpost (Objective Caml) などがあるようである．

参考文献

1）John, D. H. and The MetaPost Development Team : METAPOST A User's Manual (2014), https://www.tug.org/docs/metapost/mpman.pdf

2）André, H. : Learning METAPOST by Doing (2005), https://staff.science.uva.nl/a.j.p.heck/Courses/mptut.pdf

（2017 年 4 月 10 日受付）

松崎公紀（正会員）■ matsuzaki.kiminori@kochi-tech.ac.jp

2005 年東京大学大学院情報理工学系研究科中途退学，2009 年より現職，博士（情報理工学）．プログラミング手法，特に高水準並列プログラミング，およびゲーム情報学に関心を持つ．