第九部分

METAPOST使用说明

1 简介

METAPOST是一个基本的作图工具，适合做出精确的图形，输出为ps格式的矢量图形，可供TEX中的插图使用。使用METAPOST的基本步骤如下：

1. 编辑METAPOST源文件。本质就是一个文本编辑的过程。
2. 把源文件提交给METAPOST的处理程序mpost.exe编译。
3. 将编译产生的输出文件在TEX文档中使用，并编译，图形会在dvi文件转换为ps或者pdf时出现。

更详细的说明请参考 [1]。以下是一个简单的METAPOST程序，你可以看看它的结果：

```
beginfig(0)
path pp ;
u := 2cm ;
Angle := 10 ;
n = 360 / Angle ;
pp := (-u ,-u )--(-u ,u )--(u ,u )--(u ,-u )-- cycle;
draw pp ;
for i = 1 upto n :
    pp := pp scaled (1/(sind(Angle)+cosd(Angle))) rotated Angle ;
    draw pp ;
endfor;
endfig ;
end
```

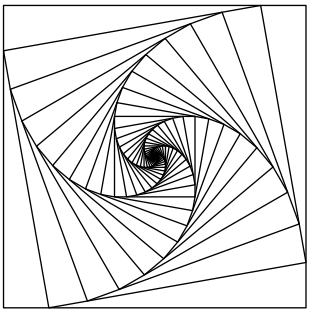

图 1: 程序输出样例

2 基本语法元素

在METAPOST里面共有以下几种元素，附有说明：

numeric 与强语言不同，这里只有一种数值形式，最大不能超过4096。

pair 表示平面上面的点。使用(x,y)来定义或者直接使用。

color 表示一种颜色，常用(r,g,b)来表示一种颜色。

path 表示一条路经，可以使用点加连接方式来表示，如：(a,b)−−(c,d)、(a,b)..(c,d)等等。

picture 表示一个图形集合，后面将介绍如何使用它做一些变换。

transform 表示一种变换方式，可以作用在pair、path、picture上面，获取变换后的图形。

string 字符串，常使用"string example"来表示。

boolean 一个bool值，和**if**等一起使用。

pen 表示画图时笔的形状。

其中可以认为，**pair**、**numeric**、**color**这三种是为了表达某一种状态，或者位置的值，可以进行通常以以下的运算。而**path**和**picture**则稍复杂，可以认为是图形的元素集合，transform则为对这些的变换方式（本质上由几个**numeric**组成）。象**pen**更像是一个**path**。

3 pair与path

对于一个**numeric**变量，可以进行常规的数值运算，还有几个辅助的函数，如三角函数**sind**、**cosd**等。而对于**pair**和**color**可以进行线性方程组运算。最简单的是比如求交点：

```
beginfig(0)
u := 2cm ;
pair a ;
draw ( 0u , 0u ) -- ( u , u ) ;
draw ( u , 0u ) -- ( 0u , 1u ) ;
a = whatever[( 0u , 0u ) , ( u , u )] ;
a = whatever[( u , 0u ) , ( 0u , 1u )] ;
dotlabel.bot ( "a" , a ) ;
endfig ;
```

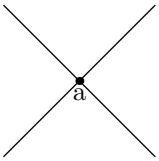

图 2: 分点示意

其中的**whatever**表示一个无名的变量，这样我们知道a是一个两直线的公共点。而结构[p1,p2]可以应用到**color**上面，表示分点的意思，而前面的数字即分比。

多个点之间采用不同的连接方式，就可以得到**path**，基本连接方式为：

path pp ;

pp := point1−−point2−−...−−pointn[−−**cycle**]

其中−−可以换为 .. 。−−表示用直线连接，而 .. 表示用尽量光滑的曲线连接（一般使用Bézier曲线）。最后可选的**cycle**表示需要闭合。由于使用了Bézier曲线，所以可以使用控制点来画，此时的点的表达式换为**control**(x1,y1) **and** (x2,y2)或者**control**(x,y)。

点还可以加上方向，方式为point{**dir**}，dir可以为**left**、**right**、**up**、**down**。

还可以使用下面的语法强制该点的斜率point{**dir** a}，a是角度值。

另外，能为曲线增加紧张程度的描述，即使用point1 ..**tension** a..point2，a的大小描述了紧张程度，$a \geqslant \frac{3}{4}$。或者point1 ..**tension** a **and** b..point2。

最后，起点和终点可以使用curl来表示卷曲程度，如point1{**curl** c}。

另外，使用METAPOST自己定义的一些宏也能够产生相应的路径，如：**fullcircle**、**halfcircle**。

4 picture与transform

所谓的**picture**是一个所有图形对象的集合，有一个标准的**picture**是当前使用于输出的**picture**，也就是**currentpicture**，可以通过对它的操作改变当前图片，如：

currentpicture := **nullpicture** ;

会把当前图片清除。

我们可以对图形作出很多变换，当然，这些变换也适用于**pair**和**path**，下面列出一些常用的变换：

scaled 按比例放大，后接放缩因子。
xscaled 按比例放大，后接放缩因子。
yscaled 按比例放大，后接放缩因子。
rotated 旋转，后接旋转角度。
shifted 平移，后接平移点的位置。
rotatedaround 绕指定点旋转。

利用好**picture**可以方便的进行迭代，下面的代码将会生成Sierpinski垫片，这是一个著名的分形例子：

```
beginfig(3);
IteralTime := 7 ;
u := 5cm ;
picture picTemp ;
path triangle ;
triangle := ( 1u , 0u ) -- ( 0u , u*sqrt( 3 ) ) -- ( -1u , 0u ) -- cycle ;
filldraw  triangle ;
picTemp := currentpicture scaled 0.5 ;
for i := 1 upto IteralTime :
    currentpicture := nullpicture ;
    draw picTemp shifted ( -0.5u , 0u ) ;
    draw picTemp shifted ( 0.5u , 0u ) ;
    draw picTemp shifted ( 0u , 0.5 * u * sqrt(3) ) ;
    picTemp := currentpicture scaled 0.5 ;
endfor;
endfig;
```

值得注意的是，有时候我们需要在METAPOST图片中插入TEX字符，这实际上也是一种**picture**。如下面代码将返回一个**picture**。

```
picture pp ;
pp := thelabel( btex \TeX's Output etex , ( 0 , 0 ) ) ;
```

其中的代码块**btex** \TeX's Output **etex**就是一种**picture**。

我们通常更愿意使用LATEX插入字符，甚至中文字符，下面的例子将为我们说明这一点：

```
verbatimtex
%&latex
\documentclass{article}
\usepackage{CJK}
\begin{CJK*}{GBK}{song}
\begin{document}
etex
beginfig(2)
draw fullcircle scaled 2cm ;
label( btex 圆心 etex , ( 0, 0 ) ) ;
endfig ;

verbatimtex
\end{CJK*}
\end{document}
etex
```

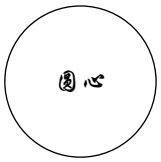

图 3: LATEX插入中文示例

注意在**beginfig**与**endfig**前后的那两段，使表示使用LATEX做预处理，预处理之后，就会形成更低层的ps代码供后面的METAPOST使用。

5 数组

数组是把相同的数据类型放在一起，这样形成的数据结构。在METAPOST使用数组很方便，如下面的声明可以产生一个数值类型的数组：

```
numeric p[] ;
```

值得注意的是，METAPOST里面不允许定义数组的大小，声明之后，到使用类似p1的时候，自然成为该数组的成员。即使是p3.1 也是该数组的成员。我们可以在一个循环里面使用数组，或者储存数据，或者计算。

比如，我们相制作绳子上面的两个波相遇的图片，可以把绳子想象成为许多段，然后通过对每一个节点的计算获得某一时刻的波形图，那么这里需要储存数据，自然而然的将由我们的数组来储存。下面是该例的代码：

```
newinternal mu , nn , tl , A , stp , edp ;
mu := 1mm ;
nn := 16 ;
tl := 100 ;
A := 10mu ;
stp := 1 ;
edp := tl+1 ;

for j := 0 upto ( tl − nn ) :
    beginfig(j)
    path pp ;
    numeric rp[] ;
    for i := stp upto edp :
        rp[i ] := 0 ;
    endfor ;
    stwp1 := j ;
    stwp2 := tl−nn−j;
    for i := 1 upto nn :
        rp[(stwp1+i)] := rp[stwp1+i] + A∗sind(i/nn∗360) ;
        rp[(stwp2+i)] := rp[stwp2+i] + A∗sind(i/nn∗360) ;
    endfor ;
    pp := (stp∗mu , rp[stp] )
    for i := stp+1 upto edp :
        .. ( i∗mu , rp[i ] )
    endfor ;
    draw pp ;
    setbounds currentpicture to (stp∗mu,−2A)−−(edp∗mu,−2A)−−(edp∗mu,2A)−−(stp∗mu,2A)−−cycle;
    endfig ;
endfor ;

end
```

6 宏定义

有几种定义宏的方式。最常见的一种，类似于函数，基本格式见下：

```
vardef macro_name( expr args ) =
    doings ;
enddef ;
```

我们这里给出一个画多边形的宏，见下：

```
vardef polygon ( expr n ) =
    save edge , i ;
    path edge ;
    edge := ( 1 , 0 )
        for i := 1 upto n-1 :
            -- ( cosd(360/n*i) , sind(360/n*i) )
        endfor ;
    edge := edge -- cycle ;
    edge
enddef ;
```

其中的**save**表示后面的变量为宏的内部变量，由此我们知道，没有这一句话，宏内更改的变量将会影响到调用宏的代码部分中的同名变量。宏体的最后一行表示宏的返回值。

另外，还有多种定义方式，如：

```
def macro_name( expr args ) =
    doings ;
enddef ;
```

7 结构控制指令

下面介绍一些结构控制语句：

for 一般具有下面的形式：

```
for counter := startNumber upto endNumber :
    Doings ;
endfor;
```

也有这样用的：

```
for counter := startNumber step stst to endNumber :
    Doings ;
endfor;
```

我们还可以通过他获得类似while的循环类型，如下：

```
forever :
    Doings ;
    exitif something ;
endfor;
```

if 用于选择，语法如下：

```
if somthing true or false :
    Doings ;
else :
    Doings ;
endif ;
```

或者，

```
if something true or false :
    Doings ;
elseif true or false :
    Doings ;
else :
    Doings ;
endif ;
```

8 基本绘图宏

下面简单介绍一些绘图命令：

draw 利用这个命令可以画出很多复杂的图形。基本格式为

drawarrow 这个命令主要用于绘制带箭头的线段，如：

```
u := 1cm ;
drawarrow ( 0 , 0 ) -- ( 2u , 0 ) ;
drawarrow ( 0 , 0 ) .. ( u/2 , u/2*sqrt (3) ) .. ( u , u ) ;
```

undraw 就是用背景色画。

label 可以在图片里面加上标签，基本格式为：

```
label.pos("String",point)
```

其中的pos为放置标签相对于点的位置，可以取top、bot、lft、rt、llft、ulft、lrt、urt。没有的时候表示居中。

dotlabel 与**label**相似，只是在该点使用点标明。

infont 用于在指定位置加入标签，但是这是最原始的命令。基本用法如下：

```
label("text" infont defaultfont scaled defaultscale, z0)
```

字符串可以使用char <**numeric** primary>，这是直接调用当前字体下面的字符。

fill 使用颜色填充指定区域，语法见下：

```
fill <path expression> withcolor <color expression>
```

color可以为0.3**white**。

unfill 就是用背景色填充。

fullcircle 获得圆路径，一般使用以下语法：

```
p = fullcircle scaled 2cm shifted point ;
```

其中，**scaled**后面是圆半径，而**shifted**后面是圆心。

harfcircle 获得半圆路径，一般使用以下语法：

```
p = halfcircle scaled 2cm ;
```

buildcycle 用于获得封闭路径，一般使用的语法如下：

```
p = buildcycle ( p1 , p2 ) ;
```

其中p1与p2表示两相交的路径。

defaultscale 设置字体放缩比例。

pickup pencircle scaled 4pt 表示设置画笔样式为圆形并放大到4pt。

cutbefore 在某路径前截断，如：

```
p := p1 cutbefore p2 ;
```

会把路径p1在p2前截断，赋值给p。

cutafter 类似cutbefore，但是取截断的后面一部分。

decimal 将数值变量化为对应的字符串，如：

```
u := 4mm ;
drawarrow ( 0 , 0 ) -- ( 7u , 0 ) ;
for i := 0 upto 6 :
    dotlabel.bot ( decimal i , ( i*u  , 0 ) ) ;
endfor ;
```

会得到一条有“刻度”的数轴。

direction of 计算路径上面某点的方向，比如说：

```
verbatimtex
%&latex

\documentclass{article}
\usepackage{CJK}
\begin{CJK*}{GBK}{song}
\begin{document}
etex
beginfig(3)
u := 2cm ;
numeric t ;
path p ;
pair z ;
drawarrow ( 0 , 0 ) -- ( 1.5u , 0 ) ;
drawarrow ( 0 , 0 ) -- ( 0 , 1.5u ) ;
p := ( 0 , 0 ) .. for i := 1 upto 3 : ( i/4*u , ((i/4)**2)*u ) .. endfor ( 1u , 1u ) ;
draw p ;
t := 3 ;
dotlabel.lrt( btex 切点 etex , point t of p  ) ;
z = whatever [ ( 0 , 0 ) , ( 1.5u , 0 )] ;
z - point t of p = whatever * direction t of p ;
draw z  -- point t of p ;
```

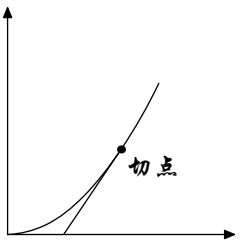

图 4: direction of示例

```
endfig ;

verbatimtex
\end{CJK*}
\end{document}
etex
```

directionpoint of 和上面的相似，就是去路径上面方向为给出方向的点的对应在路径上的参数，用法如下：

numeric t ;
path p ;
p := (0 , 0) .. **for** i := 1 **upto** 3 : (i/4∗u , ((i/4)∗∗2)∗u) .. **endfor** (1u , 1u) ;
t := directionpoint (1/2 , 1) of p ;

*****part** 例如**xpart**，**ypart**，还有**redpart**，**bluepart**，**greenpart**，可以取出**pair**或者**color**对应的分量。
dashed 本质上面该命令产生一个图片，用于画出虚线等各种线性。例如：

draw p **dashed** withdots **scaled** 2 ;
draw p **dashed** evenly **scaled** 2 ;

dashpattern 用于产生**dashed**可以使用的线性，常常这样定义：

draw p **dashed dashpattern**(**on** 4pt **off** 4pt **on** 4pt)

9 常用运算宏

sind,cosd 计算角度的三角函数。
dotprod 计算两点的点积。
div 整数除法。
length 取得路径的长（指参数表示的最大值）。

10 常用内部变量

currentpicture 使用的例子见前面。
linecap 设置线的终结方式，可取**rounded**、**squared**、**butt**。
linejoin 设置线的接头方式，可取**mitered**、**rounded**、**beveled**。
miterlimit 控制**mitered**下线延伸的长度。

11 几个小例子

画数据点的例子，其中fern.dat是数据文件，一行表示一个数据点。

beginfig(0)
draw begingraph (4**cm** , 4**cm**) ;
gdraw ”fern.dat” plot **btex** $\cdot$ **etex** ;
Gmarks := 1 ;
endgraph ;
endfig ;

Peano曲线的作法，使用了迭代：

```
beginfig(0)
u := 2cm ;
IteralTime := 4 ;
path p , pp ;
p := ( −u , −u ) −− ( −u , u ) −− ( u , u ) −− ( u , −u ) ;
p := p scaled 0.5 ;
for i := 1 upto IteralTime :
    pp := reverse p rotated −90 shifted ( −u , −u ) ;
    pp := pp −− p shifted ( −u , u ) ;
    pp := pp −− p shifted ( u , u ) ;
    pp := pp −− reverse p rotated 90 shifted ( u , −u ) ;
    p := pp scaled 0.5 ;
endfor ;
draw pp ;
setbounds currentpicture to ( −2u , −2u ) −− ( −2u , 2u ) −− ( 2u , 2u ) −− ( 2u , −2u ) −− cycle ;
endfig ;
```

想想这段代码怎么改成追光灯？？

```
verbatimtex
%&latex
\documentclass{article}
\usepackage{mflogo}
\begin{document}
\Huge
etex

beginfig(0)
picture myp ;
path pc ;
myp := thelabel( btex Welcome to \MP etex , ( 0 , 0 ) ) ;
pc := fullcircle scaled ((ypart ulcorner myp)−(ypart llcorner myp)) ;
 fill  bbox myp ;
unfill  pc ;
draw myp ;
endfig ;

verbatimtex
\end{document}
etex

end
```

12 结束语

METAPOST是一种强大的绘图语言，当你遇到困难的时候不要气馁，一般来说理解了上面的基本语法，以及基本宏，就能够做出比较复杂的图形了，但是还需要多多了解其它的宏，这可以在 [1]里面找到详细的说明，另外 [2]中有大量的例子，是试验的好材料。

参考文献

[1] John D. Hobby:A User's Manual for METAPOST

[2] André Heck:Learning METAPOST by Doing