MetaPost Tutorial

Study group material prepared by Adrian

21st January 2005

1 Background

- Developed by John Hobby in AT&T Bell Labs, base on Donald Knuth's Metafont language
- Web site: http://cm.bell-labs.com/who/hobby/MetaPost.html
- Papers and documentations:
  - John D. Hobby, *A METAFONT-like System with PostScript Output*, Tugboat, the TEX User's Group Newsletter, 10(4), 1989.
  - **John D. Hobby, *A User's manual for MetaPost,* AT&T Bell Laboratories Computing Science Technical Report 162, 1992. (just 87 pages)**
  - John D. Hobby, *Introduction to MetaPost,* Proceedings of EuroTEX '92, 1992.
  - John D. Hobby, *Drawing Graphs with MetaPost,* AT&T Bell Laboratories Computing Science Technical Report 164, 1992.

2 Concept

- Build in feature of most LATEX packages
- Two-dimensional Cartesian coordinate system, (x,y)
- Way of use:
  1. Write a plain text file describing the picture (e.g. picture.mp)
  2. Compile to give output in encapsulated postscript format
     `$ mpost picture.mp`
  3. Rename the picutre
     `$ mv picture.1 picutre.eps`
  4. Include it in LATEX such as:

```
\usepackage{graphicx}
....
\includegraphics[width=1.0\columnwidth,keepaspectratio]{picture.eps}
```

- If you want MetaPost inside the LATEX source, you can try "EMP" or encapsulated MetaPost

3 Basics

- Begin each figure with `beginfig(`*n*`)` and end each figure with `endfig`
  - *n* is the figure number. Where each output EPS figure will be stored as `filename.`*n*
- You can have multiple figures in a file
- End the whole file with the line end

- Example:

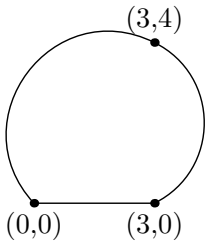
```
beginfig(101);
draw fullcircle scaled 2cm;
endfig;
beginfig(102);
draw fullcircle scaled 2cm xscaled 1.5;
endfig;
end
```

- Metapost draws on a canva and it knows all TEX units
  - Centimeter (cm), Millimeter (mm), Points (pt), Picas (pc), Inches (in), ...
  - Maximum size it can handle: $\pm$4096pt $\times$ $\pm$4096pt (around $\pm$5 feet $\times$ $\pm$5 feet)
- Data types
  - `numeric`, e.g. 1, 1.5, 2
  - `pair` (coordinates), e.g. (-10,10)
  - `path` (lines and curves)
  - `transform`, e.g. scaling, rotating, shifting
  - `string`, double quoted
  - `boolean`, construct by `true`, `false`, `and`, `or`, `not`, `=`, `<>`, `<`, `<=`, `>`, `>=`
  - `color`, `picture`, `pen`
- Declare your data type before use, or MetaPost will guess what it is (maybe wrong)
- Usually coordinates are represented as `z`*n* where *n* can be any number, e.g. `z0`, `z1`, `z2`
  - Once declared, `x`*n* and `y`*n* correspond to its *x*-coordinate and *y*-coordinate immediately!
- Operators
  - Calculations: `+`, `-`, `*`, `/`, `**` (exponential), `-` (negative), `++` ($\sqrt{a^2+b^2}$), `+-+` ($\sqrt{a^2-b^2}$)
  - String concatenation: `"abc" & "def"`
  - Substring: `substring (0,1) of "abcdef"`
  - Square root: `sqrt(2/3)`
  - Mediation: `0.5[6,7]` equals to `6.5`, `0.5[(0,0),(4,4)]` equals to `(2,2)`
  - x/y coordinate: `xpart (0,0)`, `ypart (0,0)`

4 Drawing lines and curves

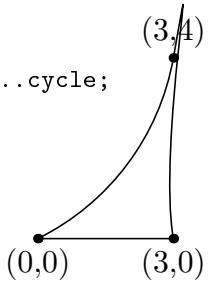
- Lines: `--`, Curve: `..`
- Example:

```
beginfig(101);
draw (0,0)--(3cm,0)..(3cm,4cm)..cycle;
dotlabel.bot("(0,0)",(0,0));
dotlabel.bot("(3,0)",(3cm,0));
dotlabel.top("(3,4)",(3cm,4cm));
endfig;
end
```



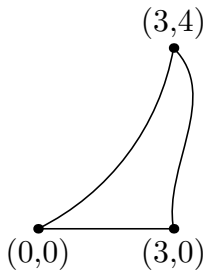
- Another example: Directions of curves

```
beginfig(102);
draw (0,0)--(3cm,0){dir 100}..(3cm,4cm){dir -100}..cycle;
dotlabel.bot("(0,0)",(0,0));
dotlabel.bot("(3,0)",(3cm,0));
dotlabel.top("(3,4)",(3cm,4cm));
endfig;
end
```



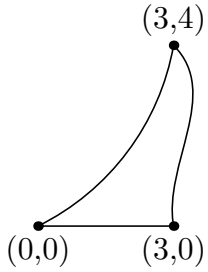
- A bit ugly at (3,4). So more mandatory directions added:

```
beginfig(103);
draw (0,0)--(3cm,0){dir 100}..{dir 135}(3cm,4cm){dir -100}..cycle;
dotlabel.bot("(0,0)",(0,0));
dotlabel.bot("(3,0)",(3cm,0));
dotlabel.top("(3,4)",(3cm,4cm));
endfig;
end
```



- Actually, repeating "cm" is so inconvenent, so we do this:

```
numeric u;
u:=1cm;
beginfig(103);
draw (0,0)--(3u,0){dir 100}..{dir 135}(3u,4u){dir -100}..cycle;
dotlabel.bot("(0,0)",(0,0));
dotlabel.bot("(3,0)",(3u,0));
dotlabel.top("(3,4)",(3u,4u));
endfig;
end
```



- Points to note:

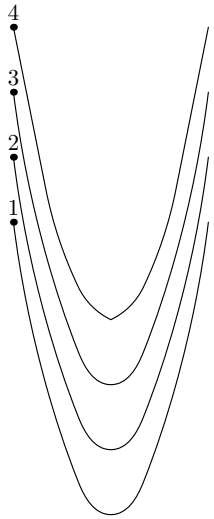
1. We can control how curve the line is, or MetaPost will make the decision for us
2. Direction of curves are expressed in degrees, and +ve degree means clockwise
3. At "emitted" end of curve, 0 degree is at the +ve *x*-axis;
   At "incident" end of curve, 0 degree is at the -ve *x*-axis.
4. We declared variables and use it (not necessary, but avoids problems)
5. Assignment of values to variables: `var:=value`
6. For convenience, MetaPost defined
   - `left` = `dir 180`
   - `right` = `dir 0`
   - `up` = `dir 90`
   - `down` = `dir -90`
7. `cycle` means to close the loop
8. We used `dotlabel` to draw labels with a dot at some coordinate. The label text is placed at the bottom (`bot`), top (`top`), left (`lft`), right (`rt`), upper left (`ulft`), upper right (`urt`), lower left (`llft`), or lower right (`lrt`). Labels without placement is centered at the coordinate. Labels without dot can be created using `label` with similar syntax.

5 Plotting graphs

- MetaPost supports loops
  - Example 1:
    `draw (0,0)..  for i=1 step 1 until 5:  ..(i,i**2); endfor;`
  - Example 2:
    `draw (0,0)..  for i=1 upto 5:  ..(i,i**2); endfor;`
  - Example 3:
    `draw (0,0)..  for i=1,2,3,4,5:  ..(i,i**2); endfor;`
  - Example 4:
    `forever:  exitunless i>=5; draw (i,i**2)..(i+1,(i+1)**2); i++; endfor`

- Example:

```
beginfig(104);
path pict[];
pict1:=(-3u,9u) for i=-3 step 1 until 3: ..(i*u,i**2*u) endfor;
pict2:=(-3u,9u) for i=-3 upto 3: ..(i*u,i**2*u) endfor;
pict3:=(-3u,9u) for i=-3,-2,-1,0,1,2,3: ..(i*u,i**2*u) endfor;
i:=3;
pict4:=(3u,9u);
forever:
    pict4:=pict4..(i*u,i**2*u);
    i:=i-1; exitif i<0;
endfor;
i:=-3;
pict5:=(-3u,9u);
forever:
    pict5:=pict5..(i*u,i**2*u);
    i:=i+1; exitif i>0;
endfor;
pict4:=pict5..reverse pict4;
for i=1 upto 4:
    draw pict[i] shifted (1cm,i*cm);
    dotlabel.top(char(ASCII("0")+i),(-3u,9u) shifted (1cm,i*cm));
endfor;
endfig;
```



- We used loops to calculate the coordinate of different points, and make a smooth curve to join them
- Something to note:

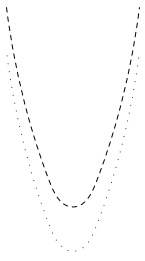
1. `forever` loop need complete statements, hence we cannot use it to build a curve like examples 1-3
2. we demonstrated how to use arrays in MetaPost
3. we demonstrated how to use path variable to store a path. The way to call an element of an array can be `pict1`, `pict.1`, or `pict[1]`.
4. we used the function `reverse` to reverse a path, as in `pict4:=pict5..reverse pict4`
5. we used `ASCII("0")` to get the ASCII number of character "0" and used `char()` to get a character by the ASCII number
6. we used `shifted (x,y)` to move the whole picture left `x` coordinates and up `y` coordinates

- Actually, we are not limited to shifted, we can have: (better to collect them into a `transform` variable)

$$
\begin{aligned}
(x,y)\ \texttt{shifted}\ (a,b) &= (x+a,y+b)\\
(x,y)\ \texttt{slanted}\ a &= (x+ay,y)\\
(x,y)\ \texttt{scaled}\ a &= (ax,ay)\\
(x,y)\ \texttt{xscaled}\ a &= (ax,y)\\
(x,y)\ \texttt{yscaled}\ b &= (x,by)\\
(x,y)\ \texttt{zscaled}\ (a,b) &= (ax-by,bx+ay)\\
(x,y)\ \texttt{rotated}\ \theta &= (x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta)\\
(x,y)\ \texttt{reflectedabout}\ (p,q) &= \text{reflection}\\
(x,y)\ \texttt{rotatedaround}\ (p,q) &= \text{rotation}
\end{aligned}
$$

- If you want dots instead of solid lines, you can have it in this way (see `transformed` is used as well):

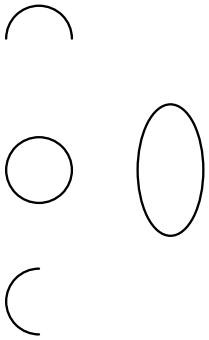
```
beginfig(105);
path pict;
transform T;
T:=identity shifted (3,9) scaled u;
pict:=(-3,9) for i=-3 step 1 until 3: ..(i,i**2) endfor;
draw pict transformed T dashed withdots;
draw pict transformed T shifted (0,1cm) dashed evenly;
endfor;
endfig;
```



6 Circles and squares

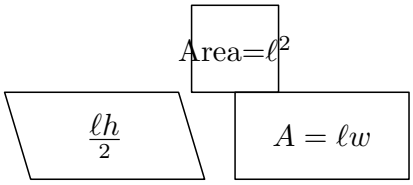
- Full circle and half circles

```
beginfig(106);
transform circle,ellipse,semi,rotsemi;
circle:=identity scaled u;
ellipse:=identity shifted (2,0) scaled u yscaled 2;
semi:=identity shifted (0,2) scaled u;
rotsemi:=identity rotated 90 shifted (0,-2) scaled u;
draw fullcircle transformed circle;
draw fullcircle transformed ellipse;
draw halfcircle transformed semi;
draw halfcircle transformed rotsemi;
endfig;
```



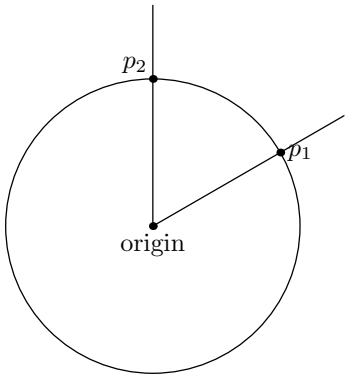
- Please remember to `rotate` before you `shifted` or `scaled`!
- Sorry, we don't have squares or rectangles. Please write your own.

```
beginfig(107);
path square; % square with side length 1 pt
square:=(-0.5,-0.5)--(-0.5,0.5)--(0.5,0.5)--(0.5,-0.5)--cycle;
draw square scaled 2u;
draw square scaled 2u xscaled 2 shifted (2u,-2u);
draw square scaled 2u xscaled 2 slanted -0.3 shifted (-3u,-2u);
label(btex Area=$\ell^2$ etex, (0,0));
label(btex $A=\ell w$ etex, (2u,-2u));
label(btex $\displaystyle \ell h \over 2$ etex, (-3u,-2u));
endfig;
```



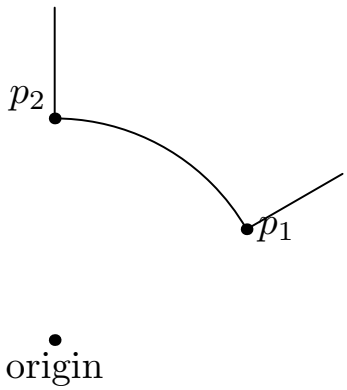
- In the above, we've shown
  1. If you want plain text, you can just use a string (double quoted)
  2. If you want something complicated, use TEX and place them between `btex` and `etex`
  3. In TEX, no display equation is supported, if you want the fractions prettier, use `\displaystyle`
  4. Save some figures, e.g. `square`, and then you can use it many times
  5. Sometimes, if you just want a square enclosing some text, use this method:
     ```
     picture text;
     text:=thelabel("Hi",(1cm,1cm));
     draw text;
     draw bbox text;
     ```
     where `thelabel` is same as label without actually drawing it and `bbox` is a function to get the enclosing rectange of a picture
     - To box everything drawn, try `draw bbox currentpicture`
- How about arcs? Let's see

```
beginfig(108);
path circle, line[];
circle := fullcircle scaled 4u;
line1 := (0,0)--(1,0) rotated 30 scaled 2.5u;
line2 := (0,0)--(1,0) rotated 90 scaled 2.5u;
draw circle; draw line1; draw line2;
dotlabel.bot("origin",(0,0));
dotlabel.rt(btex $p_1$ etex, line1 intersectionpoint circle);
dotlabel.ulft(btex $p_2$ etex, line2 intersectionpoint circle);
endfig;
```



- We can actually use `intersectionpoint` to find the intersections! So we can use this cue to *build* an arc

```
beginfig(109);
path circle, line[];
circle := fullcircle scaled 4u;
line1 := (0,0)--(1,0) rotated 30 scaled 3u;
line2 := (0,0)--(1,0) rotated 90 scaled 3u;
pair arc[];
arc1 = circle intersectiontimes line1;
arc2 = circle intersectiontimes line2;
draw subpath(xpart arc1,xpart arc2) of circle;
draw subpath(ypart arc1,length line1) of line1;
draw subpath(ypart arc2,length line2) of line2;
dotlabel.bot("origin",(0,0));
dotlabel.rt(btex $p_1$ etex, point xpart arc1 of circle);
dotlabel.ulft(btex $p_2$ etex, point ypart arc2 of line2);
endfig;
```

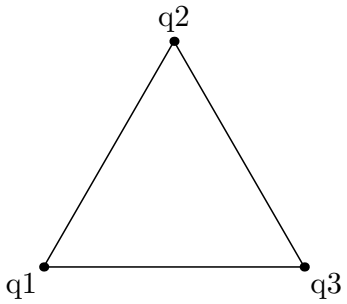


- Points to note:

  1. `intersectiontimes` gives the "time" pair, where first and second value correspond to first and second path respectively
  2. time can be regarded as the parameter of the position along a path
  3. `length` of a path is the time at its endpoint. Starting point's time is always 0.
  4. `subpath(a,b) of path` gives the partial curve from position `a` to position `b`
  5. If no intersection can be found, `intersectiontimes` gives (-1,-1)

- Actually, we can use similar technique to draw equilaterial triangles:

```
beginfig(110);
path p[];
p1=(0,0)--(1,0);
p2=((0,0)--(1,0)) rotated 60;
p3=((0,0)--(1,0)) rotated 120 shifted (3cm,0);
pair q[];
q11=point 0 of p1;
q12=point length p1 of p1;
q21=point 0 of p2;
q22=point length p2 of p2;
q31=point 0 of p3;
q32=point length p3 of p3;
q1=whatever[q11,q12]=whatever[q21,q22];
q2=whatever[q21,q22]=whatever[q31,q32];
q3=whatever[q31,q32]=whatever[q11,q12];
draw q1--q2--q3--cycle;
dotlabel.llft("q1",q1);
dotlabel.top("q2",q2);
dotlabel.lrt("q3",q3);
endfig;
```



- Definitely, it is not to most compact way to draw a equilaterial triangle. But you can see how great MetaPost is!

  – `q1=whatever[q11,q12]=whatever[q21,q22]`
  means to define `q1` as the point of intersection between `q11--q12` and `q21--q22`

  – The easiest way to draw a equilaterial triangle is

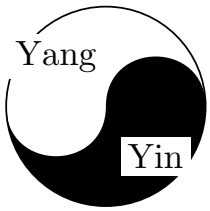
```
path p[], triangle;
p1:=(0,0)--(1,0);
p2:=p1 rotate 60;
p3:=p1 rotate 120 shift (1,0);
draw p1--p2--p3--cycle;
```

7 What's more?

- Actually there are much more features in MetaPost

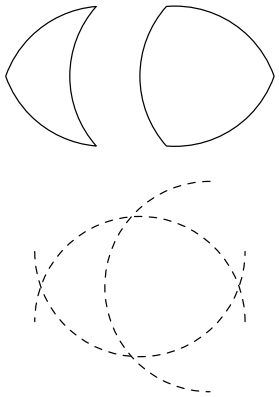
- Example: Filling and unfilling

```
beginfig(111);
path p;
p:=(-1,0)..(0,-1)..(1,0);
fill (p{up}..{down}(0,0){down}..{up}cycle) scaled 1cm;
draw (p..(0,1)..cycle) scaled 1cm;
picture yin,yang;
yin:=thelabel("Yin",(0.5cm,-0.5cm));
yang:=thelabel("Yang",(-0.5cm,0.5cm));
unfill bbox yin; draw yin;
unfill bbox yang; draw yang;
endfig;
```



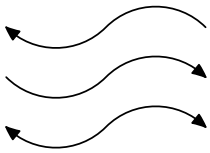
- Example: Building from intersections

```
beginfig(112);
path p[];
p1:=halfcircle scaled 3cm;
p2:=halfcircle rotated 180 scaled 3cm shifted (0,1cm);
p3:=halfcircle rotated 90 scaled 3cm shifted (1cm,1cm);
p4:=buildcycle(p1,p2,p3) shifted (-0.5cm,3cm);
p5:=buildcycle(p3,p2,p1) shifted (0.5cm,3cm);
draw p1 dashed evenly;
draw p2 dashed evenly;
draw p3 dashed evenly;
draw p4;
draw p5;
endfig;
```



- Example: Arrows

```
beginfig(113);
path p;
p:=(-1cm,0){dir -45}..(0,0){dir 45}..(1cm,0);
drawarrow p;
drawarrow reverse p shifted (0,0.5cm);
drawdblarrow p shifted (0,-0.5cm);
endfig;
```



- Clipping. grouping. macros, coloring, pen style, self-defined dash style

8 Why learning MetaPost?

- I learn it because I want to draw a Markov Chain in my paper
- Don't be lazy! If you picture is really complicated, write a program to help you generate all these drawing commands (Perl can do a good job!)