

# MathsPIC: A filter program for use with PICTEX

---

Richard W. D. Nickalls

*Department of Anaesthesia,  
City Hospital, Nottingham, UK.*

*dicknickalls@compuserve.com*

## Abstract

This article presents an overview of the mathsPIC utility for the PICTEX drawing engine. mathsPIC facilitates the drawing of mathematical diagrams by allowing the manipulation of points by name rather than by coordinates. Some familiarity with the PICTEX package is assumed.

*This article is a slight modification/update of the original article presented at the EuroTEX99 meeting in Heidelberg, Germany (September, 1999).*

## 1. Introduction

mathsPIC<sup>1</sup> is a filter program for use with the PICTEX drawing engine and its various addons<sup>2</sup>. It parses a plain text input-file (the mathsPIC file), and generates a plain text output-file containing PICTEX and TEX commands, which can then be TEXed (or L<sup>A</sup>TEXed) in the usual way. Spaces and the comment % symbol are used in the same way as TEX, although unlike TEX, mathsPIC commands are *not* case-sensitive; PICTEX commands can also be freely used in the mathsPIC file. mathsPIC also returns various parameter values in the output file, e.g. angles, distances between points, center and radius of inscribed and exscribed circles etc., since such values can be useful when making adjustments to a diagram.

---

<sup>1</sup> CTAN/[tex-archive/graphics/pictex/mathspic/](http://tex-archive/graphics/pictex/mathspic/)

<sup>2</sup> See particularly [pictexwd.sty](http://tex-archive/graphics/pictexwd.sty) and other related files by Andreas Schrell in CTAN/[tex-archive/graphics/pictex/addon/](http://tex-archive/graphics/pictex/addon/).

The motivation for `mathsPIC` stems from the fact that, while `PiCTeX` is an extremely versatile system for drawing figures, and offers the convenience and advantages of having the graphics code within the `TEX` document itself (e.g. printer-independence), it does require you to specify the coordinates of most points. This can make it quite awkward to use with complicated diagrams, particularly if several coordinates have to be re-calculated manually each time the diagram is adjusted.

For example, the `PiCTeX` commands for drawing Figure1, namely a triangle  $ABC$  with  $AB$  5cm,  $AC$  3cm, and included angle  $BAC$  40 degrees, and its incircle, are as follows.

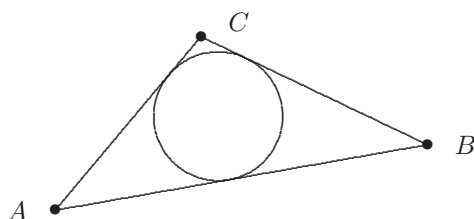


Figure 1:

```

\put {$\bullet$} at 0 0 % draw point A
\put {$\bullet$} at 4.9240 0.8682 % draw point B
\put {$\bullet$} at 1.9283 2.2981 % draw point C
\plot 0 0 4.9240 0.8682 1.9283 2.2981 0 0/ % draw triangle
% draw incircle
\circulararc 360 degrees from 3.0086 1.245 center at 2.1568 1.245

```

Although point  $A$  can be placed at the origin for convenience it is then necessary, since in this particular example  $AB$ ,  $AC$ , and the included angle are defined (see above), to resort to geometry and a calculator to determine points  $B$  and  $C$ . It is then necessary to recall the coordinates of all the points in order to write the `\plot` command. Finally, the `\circulararc` command requires even more geometry and calculation to figure out the radius of the incircle, the coordinates of its center, and the coordinates of the starting point of the arc drawing routine. Furthermore, if the initial diagram is not a suitable shape or size, the calculator has to be used again for any adjustments. In practice, therefore, `PiCTeX` requires a certain amount of planning and calculation for all but the simplest of diagrams.

`mathsPIC` provides an environment for manipulating named points, which makes `PiCTeX` convenient to use and mathematical diagrams easy to create.

For example, the equivalent `mathsPIC` commands for drawing Figure 1 are as follows.

```
point(A){0,0}           % point A at origin
point(B){A,polar(5,10deg)} % point B 5cm from A
point(C){A,polar(3,50deg)} % point C 3cm from A, BAC = 40 deg
drawPoints(ABC)       % draw points A B C
drawTriangle(ABC)
drawIncircle(ABC)
```

## 2. Points

Each point is associated with a point-name which is defined using a `point` command. Once defined, points can be referred to by name.

A point-name *must* begin with a *single* letter, and may have up to a *maximum* of two following digits. The following are valid point-names: *A*, *B*, *C3*, *d45*. Since `mathsPIC` is not case sensitive the points *d45* and *D45* are regarded as being the same point. Point-names can be either separated by spaces, or simply run together. Where coordinates need to be specified, the point is defined using the following command: `point(name){x,y}`. Existing point-names can be re-allocated new coordinates using the equivalent `point*` command.

The default point-character is the symbol  $\bullet$  (`\bullet`). However, `mathsPIC` allows the optional use of any `TEX` character to represent a point, by including it in a square bracket. For example, the point *A*(5,10) can be represented by the  $\square$  symbol<sup>3</sup> by defining it as follows.

```
point(A){5,10}[\Box]
```

Note that when lines are drawn to a point, the line will (unless otherwise instructed) extend to the point location. However, this can be prevented by allocating an optional circular line-free zone to a point by specifying the radius in the square bracket. For example, lines to a  $\square$  symbol at point *A* (above) can be prevented from being drawn through the box to its center, by allocating a 5 unit line-free zone to the point, as follows.

```
point(A){5,10}[\Box$,5]
```

If only the radius is specified (e.g. [5]) then the line-free zone will be applied to the default point-character (`\bullet`). However, if there is no character

<sup>3</sup> Available in the `latexsym` package.

before the comma (e.g. `[,5]` or `[ ,5]`) then the point-character will be the space character.

Points can also be defined in relation to other points or lines; for example, as the midpoint of a line, or the intersection of two lines.

Points are interpreted according to their grouping and context. Thus two points represent either a line or its length. For example, `drawCircle(C3,AB)` means draw a circle, center  $C3$ , with radius the length of line  $AB$ . Three points represent either a triangle, an angle, or a line, depending on the circumstances.

The order of points in `mathsPIC` commands is often significant. For example, the command `point(D){PointOnLine(AB,23)}` defines the point  $D$  as being 23 units *from A in the direction of B*. Some examples of valid `mathsPIC` point commands are as follows.

```
point(A){5,5}
point(B2){22,46}[$\triangle$,6]
point(C){3,8}[4]
point*(A){A,shift(3,0)}
point(D){midpoint(AB)}
point(e2){intersection(AB,CD)}
point(f){PointOnLine(AB,+5)}
point(g){perpendicular(C,AB)}
point(h){D,shift(3,-2)}
point(j){e2,polar(3,122 deg)}
point(k){F,polar(4, 0.5 rad)}
point(m){circumcircleCenter(ABC)}
point(n){incircleCenter(ABC)}
point(p){excircleCenter(ABC,BC)}
```

The point-symbol is drawn using the `drawPoint` or `drawPoints` command; for example, `drawPoint(A)`, or `drawPoints(ABCD)`.

### 3. Variables

Although magnitude can be specified by defining two points, numeric variables can be more conveniently defined using the `variable(name){value}` command. The *value* in the braces can be either a numeric (e.g. 4.32), a variable name (e.g.  $r3$ ), or two points (e.g.  $AB$ ) meaning the distance between the two points. Thus the command `variable(r3){20}` allocates 20 to the variable  $r3$ , which could then be used, for example, in the circle command

`drawcircle(C3,r3)`. New values can be re-allocated to an existing variable-name using the equivalent `variable*` command. Variables can also be manipulated mathematically using the terms `advance`, `multiply`, `divide`, and `mod`. Thus the command `variable(t1){t0,advance(-4)}` is equivalent to  $t_1 = t_0 - 4$ .

However, one must be careful not to mistake variables for points, and vice versa, since they both have the same name structure. When both are necessary, using upper case for points and lower case for variables is a convenient strategy. Some examples of valid `mathsPIC` `variable` commands are as follows.

```
variable(r){5}
variable*(r23){-6}
variable(p3){x}
variable(y4){y1,divide(9)}
variable(a2){angle(ABC)}
variable(x){area(ABC)}
variable(y1){Ypoint(A)}
```

## 4. Lines

Lines are drawn using the `drawLine` or `drawLines` command. For example, a line from  $P_1$  to  $P_2$  is drawn with the command `drawLine(P1P2)`. If a line is to be drawn through several points (say,  $J_1, J_2, J_3, J_4, J_5$ ) and can all be drawn ‘without lifting the pen’, then this can be achieved in one go with `drawLine(J1J2J3J4J5)`. Several unconnected lines can be drawn using one command by separating the line segments with commas; for example, `drawLines(J1J2,J3J4J5,J1J3)`.

## 5. Text

Text is typeset using the `text` command and, by default, is centered both horizontally and vertically at a defined point. For example, the box symbol  $\square$  would be placed at the point-location  $Z$  using the command `text($\Box$){Z}`. Optionally, text can be placed relative to a point using appropriate combinations of the P<sub>1</sub>CT<sub>1</sub>E<sub>1</sub>X options `B t b l r` to align the **B**aseline, **t**op, **b**ottom, **l**eft, **r**ight edges respectively of the text box with the point-location (see Wichura, 1992). For example, the text box This is point P would be aligned such that the right edge of the text box would be centered vertically at the point  $P$ , using `text(This is point P){P}[r]`.

Alternatively, text can be located relative to the point-location using the optional `shift(dx, dy)` or `polar(r,  $\theta$ )` commands. For example, points  $P_1P_2P_3$  could have their labels located 4 units from each point as follows.

```
text($P_1$){P1,shift(-4,0)}
text($P_2$){P2,polar(4,10 deg)}
text($P_3$){P3,polar(4,0.29088 rad)}
```

## 6. Using P<sub>I</sub>C<sub>T</sub>E<sub>X</sub> and T<sub>E</sub>X commands

In order to protect P<sub>I</sub>C<sub>T</sub>E<sub>X</sub> commands within the `mathsPIC` file from being acted on by `mathsPIC` they are included as the argument for the `pictex{...}` command, which simply copies its argument unchanged into the output file. For example, drawing with dashed lines is enabled using the P<sub>I</sub>C<sub>T</sub>E<sub>X</sub> command `\setdashes`, which is used in the `mathsPIC` file as `pictex{\setdashes}`.

In practice, it is also useful to include in the `mathsPIC` file any T<sub>E</sub>X or L<sup>A</sup>T<sub>E</sub>X headers and footers which would otherwise have to be added manually to the output file before L<sup>A</sup>T<sub>E</sub>Xing the file. This is achieved using the similar `tex{...}` and `latex{...}` commands. For example, when using L<sup>A</sup>T<sub>E</sub>X a typical format for a `mathsPIC` file might be as follows.

```
latex{\documentclass[a4paper]{article}}
latex{\usepackage{pictexwd,latexsym,amssymb}}
latex{\begin{document}}
pictex{\beginpicture}
paper{units(mm),xrange(0,50),yrange(0,50),axes(LB)}
...
...
pictex{\endpicture}
latex{\end{document}}}
```

## 7. Input and output files

The following example `mathsPIC` file (`triangle.m`) illustrates how some of these commands are used to draw Figure 2. Setting up the P<sub>I</sub>C<sub>T</sub>E<sub>X</sub> plotting area and axes is greatly simplified by the `mathsPIC` one-line `paper` command. Note that the dashed line  $AD$  is drawn after invoking the P<sub>I</sub>C<sub>T</sub>E<sub>X</sub> `\setdashes` command, and then `\setsolid` is used before drawing the right-angle symbol.

Also, the points  $A$ ,  $B$ ,  $C$  are defined using the  $\text{\TeX}$   $\odot$  symbol, in conjunction with a line-free zone of 1.2 mm in order to make the lines go to the edge of the symbol—the radius of such  $\text{\TeX}$  symbols has to be determined by trial and error.

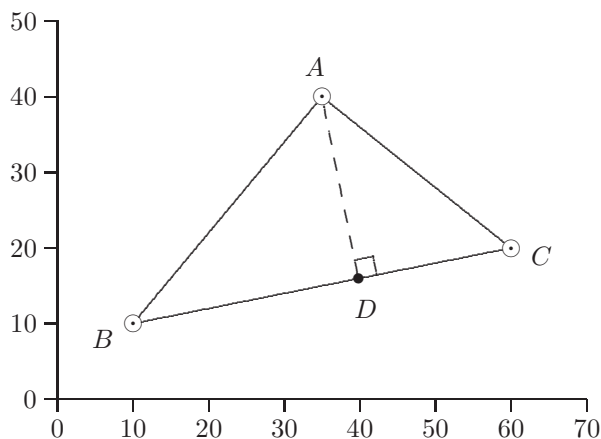


Figure 2:

```

pictex{\beginpicture}
paper{units(mm),xrange(0,70),yrange(0,50),axes(LB),ticks(10,10)}
point(A){35,40}[$\odot$,1.2]
point(B){10,10}[$\odot$,1.2]
point(C){60,20}[$\odot$,1.2]
point(D){perpendicular(A,BC)} %% from point A to line BC
drawPoints(ABCD)
drawLine(ABCA)
pictex{\setdashes}
drawLine(AD)
pictex{\setsolid}
drawRightangle(ADC,2.5)
text($A$){A,shift(-1,4)}
text($B$){B,shift(-4,-2)}
text($C$){C,shift(4,-1)}
text($D$){D,shift(1,-4)}
showLength(AD)
showLength(BC)
showArea(ABC)
pictex{\endpicture}

```

When the above file is processed by `mathsPIC` the output file is as follows. Note how the `PiCTEX` commands are accompanied by their `mathsPIC` commands (commented out), some of which have additional information added (e.g. the coordinates of a derived point—see `%% point(D)...`). Being able to compare the `mathsPIC` and resulting `PiCTEX` commands in the output file is particularly useful when debugging. Note also how the `show` commands return the lengths  $AD$ ,  $BC$ , and the area  $ABC$ .

```
%% triangle.m (Figure 2)
\beginpicture
%% paper{units(mm),xrange(0,70),yrange(0,50),axes(LB),ticks(10,10)}
\setcoordinatesystem units <1mm,1mm>
\setplotarea x from 0 to 70 , y from 0 to 50
\axis left ticks numbered from 0 to 50 by 10 /
\axis bottom ticks numbered from 0 to 70 by 10 /
%% point(A){35,40}[$\odot$,1.2]
%% point(B){10,10}[$\odot$,1.2]
%% point(C){60,20}[$\odot$,1.2]
%% point(D){perpendicular(A,BC)} ( 39.80769 , 15.96154 )
%% drawPoints(ABCD)
\put {$\odot$} at 35 40 %% A
\put {$\odot$} at 10 10 %% B
\put {$\odot$} at 60 20 %% C
\put {$\bullet$} at 39.80769 15.96154 %% D
%% drawLine(ABCA)
\plot 34.23178 39.07813 10.76822 10.92187 / %% AB
\plot 11.1767 10.23534 58.8233 19.76466 / %% BC
\plot 59.06296 20.74963 35.93704 39.25037 / %% CA
\setdashes
%% drawLine(AD)
\plot 35.23534 38.8233 39.80769 15.96154 / %% AD
\setsolid
%% drawRightangle(ADC,2.5)
\plot 42.25914 16.45183 41.76886 18.90328 /
\plot 39.3174 18.41299 41.76886 18.90328 /
%% text($A$){A,shift(-1,4)}
\put {$A$} at 34 44
%% text($B$){B,shift(-4,-2)}
\put {$B$} at 6 8
%% text($C$){C,shift(4,-1)}
\put {$C$} at 64 19
%% text($D$){D,shift(1,-4)}
```

```

\put {$D$} at 40.80769 11.96154
%% showLength(AD) 24.51452
%% showLength(BC) 50.9902
%% showArea(ABC) 625.0001
\endpicture

```

## 8. Arrows

Arrows can be drawn in all possible orientations, will *stretch* between points, and arrowheads are readily customised using the `mathsPIC setArrowshape` command (see also Salomon, 1992). Curved arrows are drawn using the `drawAngleArrow` command, which takes parameters for the angle, radius of arc, direction, and whether the angle is internal or external (see Figures 3 and 4).

```

%% arrow1.m (Figure 3)
pictex{\beginpicture}
paper{units(mm),xrange(5,45),yrange(5,45)}
point(A){30,30}
point(P){10,10}
point(B){30,10}
drawPoints(APB)
drawLine(APBA)
text($A$){A,shift(1,5)}
text($B$){B,shift(5,0)}
text($P$){P,shift(-5,0)}
drawAngleArrow{angle(BPA),radius(11),anticlockwise,internal}
text($\psi$){P,polar(7,22.5 deg)}
drawRightangle(ABP,2.5)
pictex{\endpicture}

```

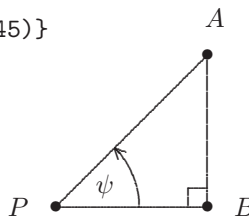


Figure 3:

Arrows are also used to link elements in a diagram, as shown in Figure 4. The right-hand diagram uses the `drawArrow` command; the small gap between the arrows and the letters  $P, Q, R, T$  being due to the 5 unit line-free radius associated with these points (see `arrow2.m`). The arrows are easily ‘stretched’ to accommodate their labels simply by adjusting the separation of the nodes in the `polar( $r, \theta$ )` commands (cf. Feruglio, 1994).

```

%% arrow2.m (Figure 4)
pictex{\beginpicture}

```

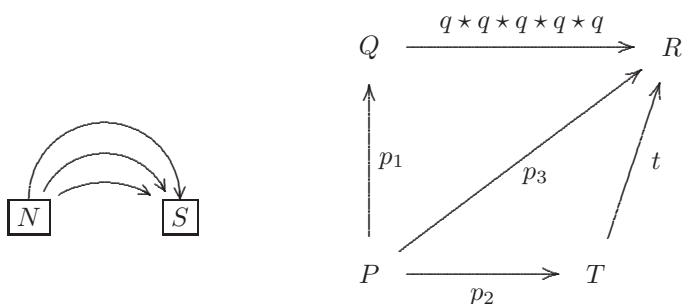


Figure 4:

```

paper{units(mm),xrange(10,110),yrange(0,45)}
%%-----Left-hand diagram-----
point(N){15,20}
point(S){N,shift(20,0)}
text(\framebox{N}){N,shift(0,-2.5)}
text(\framebox{S}){S,shift(0,-2.5)}
point(Z){midpoint(NS)}
drawAngleArrow{angle(NZS),radius(NZ),clockwise,internal}
point(N1){N,shift(2,1)}
point(S1){S,shift(-2,1)}
point(Z1){Z,shift(0,-3)}
drawAngleArrow{angle(N1Z1S1),radius(N1Z1),clockwise,internal}
point(N2){N1,shift(2,-0.5)}
point(S2){S1,shift(-2,-0.5)}
point(Z2){Z,shift(0,-10)}
drawAngleArrow{angle(N2Z2S2),radius(N2Z2),clockwise,internal}
%%-----Right-hand diagram-----
point(P){60,10}[$P$,5]
point(Q){P,polar(30,90 deg)}[$Q$,5]
point(R){Q,polar(40,0 deg)}[$R$,5]
point(T){P,polar(30,0 deg)}[$T$,5]
drawPoints(PQRT)
drawArrows(PQ,QR,PT,TR,PR)
point(P1){midpoint(PQ)}
text($p_1$){P1,shift(3,0)}
point(P2){midpoint(PT)}
text($p_2$){P2,shift(0,-3)}
point(P3){midpoint(PR)}
text($p_3$){P3,shift(2,-2)}

```

```
point(T1){midpoint(TR)}
text($t$){T1,shift(3,0)}
point(Q1){midpoint(QR)}
%% use a macro for the label
latex{\newcommand{\q}{\q \star q \star q \star q \star q}}
text(\q){Q1,shift(-1,3)}
pictex{\endpicture}
```

## 9. Circles

mathsPIC allows the point-symbol to be designated as a circle using the optional `[circle,r]` to the `point` command, which then allows the circles not only to have a line-free zone, but also to be drawn using the `drawPoints` command as shown in Figure 5. This greatly simplifies the drawing of directed graphs, trees and equivalent structures.

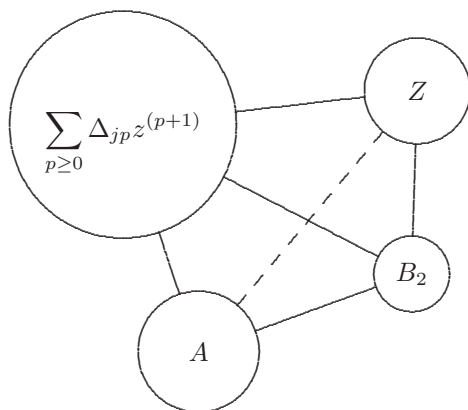


Figure 5:

```
%% points4.m (Figure 5)
pictex{\beginpicture}
paper{units(mm),xrange(0,65),yrange(0,55)}
point(A){30,10}[circle,8]
point(B1){A,shift(-10,30)}[circle,15] %% big circle
point(B2){A,polar(30,20 deg)}[circle,5]
point(Z){A,polar(45,50 deg)}[circle,7]
```

```

drawPoints(AB1B2Z)
drawLines(AB1,AB2,B1B2,B1Z,ZB2)
pictex{\setdashes}
drawLine(AZ)
text($A$){A}
% use a macro for the maths
latex{\newcommand{\BB}{\sum_{p\ge 0}\Delta_{jp}z^{(p+1)}}}
text(\vbox{\BB}){B1,shift(0,-2)}
text($B_2$){B2}
text($Z$){Z}
pictex{\endpicture}

```

Points on circles (and their labels) are most easily defined and positioned using the `polar` parameter, as shown in the `mathsPIC` file for Figure 6. Note the use of the `variable` command to define the radius  $r_2$ .

```

% circle.m (Figure 6)
pictex{\beginpicture}
paper{units(mm),xrange(0,55),yrange(0,55)}
point(C){30,30}[$\odot$,1.2]
variable(r2){20} % radius
drawCircle(C,r2) %% center C
point(P){C,polar(r2,250 deg)}
point(Q){C,polar(r2,120 deg)}
point(R){C,polar(r2,-30 deg)}
drawPoints(CPQR)
drawLines(PCRQP)
showAngle(PQR) % alpha
showAngle(PCR) % beta
text($P$){P,polar(5,30 deg)}
text($Q$){Q,polar(5,160 deg)}
text($R$){R,polar(5,-30 deg)}
drawAngleArrow{angle(PCR),radius(8),anticlockwise,internal}
text($\beta$){C,polar(5,285 deg)}
drawAngleArrow{angle(PQR),radius(12),anticlockwise,internal}
text($\alpha$){Q,polar(8,-65 deg)}
pictex{\endpicture}

```

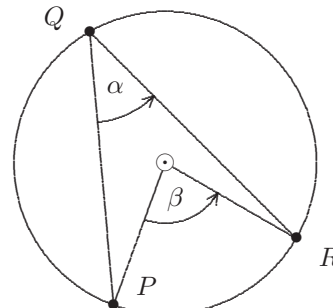


Figure 6:

Note that the returned values in the output file from the `showAngle` commands (see below) for Figure 6 indicate that  $\beta$  is twice  $\alpha$  as expected.

```

% showAngle(PQR) 40 degrees / radians 0.6981316 % alpha

```

```
%% showAngle(PCR) 79.99999 degrees / radians 1.396263 % beta
```

mathsPIC offers a range of circle commands (`drawIncircle`, `drawCircumcircle`, `drawExcircle`) specifically for geometry diagrams (cf. Cameron, 1992), as shown in Figure 7. This example starts with the triangle  $ABC$ , and later points  $D$  and  $E$  are constructed in order to extend lines  $AC$  and  $AB$ .

```
%% incircle.m (Figure 7)
```

```
pictex{\beginpicture}
paper{units(mm),xrange(0,70),yrange(25,70)}
point(A){10,40}
point(B){30,35}
point(C){40,55}
drawPoints(ABC)
drawTriangle(ABC)
drawCircumcircle(ABC)
drawIncircle(ABC)
%% extend AC and AB
point(D){pointOnLine(AC,50)}
point(E){pointOnLine(AB,50)}
drawLines(CD,BE)
drawExcircle(ABC,BC)
text($A$){A,shift(-3,-4)}
text($B$){B,shift(0,-5)}
text($C$){C,shift(2,5)}
point(P1){circumcircleCenter(ABC)}[$\odot$,1.2]
point(P2){incircleCenter(ABC)}[$\odot$,1.2]
point(P3){excircleCenter(ABC,BC)}[$\odot$,1.2]
drawPoints(P1P2P3)
pictex{\setdashes}
drawLine(P1P2P3P1)
pictex{\endpicture}
```

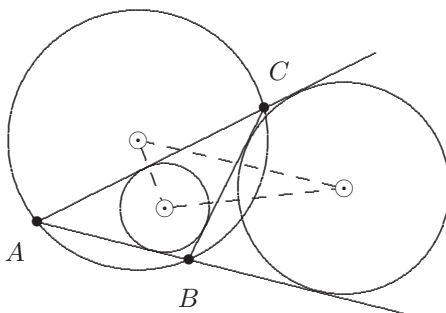


Figure 7:

## 10. Functionally connected diagrams

When constructing diagrams it is often useful to write the mathsPIC file in such a way that new points are related to earlier points, since the structure of the diagram is then maintained even when points are moved. This is demonstrated in Figure 8, where the mathsPIC code for the two diagrams differs *only*

in the angle of the line  $AB$  (left diagram, 60 degrees; right diagram, 5 degrees) as defined in the `point(B){...}` command as follows.

- Left-hand diagram: `point(B){A,polar(45,60 deg)}`
- Right-hand diagram: `point(B){A,polar(45,5 deg)}`

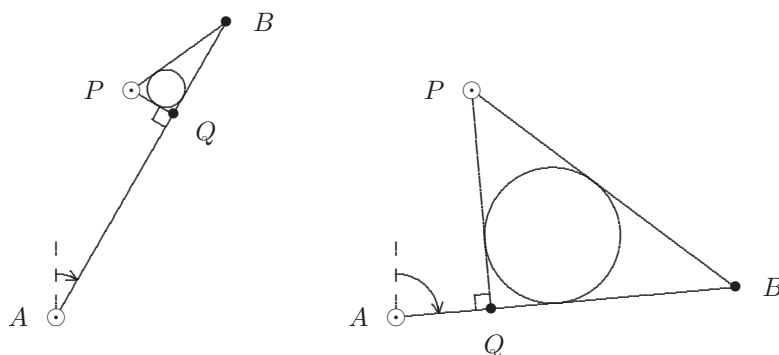


Figure 8: The `mathsPIC` code for the two diagrams differs *only* in the angle of the line  $AB$  as defined in the `point(B){...}` command (see `dynamic.m`).

```

%% dynamic.m (Figure 8)
pictex{\beginpicture}
paper{units(mm),xrange(5,120),yrange(0,45)}
%% left-hand diagram
point(A){15,5}
point(P){A,shift(10,30)}
point(B){A,polar(45,60 deg)}
point(Q){perpendicular(P,AB)}
drawRightangle(PQA,2)
drawPoints(ABPQ)
drawLine(ABPQ)
drawIncircle(PQB)
text($A$){A,shift(-5, 0)}
text($B$){B,shift(5, 0)}
text($P$){P,shift(-5, 0)}
point(S){pointOnLine(QP,-5)}
text($Q$){S}
%% now draw vertical line and angle

```

```

point(N){A,shift(0,12)}
pictex{\setdashes}
drawLine(AN)
pictex{\setsolid}
drawAngleArrow{angle(NAB),radius(7),clockwise,internal}
pictex{\endpicture}

```

## 11. Inputting files and recursion

mathsPIC allows the recursive input of blocks of commands as files, using the `inputfile` command to input files containing mathsPIC commands. In practice, this facility is equivalent to a ‘DO-LOOP’ in a program. For example, the file `myfile.dat` would be input six times sequentially using the command `inputfile(myfile.dat) [6]`.

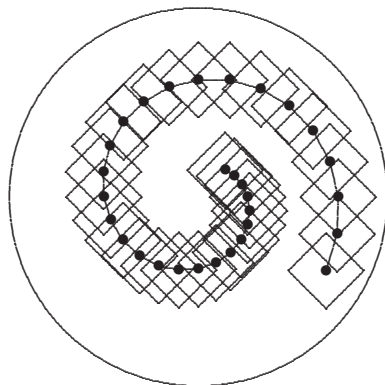


Figure 9:

Figure 9 was produced by the following code (`spiral.m`), which inputs a small file of mathsPIC code (`spiral.dat`) recursively 30 times using the `inputfile(spiral.dat) [30]` command. Note the use of the `variable*` and `point*` commands to re-allocate variables and points recursively, as well as the use of the `advance` term.

```

%% spiral.m
pictex{\beginpicture}

```

```

paper{units(mm), xrange(0,60), yrange(0,60)}
point(C){30,30} % circle center
drawcircle(C,25)
variable(a){315} % angle deg
variable(r){20} % radius of spiral
variable(s){5} % square, semi-diagonal
point(T){C,polar(r,330 deg)}
inputfile(spiral.dat)[30]
pictex{\endpicture}

%% spiral.dat
variable*(r){r,advance(-0.5)} % let r=r-0.5
variable*(a){a,advance(15)} % let a=a+15 deg
point*(P){C,polar(r,a deg)}
drawpoint(P)
drawline(TP)
point*(T){P} % let T=P
%% make a square centered on P
point*(Q1){P,polar(s,0)}
point*(Q2){P,polar(s,90)}
point*(Q3){P,polar(s,180)}
point*(Q4){P,polar(s,270)}
drawline(Q1Q2Q3Q4Q1)

```

Data-files which do not contain `mathsPIC` commands can be input using the `inputfile*` command. This command inputs files verbatim, and so can be used for inputting files containing points for plotting curves and `PTCTEX` commands.

## 12. Conclusion

`mathsPIC` facilitates the drawing of `PTCTEX` diagrams mainly because it provides an environment for manipulating named points, and makes it easy for points to be defined in terms of other points. Geometry diagrams can therefore be constructed in an intuitive way, much as one might with a compass and ruler; for example, constructing a point at a certain position simply to allow some other point to be constructed, perhaps to draw a line to. In other words, `mathsPIC` offers the freedom to create ‘hidden’ points having a sort of scaffolding function. In particular, this facility allows diagrams to be constructed in such a way that they remain functionally connected even when points are moved.

Note that `mathsPIC` can also be viewed as a handy tool for exploring elementary geometry since its `show` commands return the values of various parameters; for example, angles, the distance between points, coordinates of derived points, areas of triangles etc. A detailed 37-page manual on `mathsPIC` is available at [CTAN/graphics/pictex/mathspic/](http://CTAN/graphics/pictex/mathspic/).

Finally, I would like to thank Apostolos Syropoulos and Bob Schumacher for beta-testing this program, and for their many ideas and suggestions.

### 13. References

- Cameron P. J. (1992): *Geometric diagrams in L<sup>A</sup>T<sub>E</sub>X*. *TUGboat* **13** (No. 2), 215–216.
- Feruglio G. V. (1994): *Typesetting commutative diagrams*. *TUGboat* **15** (No. 4), 466–484.
- Salomon D. (1992): *Arrows for technical drawing*. *TUGboat* **13** (No. 2), 146–149.
- Wichura M. J. (1992): *The P<sub>r</sub>CT<sub>E</sub>X manual*. Pub: Personal T<sub>E</sub>X Inc., 12 Madrona Avenue, Mill Valley, CA 94941, USA. <http://www.pctex.com/>