The unknown *picture* environment

Claudio Beccari

Abstract

The old *picture* environment, introduced by Leslie Lamport into the LaTeX kernel almost 20 years ago, appears to be neglected in favor of more modern and powerful LaTeX packages that eliminate all drawbacks of the original environment. Nevertheless it is still being used behind the scenes by a number of other packages. Lamport announced an extension in 1994 that should have removed all the limitations of the original environment; in 2003 the first version of this extension appeared; in 2004 the first stable version was released; in 2009 it was actually expanded with new functionality. Nowadays the *picture* environment can perform like most simple drawing programs, but it has special features that make it invaluable.

1 Introduction

Plain TeX, as described in *The TeXbook* [5], contained a simple way to draw simple graphics with tex. When LaTeX was first published in 1984, it contained an environment suitable for relatively complex graphics; Lamport's handbook [6] described its workings and commands. But all this seems to have fallen into complete oblivion.

Many users of LaTeX related forums keep asking questions such as "How can I produce such and such a symbol"; I keep answering "Use the *picture* environment". Apparently nobody follows my suggestions, which of course they are free to avoid; but they would save time if they spent no more than 15 minutes in reading the environment description in Lamport's handbook [6]. The second edition of this handbook announces the extensions and discusses the eliminated drawbacks; these extensions were eventually realized only in 2003 by Gäßlein and Niepraschk [4]. In 2004 the same authors released a stable version. In 2009, with the contribution of the third author Tkadlec, they released an enhanced version that added quite a few new commands that substantially extend the *picture* environment functionality. But, except for those very latest additions in 2009, everything else was already documented by Lamport in his second edition.

Not longer than a few days ago a forum participant asked how he could draw a square wave and a saw tooth wave of suitable size for setting them in line with his text: and .

Editor's note: First published in ArsTeXnica #11, April 2011. Reprinted with permission, in slightly different form.

Here is the whole thing required to produce the two simple commands, using the recent pict2e extension package's new commands:

```
\newcommand*\sqwave[1][0.125ex]{{%
\unitlength=#1\relax
\picture(20,10)
\polyline(0,0)(6.5,0)(6.5,15)(13,15)%
        (13,0)(19.5,0)
\endpicture}}

\newcommand*\sawtooth[1][0.125ex]{{%
\unitlength=#1\relax
\picture(20,10)
\polyline(0,0)(6.5,0)(6.5,15)(13,0)%
        (19.5,0)
\endpicture}}
```

Most of the time suggestions are given by and to the forum participants to use external drawing programs, or to use the sophisticated PSTricks [12] or TikZ [11] packages. The former solution is generally to be avoided, because if some lettering is needed, it requires extra work to use the same fonts as those used in the document. The latter two packages are certainly capable of doing marvelous and complicated drawings, but require considerable time with the documentation and a steep learning curve.

Also, the *picture* environment has a unique feature: it can produce drawings of zero width and/or height. This special feature makes them valuable for packages such as eso-pic [9], crop [3], layout [8], layouts [15], and others. These packages draw things on the page that do not require any external package, and therefore don't have any dependency. These drawings occupy no space, although they have a specific position on the page; their contents reach whatever point on the page, as background images or marks that do not interfere with the positioning of other page elements.

In particular eso-pic (and similar packages for setting watermarks) and crop exploit this functionality specifically for setting a background picture or the crop marks in the correct positions without interfering with the other elements of the page.

At the same time within the *picture* environment it is possible to use cubic Bézier curves and to draw polygons, arches and sectors, oval frames whose corner curvature can be freely specified, white or filled circles of any dimension. The \polyline macro used in the above example makes it very easy to draw polylines with any number of corners and any segment slope, to the point that if the nodes are sufficiently close, it is possible to draw "smooth" curves whose points may be calculated with any number-crunching personal or mainframe computer.

2 In detail

Everybody can agree that the original *picture* environment, created by Lamport with the very first version of LaTeX, was pretty rudimentary, but there was practically nothing else to use in its place. The straight lines could have slopes that were ratios of relatively prime integer numbers not exceeding 6 in absolute value. For vectors the limitation was even stricter; the slopes could not be specified with integer numbers larger than 4 in absolute value. Why were there such strange limitations? Because straight lines were made up through the juxtaposition of small segments 10 pt ($\approx$ 3 mm) long taken from a special font; this same special font contained also the vector arrow tips that occupied a large part of the available positions; and, remember, at that time the typesetting engine could deal only with 128-glyph fonts, so that the available short segments and arrow tips, plus a selection of closed and filled circles and/or quarter circles placed strict limitations on the drawing performance.

Patient programmers created extension packages such as `curves` [7], that could overcome such limitations by drawing lines of any slope and circles of any diameter by juxtaposing an "infinity" of small dots. For plain TeX there existed another package, PiCTeX [14], that with a suitable interface could work also with LaTeX. It performed well on large mainframes with large memory capabilities, but worked very poorly on the desktops of that age, the eighties, when a 20 MiB hard disk was a luxury and 640 KiB RAM was almost the maximum available. These packages mostly saturated the RAM and drawing was virtually impossible on personal computers.

Some progress was made when the PostScript format became available; some drawing packages (again `curves`) exploited TeX's `\special` to write raw PostScript drawing commands in the output, so that the actual action of drawing was demanded of the screen or printer driver. Nevertheless powerful extensions in this directions, such as PSTricks, appeared much later. Drawing with external programs and importing the resulting artwork was therefore a necessity, but not an easy task.

Things evolved in the right direction when personal computers, having become the universal complement of any person needing to write anything, started having a more user friendly interface, more RAM, larger hard disks, and better programs, the TeX system included. The nineties, besides the important passage from TeX 2.x to TeX 3.0, gave us LaTeX 2_ε, PostScript fonts, and the drawing instrument METAPOST. This program used more or less the same philosophy that led Knuth to develop METAFONT, in order to draw the TeX system default fonts; METAPOST produced a simplified output PostScript code that was understood also by the newborn typesetting program `pdftex`. METAPOST was, and still is, fully compatible with the rest of the TeX system typesetting engines, so that all the TeX and LaTeX features could be used for putting any lettering on the METAPOST output files.

Meanwhile LaTeX went on with its small drawing environment, without exploiting the new possibilities with the PostScript format and the PDF portable document format, until Gäßlein and Niepraschk wrote the *picture* extension announced by Lamport some ten years before.

2.1 2009 extensions to *picture*

Let us now discuss the enhancements introduced by the extension realized by Gäßlein and Niepraschk. Since these changes are so recent, some commands are not described in [6].

1. First of all, the enhancements rely on the drivers that are being used to display or to print the document. More precisely, when the `latex` program is used, the `\special` commands to the driver contain only PostScript language commands; this implies a transformation of the resulting DVI file into a PostScript one by means of `dvips`, and possibly a second transformation to the PDF format by means of (for example) `ps2pdf`. On the other hand, if the document is processed with `pdflatex`, then the `\special` commands contain only the PDF language commands. Therefore the extension is fully compatible with the typical output formats provided by the most popular typesetting engines, and this is fully automatic so users need not bother about the details.
2. The output file size very often is smaller since the actual drawing computations are performed by the suitable drivers.
3. One of the limitations of the original environment was the slope of lines and vectors. In the first implementation of the extension the slope coefficients had to be integers not higher than 1000 in absolute value, thus implementing Lamport's description of 1994. The 2009 enhancements, however, remove even this limitation, and the slope coefficients can be any fractional decimal number (well, yes, not too large, not higher than 16 383.999 98 which corresponds to the the largest dimension in points that any TeX system typesetting engine can

handle). Now line and vector slopes should not have any detectable limitation.

4. The above is valid also for vectors; even better, now it's possible to pass an option to the package so that it can draw the arrow tips in "LaTeX style" or in "PostScript style". In LaTeX style the joining sides to the arrow tip are slightly concave, and the arrow shaft is straight; in PostScript style they form a polygon that resembles a stealth aircraft.
5. Circles and quarter circles were available in a limited set; now they can be drawn in any size, both filled and unfilled.
6. Line thicknesses could previously be specified only as `\thinlines` (default) and `\thicklines` (twice as thick as `\thinlines`), and only vertical and horizontal lines used the thickness specified with `\linethickness` ⟨*dimension*⟩. Now `\linethickness` can modify the thickness of all sorts of lines, Bézier splines included.
7. "Ovals", frames with rounded corners, could have the corner quarter circle with an automatic setting of its radius, in any case not larger than about 15 pt (about 5 mm), and they could not use a radius dimension specified by the document. Of course this radius should not exceed the half length of the shorter frame sides (that is, half of the distance of the longer straight lines that form the longer sides of the frame) but the radius can now be specified as an optional argument to the `\oval` command so that the created frame can have a very different look when a smaller radius is chosen compared to the same-sized frame with a larger corner radius.
8. Quadratic and cubic Bézier splines are now generated with the driver commands and they result in smooth curves, not lines with a rough contour due to the juxtaposition of many small dots. The possibility of specifying the number of dots is available even now, but it is mostly for backwards compatibility — although, even now, dotted splines might be used for special purposes. In any case they do not suffer any magnification when seen on the screen; they are scalable vector strokes. The previous command `\bezier` is maintained with its compulsory specification of the number of points to use, but two new commands, with an optional specification of the number of points, are introduced, `\qbezier` for tracing quadratic Bézier splines, and `\cbezier` for tracing cubic Bézier splines; this last command was not described in [6], and is a completely new command to the package.
9. Up to this point the traditional commands have been discussed and the differences with the original environment described. The last extension of `pict2e`, published in the second half of 2009, adds some other commands that draw other lines but in general don't require the use of `\put` to place these lines in a special position. Of course they may be shifted with `\put`, which might come in handy when fine-tuning the picture, but the `\put` is not necessary.
10. A first exception to the above statement is the new macro `\arc` that is a generalization of `\circle` (both starred and non-starred forms; in both cases the starred form produces a filled contour) which requires putting the arc center in a specific position, so that the whole command must be set as an argument to `\put`. The `\circle` command is used like this:

    `\put(`⟨*x*⟩`,`⟨*y*⟩`){\circle`⟨***⟩`{`⟨*diameter*⟩`}}`

    and similarly with the `\arc` command:

    `\put(`⟨*x*⟩`,`⟨*y*⟩`){\arc`⟨***⟩`[`⟨*ang1*⟩`,`⟨*ang2*⟩`]{`⟨*radius*⟩`}}`

    The arc has its center at point `(`⟨*x*⟩`,`⟨*y*⟩`)`, and it will go from the angle ⟨*ang1*⟩ to the angle ⟨*ang2*⟩; angles are in sexagesimal degrees and are positive in the anticlockwise direction; if the optional angles are not specified, the full circle is drawn. The arc is drawn from the smaller angle to the larger one, so that the order in which ⟨*ang1*⟩ and ⟨*ang2*⟩ is not important.
11. The following commands do not require `\put`:

    ```
    \Line(⟨x1⟩,⟨y1⟩)(⟨x2⟩,⟨y2⟩)
    \polyline(⟨x1⟩,⟨y1⟩)(⟨x2⟩,⟨y2⟩)...(⟨xN⟩,⟨yN⟩)
    \polygon(⟨x1⟩,⟨y1⟩)(⟨x2⟩,⟨y2⟩)...(⟨xN⟩,⟨yN⟩)
    \polygon*(⟨x1⟩,⟨y1⟩)(⟨x2⟩,⟨y2⟩)...(⟨xN⟩,⟨yN⟩)
    ```

    The first command is simply the segment that joins the two points identified by the two pairs of coordinates. The second command is a sequence of segments that join with one another in the order of the N specified pairs of coordinates; we have seen it at work in the example shown in the introduction. The third command is a closed polygon whose vertices are sequentially shown by the N pairs of coordinates. The fourth command is similar but draws a filled polygon.
12. In order to draw the various lines and curves, the internal commands make use of the "turtle graphics" commands used within both the PostScript and PDF languages. These elementary commands are available to the user also through package `pict2e`; they are:

    ```
    \moveto(⟨x⟩,⟨y⟩)
    \lineto(⟨x⟩,⟨y⟩)
    \curveto(⟨x2⟩,⟨y2⟩)(⟨x3⟩,⟨y3⟩)(⟨x4⟩,⟨y4⟩)
    ```

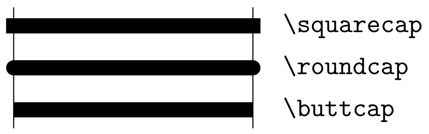

Figure 1: Ending styles for line segments (of equal width)

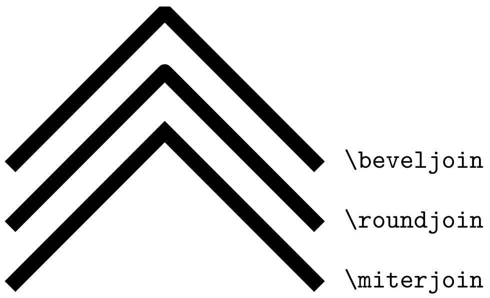

Figure 2: Join styles for line segments

and a few more that the reader may find in the documentation [4]. These commands may be used in any order, except `\moveto` that must fix the first position of the drawing pen. In order to finish the path it is optional to use `\closepath` in order to draw a line from the last point to the initial one, but then it is necessary to use either `\strokepath` to draw the path or `\fillpath` in order to fill the path with the default color.

13. The initial and final points of an open path may be controlled with the commands `\buttcap` (cut the path at the end points), or `\roundcap` (adjust the end points with a filled semicircle), or `\squarecap` (adjust the end points with a filled half square); in general with line art the `\roundcap` should be preferable, but sometimes it's better to use one of the other two kinds of end point finishing. See figure 1.
14. Similarly the joins between adjacent segments of a polyline or a polygon may be adjusted with the three commands `\miterjoin`,[1] `\roundjoin`, and `\beveljoin`, as shown in figure 2.

3 Examples

There are dozens of examples in the GuIT documentation [2], where every line art picture has a small legend containing the author name and the program used for producing it. This book is a collective effort of the Italian TeX users group, and is downloadable from the GuIT site `http://www.guitex.org/home/images/doc/guidaguit.pdf`. There, the interested reader can find plenty of ideas and useful "tricks".

[1] In the documentation, [4], this command is erroneously spelled `\mitterjoin`.

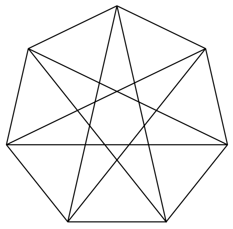

Figure 3: A heptagon with seven vertices and inscribed star

Here we present a few examples, sometimes with their source code, in order to see the modern *picture* environment at work.

A heptagon We compute the vertices of a heptagon inscribed into a circle with a diameter of 6 cm by means of a pocket calculator:

$$v_1 = (1.3017, -2.7029) \qquad v_5 = (-2.3455, 1.8705)$$
$$v_2 = (2.9248, -0.6676) \qquad v_6 = (-2.9248, -0.6676)$$
$$v_3 = (2.3455, 1.8705) \qquad v_7 = (-1.3017, -2.7029)$$
$$v_4 = (0, 3)$$

Then we set up the picture environment (within a *figure* environment, so we don't need to do anything to limit the scope of the `\unitlength` assignment) with the following code:

```
\unitlength=5mm
\begin{picture}(6,6)(-3,-3)
\polygon(1.3017,-2.7029)(2.9248,-0.6676)%
    (2.3455,1.8705)(0,3)(-2.3455,1.8705)%
    (-2.9248,-0.6676)(-1.3017,-2.7029)
\polyline(1.3017,-2.7029)(0,3)%
    (-1.3017,-2.7029)(2.3455,1.8705)%
    (-2.9248,-0.6676)(2.9248,-0.6676)%
    (-2.3455,1.8705)(1.3017,-2.7029)
\end{picture}
```

Figure 3 contains also the seven pointed star inscribed in the heptagon.

Splines We draw some splines inside a square with sides 6 cm long; a quadratic spline has its two nodes at the square base vertices, and the control node at the center of the upper side. A cubic spline uses the four square vertices as end and control nodes:

```
\unitlength=6.5mm
\begin{picture}(6,6)(-3,-3)
\put(-3,-3){\framebox(6,6){}}
\polyline(-3,-3)(0,3)(3,-3)
\polyline(-3,3)(3,3)(-3,-3)(3,-3)
\linethickness{1.5pt}
\qbezier(-3,-3)(0,3)(3,-3)
\cbezier(-3,3)(3,3)(-3,-3)(3,-3)
\end{picture}
```

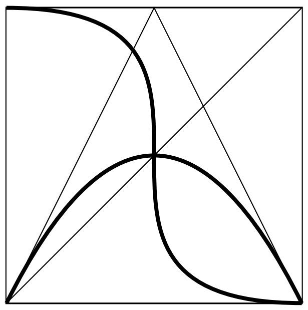

Figure 4: Quadratic and cubic splines

Figure 4 displays the result; observe the effect of the `\linethickness` assignment on the splines themselves. If you can read this document on the screen, you can magnify the picture and check the vector nature of the splines. Figure 4 contains also the polylines that join the nodes and control points, so that it's easier to see the effect of these "control segments".

An electric circuit Many years ago, at the end of the 1980s, when I had available only the *picture* environment, I needed to draw circuit diagrams. In fact, I had so many circuit diagrams to insert in my book that I needed to create suitable macros for drawing the circuit components and their connections to the various circuit nodes; of course every component had to be identified with a symbol and optionally should be assigned a value with the proper units.

Nowadays there are modular packages that work with Ti*k*Z (`circuitikz` [10]) and PSTricks (`pst-circ` [13]), but at that time there was nothing, or at least nothing I was aware of.

In my department there was a very good expert of technical drawing, and for my previous books I had asked him to draw my circuits; these drawings had to be glued to the camera ready copy, because at that time it was very difficult to insert graphical files into a document; not impossible, but difficult. The publisher, in any case, did not want any kind of file; he wanted only the camera ready copy. This procedure was pretty lengthy: draw my circuits by hand, pass them to the technician with suitable descriptions about dimensions, lettering, line thicknesses, and the like; after the drawings were done, careful checking of the correctness, the proper position of the labels and indices, and any possible typos; start again with the second draft, and so on.

Therefore I decided to write a personal package containing all the circuit macros, to work as an interface between the user and the *picture* environment with its internal macros. It took about two weeks; afterwards, I had an almost complete circuit-drawing TeX program. At that time, of course, arbitrary sloped lines were done by juxtaposing a multitude of little dots, as well as quarter, half and full circles of any diameter. Single-port components were automatically drawn as vertical or horizontal elements; connections automatically made the necessary bends in order to reach the destination nodes; two-port and four-pole devices were set in the proper orientation in order to avoid crossing their connections; operational amplifiers, nullators, norators and nullors were correctly designed; block diagrams with their signal flows, their branching nodes, their summing points, and so on, were at hand. The unit length was parametrized to the current font 'ex' unit, so that the circuit diagram would scale together with the size of the surrounding text font.

I saved much time using my macros and the technical expert eventually congratulated me, admitting that my drawings were more consistent than his own.

When the important `pict2e` package became available in 2003, I started to eliminate all references to the old tiny-point-overlay technique, and promptly switched to the new technology.

I eventually added logical components as well, so that this private package is almost complete. What is provided by `circuitikz` is much more complete, and this is the main reason why my package remains private. I keep using it for no other reason than compatibility with the past book files I wrote long ago, from which I often pick up some parts in order to assemble short tutorials for students who ask me for explanations.

The package is too large to publish here; therefore, I will not show how the user commands are realized with the internal modern *picture* environment ones. I show just the user-level code for drawing the circuit diagram of a band elimination filter:

```
\begin{circuito}(75,35)
\hconnect(0,0)(19,0)\HPolo(20,0)(65,0)
\polo(20,25)[30,25]\polo(65,25)[75,25]
\hconnect(66,0)(75,0)
\R(75,0)(75,25){R\ped{L}}
\E(0,0)(0,25){E}
\R(0,25)(19,25){R\ped{G}}
\serie*(30,0)(21,25)C{C_1}-L{L_1}
\parallelo(30,25)(55,25)L{L_2}-C{C_2}
\serie(55,0)(64,25)C{C_3}-L{L_3}
\nodi(55,0)(55,25)(30,25)(30,0)
\end{circuito}
```

and you can see the result in figure 5. As you can see the component and connection macros are very user friendly and the total amount of code for drawing

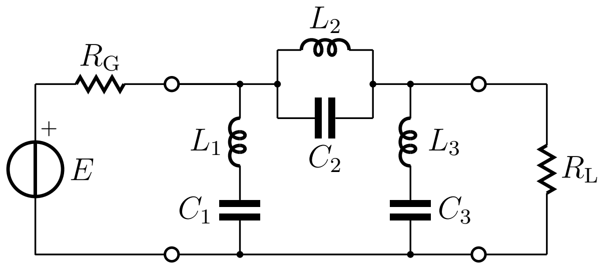

Figure 5: A band elimination filter

a complicated circuit diagram[2] is very limited. If you are reading this file on screen you can magnify the image of figure 5 and again verify that the whole drawing is made of scalable vectors. You can also recognise that the resistors are drawn by means of the `\polyline` command with the `\miterjoin` specification for the connection of the various segments.

A Cartesian diagram While teaching the synthesis of electrical circuits I often needed Cartesian diagrams of their performance; in figure 6 the squared magnitude of a fifth order elliptical filter characteristic function is plotted. The name "elliptical" derives from the use of elliptical integrals and functions of the first and second kinds. The diagram is only qualitative; although it would not have been a problem to compute the actual points by means of a suitable program, for a qualitative diagram the extreme points and the peaks and zeros should be sufficient. The whole diagram had to be also shown as a slide, therefore a `beamer` presentation was made containing the same code:

```
\unitlength=0.9mm
\begin{picture}(80,60)(-40,-5)
  \VECTOR(-40,0)(40,0)
  \Zbox(40,-2)[tr]{\omega}
  \VECTOR(0,-1)(0,55)
  \Zbox(-1,55)[tr]{|F|^2}
  \multiput(-35,5)(4,0){18}%
    {\line(1,0){2}}
  \Zbox(2,7)[bl]{1}
  \multiput(-35,45)(4,0){18}%
    {\line(1,0){2}}
  \Zbox(1,46)[bl]{H^2}
  \multiput(-2,15)(4,0){4}%
    {\line(1,0){2}}
  \Zbox(1,16)[bl]{H}
  \LINE(-10,0)(-10,-1)\Zbox(0,-2)[t]{0}
  \Zbox(-10,-2)[t]{-1}
  \LINE(10,0)(10,-1)\Zbox(10,-2)[t]{1}
```

[2] Actually the circuit diagram is not complicated at all; the complication is hidden behind the user macros, especially those for inductors, where the various Bézier cubic splines are properly described and connected to one another.

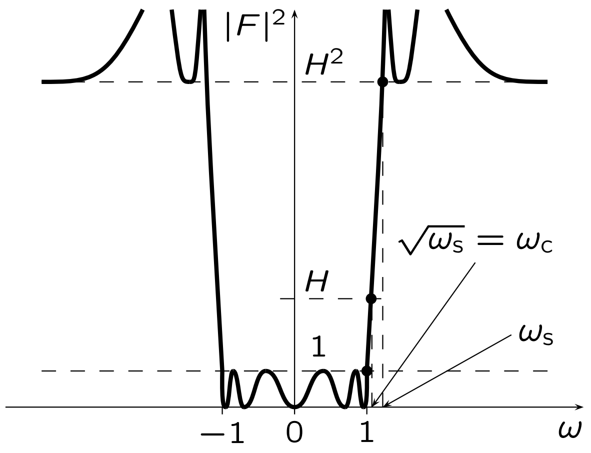

Figure 6: The squared magnitude of the characteristic function of an elliptical filter

```
{\linethickness{1.5pt}%
  \cbezier(-12.5,55)(-12,40)(-12,40)%
        (-10,5)
  \cbezier(-10,5)(-10.1,0)(-9.75,0)%
        (-9.5,0)
  \cbezier(-9.5,0)(-9,0)(-9,5)(-8.5,5)
  \cbezier(-8.5,5)(-7.75,5)(-7.75,0)%
        (-7,0)
  \cbezier(-7,0)(-5.5,0)(-5.5,5)(-4,5)
  \cbezier(-4,5)(-2,5)(-2,0)(0,0)
  \cbezier(-13,55)(-13.75,45)(-13.75,45)%
        (-14.5,45)
  \cbezier(-14.5,45)(-15.75,45)%
        (-15.75,45)(-17,55)
  \cbezier(-21,55)(-26,45)(-28,45)%
        (-35,45)
%
  \cbezier(12.5,55)(12,40)(12,40)(10,5)
  \cbezier(10,5)(10.1,0)(9.75,0)(9.5,0)
  \cbezier(9.5,0)(9,0)(9,5)(8.5,5)
  \cbezier(8.5,5)(7.75,5)(7.75,0)(7,0)
  \cbezier(7,0)(5.5,0)(5.5,5)(4,5)
  \cbezier(4,5)(2,5)(2,0)(0,0)
  \cbezier(13,55)(13.75,45)(13.75,45)%
        (14.5,45)
  \cbezier(14.5,45)(15.75,45)(15.75,45)%
        (17,55)
  \cbezier(21,55)(26,45)(28,45)(35,45)
}%
  \put(10.6,15){\circle*{1.5}}
  \put(12.2,45){\circle*{1.5}}
  \put(10,5){\circle*{1.5}}
  \multiput(12.2,0)(0,4){11}%
    {\line(0,1){2}}
  \multiput(10.7,0)(0,4){4}%
    {\line(0,1){2}}
  \VECTOR(25,20)(10.7,0)
  \VECTOR(30,10)(12.2,0)
  \Zbox(25,21)[b]%
  {\sqrt{\omega\ped{s}}=\omega\ped{c}}
  \Zbox(31,10)[lc]{\omega\ped{s}}
\end{picture}
```

Two custom commands, `\Zbox` and `\VECTOR`, were defined in order to speed up data input. The former is short for inserting a zero dimension box at the proper coordinate, and the latter is similar to the standard `\Line` command but applied to vectors. The other unusual macro `\LINE` is completely equivalent to `\Line`; I had merely defined it before the 2009 enhancement of the `pict2e` package.

Figure 6 displays the whole diagram as scalable vector graphics, and as you can see the important messages about the filter characteristic function properties are fully and clearly expressed. This is just one example among the many such diagrams I used in my books and presentations. It was well worth the little time spent in defining the service macros. Of course I could have used the `plothandlers` module of the Ti*k*Z library, or the `tikz-3dplot` Ti*k*Z extension package, not to mention the modules associated with PSTricks. However, I saved myself the study of the 700-plus pages of Ti*k*Z documentation or a similar amount for PSTricks. The fonts in figure 6 are just the ones I designed myself for presentations; they were fully described in [1], and are available on any complete recent distribution of the TEX system.

4 Conclusion

As mentioned in the introduction, the *picture* environment is a very simple one, with but few and specific drawing commands; the documentation is so simple that less than 10 pages are sufficient. At the same time these simple commands may be used to create more complex macros and eventually result in professional drawings. Certainly this environment cannot compete with more elaborate ones, such as those provided by the packages Ti*k*Z and PSTricks; but the latter require a steep learning curve, while the former can be mastered in a few minutes. Very often the results obtained with the *picture* environment, completed by the recent enhancements provided by the `pict2e` package, are fully acceptable: it's possible to create complicated diagrams as well as simple symbols; it's possible to use this environment to place background or foreground images or symbols "wherever" on the page, even outside the margins; in [2] it is shown also how to make strange and unusual tables, Cartesian diagrams, and any sort of mix between line art and included pictures. In any case, if any lettering is placed in the drawing, it surely uses the same fonts as those used for the text, thus eliminating the usual risk that occurs when using external drawing software.

I believe that beginners would find this enhanced environment the right first step for programmed drawings; with minimum effort they can reach very good results.

References

[1] Claudio Beccari. lxfonts: LATEX slide fonts revived. *TUGboat*, 29(2), 2008. Reprinted from ArsTEXnica #4, 2007.

[2] Claudio Beccari, editor. *Introduzione all'arte della composizione tipografica con LATEX*. GuIt, 2011.

[3] Melchior Franz. The `crop` package, 2003. `http://mirror.ctan.org/macros/latex/contrib/crop`.

[4] H. Gäßlein, R. Niepraschk, and J. Tkadlec. The `pict2e` package, 2011. `http://mirror.ctan.org/macros/latex/contrib/pict2e`.

[5] Donald E. Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1990.

[6] Leslie Lamport. *LATEX: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, 2nd edition, 1994.

[7] Ian Maclaine-cross. The `curves` package, 2009. `http://mirror.ctan.org/macros/latex/contrib/curves`.

[8] Kent McPherson. The `layout` package (part of the `tools` bundle), 2000. `http://mirror.ctan.org/macros/latex/required/tools`.

[9] R. Niepraschk. The `eso-pic` package, 2010. `http://mirror.ctan.org/macros/latex/contrib/eso-pic`.

[10] M. A. Redaelli. CircuiTi*k*Z, 2011. `http://mirror.ctan.org/graphics/pgf/contrib/circuitikz`.

[11] Till Tantau. The Ti*k*Z and PGF packages, 2010. `http://mirror.ctan.org/graphics/pgf/base/doc`.

[12] Timothy van Zandt. PSTricks: PostScript macros for generic TEX, 2011. `http://mirror.ctan.org/graphics/pstricks/base/doc`.

[13] Herbert Voß. The `pst-circ` PSTricks package, 2011. `http://mirror.ctan.org/graphics/pstricks/contrib/pst-circ`.

[14] Michael Wichura. *PiCTEX*. Department of Statistics, University of Chicago, 1987.

[15] Peter Wilson and Will Robertson. The `layouts` package, 2009. `http://mirror.ctan.org/macros/latex/contrib/layouts`.

◇ Claudio Beccari
Villarbasse (TO), Italy
`claudio dot beccari (at) gmail dot com`