PSImage: Macros Postscript para figuras y diagramas

Jaime E. Villate
Faculdade de Engenharia
Universidade do Porto
Rua dos Bragas, 4050-123 Porto, Portugal
villate@fe.up.pt

Octubre de 1999

Resumen

Son discutidos algunos ejemplos de figuras y diagramas que pueden aparecer en un documento técnico o una tesis y se presenta una metodología que permite obtener consistencia en todos los diagramas de un documento. Con la metodología propuesta es posible también producir muchas figuras en un espacio de memoria reducido, y realizar cambios globales en el estilo de todas las figuras y diagramas de un documento. El lenguaje PostScript es usado por ser bastante apropiado para la creación de gráficas vectoriales y por existir una implementación libre (ghostscript) de uso muy extendido. Las macros presentadas pueden ser usadas en cualquier sistema operativo.

1 Introducción

Existen actualmente varias herramientas que permiten crear figuras de elevada calidad. Sin embargo la mayor parte de manuales libres que han aparecido últimamente no presentan ninguna figura. La principal dificultad es que la creación de una ilustración requiere bastante tiempo y algún talento artístico; el tipo de figura que suele aparecer con mayor frecuencia es el llamado "screenshot" que puede ser creado fácilmente con unos pocos toques en los botones del ratón, pero pueden producir figuras con tamaños de varias centenas de kilo-bytes.

Para obtener un buen resultado gráfico es conveniente usar un formato vectorial para crear diagramas y figuras. El sistema mas usado en Linux para producir documentos que serán imprimidos es el LaTeX[1], que puede ser obtenido a partir de una versión mas general del documento, por ejemplo en lenguaje SGML. Cuando existen figuras en un documento LaTeX, normalmente se usa el programa *dvips*[2] para obtener un fichero PostScript que pueda ser imprimido. La especificación del lenguaje PostScript es libre, y existe una implementación libre de este lenguaje, llamado *ghostcript*[3]. El lenguaje PostScript es muy flexible y permite producir la misma figura en muchas formas diferentes, incluso como un *bitmap*. Por ejemplo, la figura 1 podría haber sido creada con *gimp*[4], pero en ese caso el resultado seria una imagen *bitmap* (a pesar de haber sido exportada en PostScript) que ocupa mucho espacio, no es fácil de modificar y su calidad puede ser bastante mala si la resolución usada no es la apropiada; otro

método mas eficiente consiste en definir la figura en forma lógica: por ejemplo los dos cilindros verticales pueden ser referidos en la figura únicamente por las coordenadas de una de las esquinas, y una llamada a una macro *Cilindro* que puede ser definida en un prólogo común a todo el documento.

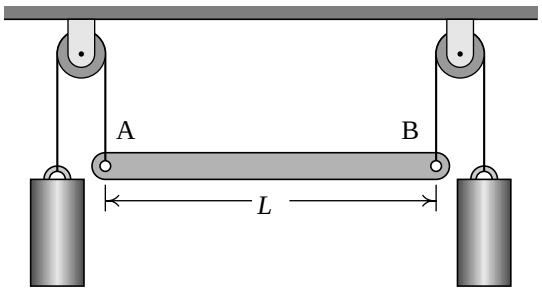

Figura 1: Ejemplo de una ilustración en un artículo técnico.

El programa *dvips* permite incluir un prólogo PostScript, donde pueden estar definidas algunas macros que serán usadas en varias figuras del documento LaTeX. De esta forma se ahorra mucho espacio en disco y se pueden modificar varias figuras con una simple alteración del prólogo.

Otro ejemplo que aparece con mucha frecuencia en documentos técnicos son las gráficas de funciones. Consideremos por ejemplo la función en la figura 2; desde el punto de vista matemático, la información de la gráfica puede ser resumida a una función simple y a la especificación de su dominio. Los pormenores de como dibujar los ejes o donde poner los nombres de las variables son secundarios, pero conviene que sean uniformes en todas las figuras del documento y que puedan ser alterados en forma global. Por ejemplo si el editor de un libro sugiere al autor que cambie el tipo de letra o el tipo de flechas que usó para los ejes de todas las gráficas en un libro, esa modificación puede ser hecha muy fácilmente si el autor definió esos parámetros en un prólogo común a todas las figuras.

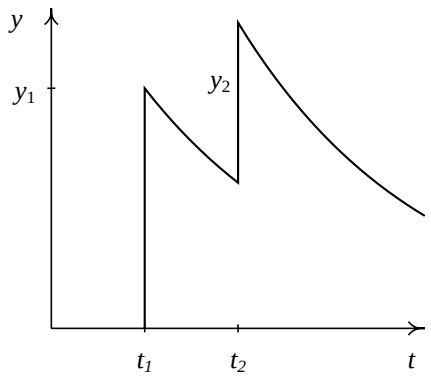

Figura 2: Ejemplo de gráfica de una función en dos dimensiones.

El programa *gnuplot*[5] es una alternativa muy buena para crear gráficas como la de la figura 2, pero en el fichero PostScript creado no queda la información lógica que fue

usada (a menos que el autor se haya tomado el cuidado de guardar comandos de gnuplot en un fichero) para generar cada gráfica y si llega a ser necesario hacer modificaciones globales, puede ser preciso volver a producir cada gráfica individualmente.

Los ejemplos presentados en este artículo fueron creados con un conjunto de macros (*psimage*) que el autor ha ido escribiendo y coleccionando a lo largo de los últimos 15 años, y que están disponibles (con licencia GPL) en la URL de *psimage*[6]. Las macros están agrupadas en un prólogo que ocupa unos pocos kilo-bytes y que ha sido usado para producir 233 figuras en un libro[7]; las 233 figuras ocupan únicamente 674 Kbytes, a pesar de algunas ser mas complicadas que la figura 1.

2 Macros en lenguaje PostScript

El lenguaje PostScript permite definir rutinas (macros) que pueden ser ejecutas repetidamente con diferentes valores de entrada. Los valores de entrada de la rutina pueden entrar a través de variables o de un *stack* que mantiene el interpretador de PostScript. Una peculiaridad del lenguaje PostScript es que usa la notación polaca inversa (RPN), en la cual primero se especifican los operandos (que entran en el *stack*) y luego el operador. Por ejemplo, para sumar 3 mas 4 se usaría el siguiente comando: 3 4 add. El número 3 es "empujado" en el stack y luego el 4 por encima de el; el operador *add* saca los dos números superiores en el stack, dejando su suma (7) en la parte superior del *stack*.

Tanto las variables como las macros se definen introduciendo un nombre simbólico precedido de un *slash*, después viene la definición y finalmente el operador *def*; por ejemplo las siguientes instrucciones definen una variable: "factor_conversión" y una macro que usa esa variable para convertir de pulgadas a centímetros:

```
/factor_conversion 2.54 def
/pul_cm {factor_conversion mul} def
```

En el segundo caso es obligatorio usar corchetes para indicar que lo que allí está no debe ser interpretado (*mul* no causará ningún efecto aún), sino guardado como una definición de una macro. Para utilizar la macro "pul_cm" se tiene que escribir primero un número que será convertido a cm; por ejemplo: 3.78 pul_cm, convierte 3.78 pulgadas en centímetros; el número 3.78 entra en el *stack*, seguido del valor numérico de la variable factor_conversion, que son luego multiplicados.

La especificación completa del PostScript se encuentra en un libro publicado por la Adobe[8], y otras referencias actualizadas son dadas en la página de *psimage*[6].

3 Algunos ejemplos

Las figuras 3 y 4 muestran un par de ejemplos usando algunas de las macros que han sido definidas en el prólogo *psimage.pro*[6]. El fichero PostScript usado para producir la figura 3 es un fichero normal de texto, que puede ser creado con cualquier editor de texto (en cualquier sistema operativo), con el siguiente contenido:

```
0.5 setlinewidth
Plotreset Setaxes
0 0 3 4.5 Domain
(t) Xlabel (f\(t\)) Ylabel
```

```
(3) 3 Ymark (1) 1 Xmark (2) 2 Xmark
1 setlinewidth
[0 0 1 3 2 0] Plot
showpage
```

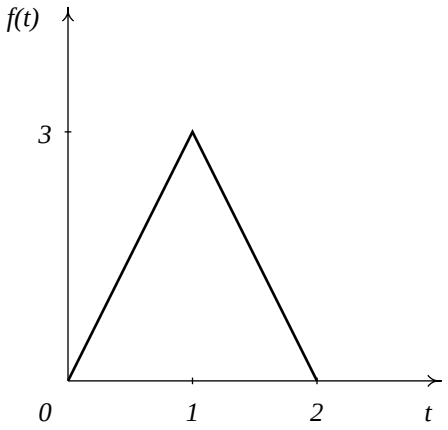

Figura 3: Función definida por las coordenadas de 3 puntos.

PostScript distingue entre letras mayúsculas y minúsculas, y los nombres de los comandos son normalmente en letras minúsculas; para distinguir las macros definidas en nuestro prólogo, usaremos nombres que comienzan por letras mayúsculas. La unidad de distancia usada por PostScript es el punto, que equivale a 1/72 pulgadas. El comando *setlinewidth* define el ancho de las líneas que serán de 0.5 puntos para los ejes y 1 punto para la función. La macro *Plotreset* la usamos para ejecutar algunas cosas necesarias antes de comenzar una gráfica. La macro *Domain* define el dominio de la función. Las macros *Xlabel* y *Ylabel* se usan para indicar lo que debe ir en cada eje. Las macros *Xmark* y *Ymark* colocan números en algunas partes de los ejes, y finalmente la gráfica de la función se obtiene dando las coordenadas de los puntos a la macro *Plot*. Las coordenadas de los puntos se dan en una lista que en PostScript se crea usando los paréntesis cuadrados. El comando *showpage* termina la definición de una página y es necesario para ordenar al interprete de PostScript que dibuje la página.

Para poder ver la figura, será preciso anexar al comienzo el prólogo que define las macros. Si por ejemplo queremos ver la gráfica usando el programa *gv*, podemos usar el siguiente comando:

```
cat psimage.pro triangular.ps | gv -
```

Donde *psimage.pro* es el prólogo, disponible en la URL de *psimage*[6], y *triangular.ps* es el fichero donde creamos la figura.

El programa *gv* (o *ghostview*) es útil para identificar la *Bounding Box* de la figura: coordenadas de los puntos inferior izquierdo y superior derecho de una caja que envuelva a la figura. Por ejemplo, quien haya creado el fichero *triangular.ps* de la forma como se indicó arriba, puede mover el cursor sobre la figura y leer las coordenadas en una caja a la izquierda en la ventana de *gv*. Conviene introducir estas coordenadas en un comentario al comienzo del fichero *triangular.ps*, de la siguiente forma:

```
%%BoundingBox: 215 507 389 676
```

Así será posible introducir la figura en un documento LaTeX, por medio del comando:

```
\includegraphics{triangular.ps}
```

(o para quien prefiera usar *epsfig*: `\epsfig{file=triangular.ps}`).

En el preámbulo del documento LaTeX deberá ser indicado el fichero donde se encuentran las macros, después de indicar que el paquete *graphics* (o *graphicx*, *epsfig*, etc) será usado:

```
\usepackage[dvips]{graphics}
\special{header=psimage.pro}
```

Hoy en dia es también muy usado el formato *pdf* (*Portable Document Format*) en vez del PostScript. En la página *www* que acompaña este artículo[6], se encuentra disponible un programa en *perl* llamado *psimagetopdf*, que permite transformar cualquiera de las figuras creadas con *psimage* en un fichero *pdf*. Por ejemplo, para crear el fichero *triangular.pdf* a partir de *triangular.ps* (después de haber definido una *Bounding Box* como se explicó arriba), usamos el siguiente comando:

```
psimagetopdf triangular.ps
```

La figura 4 muestra otro ejemplo en el cual es muy útil el uso de macros: un circuito eléctrico. El fichero usado para producir la figura 4 es el siguiente:

```
%!PS-Adobe-2.0
%%Title: circuito.ps
%%Creator: Jaime E. Villate
%%CreationDate: October 1999
%%BoundingBox: 110 505 285 630
12 /Times-Roman Font 120 610 moveto
0 -80 rlineto 40 0 Resistor
currentpoint -20 5 rmoveto
(300 [W]) Center Show moveto
60 0 Femp currentpoint -30 -20 rmoveto
(1,5 V) Center show moveto 30 0 rlineto
80 90 Resistor currentpoint
5 -45 rmoveto (50 [W]) Show moveto -70 0 rlineto
120 610 moveto 70 0 Resistor currentpoint
-35 5 rmoveto (100 [W]) Center Show
moveto 40 270 Openswitch 70 180 Resistor
35 10 rmoveto (R_{3}) Center Show
stroke showpage
%%EOF
```

En este ejemplo han sido usadas tres macros *Resistor*, *Femp* y *Openswitch* para dibujar resistencias, baterías (pasando primero por el terminal positivo) y interruptores abiertos, a las cuales se les debe dar una distancia y un ángulo. Las macros *Center* y *Font* sirven para escribir texto que puede tener subíndices o superíndices y letras griegas (entre paréntesis cuadrados). La letra griega Ω en PostScript tiene el mismo código que la letra W; para usar estas macros es preciso definir la fuente y el tamaño de letra, por medio de *Font*. Note que existe también un comando *show* de PostScript que permite escribir texto, pero no acepta las extensiones de índices y letras griegas permitidas por *Show*. El comando *currentpoint* de PostSript guarda en el *stack* las

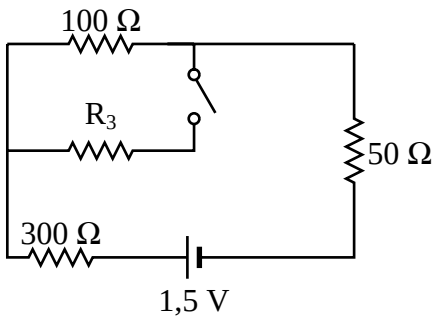

Figura 4: Circuito eléctrico con resistencias, interruptor y batería.

coordenadas actuales del cursor, que serán usadas mas tarde para saltar a ese punto usando el comando *moveto*. El comando *rmoveto* mueve el cursor en relación al punto actual, y *rlineto* hace lo mismo pero también dibujando una línea entre el punto anterior y el nuevo punto; este último comando y *show* realmente no dibujan nada, sino definen un conjunto de puntos de la página (llamado *current path*) que solo serán dibujados cuando se dé el comando *stroke*, el cual dibuja lo que esté definido en el *current path*.

En este caso, si por ejemplo quisiéramos cambiar el diagrama de circuito usado para todas las resistencias por algo diferente como un bloque rectangular, bastaba modificar la macro *Resistor* en el prólogo *psimage.pro*, o redefinirla al comienzo de la figura.

4 Conclusiones

Hemos mostrado una metodología para producir figuras de alta calidad usando PostScript en cualquier sistema operativo. Dimos algunos ejemplos que usan unas macros creadas por el autor, y que son distribuidas libremente (licencia GPL) a partir de la página de *psimage*[6]. Los ejemplos dados solo tocan una pequeña parte de las posibles aplicaciones de las macros incluidas en *psimage.pro*.

Existen muchos sistemas que usan una metodología semejante a la que fue usada aquí. Los mas comunes son *Pstricks* y *Metapost*. Sin embargo debo admitir que no los he usado (*psimage* ha sido suficiente para producir las figuras que he necesitado en los últimos años) y por eso no puedo hacer una comparación exhaustiva; el espíritu del *software* libre no se basa en la competición entre programas de diferentes autores, sino que existen una infinidad de herramientas que son puestas a disposición por diversos autores, y de esta diversidad de métodos para realizar una misma tarea van surgiendo proyectos de mayor dimensión que sintetizan lo mejor de cada herramienta. Bajo ese espíritu de colaboración pongo a disposición *psimage* esperando que seja útil para autores de documentos técnicos.

Un rasgo común que puedo identificar en varios de los sistemas usados para crear figuras vectoriales, es la tendencia a crear sus propios lenguajes, que no llegan a ser tan completos como el PostScript. Mi opinión es que es mas conveniente usar directamente PostScript que es un lenguaje mas universal, libre y con muchas mas posibilidades.

Quizás la mayor dificultad para usar estas macros será la falta de un interfaz gráfico que ayude a crear y modificar las figuras. En el futuro espero poder crear ese tipo de interfaz o aprovechar interfaces ya existentes como los de *gimp* o *xfig*, produciendo ficheros que usen macros existentes en *psimage.pro*.

Referencias

[1] Ver por ejemplo http://www.tex.ac.uk/

[2] Dvips es el programa usado para producir ficheros PostScript a partir de ficheros *dvi*. Escrito por Tomas Rokicki.

[3] Ver: http://www.cs.wisc.edu/∼ghost/.

[4] Gimp es el programa de manipulación de gráficos de GNU (*GNU Image Manipulation Program*); fué creado inicialmente por Peter Mattis y Spencer Kimball. Ver: http://www.gimp.org

[5] Gnuplot fué escrito inicialmente por Thomas Williams y Colin Kelley. Ver: http://www.cs.dartmouth.edu/gnuplot_info.html

[6] http://www.fe.up.pt/∼villate/psimage.html

[7] J. E. Villate, *Electromagnetismo*, McGraw-Hill, Lisboa, 1999.

[8] Adobe Systems Incorporated, *PostScript Language Reference Manual*, Addison-Wesley, Reading, Massachusetts, 1985.