DVIPDF and Graphics

Sergey Lesenko
Institute for High Energy Physics
Scientific Information Department
142284 Protvino, Moscow Region, Russia
lesenko@mx.ihep.su

Abstract

This paper describes how the *dvipdf* program inserts BMP, JPEG, PNG and EPS graphics into its output. It also discusses geometric transformations of image, text and rule objects.

Introduction

For today's scientific publications, we have to provide for accessibility via the Internet, and need an effective presentation format for our documents which supports a wide range of hypertext and graphic features. One option is the Portable Document Format (PDF) [1], probably the most popular system for distributing complex formatted documents. PDF permits us to use color, geometric transformations and included images (vector and bitmap). This paper describes how a document in PDF with such graphic features may be prepared using TEX and *dvipdf*.

Graphic commands

Although TEX has only a limited capability to deal with graphics, it has the wonderful property of the `\special` command, which allows us to perform arbitrary tasks at the level of driver programs, which include *dvipdf*. This is based on *dvips* [2], and it would be preferable to support the same commands. However, *dvipdf* is oriented to a more confined output format, and a new set of `\special` commands has been introduced to permit better performance. The syntax for these has already discussed in [8], and all the `\special` commands for graphics are listed in Table 1

Only a minimal number of parameters for the DVI file is required, and the full set needed in the PDF output is computed by *dvipdf*. This means that writing macros or incorporating the `\special` commands into style packages is not difficult. The `color` [3], `graphics` [4] and `graphicx` [4] packages, for instance, have 'dvipdf' drivers which make use of the appropriate `\special` commands.

A particular problem with inclusion of bitmap graphic formats like BMP, JPEG and PNG is that the user has to provide TEX with the image size (directly or via an additional *bb* file). Encapsulated

Geometric	
Begin rotation	`pdf: /ROT` *angle* `<<`
End rotation	`pdf: /ROT >>`
Begin scaling	`pdf: /SC` *hscale vscale* `<<`
End scaling	`pdf: /SC >>`
Color	
Begin color	`pdf: /C` *Yellow* `<<`
End color	`pdf: /C >>`
Page	
Page color	`pdf: /BG` *Blue*
Page rotate	`pdf: /ROTPAGE` *angle*
EPS	
Mode	`pdf: /IMAGE` *rgb*16*m*
Resolution	`pdf: /RES` 600
Insertion	`pdf: /GRAPH` *file llx lly urx ury*
BMP, JPEG and PNG	
Insertion	`pdf: /GRAPH` *file width height*

Table 1: `\special` commands supported by *dvipdf*

PostScript files have a `%%BoundingBox` line which TEX can read, but it cannot (easily) read binary formats. Scaling and rotation can use the commands of the standard LATEX graphics package.

For example, the following code inserts a JPEG image, rotates it, and scales it. Note the optional argument to `\includegraphics` to specify the original image size:

```
\resizebox*{2in}{!}{%
\rotatebox{90}{%
\includegraphics[12cm,9cm]{photo.jpg}}}
```

Basic algorithms

Since geometric transformations in PDF are one of its most powerful features, we will begin our description of *dvipdf* innards in this area.

To allow both single level transformations and nested transformations, *dvipdf* supports a 16 level stack. For each level eight parameters are recorded: the initial coordinates (x, y) of the reference point for the layer, offsets to its position after transformation (dx, dy) and factors (a, b, c, d) for the current transformation matrix (CTM). Each new transformation (or layer) pushes the current parameters and calculates new parameters. Corresponding returns from each layer pops the parameters. To define the coordinate of each point on current layer, *dvipdf* computes its position (ddx, ddy) in relation to the reference point and adds the coordinate of the reference point ($x + dx$, $y + dy$).

The transformations for points apply equally to other types of object (rule, text and image). Let us consider the simplest object (a rule); this may be dealt with as a rectangle or a region — a rectangle is described with two points, and a region with four points. To optimize the output, the two types of rule (with rotating and without it) are treated differently. Rules without rotating are treated as rectangles and described with two points A, B (Fig. 1(a)). This requires four arguments (ulx, uly, w, h): base point, width, and height. Rotated rules are treated as regions with four points A, B, C, D (Fig. 1(b)). This requires eight parameters: (llx, lly, lux, luy, urx, ury, ulx, uly)

Apart from geometric transformation, there are other PDF objects that use the same algorithm (although they are used only during interactive viewing and are not printed). These objects are link annotations and bookmarks. To specify a link annotation, and the destination for a link annotation or bookmark, we need to know the rectangle of the location. To compute the rectangle, we firstly utilise the algorithm for the definition of a region, and then

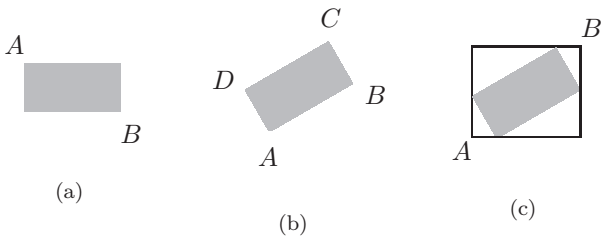

Figure 1:

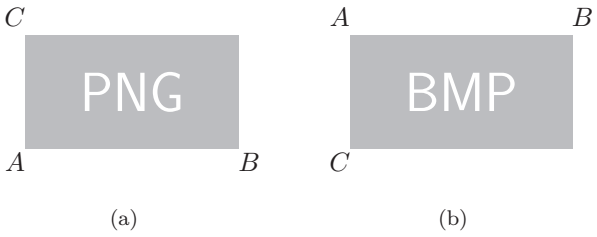

Figure 2:

calculate a bounding box (llx, lly, urx, ury) for this region. This is defined using two points A, B (Fig. 1(c)).

Now we can consider transformation of text and images. These objects are managed using the PDF page marking operators **Tm** (for text) and **cm** (for images). They are supported in *dvipdf* in similar ways, using the geometric figure of a parallelogram. It is described by three points (Fig. 2(a)). The first point is lower-left corner, the second point is the lower-right corner and the third point is the upper-left corner. The first point A is defined as a base point (x, y) and other two points B, C as offsets (a, b) and (c, d) from the base point. These six parameters (a, b, c, d, x, y) allow us to do all the transformations that are needed.

Since a BMP image has a bottom-to-top order and it is saved during the initial parse, its parallelogram is described by other points (Fig. 2(b)), where the base point A is the upper-left corner.

Color management for text (here 'text' includes rules too) is nested. The Current version of the color stack permits us to reverse the order of pages and to produce separated pages.

Images

It is usually impossible to know exactly with which resolution our document will be viewed or printed by its eventual receiver. So when we deal with bitmap image formats like BMP, JPEG and PNG, the best

```
\setres{300}
\setimage{rgb16m}
\fboxsep = 0 cm
\fboxrule = 0.6cm
\baselineskip = 0pt

\def\epsm{\textcolor[named]{Green}{\fbox{%
\scalebox{-1}[1]{%%
\includegraphics{tiger.eps}}}}}


\def\new{\scalebox{-1}[1]{%
\rotatebox{30}{\resizebox*{2in}{!}{\epsm}}}%
\rotatebox{30}{\resizebox*{2in}{!}{\epsm}}}
\def\newt{\scalebox{1}[-1]{\new}}
\resizebox*{8cm}{!}{%
\textcolor[named]{ForestGreen}{\fbox{%%
\vtop{\hbox{\new}\hbox{\newt}}}}}
```

Figure 3:

we can do is preserve the original resolution in the PDF output. This allows us to avoid getting concerned with intermediate conversion of resolution. The same principle is also applied to the mode of the image, which is simply preserved (i.e., mono, gray, RGB, CMYK and Indexed).

Dealing with EPS images is a different matter. This type of vector image is inserted after processing with GhostScript [6]. Ghostscript allows us to create a bitmap image, with appropriate settings of mode and resolution. How do we establish those settings? To work them out, we need to set the appropriate resolution (R)and mode via `\special` commands. *dvipdf* can then compute the real resolution R_x and R_y for Ghostscript:

$$R_x = R \cdot \frac{W_{actual}}{W_{original}}$$

$$R_y = R \cdot \frac{H_{actual}}{H_{original}}$$

where $W_{original}$ and $H_{original}$ are the width and height, (as derived from the BoundingBox in the EPS file), and W_{actual} and H_{actual} are the sizes of the parallelogram sides. In some images the form may not be a rectangle, so they are calculated as the distance between corners.

Using the current version of Ghostscript, bitmap images are produced corresponding to the page size, so the possibility has been added of producing them with the rectangle as the BoundingBox.

At present we ask Ghostscript to produce temporary files, and these are then placed in the output (in future versions it should be possible to merge Ghostscript's output stream directly into *dvipdf*'s output). We can select an option to save the temporary files and to re-use them for the next run of *dvipdf*, if the same parameters are used for images. The unique parameters for each image are recorded as EPSF structured comments (file name, date, time and file size, and parameters for for Ghostscript processing, i.e., mode and X, Y resolutions). We can generate these parameters and compare them during the next processing by *dvipdf*, and name temporary files accordingly. We adopt the simple solution of computing a Cyclic Redundancy Check value (CRC) [7] on the basis of the parameter set and use this CRC as the basis for naming temporary files. We can thus avoid calling Ghostscript wherever possible, and so reduce the time taken to prepare our PDF document.

Repeated images (for example a logo on every page of a document) are organized as references to a common object with just one instance. For bitmap images the basis of deciding if something can be stored as a common object is just the name of graphics file — rotating and resizing are done at the time of the object instance. For EPS images, we also need to consider resolution and mode. To illustrate this use of common objects, four tigers are placed together in Fig. 4, with the TEX code used to produce

the picture. Only one copy of the graphics file is included in the output.

Acknowledgements

I would like to thank Michel Goossens and Mimi Burbank for their support of this project, Laurent Siebenmann for test samples, David Carlisle and Sebastian Rahtz for adapting their graphics and hypertext packages to *dvipdf*, and Sebastian Rahtz for editing this paper and helping with the English.

I am very grateful to Tim Bienz for his patient explainations of PDF, and Peter Deutsch for his useful remarks.

Finally thanks to Tomas Rokicki for his wonderful *dvips* which has been the starting point for this project.

References

[1] Tim Bienz and Richard Cohn, *Portable Document Format Reference Manual*, Adobe Systems Incorporated, 1993, Addison-Wesley Publishing Company. ISBN 0-201-62628-4. This document is available from `http://www.adobe.com/supportservice/devrelations/PDFS/TN/PDFSPEC.PDF`

[2] Tomas Rokicki, *Dvips: A TEX Driver*, distributed with *dvips*, version 5.58, 1994. electronic distribution from `labrea.stanford.edu`.

[3] David Carlisle, *The color package* (on CTAN)

[4] David Carlisle and Sebastian Rahtz, *The graphics package* (on CTAN)

[5] David Carlisle and Sebastian Rahtz *The graphicx package* (on CTAN)

[6] L. Peter Deutsch, *Aladdin Ghostscript* version 4.03, 1996 electronic distribution `ftp.cs.wisc.edu://pub/ghost/aladdin`

[7] L. Peter Deutsch, *RFC 1952: GZIP 4.3 specification*, `ftp://ftp.uu.net/graphics/png/documents/zlib/zdoc-index.html`

[8] Sergey Lesenko, *The DVIPDF Program,* TUGboat 17, 3, September 1996, pages 252–254.