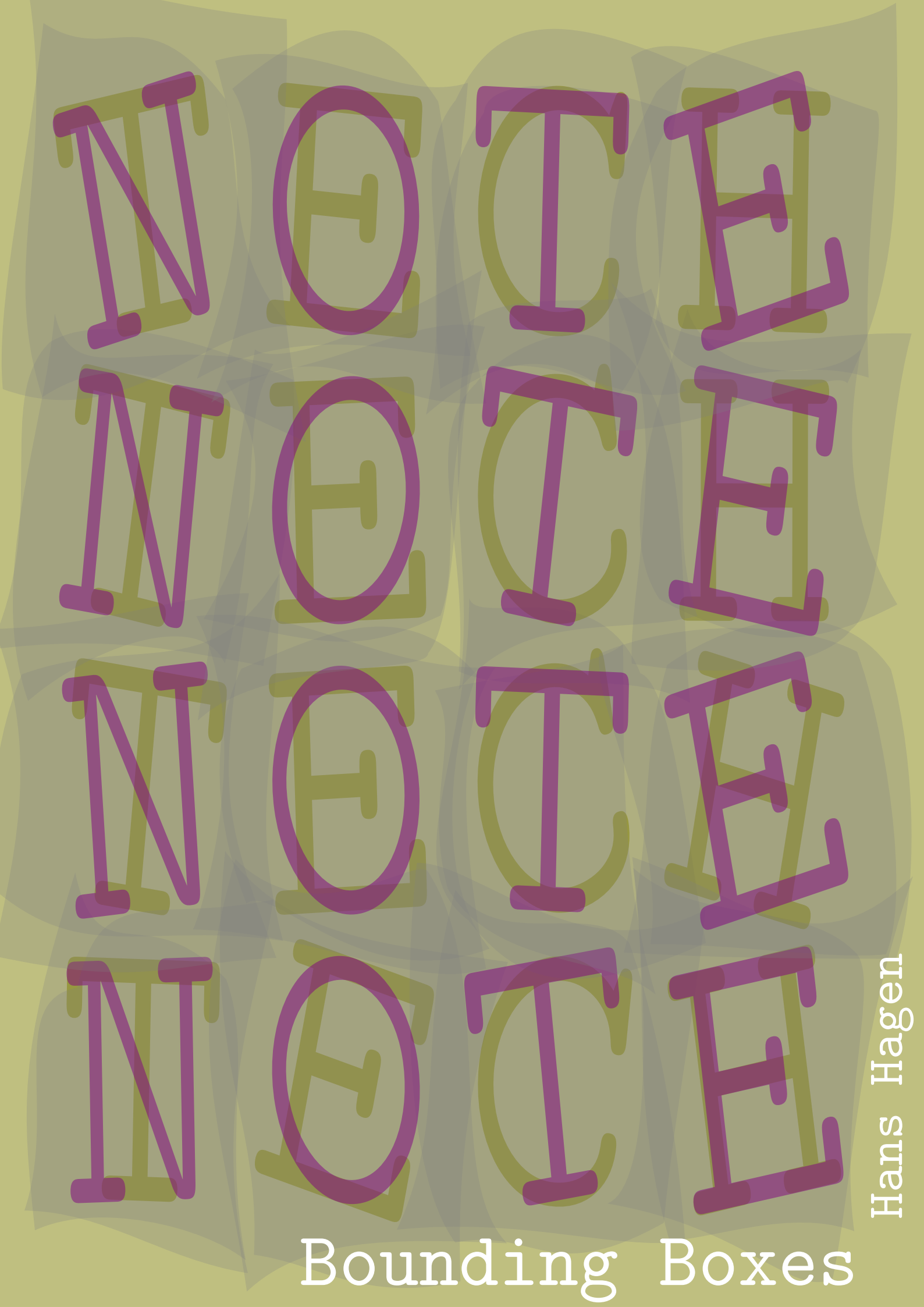

Bounding Boxes

Hans Hagen

For most ConTEXt users, placement of graphics is something that "just happens" to work. In their document source you may find commands like:

```
\placefigure
  {An example figure.}
  {\externalfigure[hacker.jpg][height=3cm]}
```

This results in a placement like:

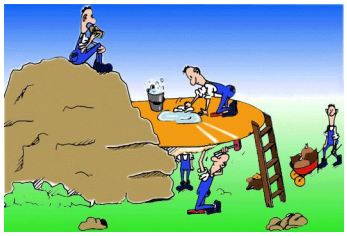

Figure 1 An example figure.

A graphic —as in this example— can be a bitmap, or a vector drawing, or the result from embedded METAPOST code. The previous example demonstrates that ConTEXt handles the spacing around the graphic.

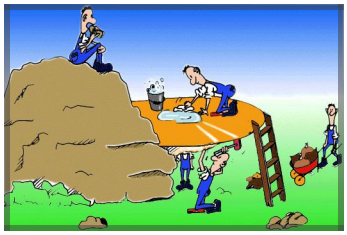

Figure 2 An example figure with frame.

In the framed example figure, the frame matches the natural size of the graphic and is drawn on top of the graphic. As you can see, no graphic content falls outside the natural bounds of the graphics. The rectangular box that determines the dimensions of a graphic is called the bounding box (although a graphic may have any shape). The bounding box is always a rectangle and its position in a carthesian system and its size is determined by two coordinates: the lower left corner and the upper right corner.

When dealing with vector graphics, the natural boundaries are often not as prominent as with bitmaps, but we still have a boundingbox. Although in the example graphic the boundingbox is tight you often cannot trust this boundingbox to be tight especially when such a graphic is made outside our control,.

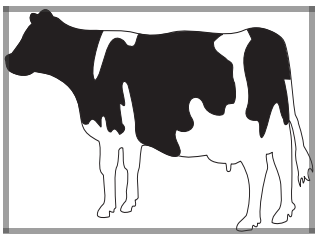

Figure 3 A vector graphic.

There can be good reasons to have a boundingbox that is not tight. To mention a few:

- the graphic is part of a series, and the sizes and scales of the individual graphics need to match
- a graphic needs a displacement from its natural position (e.g. part of a graphic falls behind a text)
- a graphic needs to bleed (i.e. part falls of a page in order to compensate for cut errors

Now, instead of messing around with the boundingbox, one can best see to what extend the typesetting engine can handle such anomalities. In most cases ConTEXt permits you to mess around with graphic placement and therefore we strongly recommend to stick to tight boundingboxes when possible, especially when you want to reuse graphics.

The bad news is that due to the fact that desk top publishing applications use a different approach (read: graphics are placed manually), most graphics that are made by non TEX users, have wrong bounding boxes. Even worse, there are applications that provide the wrong and/or inaccurate dimensions, and (believe it or not) in many cases we have to rebound third party graphics before they can be used.

Real bad situations can occur when a graphic is not positioned at all. In a drawing program one normally sets up a canvas (say a sheet of letter paper) and starts drawing on it. When it's time to save the graphic, well, the graphic is just saved. Clipping or bounding then takes place in the desk top publishing application. In the worst case, the graphic is positioned outside the paper, and it may become real messy when scratch bits and pieces are still there and positioned somewhere out of sight (that is: they will show up when the bounding box is wrong).

This phenomena is obscured by the fact that programs that deal with such graphics can recognize these flaws and compensate for them, thereby obscuring bugs and flaws. In that case it may be hard to communicate such a bug to the designer, simply because he or she may lack the technical knowledge and quite certainly is not going to look inside the PostScript or pdf file.

In one of our projects one of the graphics is generated by ConTEXt, by TEX and METAPOST in tandem. This makes sense because each chapter in a huge series of

books has one of these graphics and making them by hand is no fun, especially since we're talking of hundreds of chapters and thousands of graphics.

In the introduction of a similar graphic is included, but that one is not generated by the typesetting engine, but made by a third party. We will use that graphic to illustrate the bounding box problem.

When we include the original graphic as it is, we get the following result. To start with, the graphic is not properly centered horizontally. In a dtp system this is handled by manual shifting.

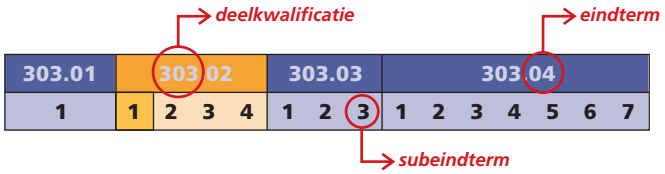

Figure 4

We've put a frame around the graph. That way you get a pretty good impression of the spacing problems.

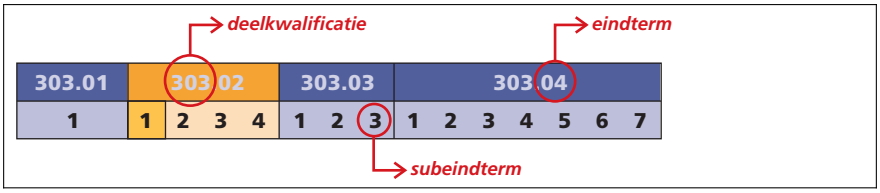

Figure 5

There is not only a horizontal spacing problem, but also a vertical one. It will be clear that compensating such margins can only be done automatically by looking into the graphic (brute force).

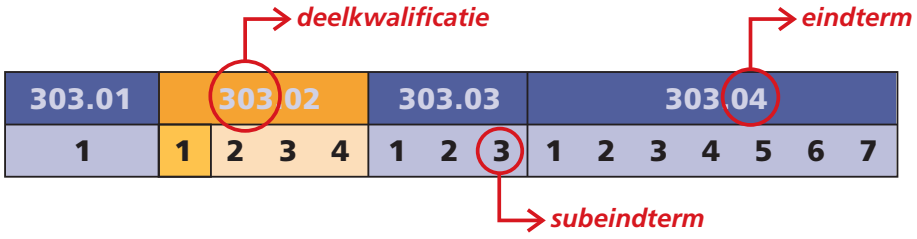

Figure 6

Figure 6 shows the same graphic scaled to the text width. As you see, such a request is not honoured in the visual sense.

What on the left may look like a small margin is again part of the graphic. Never put such margins in a graphic: ConTEXt can do that for you if needed.

We've used Acrobat to crop the graphic and this time we get a properly centered graphic.

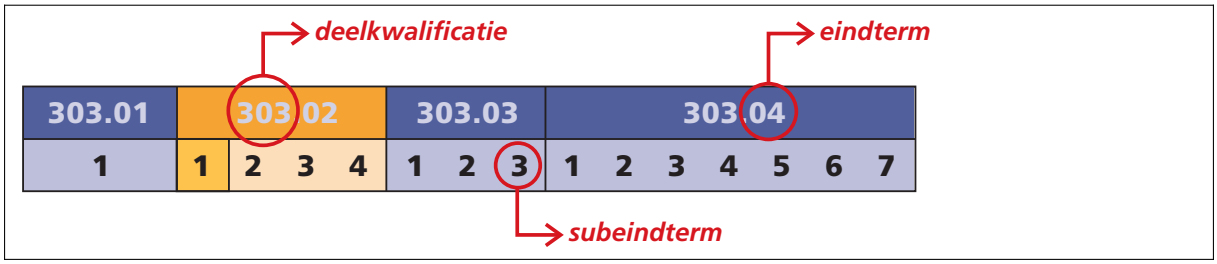

Figure 7

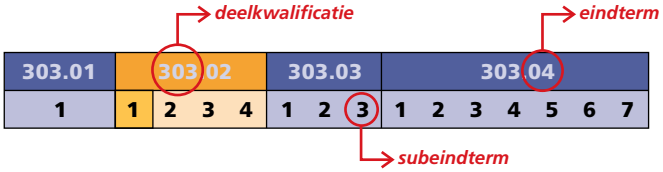

Figure 8

When graphics have text like this, it makes sense to let the bottom bounding line be the baseline of the text, and the top bounding line the proper strut height (for not ConTEXt users: a strut is an invisible character with only height and depth, totaling to the lineheight). In practice this wish is hard to (explain and therefore) honor.

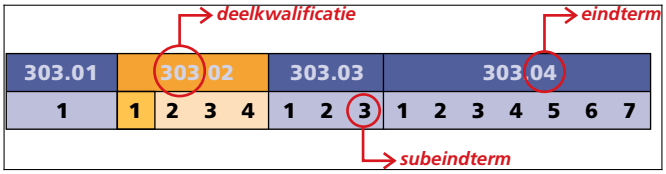

Figure 9

This time we can properly scale to the line width and even now we get inaccuracies. It would make sense to make the right boundingbox align with the right boundary of the boxes and so let the text stick slightly out of that box.

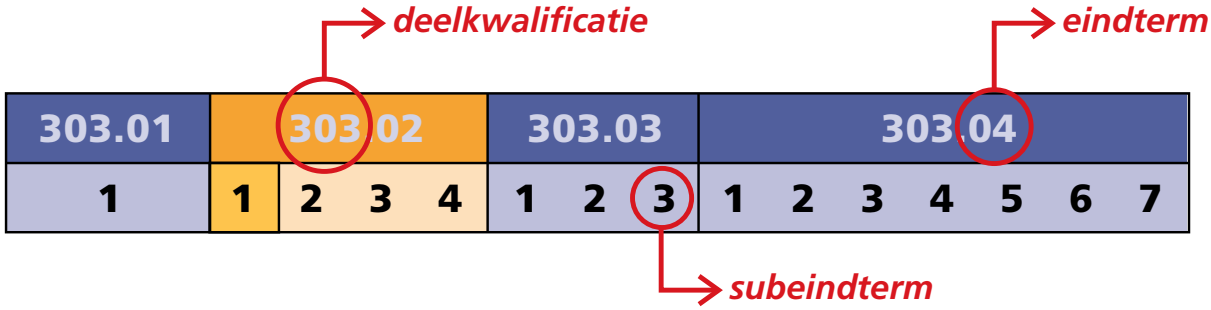

Figure 10

It would have made sense to make this graphic in ConTEXt as well, if only because we then have more control over the boundingbox and so let the text stick out.

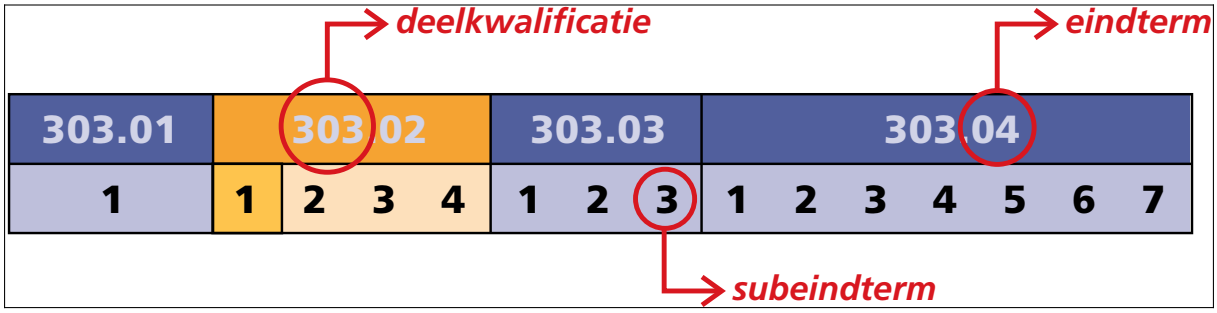

Figure 11

Here is the alternative that is used in each chapter introduction. As said, this variant is generated by METAPOST based on an xml database and a chapter specific specification.

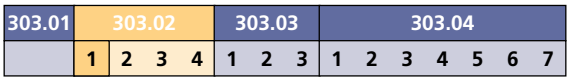

Figure 12

This time there is no difference between the framed variant and the frameless one. The frame is obscured by the frame that is part of the graphic itself.

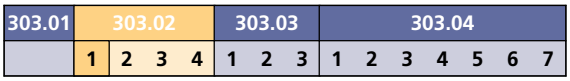

Figure 13

This alternative is suited for scaling to the width of a page. In practice this will not happen. Some instances of the graphic are way to wide for the page, and are clipped in a sensible way.

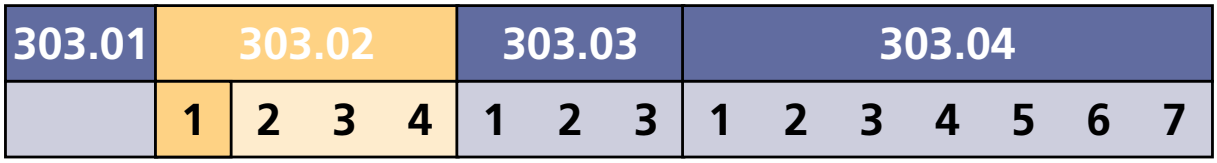

Figure 14

Again there is no difference when we add a frame. No margins are part of the graphic. This time we have no text outside the boxes, but even then we would have had proper boundingboxes.

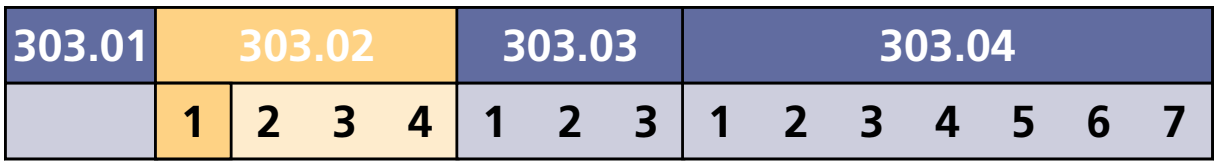

Figure 15

In this particular project, the graphics can be quite large, especially because nearly all of them are high resolution bitmaps. Each book has 150 to 250 pages and

the resulting filesize is between 500 and 1000 MBytes. The typesetting is done automatically on a dedicated ConTEXt server, is based on xml input, and uses the eXaMpLe framework to control the typesetting process over inter- and intranet.

To keep transmission of files within reasonable time, intermediate versions use low resolution graphics. These are automatically generated when new graphics are posted to the server. The annotated graphic shown previously also entered this downsampling process and the result showed up as white area.

Why was this? We use our pstopdf script (which is part of the ConTEXt distribution) to feed the graphic to GhostScript. This script filters all kind of disturbing fuzzy and application specific code from vector graphic, but only when they are in PostScript format. A pdf graphic is fed to GhostScript asis. It took a while to figure out why this graphic was so troublesome and only looking into the pdf file helped us out: negative paper is used. In such situations the pdf interpreter has to take the crop and art box into account and in this graphic these are slightly confusing.

This can best be demonstrated with a graphic, see figure 18. The mediabox (think of the paper) has negative dimensions, and this confuses GhostScript too much. I'm not sure if paper extending in quadrant 2, 3 and 4 makes sense, but it is definitely weird. In this respect there is a difference between placed pdf (like eps) and an independent page. In my opinion it's best not to gamble and make sure that the graphic ends up in the first quadrant. Whether we have to do with a bug in the graphic or in GhostScript is subjected of discussion.

After this experience, we decided to extend the XPDF companion program PDFInfo to report the boxes (a similar extension is part of future versions of this program). The problematic graphic resulted in the following report. Note that the graphic is placed on negative paper (media).

```
Creator         Adobe Illustrator 10
Producer        Adobe PDF library 5.00
CreationDate    11/23/03 14:08:49
ModDate         12/23/03 16:37:54
Tagged          no
Pages           1
Encrypted       no
Page size       327.535 x 69.0819 pts
Media box       [-94.5 -654 500.5 188]
Crop box        [-5.12834 -47.2649 322.407 21.817]
Bleed box       [-5.12834 -47.2649 322.407 21.817]
Trim box        [-5.12834 -47.2649 322.407 21.817]
Art box         [87.4258 607.25 399.235 188]
File size       59115 bytes
Optimized       no
PDF version     1.5
```

In this case the graphic is made in Adobe Illustrator, and since this is a complex and highly configurable program, the box info shown here demonstrates that you'd better know what you're doing when you export a graphics as placeable pdf. Since nowadays pdf is also Illustrators file format, it means that you cannot simply consider the stored graphic to be placeable. The data file contains editor specific information, so, before you use the graphic in other applications (especially in applications of other vendors), you may need to prepare the graphic for placement.

Another aspect to keep in mind is that when a graphic is not accurate, it may have consequences for the boundingbox as well. In the dtp variant, the small square is slightly too large.

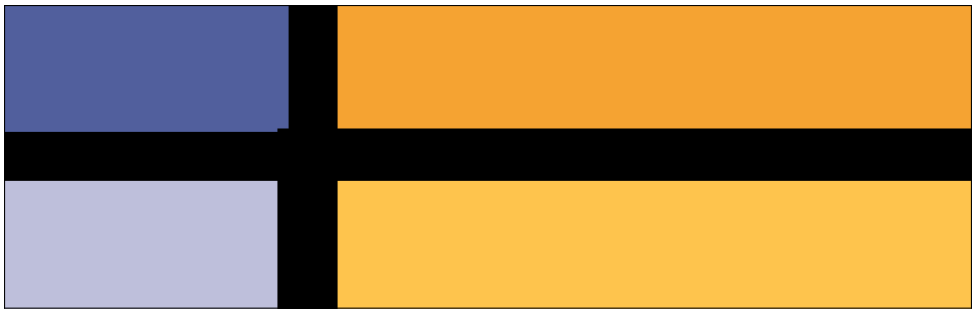

Figure 16 Not all vector graphics are accurate (dtp variant, 1).

If this happens at one of the edges, it does influence the boundingbox. If you are a regular TEX user, you probably are aware of the fact that TEX is pretty precise (which by the way is no guarantee for pretty documents). Even small inaccuracies in third party graphics then may give you an uneasy feeling when looking at the result.

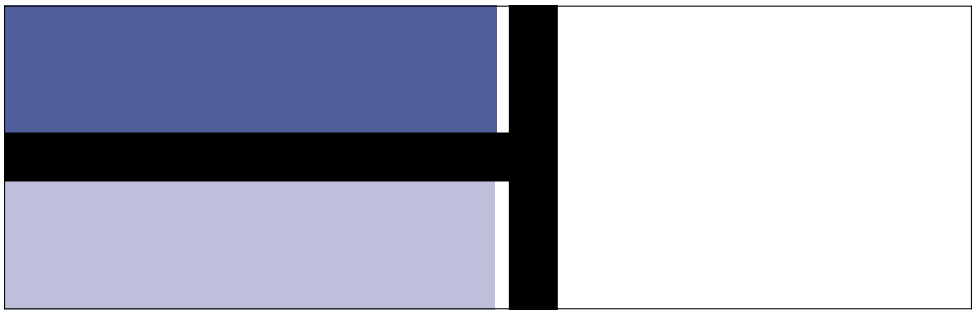

Figure 17 Not all vector graphics are accurate (dtp variant, 2).

Such inaccuracies result in unsufficient snapping in drawing programs, in no too precise coordinates in the pdf (or eps) file, in wrong rounding, or most probably, in the author of the graphic. METAPOST drawings are (if you do it right) extremely precise, but this program is suited for only a particular class of graphics. It's symbolic definition of graphics (e.g. the use of named coordinates) also kind of prevents snapping problems. Whatever program you use for drawing the graphics: you need an eye for detail. Also, you'd better make sure that you know how (and in what application) the graphic is to be (re)used.

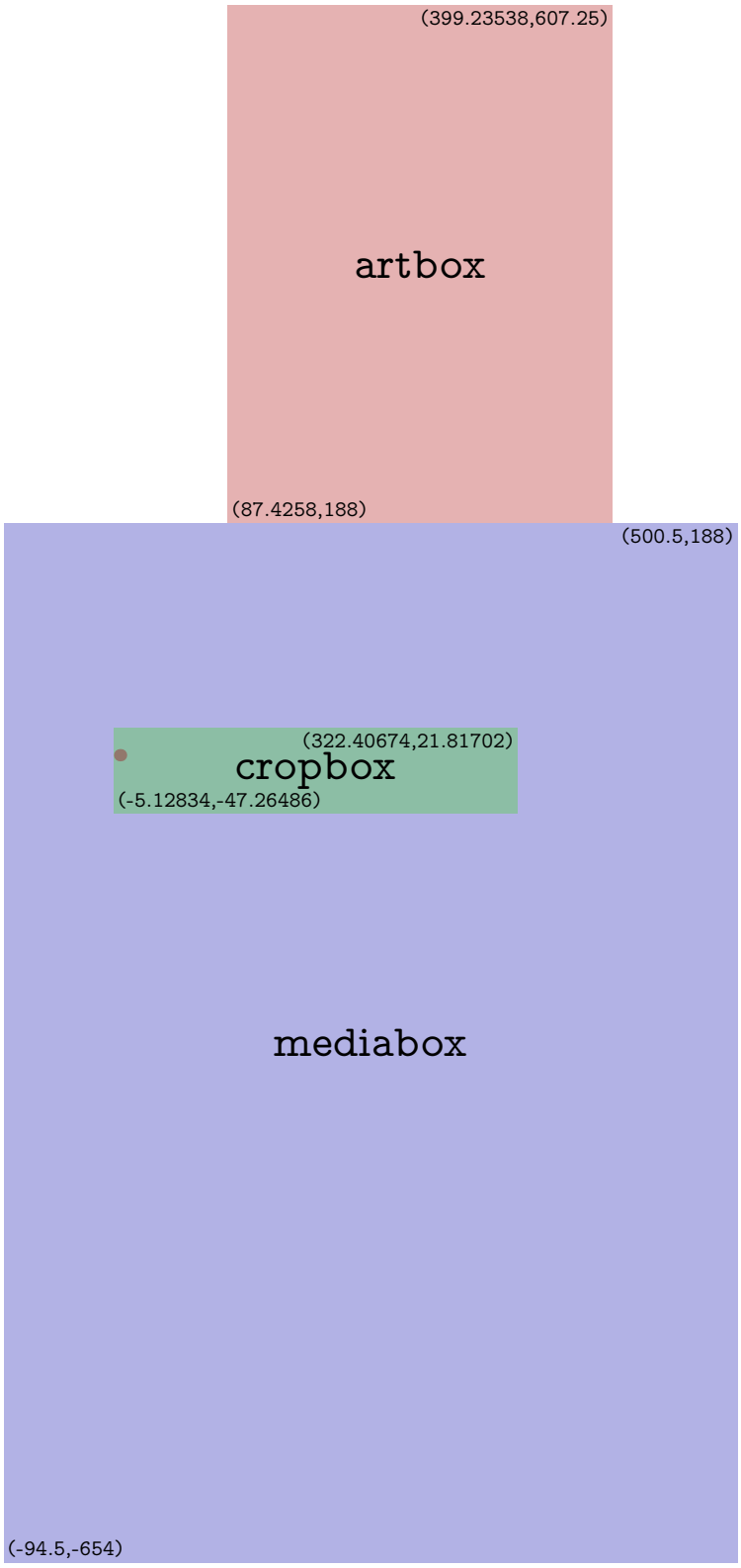

Figure 18 The boxes that define the location and size of the graphic.