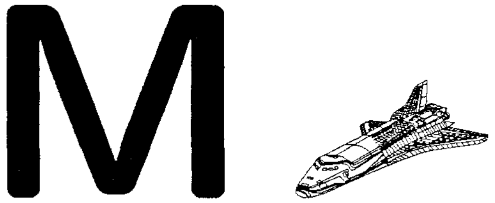
Integration of graphics into TEX

Friedhelm Sowa
Heinrich-Heine-Universität Düsseldorf
Universitätsrechenzentrum
Universitätsstraße 1
D–4000 Düsseldorf
FRG
Bitnet: `tex@dd0rud81`

Abstract

This paper is a practical approach to methods of integrating graphics into TEX. The problems that result from this integration and possible solutions are also discussed.

During the last two years we have had a lot of discussions during meetings, in publications and within electronic lists, about the question of how to integrate pictures into TEX. It is a very interesting question, indeed! The different answers to this question are also interesting because they lead to the interdependencies between TEX and driver programs. We can say that it is no problem in general to typeset pictures with TEX anywhere in a document by using different methods like the LATEX picture environment or similar macro packages, the `\special` primitive within a box, or something else. But when trying to typeset complex pictures, there is a great probability of failure because of TEX's limited capability in this area. When avoiding that effect by using the `\special` primitive, we surely need a driver that knows how to handle the special information. Nevertheless, it sounds very queer to talk about programs with special features that have to interpret a DeVice Independent `dvi` file. In one of the TEXhax issues of 1989 Stephan v. Bechtolsheim wrote "TEX was made for typesetting text...." Maybe he remembered the operation codes of the `dvi` file format when writing that. He is right, anyway. There is no correct driver in use that is not able to print:



Though it is true that not all printers can print this.

What is the difference between that character on the right and the characters on the left in general, except the dimensions, from a drivers point of view?

The difference in usage between the inch high "M" and the shuttle is illustrated below, where the "M" code is on the left and the shuttle code is on the right:

```
\font\origin=cminch   \font\atesta=atesta
{\origin M}           {\atesta !(}
```

A lot of people have considered converting graphic output from different systems to fonts. This way of integrating graphics into TEX seems to minimize the existing problems.

The Graphic Sources

As this paper should not be a theoretical but a practical approach to the problem, it is necessary to have a look at the typical situation for the need of graphics integration.

Eleven years after the birth of TEX, there is someone, somewhere in the academic world, writing a paper. The visualization of the scientific results is done by a dedicated system. For a lot of different disciplines there are a lot of different programs available on the market which convert data into a picture. Often those systems are embedded in a single purpose system. In any case, it is just that system which is the best for,

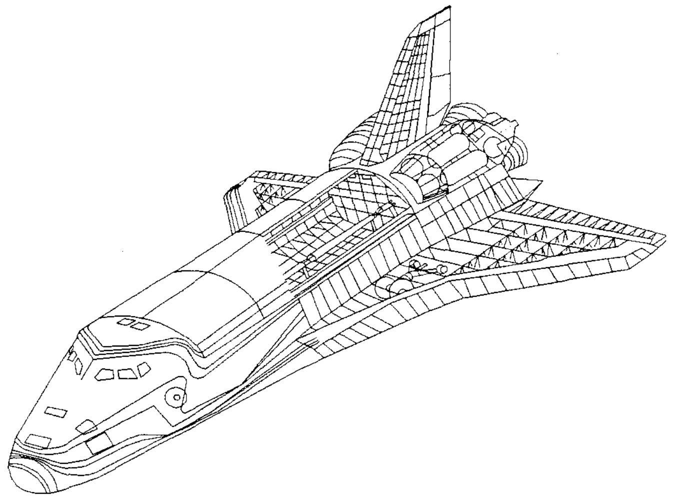
1. chemistry
2. physics
3. mathematics
4. statistics
5. engineering
6. economics
7. geography
8. etc.,

otherwise it would not be used. As long as an institute has computer freaks to do the programming, there will always be a "best" program for any particular area of science. In any case, the system produces pictures, that can be printed on a plotter as well as on a laser printer or a matrix printer. Some systems use a device independent metafile, others produce vector graphics, or even raster graphics immediately.

In the decade of micros, the importance of raster graphics (compressed or uncompressed) grew by the use of scanners, cameras, screen dumps and paint programs. The difficulties of making pictures was decrease at the same time. Even the digitization of "real pictures," like photographies, became possible.

Now back to the lonesome scientist who is using TEX to write his paper and trying to describe what is to be seen on the output of a graphic system. Usually there is someone who asks the question, "Hi, I'm using TEX for my paper and I want to include some pictures. How does it work?"

The individual situation It works, but how is not important here. Only the dimensions of the picture in the final print are important. Either the user wants to print it on his own 300 dots per inch (dpi) laser printer or on the publisher's 1200 dpi typesetter. That is the most important item for the layout of the document. The picture will be included into the document for a certain print device by telling TEX the picture's `tfm` information. Although the files around TEX are device independent the picture in it's original form is not device independent with respect to the document's purpose. Of course, there is a formal device independency concerning the `tfm` files and the `pk` files or `pixel` files, but depending on the resolution, the picture should be printed in different sizes. Using a high resolution device the picture can be generated larger than for a low resolution device because there are more dots available for the same amount of space.



This picture was generated for a 300 dpi device. The same picture on the previous page was generated for a Tektronix display and included for purposes of demonstration. The higher resolution allows for more detail. Expressing this thought in one sentence,

> A complex graphic on a high resolution device requires less space than the same graphic on a low resolution device to see the details.

The user has to decide, depending on the resolution of the output device, what size the picture should be on the page. This is only true if the source of the picture is a vector oriented graphic. Bitmap graphics are fixed in size. Changes to an existing black/white bitmap by scaling will change the quality of the picture too, unless the run of the black bits are converted to vectors. In this case the circle is closed again.

The situation in general. The problem of integrating pictures into a document is heightened by facts, which are basically the same for every author. Those facts are:

1. the kind of the graphic
   (a) vector graphics
   (b) black/white bitmaps
   (c) bitmaps with grey pixels
2. the resolution of the output device
3. the available driver program
4. the ability of TEX to typeset text
5. the motivation of the author

What the author needs is a link between his graphics and his text. There is no question that he himself has to do something to make the link. The simplest way is to start a program that converts the graphic file into a font for the author's output device by generating the `tfm`'s and `pk` or `pixel` files and writes some TEX instructions to typeset the picture. So he only has to say something like `\setpicxy` somewhere in his text. That should be easy enough for everyone.

It was already mentioned that some vector oriented graphic systems generate metafiles like GKS, Phigs etc. Is it wrong to say that every system of that kind generates metafiles in any way when processing a picture? Certainly not. The aim of those systems is to draw some lines or curves on a sheet of paper. This is done by either external driver programs or special subroutines for specified devices. At this point, we find common instructions for printers and plotters. For those devices, a fixed instruction set is defined, exists for a long period of time and does not need any standardization committee.

It is a standard in an isolated area, acknowledged by a lot of driver programs or subroutines.

This is the relevant interface for a font generating program. The program should know the set of instructions of two or three manufacturers to solve the integration problems of about 80% or even 90% of the TEX users who make their pictures with such a graphics system.

There is a similar situation in the world of bitmaps. Many abbreviations like TIFF, PCX, IMG or CUT represent the package of more or less compressed bitmaps. A lot of conversion programs allow one to use the appropriate package for an interesting picture. Here it is probable the problem will be solved up to 99.99% of the time.

The Graphic Fonts

When talking about fonts within the TEX community, everybody remembers METAFONT. The readers of *TUGboat* or the Exeter proceedings know, that there are some programs that use the sister of TEX to make fonts from graphic files. He should also know, that some approaches were made without METAFONT to generate graphic fonts. The main reason to avoid the usage of METAFONT is the ability of TEX to run on any machine without METAFONT. The user knows it, and he typically utilizes TEX, not METAFONT. He only wants one program to include the picture, not a complex system.

Driver compatibility. It was already mentioned that the typical user does not need device independence. He only needs the driver selected to print the final document. This sounds a little bit like a driver restriction. And indeed it is. A correct driver should follow the recommendations of the `dvi` driver standards committee. This is not a wish but a law. Especially on small systems there is a restriction concerning the size of font files. A limit of 64KB is to be found in a lot of programs. This is the smallest common denominator when generating graphic fonts.

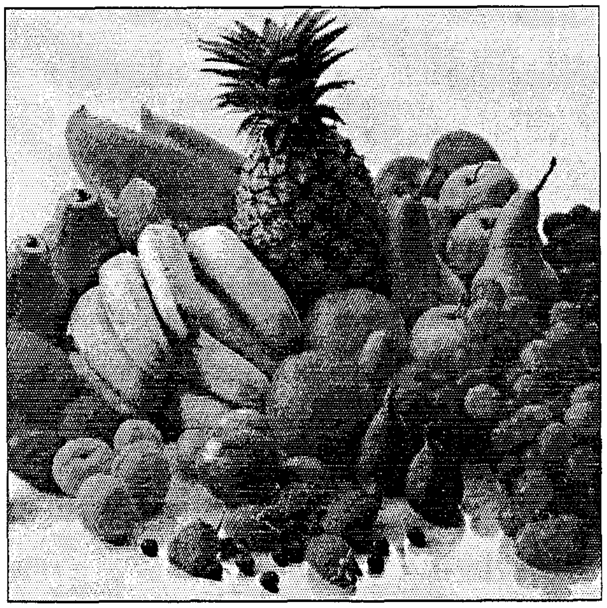
It may be that there is a difference among drivers concerning the ability to print large characters. Here is the next guideline for generating fonts.

As a conclusion of those facts we can say: If a driver program is able to handle

1. the number of fonts TEX allows
2. fonts with characters not larger than 1 inch
3. fonts not exceeding the 64KB limit

then it is able to print documents that include graphics.

The suitable program. A practical approach to the problem of including graphics includes the development of a program too, especially for an employee of a computer center in a university. After gathering information about the structure of `tfm` files, `pixel` files and `pk` files on the one hand and about TIFF and CUT format on the other hand, the foundation stone for BM2FONT was laid. It should be able to handle bitmap formats and simple bitmaps. The latter ones are generated by a special vector conversion program on a central computer system, where the vector approach was done for historical reasons.



Since a lot of students and scientists are working on personal computers BM2FONT was written for DOS. Here we find the origin of the other kinds of bitmaps. After long discussions with the WEB system and the Pascal compiler the program got a consolidated status. It produced a lot of `tfm` files and `pk` files and files like:

```
\newbox\obstbox
\newdimen\obstw
\font\aobst=aobst
\font\bobst=bobst
\font\cobst=cobst
\font\dobst=dobst
\setbox\obstbox=\vbox{\hbox{%
\aobst !()+}}
\obstw=\wd\obstbox
\setbox\obstbox=\hbox{\vbox{\hsize=\obstw%
\parskip=0pt\offinterlineskip\parindent0pt%
\aobst !()+\vskip0pt%
,\bobst !()\vskip0pt%
+,\cobst !(\vskip0pt%
)+,\dobst !}}
```

```
\def\setobst{\box\obstbox}
```

Using `\input obst` to include the TEXnical description of the graphic into the TEX source file and `\setobst` to typeset the picture the job of integration is done.

Everything worked fine and there were no new problems concerning the available driver programs. Only the known errors appeared.

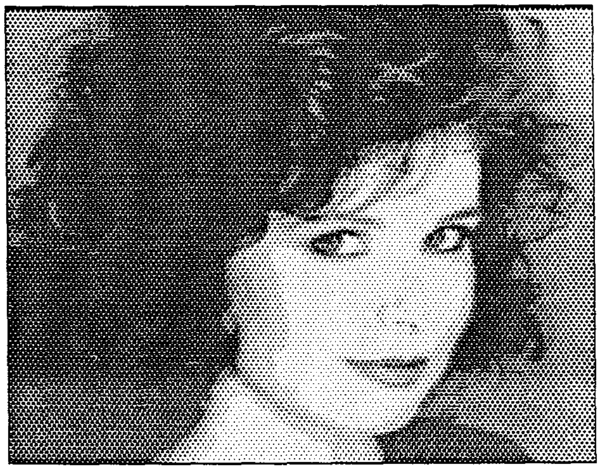
Halftone Graphics

The conversion of black/white bitmaps to fonts is a rather boring job. Pixels p with values in the range $0 \leq p \leq 255$ present additional fun before that conversion. In this case, a temporary bitmap must be generated, where the original pixels are represented by areas partially filled with black or white pixels, which are usually the only types of pixels available on common devices. In ***TUGboat***, there have been interesting articles on this (Volume 8, number 2 and 3 by Don Knuth and Adrian F. Clark.) They used fixed patterns for defined devices within very special fonts. The conclusion of those articles was the need of a big TEX for small patterns. With that solution, the still life on the previous page would require 79,341 characters for the same size picture.

The other suggestion of Knuth was a better one. He proposed a picture specific font that includes the whole picture. By assigning larger areas to the characters of the font, he reduced the amount of characters in the input file. But there was no consideration of driver limitations and, as the father of METAFONT, he was not willing to reconsider this, as it was not a solution for any acceptable output device. Nevertheless it was the right idea to solve the problem.

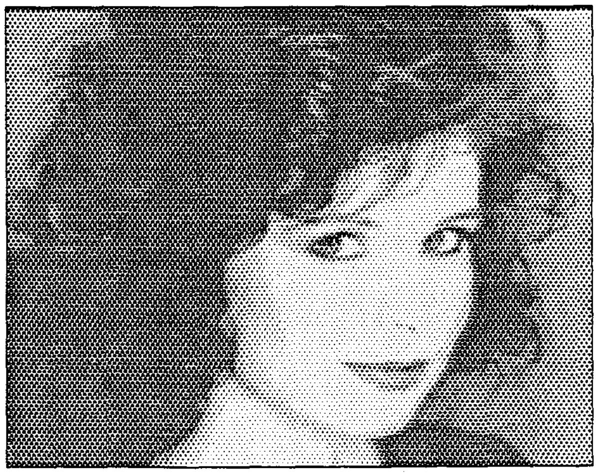
Another aspect of the problem is the subjective behavior of the human eye, which tries to see a homogeneous picture, and not the single pixels. This can be achieved by representing different grey shades by black dots, which differ in size. This is not a new idea, but it is difficult to achieve good results in a general way for different devices with resolutions between 300 and 1200 dpi.

Generating grey patterns. To generate black dots, we take a n x m square of pixels. For every possible grey shade, a new pixel in this square, close to the last blackened pixel, is turned from white to black. This is the principal way to generate black dots with different sizes. The grey values of the source file are to be scaled up or down to the available grey shades, and then the picture is composed with a well structured number of more or less black squares.



Usually the original file comes from a system that uses a scanner or a video camera. To get the details off a picture, it should be digitized with a high contrast. This means in practice that 256 grey values give better results than 16 grey values. Most systems follow this rule, but when scaling the usable grey shades down, depending on the square's size, it is necessary to distribute the rounding errors of the pixel to the neighboring pixels. Experience has shown that error distribution gives a smooth transition between areas of nearly same grey values.

The picture above was taken by a video camera system that uses 256 colors. It was converted by using a 4 x 4 square. The available 16 grey shades are reduced to 12 because the printer seems to overlap the pixels and produces solid black dots quite early. Beside that manipulation, a correction of the lines of patterns was necessary. As the pixels of the camera system are higher, rather than wider, the last line of the square was ignored. This is a good method to avoid long faces. The following example shows the result of error distribution. The changing of colors became smooth, but the contrast in the picture was reduced.



In this example, the distribution was done by the Floyd-Steinberg algorithm, that gives $\frac{3}{8}$ of the error to the side and down and $\frac{1}{4}$ diagonally down.

Simulating more patterns. The last two examples were made with this described method, but a little trick was added. Four pixels of the source file had to build one "grey dot" in the temporary bitmap. This gave the opportunity to use one dither matrix for any size of squares. We had to start in the top left corner of the matrix to find the position of the pixel in the *n* x *m* square which had to become black.

	column						
row	1	2	3	4	5	6	7
1	01	03	06	10	18	26	38
2	02	04	07	12	19	28	40
3	05	08	09	14	21	30	42
4	11	13	15	16	23	32	44
5	17	20	22	24	25	34	46
6	27	29	31	33	35	36	48
7	37	39	41	43	45	47	49

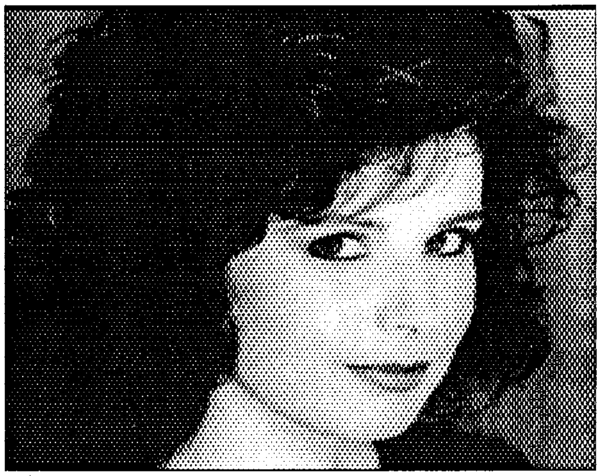
This resulted in four different patterns for each grey value, because the black pixels of one "grey dot" should be close together. The position of pixel 1 in the first pattern is the upper left corner of the square, in the second pattern the lower right corner, in the third pattern the lower left corner and in the last pattern the upper right corner. So the original grey pixels can grow together considering the values of the neighbors.

	pattern			
shade	1	2	3	4
1	●			
2	●	 ●		
3	●	 ●	 ●	
4	●	 ●	 ●	●
5	● ●	 ●	 ●	●
6	● ●	 ● ●	 ●	●

⋮

	pattern			
shade	1	2	3	4
20	●● ●● ●	● ●● ●●	● ●● ●●	●● ●● ●
21	●●● ●● ●	● ●● ●●	● ●● ●●	●● ●● ●
22	●●● ●● ●	● ●● ●●●	● ●● ●●	●● ●● ●
⋮	⋮	⋮	⋮	⋮
32	●●● ●●● ●●	●● ●●● ●●●	●● ●●● ●●●	●●● ●●● ●●
33	●●● ●●● ●●●	●● ●●● ●●●	●● ●●● ●●●	●●● ●●● ●●
34	●●● ●●● ●●●	●●● ●●● ●●●	●● ●●● ●●●	●●● ●●● ●●
35	●●● ●●● ●●●	●●● ●●● ●●●	●●● ●●● ●●●	●●● ●●● ●●
36	●●● ●●● ●●●	●●● ●●● ●●●	●●● ●●● ●●●	●●● ●●● ●●●

Still this method was not good enough. More grey shades were necessary to maintain the contrast in spite of error distribution. In fact, we did not have 16 grey shades available for one "grey dot" but four times as many. So, we multiplied the shades by 4 and changed pattern generation. Instead of making four patterns for one grey shade with the same number of black pixels, we produced four patterns for a certain grey value by decreasing the number of black pixels in every pattern by one. The result was much better than those of the older version.



This version seems to consider the neighboring pixel values best. When making the examples for the effect of error distribution, the old version was used. The new method for the example gave the same results when the error distribution was eliminated in the program.

Bibliography

Burger, Peter, and Duncan Gillies, 1989, ***Interactive Computer Graphics***, Addison Wesley Publishing Co.

Childs, Bart, Alan Stolleis and Don Berryman, 1989, "A portable graphics inclusion" in ***TUGboat*** Volume 10, Number 1, page 44 – 46

Clark, Adrian F. 1987, "Halftone output from TEX" in ***TUGboat*** Volume 8, Number 3, page 270 – 274

Heinz, Alois. 1990, "Including pictures in TEX" in Malcolm Clark, ed., ***TEX Applications, Uses, Methods: TEX88 Proceedings***, page 141 – 151, Ellis Horwood Series in Computers and their Applications

Knuth, Donald E. 1986, ***The TEXbook, Computers & Typesetting***, Volume A. Addison Wesley Publishing Co.

Knuth, Donald E. 1986, ***The* METAFONT*book, Computers & Typesetting***, , Volume C. Addison Wesley Publishing Co.

Knuth, Donald E. 1987, "Fonts for digital halftones" in ***TUGboat*** Volume 8, Number 2, page 135 – 160

Messinger, Heinz. 1982, ***Langenscheidts Großwörterbuch, "Der kleine Muret-Sanders", Deutsch-Englisch***, Langenscheidt

Pickrell, Lee S. 1990, "Combining graphics with TEXon IBM PC-compatible systems and LaserJet printers" in ***TUGboat*** Volume 11, Number 1, page 26 – 31

Pickrell Lee S. 1990, "Combining graphics with TEXon IBM PC-compatible systems and LaserJet printers, Part II" in ***TUGboat*** Volume 11, Number 2, page 200 – 206

Rogers, David F. 1989, "Computer graphics and TEX, a Challenge" in ***TUGboat*** Volume 10, Number 1, page 39 – 44

Simpson, Richard O. 1990, "Nontraditional Uses of METAFONT" in Malcolm Clark, ed., *TEX Applications, Uses, Methods: TEX88 proceedings*, page 259 – 271, Ellis Horwood Series in Computers and their Applications

Wilcox, Patricia. 1989, "METAPLOT: Machine-independent line graphics for TEX" in ***TUGboat*** Volume 10, Number 2, page 179 – 187