Stereographic Pictures Using TEX

Reinhard Fößmeier

Abstract

This article shows how to produce stereographic pictures ("magic eye" pictures) using TEX. While it is possible (though slow) to produce such pictures from black and white dots, it is easier to use ordinary glyphs as picture elements.

Resumo

Tiu ĉi artikolo montras kiel produkti kvazaŭ-tridimensiajn bildojn (konatajn sub la nomo "magia okulo") per TEX. Estas eble (kvankam malrapide) konstrui tiajn bildojn el nigraj kaj blankaj punktoj, sed estas pli facile uzi ordinarajn pres-signojn kiel bilderojn.

Introduction

Recently, a new kind of stereographic picture has become rather well known, by names such as "The Magic Eye" or "Stare-E-O". Books containing such pictures (e.g., [1], and other volumes from the same series) for some time were the best-selling non-fiction books in Germany, and no doubt things were similar in a number of other countries.

The new technique differs from the conventional way to present stereographic views: Instead of having two independent views of the same scene, taken from slightly different angles, all the data here are contained in one picture, which must be looked at with a squint, so the visual axes cross behind (or in front of) the picture, and intersect the picture plane at a certain distance.

Since much has been written about it, I won't give here a detailed description of the phenomenon. Suffice it to say that the secret of the 3D effect is that the graphical information in the picture is roughly periodic, the period being the distance between the section points of the visual axes and the picture plane. If the periodicity is perfect, the whole picture is seen as flat. Everywhere the periodicity is disturbed, details stand out in relief, seeming closer

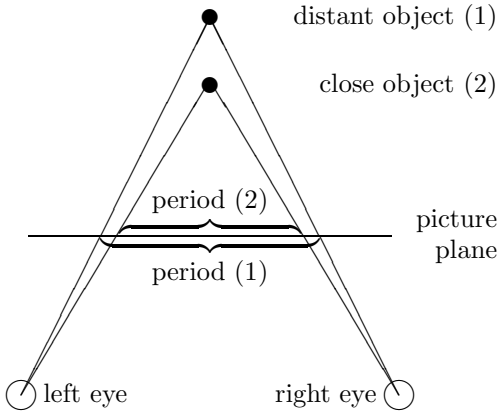

Figure 1: How to look at a magic picture

if the periodicity is less and more distant if it is greater (cf. figure 1).

The `stereo.sty` style

The trick in producing a stereo picture is to start (e.g., at the left) with a certain pattern and to continue it periodically to the right. The period varies according to the distance at which a certain point is to appear: The closer to the eye, the shorter the period. (We assume here that the visual axes cross behind the picture.)

The input form of a picture is a two-dimensional grid of numbers, each number indicating the height of the point it denotes. The number 0 stands for the lowest (most distant) plane. Only numbers between 0 and 9 are allowed. A semicolon delimits a line. The data shown in figure 2 describe a small rectangle that "floats" one unit above the ground plane.

```
000000000000000000000000000000;
000000000000000000000000000000;
000000000000000000000000000000;
000000000000111111000000000000;
000000000000111111000000000000;
000000000000111111000000000000;
000000000000111111000000000000;
000000000000111111000000000000;
000000000000111111000000000000;
000000000000000000000000000000;
000000000000000000000000000000;
000000000000000000000000000000;
```

Figure 2: Input data for a very simple picture

While it would be possible to construct patterns from black and white dots and to draw them, e.g., as `\vrule`s, this process is very slow and consumes a lot of TEX's resources. Unless the pictures are to show very fine details, it is easier to use patterns built from normal TEX glyphs, such as letters and figures. To facilitate the calculation of the period length, we use a mono-spacing font, `\tt`.

Each picture element is produced by a call of the `\Pixel` macro. Its only argument denotes the "altitude" of the point. According to this argument, the pattern (contained in `\Pat`) is shifted around and the correct glyph `\A` for the new element is printed:[1]

```
 1 \newcount\alt
 2 \def\Pixel#1{%
 3    \def\Head##1##2!{##1}%
 4    \def\Tail##1##2!{##2}%
 5    \edef\A{\expandafter\Head\Pat!}%
 6    \edef\Rest{\expandafter\Tail\Pat!}%
 7    \edef\T{\Rest}%
 8    \alt=#1
 9    \loop\ifnum\alt>0
10        \edef\A{\expandafter\Head\T!}%
11        \edef\T{\expandafter\Tail\T!}%
12        \advance\alt -1
13    \repeat
14    \edef\Pat{\Rest\A}%
15    \A%
16 }
```

To produce a line of pixels, this macro must be iterated until a semicolon is reached. This is done by the following macro, `\Line`, which calls itself recursively until it sees a semicolon:

```
17 \def\Line#1{%
18     \if #1;\vskip 0pt \else
19         \Pixel#1%
20         \expandafter\Line
21     \fi
22 }
```

We now could simply give an initial value for `\Pat` and bracket each line of the picture data by `\Line`$\cdots$`;`. The pattern, however, loses details each time it is shifted to a shorter period, and cannot recover the lost information when periods become longer again. We could end up with a pattern consisting of only one sort of glyph. So it is better to restore a reasonable value at the beginning of each line. This is

[1] The linenumbers in the code segments are added for clarity and are *not* part of the macro text.

done in the following macro, `\DLine`; we simply use the same start pattern for all lines.

```
23 \def\DLine#1;{%
24     \edef\Pat{\StartPattern}%
25     \Line#1;%
26 }
```

Now let's have a look at the result of the following calls, shown in figure 3:

```
27 \edef\StartPattern{A-CeL'+MX-/()pd=}
28 \DLine 00000000000000000000000000000000;
```

... (as in figure 3)

```
A-CeL'+MX-/()pd=A-CeL'+MX-/()p
A-CeL'+MX-/()pd=A-CeL'+MX-/()p
A-CeL'+MX-/()pd=A-CeL'+MX-/()p
A-CeL'+MX-/(pd=A-CCeL'+MX-/(pd
A-CeL'+MX-/(pd=A-CCeL'+MX-/(pd
A-CeL'+MX-/(pd=A-CCeL'+MX-/(pd
A-CeL'+MX-/(pd=A-CCeL'+MX-/(pd
A-CeL'+MX-/(pd=A-CCeL'+MX-/(pd
A-CeL'+MX-/(pd=A-CCeL'+MX-/(pd
A-CeL'+MX-/()pd=A-CeL'+MX-/()p
A-CeL'+MX-/()pd=A-CeL'+MX-/()p
A-CeL'+MX-/()pd=A-CeL'+MX-/()p
```

Figure 3: Picture from the data in figure 2

The effect of figure 3 is not very convincing as the picture is simply too small: it does not even show two full periods of the pattern. Now we could inflate the data to make the picture larger; on the other hand, TEX can do this for us. We just have to change the following lines in our macro definitions:

```
19          \Pixel#1\Pixel#1\Pixel#1%
25      \Line#1;\Line#1;%
```

to blow up each pixel to a 3×2 matrix. (To do this with loops is left as an exercise to the reader.) The above data (trimmed somewhat to fit into the column) then yield the picture in figure 4.

Now, if you have a trained magic eye, you should be able to see the floating rectangle in the middle of figure 4, and a somewhat more informative picture in figure 5. (If you have no experience with such pictures, ask somebody who has, or simply try staring at it, or forget about the whole thing—it's not really vital after all!) Figure 5 shows that it can be useful to compress the picture vertically, by specifying a negative `\vskip` in line 18.

Why No Dot Patterns?

To be really effective and bring out finer details, magic pictures have to use patterns with a finer resolution than that of glyphs. Basically, this can be done with TEX, e.g. by `\vrule`s and `\kern`s for black and white dots[2]. Details on how to do this can be found in [2]. This process, however, not only is very slow but also consumes a lot of TEX's internal memory. Unless Big TEX is used, pictures are restricted to a rather small size.

Conclusion

Given that TEX's primary domain is typesetting texts, it is no surprise that the easiest way to produce magic pictures with TEX is through the use of character symbols. The pictures produced this way do not come up to the quality of dot graphics but certainly do have a charm of their own.

Possible extensions of the macros presented are:

- use of loops to inflate pixels by variable factors;
- reading picture data from an external file, possibly without the need to insert macro calls into the data;
- starting each line with a copy of the pattern, so details at the left edge of the picture can be seen;
- use of different patterns for each line;
- automatic construction of suitable random patterns.

References

[1] Das magische Auge. Dreidimensionale Illusionsbilder von N. E. Thing Enterprises. München: arsEdition, 1994. Original: Magic Eye, Kansas City: Andrews and McMeel, 1993.

[2] R. Fößmeier: X Bitmaps in TEX. TUGboat 12 (1991), 2, 229–232.

◇ Reinhard Fößmeier
iXOS Software GmbH
Bretonischer Ring 12
DE-85630 Grasbrunn
Germany
Reinhard.Foessmeier@ixos.de

[2] I am indebted to Bernd Raichle, raichle@azu.informatik.uni-stuttgart.de, for this hint.

Figure 4: Picture from the data in figure 2, with inflated pixels

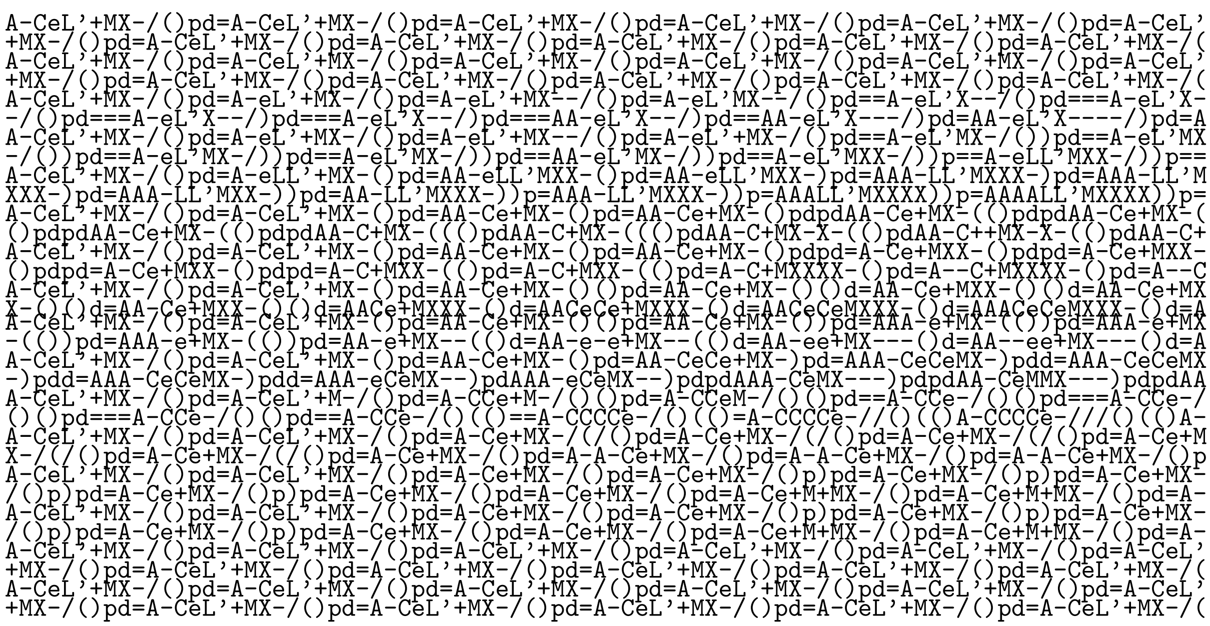

Figure 5: A more technical picture