A TEX Autostereogram Generator

Jacques Richer

Introduction

The Plain TEX code `autostereogram.tex` generates autostereograms. An autostereogram (often called simply stereogram) is a *single* picture that shows objects in 3D but does not require any special device for viewing (other than a normal pair of eyes, of course). Autostereograms have been the subject of a recent craze throughout the world. They now fill entire color albums, such as the following beautiful books:

- *Stereograms*, by Cadence Books, 1994
- *Ultra-3D*, by Montage Publications, 1994
- *Interactive Pictures*, by Benedikt Tashen Verlag, 1994
- *Hidden Dimensions*, by Dan Dyckman, Harmony Books, New York, 1994 (*this is a funny 3D puzzle book*)

They have also been made into postcards and posters. In this article, I show how the basic technique for generating such pictures can be implemented in TEX. I will first explain how `autostereogram.tex` is used and what it does. The TEX coding itself will be described last.

Generalities

The image contents (depth information) is specified to the generator in the form of an ordinary text file, hereafter called the *relief data file*, containing several rows of single digit numbers. These numbers indicate at which height or depth each pixel should be perceived by the viewer. In the original version of this code, no extra file was needed as the image data was generated from within the TEX code itself (by the `\relief` macro), through nested `\if\else\fi` constructs; the external file approach is much more flexible, and easier to use.

The output pixels are identical size `\hbox`es that may contain anything TEX can fit in them: a rule box, a character, or even whole text paragraphs! The whole picture is obtained by tiling the surface with copies of a small number of such `\hbox`es. These basic tiles determine the *texture* of the image, which is totally independent from its 3D contents. Here, I use four tiles consisting in ♡s, in two sizes, the TEX logo, and an empty box, to help produce a lighter texture. Color can be used (here ♡s will be red, if `PSTricks` is available), and is highly recommended.

The image is generated line by line, lines being totally independent from each other. The algorithm consists in repeating horizontally an arbitrary initial pattern of basic tiles, at a repetition rate that is *modulated* by depth information. More precisely, from an initial set of m pixels p_0, $p_{-1}, \ldots, p_{-(m-1)}$ on a line, randomly selected from our basic tile set, one generates n new pixels, where $n \gg m$, using the recurrence relation $p_i = p_{i-\sigma_i}$; σ_i is an integer valued monotonic function of depth d_i at point i. Here, I take that function σ_i to be simply equal to $m - d_i$; clearly, if all depths are equal to 0, for all lines, the final output will consist in identical copies of the first m-pixel wide vertical stripe; hence m is called the *period* of the image. When depths vary, one gets a horizontally distorted version of the zero-depth pattern.

The m starting pixels are not under user control, as far as their depth is concerned, and printing them will lead to more pixels appearing in the picture than the user provided data for; so it may be preferable not to print them. Here they are not printed.

Depth is perceived when the eyes lock their relative orientation so that their aiming points are always separated horizontally by m pixels, or an integer multiple k (k may be negative!) of that distance. Apart from this constraint, *the eyes can move freely and explore the whole picture.* A sort of recursive effect is seen if $|k| > 1$; this is discussed in the paper cited below. For the purpose of this presentation, it is simplest to assume that we are viewing adjacent m-pixel wide stripes. If the two stripes are identical, the 3D impression is that of a flat area floating at some distance away from us; this distance depends on the so-called *vergence angle* between the two lines of sight, that angle depending in turn on the physical width of a *period.*

If the left eye locks onto any stripe — which column it starts in does not matter, as long as it is not too close to the vertical sides — and the right eye locks onto the one immediately to its right, pixels with larger values of d_i are perceived as being closer to the observer than pixels with smaller d_i; if the eyes are crossed ($k < 0$ — left eye locks onto right stripe, and *vice-versa*), larger d_i pixels appear further away. Personally, I find it more difficult to hold the lock with eyes crossed, but that difficulty depends very much on the width of a period and the viewing distance. For the sake of simplicity, I decided to call d_i values depths, even if the word heights would often be more appropriate.

The translation invariance of the algorithm makes it possible to build arbitrarily large pictures.

With autostereograms, 3D perception always occurs as a result of *local distortions* in the repetition cycle; thus an autostereogram covering an entire wall could show 3D in any of its parts. It is easy to see the relationship between autostereograms and stereographic *image pairs* if one thinks in terms of such local disparities between two consecutive stripes; one important difference is that the autostereogram presents us with a *near continuum of stereo pairs* (unfortunately of much lower information content).

For an excellent introduction to the subject of autostereogram generation, by the inventor of the technique, see: Christopher W. Tyler & Maureen B. Clarke, "The Autostereogram", in the Proceedings of the SPIE–The International Society of Optical Engineering, SPIE vol. 1256, "Stereoscopic Displays and Applications", J. O. Merritt & S. S. Fischer, editors, 1990, pages 182–197.

The relief data

To use `autostereogram.tex`, one must supply a relief data file. The first line of that text file should contain the horizontal size of the picture n_{columns}, in pixels, and nothing else; the second line should contain the vertical size n_{lines}, in pixels. The third line must supply the period m. It should be as large as possible, subject to constraints to be discussed below.

The fourth line should contain a small integer g that will tell the generator what depth level is to be interpreted as the *ground plane*. Then should follow n_{lines} lines each containing exactly n_{columns} decimal digits, the d_is (no spaces). The code could be easily generalized to allow an arbitrary number of depth levels. For the following discussion, let's assume that the d_is are in the range $d_{\min}..d_{\max}$.

If $g \le d_{\min}$, relief will be towards the parallel eye viewer, and if $g \ge d_{\max}$, relief will be away from him/her. Values from $d_{\min}$ to $d_{\max}$ will make it possible to display both receding and advancing objects. As the foregoing sentences imply, g is not limited to the $d_{\min}..d_{\max}$ range, but it is best not to deviate too much from that range. With $g \ne 0$, the algorithm becomes

$$p_i = p_{i-(m-[d_i-g])}$$

and it is necessary to ensure that $m + g + 1 \le i + d_i$ and $d_i \ll m + g$, for any $i > m$, and $d_{\min} \le d_i \le d_{\max}$. If $i' = i - (m+1)$ (i' measures the horizontal distance from the right edge of the starting stripe), we must then have $i' \ge g$ for nontrivial pixels: problems can arise if $g > 0$. To prevent such problems the code will extend the initial stripe by exactly g columns, using the ground plane

```
60
17
10
0
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000002222000000000000000000
000000000000000000000000000000000000002222000000000000000000
000000000000000000000000000000000000002222000000000000000000
000000000000000000000000000000000022222222222220000000000000
000000000000000000000000000000000022222000222220000000000000
000000000000000000022222222000000022222000222220000000000000
000000000000000000022222220000000022222222222220000000000000
000000000000000000022222000000000000002222000000000000000000
000000000000000111222221110000000000002222000000000000000000
000000000000000111222211110000000000002222000000000000000000
000000000000000111222111110000000000000000000000000000000000
000000000000000111221111110000000000000000000000000000000000
000000000000000111211111110000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
```

Figure 1: Input data for a 3-level picture.

recurrence formula $p_i = p_{i-m}$. The initial stripe (more precisely, one line of it) will then consist (conceptually) of pixels $p_0, p_{-1}, p_{-2} \ldots, p_{m+g-1}$.

The other constraint, $d_i \ll m + g$, is more subtle: $d_i < m + g$ is necessary, but not sufficient, because if the shift $\sigma_i = m - [d_i - g]$ is too small, many of the initial pixels on the current line will never be used, and the output line could easily collapse to a series of identical pixels. So the the input deviations $d_i - g$ should be kept small. There is an inevitable loss of information in the recurrence relation, and it is necessary to have as much information as possible to begin with. This is one reason why a large m is preferable; another reason is that the perceived angular resolution is roughly inversely proportional to m since it is equal to the width of a pixel (which is normally made smaller when m is larger) divided by the viewing distance.

Figure 1 shows a small multilevel example (too small to yield good results). To generate the real example shown in Figure 3, I applied METAFONT to the very nice Irish font `eiad10` to generate a simple 2-level input file. The (UNIX style) commands

```
mf "\mode:=aps; " input eiad10
gftype -i ./eiad10.723gf > eiad.txt
```

produced a text file `eiad.txt` containing drawings of all the characters in the font, where each black pixel is represented by a `*` and each white pixel by a space. From this file I extracted the letter '`b`', which I then edited to change spaces into `0`s and `*` into `1`s, and padded with extra `0`s on all four sides to make it into a rectangular array including a small margin all around. I finally added the four parameter lines, to get the file `eiadb.dat` shown in Figure 2, which `autostereogram.tex` converted to Figure 3.

Note that if the characters are not fat enough, their thinnest parts will not be visible in 3D because of the coarse resolution imposed by the use of character boxes and integer σ_is. It is necessary to call METAFONT with a high resolution `mode` or to require

```
72
68
15
0
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000111000000000000000000000000000000000000000000000
000000000000000000000000111110000000000000000000000000000000000000000000
000000000000000000000000111111000000000000000000000000000000000000000000
000000000000000000000001111111000000000000000000000000000000000000000000
000000000000000000000011111111000000000000000000000000000000000000000000
000000000000000000000111111111100000000000000000000000000000000000000000
000000000000000000001111111111100000000000000000000000000000000000000000
000000000000000000011110111111100000000000000000000000000000000000000000
000000000000000011111100111111100000000000000000000000000000000000000000
000000000000001111111000111111100000000000000000000000000000000000000000
000000000000001111100000111111100000000000000000000000000000000000000000
000000000000001100000000111111100000000000000000000000000000000000000000
000000000000000000000000111111100000000000000000000000000000000000000000
000000000000000000000000111111100000000000000000000000000000000000000000
000000000000000000000000111111100000000000000000000000000000000000000000
000000000000000000000000111111100000000001111111100000000000000000000000
000000000000000000000000111111100000001111111111111100000000000000000000
000000000000000000000000111111100000111110000000111111000000000000000000
000000000000000000000000111111100001110000000000001111110000000000000000
000000000000000000000000111111100011100000000000000111111000000000000000
000000000000000000000000111111100110000000000000000001111100000000000000
000000000000000000000000111111101100000000000000000001111110000000000000
000000000000000000000000111111111100000000000000000000111111000000000000
000000000000000000000000111111111000000000000000000000011111100000000000
000000000000000000000000111111110000000000000000000000011111110000000000
000000000000000000000000111111100000000000000000000000011111111000000000
000000000000000000000000111111100000000000000000000000001111111000000000
000000000000000000000000111111100000000000000000000000001111111100000000
000000000000000000000000111111100000000000000000000000001111111100000000
000000000000000000000000111111100000000000000000000000001111111110000000
000000000000000000000000111111100000000000000000000000000111111110000000
000000000000000000000000111111100000000000000000000000000111111110000000
000000000000000000000000111111100000000000000000000000000111111110000000
000000000000000000000000111111100000000000000000000000000111111111000000
000000000000000000000000111111100000000000000000000000000111111111000000
000000000000000000000000111111100000000000000000000000000111111111000000
000000000000000000000000111111100000000000000000000000000111111111000000
000000000000000000000000111111100000000000000000000000000111111111000000
000000000000000000000000111111100000000000000000000000000111111111000000
000000000000000000000000111111100000000000000000000000000111111111000000
000000000000000000000000111111100000000000000000000000000111111111000000
000000000000000000000000111111100000000000000000000000000111111111000000
000000000000000000000000011111100000000000000000000000000111111110000000
000000000000000000000000011111100000000000000000000000000111111110000000
000000000000000000000000011111110000000000000000000000000111111110000000
000000000000000000000000011111110000000000000000000000001111111110000000
000000000000000000000000001111110000000000000000000000001111111100000000
000000000000000000000000001111110000000000000000000000001111111100000000
000000000000000000000000000111111000000000000000000000001111111000000000
000000000000000000000000000111111000000000000000000000011111111000000000
000000000000000000000000000011111000000000000000000000011111110000000000
000000000000000000000000000001111100000000000000000000011111100000000000
000000000000000000000000000000111110000000000000000000111111000000000000
000000000000000000000000000000111110000000000000000001111110000000000000
000000000000000000000000000000011111000000000000000001111100000000000000
000000000000000000000000000000000111100000000000000111111000000000000000
000000000000000000000000000000000011110000000000001111110000000000000000
000000000000000000000000000000000001111110000000111111000000000000000000
000000000000000000000000000000000000011111111111111100000000000000000000
000000000000000000000000000000000000000011111111100000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
```

Figure 2: Character 98 from font `eiad10`.

some magnification, (*e.g.*, `mf "\mode:=localfont; mag=magstep(4); " input eiad10`). Our implementation of `gftype` truncates its output at 80 columns; this limits the range of usable magnifications.

The input file represents the whole picture (the same number of pixels will be printed); the relative positions of elements in this "picture" will correspond to their relative positions in the 3D output. There will appear to be a small leftward shift due to the left-right asymmetry in the recurrence relation. Automatically correcting for this asymmetry would require dropping user data on the right and/or adding new ground plane pixels on the left. Another solution is to put in a small rightward shift in the input file to begin with. Here the code makes no centering correction, and leaves that problem to the user.

The code

Most of the code is trivial. First we declare a few counter and dimension variables, most of which have an obvious meaning:

```
% Load PSTricks, if available, for color
%\input pstricks.tex
\newcount\lines\newcount\columns
\newcount\nx\newcount\ny
\newcount\ThisPixelColor
\newcount\ReferenceDepth
\newcount\Period
\newcount\StartingStripeWidth
\newdimen\dx\dx=3pt
\newdimen\dy\dy=\dx
\newread\DepthData\newlinechar=`\^^J
```

Counters `\lines` and `\columns` are read from the data file; `\columns` will be augmented by the program to include the necessary number of invisible pixels, `\StartingStripeWidth`. Dimension `\dx` specifies the pixel size, if we are drawing black and white *square* pixels (a random dot autostereogram); one may set `\dy` to a different value; the code that uses `\dx` and `\dy` is commented out because we will use characters instead, as texturing elements.

`\Period` is the period m of the image, as defined above. It should be as large as possible, for the reasons given in the previous section, but it is limited by the requirement that `\Period` times the pixel width should be of the order of an inch or two (this is about the distance between points your eyes will converge to on the paper; it must be neither too small nor too large, for good comfortable viewing).

A random number generator is used to create the initial stripe on the left hand side of the figure. The rest of the picture will be obtained by copying elements from that stripe. The code for the random number generator, using a simulated shift register, is taken from an article by Hans van der Meer in *TUGboat*, Volume 15 (1994), No. 1, pages 57–58.

```
\catcode`\@=11
\newcount\@SR
\def\@SRconst{2097152}
\def\SRset#1{\global\@SR#1\relax}
\def\@SRadvance{\begingroup
  \ifnum\@SR<\@SRconst\relax \count@=0
  \else \count@=1
  \fi
  \ifodd\@SR \advance\count@ by 1 \fi
  \global\divide\@SR by 2
  \ifodd\count@
      \global\advance\@SR\@SRconst\relax
  \fi
```

```
    \endgroup}
```

It is necessary to initialize the register with some seed value, and to step it a number of times before its output becomes "random". We also define macros `\SRtest` and `\SelectPattern` that use the generator to select randomly between 2 and 4 arguments, respectively.

```
\SRset{1141651}
\nx=20
\loop\ifnum\nx>0
  \@SRadvance\advance\nx by-1\repeat

\def\SRtest#1#2{\@SRadvance
       \ifodd\@SR #1\else #2\fi}
\def\SelectPattern#1#2#3#4{%
  \SRtest{\SRtest{#1}{#2}}%
         {\SRtest{#3}{#4}}}
\catcode`\@=12
```

Different seed values will result in different texturing patterns, so one should experiment with seeds if the output shows distracting accidental defects, like large holes, or lines that seem to be much darker or lighter than the average (this particular kind of defect could also be the result of excessive loss of information, as explained before; use of a larger `\Period` should help cure this).

A test is introduced, `\ifPrinting`, to distinguish between invisible starting pixels, and the printing ones. Continuously testing this flag slows down operations, but it simplifies the code structure.

```
\newif\ifPrinting
```

Next comes a set of macros defining the *basic tiles*: boxes `\abox`, `\bbox`, `\cbox`, `\dbox`, and corresponding macros `\A`, `\B`, `\C`, `\D`. Here, just about anything is allowed, as long as the pixels all have the exact same dimensions. Their aspect ratio can be adjusted, if desired, to compensate for the non-square aspect ratio of characters in the relief data file, when it is being edited. Best results are obtained when pixels are very small, so here I use 3.5pt and 4.5pt characters. The heights and widths of boxes `\abox`...`\dbox` are all coerced to 4.5pt, and depths are coerced to 0pt. Note that this will make the TeX logo overflow into neighboring pixels, but that doesn't matter since the result looks good (at least to me). Box `\bbox` is left empty, to produce a ligther texture.

```
% One way to add a little color:
\ifx\PSTricksLoaded\endinput
  \immediate\write16{Using pstricks}
\else
  \def\black{}
  \def\red{}
\fi

\font\TeXlogo=cmr5 at 3.5pt
\font\Cards=cmsy6 at 4.5pt
\font\Cardlets=cmsy6 at 3.5pt
\newbox\abox\newbox\bbox
\newbox\cbox\newbox\dbox

\setbox\abox=\hbox{%
  \black\kern-1.5pt\TeXlogo\TeX}
\setbox\bbox=\hbox{}
\setbox\cbox=\hbox{\red\Cards\char"7E}
\setbox\dbox=\hbox{\red\Cardlets\char"7E}
\wd\abox=4.5pt
\ht\abox=\wd\abox\dp\abox=0pt
\ht\bbox=\ht\abox
  \wd\bbox=\wd\abox
  \dp\bbox=\dp\abox
\ht\cbox=\ht\abox
  \wd\cbox=\wd\abox
  \dp\cbox=\dp\abox
\ht\dbox=\ht\abox
  \wd\dbox=\wd\abox
  \dp\dbox=\dp\abox
```

The following macros must not insert extra spaces:

```
\def\A{%
  \leftappenditem{\A}\to\CurrentLine%
  \ifPrinting\copy\abox\fi}
\def\B{%
  \leftappenditem{\B}\to\CurrentLine%
  \ifPrinting\copy\bbox\fi}
\def\C{%
  \leftappenditem{\C}\to\CurrentLine%
  \ifPrinting\copy\cbox\fi}
\def\D{%
  \leftappenditem{\D}\to\CurrentLine%
  \ifPrinting\copy\dbox\fi}
```

Boxes containing a black and a white square are also defined: `\noir` and `\blanc`, plus macros `\K` and `\W`; these can be used to generate random dot autostereograms of the sort presented in Christopher Tyler's early papers.

```
\newbox\blanc\newbox\noir
\setbox\blanc=\hbox to \dx{%
   \vrule height\dy
   depth0pt width0pt\hfil}
\setbox\noir=\hbox to \dx{%
   \vrule height\dy depth0pt width\dx}

\def\W{%
  \leftappenditem{\W}\to\CurrentLine%
  \ifPrinting\copy\blanc\fi}
\def\K{%
  \leftappenditem{\K}\to\CurrentLine%
  \ifPrinting\copy\noir\fi}
```

To implement the $p_i = p_{i-\sigma_i}$ formula, my original plan was to use some of TEX's tables (lccode, uccode, etc...) as arrays to store these values, but I eventually opted for a well documented more conservative approach: I use the list manipulation techniques presented in Appendix D of the TEXbook. An algorithm based on code tables would certainly show better performance. A list called \CurrentLine holds symbolically the contents of the output line that is currently being worked on, in the form of a list of single letter macro tokens. Every time one of the basic tiles has been selected, through a call of the form \select σ_i \of\CurrentLine, the corresponding macro token (\A, \B, \C, \D — or \K, \W) is inserted with \leftappenditem at the head of the \CurrentLine list,

```
\toksdef\ta=0 \toksdef\tb=2
\long\def\leftappenditem#1\to#2{%
      \ta={\\#1}%
      \tb=\expandafter{#2}%
      \xdef#2{\the\ta\the\tb}}
```

and the same macro is also immediately executed to copy the box contents at the current position in the printed output line.

The macro \clearline resets the \nx column counter, and initializes the list to \outofrange; \outofrange is used as a sentinel and is useful for trapping errors and to accelerate the workings of another important macro, \GobbleRest.

```
\def\outofrange{\immediate\write16{%
     ^^JOut-of-range relief:
     left limit reached^^J}}

\def\clearline{%
   \gdef\CurrentLine{\\\outofrange}%
   \global\nx=0}
```

The macro \select works basically as explained in the TEXbook (p. 379), except for a little change that dramatically increases its speed (speed is important in the present application).

```
\def\select#1\of#2{%
  \gdef\result{\outofrange}%
  \gdef\\##1{\advance#1-1 %
          \ifnum#1=0 %
            \def\result{##1}%
            \let\\=\GobbleRest%
          \fi}%
   #2\result}
```

```
\def\GobbleRest#1\outofrange{}
```

It does a backward search in the list of p_is for the $(i - \sigma_i)^{\text{th}}$ element. It defines \\ to be a macro that gobbles the next token, doing nothing else, until the sought after list element has been reached; at that point, it saves the token for later reinsertion in the input stream, and redefines \\ to be a macro that eats everything else in the stream, in one step, up to the end-of-list marker \outofrange. The execution time is no longer proportional to the length of the list, but instead it is roughly proportional to the value of \Period (assuming most pixels have $\sigma_i \approx m$)

The top level macro that selects what will appear at the current position is \MakeThisPixel. It gets depth values d_i from \relief, which was originally written to supply a fixed, hard coded pattern; now it gets the data from the current relief input line \DepthDataLine, with the help of \GetNextInputDigit.

```
\def\MakeThisPixel{{%
 \ThisPixelColor\expandafter=\relief%
 \shift%
 {\select\ThisPixelColor\of\CurrentLine}%
 }}

\def\relief{\expandafter%
  \GetNextInputDigit%
    \DepthDataLine\endofline}

\def\GetNextInputDigit#1#2\endofline{%
     #1\relax\gdef\DepthDataLine{#2}}
```

The heart of the autostereogram algorithm is the trivial (!) \shift macro, whose rôle should be obvious by now: it computes σ_i.

```
\def\shift{%
  \advance\ThisPixelColor
          by-\ReferenceDepth
  \ThisPixelColor=-\ThisPixelColor
  \advance\ThisPixelColor by\Period}

% ******** End of definitions ********
```

After reading the main picture parameters, and doing a simple feasibility check, the code ends with a very simple main loop over lines and columns. The width of the required invisible stripe is calculated, and added to the user specified number of columns.

```
% Now do the work
\openin\DepthData=eiadb.dat
\read\DepthData to\DepthDataLine
         \columns=\DepthDataLine
```

```
\read\DepthData to\DepthDataLine
           \lines=\DepthDataLine
\read\DepthData to\DepthDataLine
          \Period=\DepthDataLine
\read\DepthData to\DepthDataLine
  \ReferenceDepth=\DepthDataLine

\ny=\lines
\nx=\Period\advance\nx by\ReferenceDepth
\ifnum\nx<0
  \immediate\write16{Illegal parameters:
     imply forward recursion.  You should
     probably raise the reference level.
     For this run, it is forced to zero.}
  \ReferenceDepth=0
\fi

\StartingStripeWidth=\Period
\ifnum\ReferenceDepth>0
  \advance\StartingStripeWidth
         by\ReferenceDepth
\fi
\advance\columns by\StartingStripeWidth
\hbadness=10000
\overfullrule=0pt
\offinterlineskip
\parindent=0pt
```

Before the main loop begins, a pair of crosses is written in the center of the page, that are one `\Period` apart*; `\wd\abox` must be replaced by `\dx`, if one is using the black and white squares.

```
\vfill
\nx=0
\line{\hfil\rlap{$+$}%
  \loop\ifnum\nx<\Period%
    \hskip\wd\abox\advance\nx by 1%
  \repeat\rlap{$+$}\hfil}
\vskip5mm
```

[To get the spacing right, it is important not to output any spurious `blank space` in those parts of the code.] These crosses can be used as a practice target, to get used to the particular periodicity of the picture. The viewer is ready to look at the real picture when he/she is able to see 3 crosses in line at the top of the image, in a stable and cozy way.

Looping over columns is done in three parts. First, `\Period` invisible pixels are generated randomly (line 11 or 12, below); `\SelectPattern` selects among four possibilities. To produce a random dot autostereogram, one would replace the `\SelectPattern` line with the `\SRtest{\K}{\W}` line, which is presently commented out, and adjust `\dx` and `\dy` as necessary. Then if necessary, a few more pixels are generated by recurrence (not at random), with $\sigma_i = m$, to complete the invisible stripe (line 14). If printed, these pixels would be perceived as being part of the reference plane. Finally, the visible part of the line is generated with the full recurrence algorithm (line 17).

```
 1. \loop\ifnum\ny>0
 2.   \clearline
 3.   \message{<\number\ny}
 4.   \read\DepthData to\DepthDataLine
 5.   \nx=0 \Printingfalse
 6.   \line{%
 7.     \hfil%
 8.     \loop\ifnum\nx<\columns%
 9.       \ifnum\nx<\StartingStripeWidth
10.         \ifnum\nx<\Period %
11. %          \SRtest{\K}{\W}%
12.            \SelectPattern{\A}{\B}{\C}{\D}%
13.         \else%
14.          {\select\Period\of\CurrentLine}%
15.         \fi%
16.       \else%
17.         \Printingtrue\MakeThisPixel%
18.       \fi%
19.       \advance\nx by 1 %
20.     \repeat%
21.     \hfil}%
22.   \message{>}%
23.   \advance\ny by -1 %
24. \repeat
25. \closein\DepthData
26. \bye
```

The reader with normal vision, who has difficulties perceiving depth in autostereograms, should experiment first with classic stereo pair images, possibly using a 3D viewer; then he/she should move on to simple high resolution black and white random dot autostereograms; these contain fewer distracting features than the character based autostereograms, but are not as pretty.

◇ Jacques Richer
CEntre de Recherche en Calcul Appliqué (CERCA),
5160, boul. Décarie, bureau 434,
Montréal, PQ, CANADA H3X 2H9
`richer@cerca.umontreal.ca`

* This part of the code had to be modifed to run within *TUGboat*; this is the unmodified version.

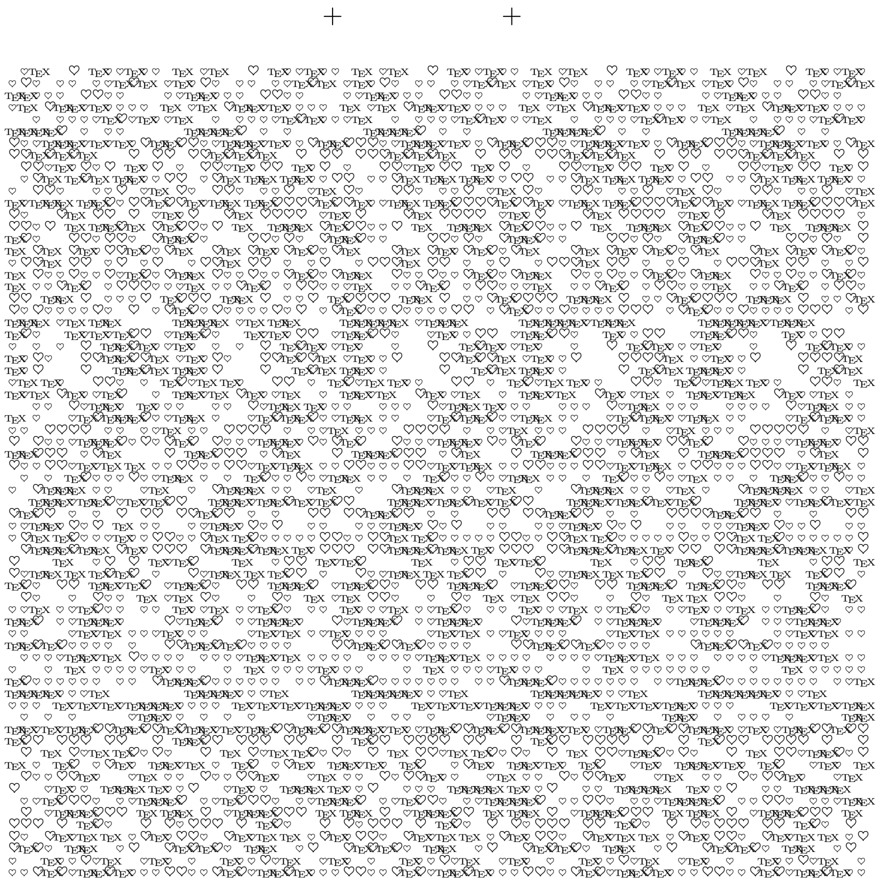
Figure 3: Output generated from Fig. 2.