



Book reviews

Mathematical Illustrations: A Manual of Geometry and Postscript

B. Casselman
Cambridge University Press
ISBN: 0521547881

I am recommending this book as a book on PostScript, which raises the question: why bother? Bill Casselman's view (and mine) is that compared to the alternatives: "*once you have made a small initial investment of effort, by far the best thing to do is to write a program in the graphics programming language, PostScript*". The code for the book, and thus for its illustrations, is available from: <http://www.math.ubc.ca/~cass/graphics/manual>. Other examples can be seen in the author's spectacular graphics on the covers of AMS notices: <http://www.ams.org/notices>.

Chapter 1 covers getting started with PostScript, including the use of the GhostScript viewer to display PostScript files, although surprisingly it makes no mention of the GhostView interface. Other ways of using PostScript files are covered in appendices.

Even at level 1 [1], PostScript is a rich language with 230 commands of which about 30% are graphics-specific (level 2 which is what is used in the book is a slight expansion, and level 3 includes direct support for PDF). PostScript follows the 'reverse polish' stack-oriented syntax of the FORTH programming language [2]. An important feature is that the language is interpreted rather than compiled, allowing dynamic redefinition of operations. The book is sprinkled with hints on programming practice, some specific to PostScript, and some applicable to many programming languages.

The recommendation for modularity is important, but it leads to longer files than can be accommodated in this review and so the examples in this review are stripped-down illustrations of my own. In the C tradition of 'hello world', the 6 lines of code on the left produce the output on the right, embedded as a graphic (after conversion to PDF to meet the production requirement of the *Gazette*).

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 200 200 420 230
/Helvetica findfont 20 scalefont setfont
205 205 moveto
(Greetings to the Galaxy) show
showpage
```

Greetings to the Galaxy

Using PostScript gives you something that is (a) more general than the \LaTeX PStricks package [5] and (b) usable in a wider range of environments. Among the ways of using PostScript are: printing, either directly or via GhostScript/GhostView, conversion to other graphics via GIMP etc, or importing into \TeX / \LaTeX .

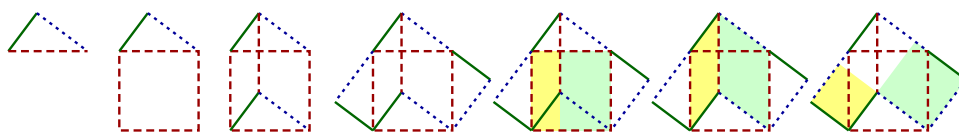
Chapter 2 covers coordinates and geometry. While this review is really about the *Manual of ... PostScript* bit, for the author, the *Geometry and ...* bit is at least as important.

Indeed he notes an ‘almost recursive’ aspect: a book that teaches you how to make a book on geometry. The author uses PostScript graphics for teaching geometry, in particular geometry related to a parallel course in computer graphics. He reports finding one or two students in each class responding with a previously unrealised enthusiasm for mathematics. On the basis of zero personal experience, I find this convincing. It may not be a priority of many readers of this review, but it is something interesting to consider if you are in a position to use it that way.

The author proclaims that the ideal (geometric) proof has no words, and no labels on diagrams. As an example, he demonstrates the principle that the area of a parallelogram is $\text{base} \times \text{height}$, and that this is (by definition and illustration) invariant under shear transformations. He then uses this to ‘prove’ Pythagoras’ theorem by a sequence of shear transformations that map the area of the square on the hypotenuse onto areas of squares whose lengths are those of the other two sides.

The graphic below differs from the book in using an alternative construction, and also in giving more emphasis to the steps in the construction than to the shear transformations. In particular, my version uses linestyle (and colour if you are reading the on-line version of the *Gazette*) to identify lengths that are equal by construction. For greater completeness, identical and complementary angles should be identified, since the key step (and the one that only works for right-angled triangles) is that the two line segments forming each of the upper boundaries are co-linear, thus allowing the second pair of shear transformations.

The PostScript file is 124 lines (too long to include) but is easy to write, using a lot of cut-and-paste as the lines for each stage are copied (and augmented) for the next stage.



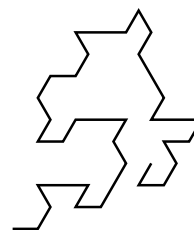
Chapter 3 describes procedures. The PostScript syntax has procedures as operators and operands within braces, assigned to a name (identified by an initial /) with the `def` command and then used by invoking the name without the /.

The example below, created for this review, plots a self-avoiding walk (on the triangular lattice to make it a little harder):

```

%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 170 190 290 290
/STEP 10 def /NS STEP neg def
/HS STEP 2 div def /NHS HS neg def
/DS HS 3 sqrt mul def /NDS DS neg def
/A {STEP 0 rlineto} def /D {NS 0 rlineto} def
/B {HS DS rlineto} def /E {NHS NDS rlineto} def
/C {NHS DS rlineto} def /F {HS NDS rlineto} def
newpath 200 200 moveto
A B C A A E A B B C B D D E D B C B A B C
A A B F E F F E A A E D F E D B
stroke showpage

```



Chapter 4 describes coordinate transformations, including both the inbuilt PostScript capabilities and the transformations required to work with a 3-D representation to produce 2-D graphics.

Chapter 5 introduces loops, allowing the real power of PostScript as a programming language to be harnessed. The asymmetric simple exclusion process (ASEP), an example near to the editor's heart [3, and references therein], is illustrated by the code and graphic below, although the random boundary conditions differ slightly from those in [3]. In Wolfram's nomenclature, this is Cellular Automaton number 184. Other Cellular Automata can be displayed by replacing the code for `func` with an encoding of the relevant rule from Table 1 in the appendix of [6].

This code also demonstrates PostScript documentation convention (in my examples preceded by `%USAGE:`) of:

initial_state_of_stack operator final_state_of_stack

```

%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 60 90 300 400
%%Title: File aseps.eps
/xdef {exch def} def
/unitrand { rand 65536 div 32768 div } def

%USAGE: prob setpsin '0 or 1'
/setspin {unitrand le {0} {1} ifelse } def

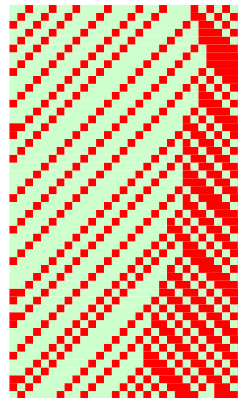
% Define sizes and positions
/yc 100 def /x0 100 def /xc x0 def
/dx 5 def dx setlinewidth
/ysize 30 def /nlim ysize 2 sub def

% Rendering of states
/A { 0.8 1 0.8 setrgbcolor block} def
/B { 1 0 0 setrgbcolor block} def
/LF { /xc x0 def yc dx add /yc exch def} def
/block {newpath xc yc moveto dx 0 rlineto stroke xc dx add /xc exch def} def
%USAGE: array Print --- output states at single time step
/Print { { 0 eq {A} {B} ifelse } forall LF} def

% Set ASEP probabilities: 'entry' and 'loss'
/alpha 0.4 def /beta 0.5 def

% USAGE: vp, vc, vn func vnew
/func { exch /tc exch def tc mul exch 1 tc sub mul add } def % RULE 184
%
/ycalc {/nc exch def
nc 1 sub xx exch get nc xx exch get
nc 1 add xx exch get func yy exch nc exch put } def
%
/xcalc {/nc exch def
nc 1 sub yy exch get nc yy exch get
nc 1 add yy exch get func xx exch nc exch put } def
%
/xcalcn {yy nlim 1 add get dup 0 eq
{pop yy nlim get /vn xdef}

```



```

    { 1 beta sub setspin mul /vn xdef}
    ifelse
    xx nlim 1 add vn put } def

/ycalcn {xx nlim 1 add get dup 0 eq
  {pop xx nlim get /vn xdef}
  { 1 beta sub setspin mul /vn xdef}
  ifelse
  yy nlim 1 add vn put } def

% starting configuration,
/xx [ 0 1 0 0 0 0 1 0 0 1 0 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0 0 1 1 1 1 0 ] def
% yy is just a placeholder
/yy [ 0 1 1 0 0 0 1 0 0 1 1 1 0 1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 ] def

% Random states (Pr = \alpha) at left boundary
/xcalc0 {xx 0 alpha setspin put } def
/ycalc0 {yy 0 alpha setspin put } def
%
/Loop { xx Print
  1 1 nlim {ycalc} for ycalc0 ycalcn
  yy Print
  1 1 nlim {xcalc} for xcalc0 xcalcn} def
1 1 25 {Loop} for
%
showpage

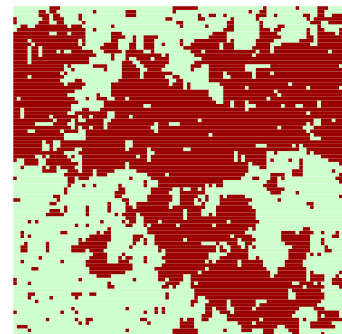
```

One aspect that is noted, but not illustrated in the book, is the use of programs in other languages to write output as PostScript. Perhaps a simpler approach is to write fragments. My example is from statistical physics:

```

%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 90 90 565 560
% File monte.ps1 Plot Ising simulation
/dx 5 def
/yc 100 def
/x0 100 def
/xc x0 def
dx setlinewidth
/A { 1 0 0 setrgbcolor block} def
/B {0 1 0 setrgbcolor block} def
/LF { /xc x0 def yc dx add /yc exch def} def
/block {newpath xc yc moveto dx 0 rlineto
  stroke xc dx add /xc exch def} def
% End of file monte.ps1
% File monte0.psd: Dummy test data
A B A A B LF A B A B B LF B A A A B
showpage
% end of file monte0.psd

```



The file on the left produces the upper graphic on the right. Replacing the ‘A B ...’ sequence by output from a larger simulation (the Ising model at criticality) leads to plots like the lower graphic on the right. This allows another form of modularity. The sequence of

choices is detached from how the cases are rendered, so that, for example, A and B could be redefined to produce + and −, or ↑ and ↓ while using the same program output. The book gives Perl scripts for combining modules, but simple cat (Unix) or copy (DOS) commands will take you a long way.

An important possibility not mentioned in Casselman's book is the ability to produce conditional graphics. The idea is to have one file that, with a single binary choice, can produce either a graphic for *Nature*, *Journal of Geophysical Research* etc, vs. one for *New Scientist*, *Scientific American* or even (shock horror!!) just readable overheads. The main choices are black-and white vs colour, thin vs. thick lines, many vs. few tick marks etc. Keeping code for both options is an example of the power of modularisation.

The book includes examples for some quite complex constructs. These include projections, text-morphing, and tools (like bubble-sort and the recursive quick-sort) that are needed for hidden surface analysis in rendering 3-D objects. However, what I found awe-inspiring was the detailed maps of world coastlines, used in part to illustrate projections. I have written a book [4] where I could (and did) use hand-coded PostScript to produce all diagrams **except maps**. Consequently, for me, the ability to use PostScript to produce world coastline maps like those in the book was eloquent testimony to the power of well-designed modularity, the capabilities of the language and the author's vision and creativity.

Producing maps may be taking PostScript programming into the realm of extreme sports, but for lesser tasks, PostScript programming is fun and effective, and Bill Casselman's book is an excellent way to get started.

References

- [1] Adobe Systems, *PostScript Language Reference Manual* (Addison-Wesley Reading Mass 1986).
- [2] L. Brodie, *Starting FORTH* (Forth Inc. Hermosa beach CA 1981).
- [3] J. de Gier and F.H.L. Essler, *Bethe Ansatz solution of the asymmetric exclusion process with open boundaries*, Phys. Rev. Lett. **95**(2005), 240601, 4pages.
- [4] I.G. Enting, *Inverse Problems in Atmospheric Constituent Transport* (CUP Cambridge UK 2002).
- [5] M. Goossens, S. Rahtz, and F. Mittelbach, *The L^AT_EX Graphics Companion: Illustrating Documents with T_EX and Postscript* (Addison-Wesley Reading Mass. 1997).
- [6] S. Wolfram (Ed.), *Theory and Applications of Cellular Automata* (World Scientific Singapore 1986).

Ian Enting

ARC centre of Excellence for Mathematics and Statistics of Complex Systems, The University of Melbourne, VIC 3010

E-mail: ienting@unimelb.edu.au