

Fun with Post Script

Gene Michael Stover

created 2005 July 13
updated 2006 April 20

Copyright © 2005–2006 Gene Michael Stover. All rights reserved. Permission to copy, store, & view this document unmodified & in its entirety is granted.

Contents

1 What is this?	1
2 Three shaded boxes	1
2.1 Encapsulated Boxes	2
3 Negative Coordinates in User Space	4
4 Simple 2-D graphs	7
5 Doodling: yin & yang	9
A Lisp program to graph 2-D points in PostScript	15
B Example 2-D graphing PostScript program	28
C Other File Formats	38

1 What is this?

Notes about Post Script, the page description language.

To learn Post Script, see *Postscript language tutorial and cookbook* [5].

2 Three shaded boxes

Let's start with a small program from page 30 of *Postscript language tutorial and cookbook* [5]. It's simple, & it's already written, so it's perfect to make sure I can print a properly written PostScript program (or display it with `ghostscript`).

Here is the source code:

```

% three boxes
% from COOKBOOK, page 30

% --- Define box procedure ---
/box
{ 72 0 rlineto
  0 72 rlineto
 -72 0 rlineto
  closepath } def

% --- Begin Program ---
newpath % First box
  252 324 moveto box
  0 setgray fill
newpath % Second box
  270 360 moveto box
  .4 setgray fill
newpath % Third box
  288 396 moveto box
  .8 setgray fill
showpage

```

The source code also at is <http://cybertiggyr.com/gene/fps/box3.ps>.

The output of the `box3.ps` program is at <http://cybertiggyr.com/gene/fps/box3.png>.

2.1 Encapsulated Boxes

To include the `box3.ps` program in this L^AT_EX document, I must convert it to an *encapsulated PostScript* file.

An *encapsulated PostScript* file is a PostScript file which follows a subset of the *Document Structuring Convention*. Namely, it adds some comments that tell L^AT_EX & similar programs the size of the image produced by a PostScript file. (Read about encapsulated PostScript & the Document Structuring Convention elsewhere. Adobe might be a good source.)

Here is the source code for the “three boxes” program, converted to encapsulated PostScript:

```

%!PS-Adobe-3.0 EPSF-3.0
%%Creator: Gene Michael Stover
%%Title: box3.eps
%%CreationDate: 2005 July 13
%%DocumentData: Clean7Bit
%%LanguageLevel: 2
%%Pages: 1
%%BoundingBox: 179 251 400 500

```

```

%%EndComments
%%BeginProlog
% Use own dictionary to avoid conflicts
10 dict begin
%%EndProlog
%%Page: 1 1

% three boxes
% from COOKBOOK, page 30
% converted to an Encapsulated PostScript file

% --- Define box procedure ---
/box
{ 72 0 rlineto
  0 72 rlineto
 -72 0 rlineto
  closepath } def

% --- Begin Program ---
newpath % First box
  252 324 moveto box
  0 setgray fill
newpath % Second box
  270 360 moveto box
  .4 setgray fill
newpath % Third box
  288 396 moveto box
  .8 setgray fill
showpage
%%Trailer
end
%%EOF

```

This source code for the “three boxes” program, converted to an encapsulated PostScript file, is also at <http://cybertiggyr.com/gene/fps/box3.eps>.

Figure 1 shows the three boxes program after converting it to an encapsulated PostScript file so I can use it in \LaTeX & \LaTeX2HTML .

The printed version looks just like the output in the PostScript Tutorial, but the version which displays in HTML is terrible. `ghostscript` displays it fine. (For the printed version, I tried it on Laser Jet 8150, Xerox Tektronix Phaser 860DP, & an Okidata C9300.)

While doing this, I learned that if `pstoimg` complains of an “illegal seek”, it means your bounding box is wrong. My bounding box was too small. A short, decent discussion of bounding boxes is in [1].

If your bounding box is too big, it won’t make any difference to the HTML version; I guess `pstoimg` crops un-drawn parts of the image. It makes a differ-



Figure 1: The output of the `box3` program as encapsulated PostScript

ence to the printed version, though. When creating new drawings, maybe it's a good idea to decide on your bounding box first & start the EPS file with that bounding box.

Why is the HTML version so terrible? Looks like `pstoimg` is assuming a light gray background, & it also converts the third box, whose lightness is 0.8, to white. The result looks wrong to the eye, though I guess it technically did what it was supposed to. Methinks it's a good lesson in how different devices render the same PostScript in different ways.

3 Negative Coordinates in User Space

Do negative coordinates in user space have meaning to PostScript? In other words, if we use the `translate` operator to move user space so that, say, $(X, Y) = (-100, -100)$ is on the page, could we draw a line from $(-100, -100)$ to $(10, 10)$?

Interestingly, I could not find the answer to this question in [6] or [6].

Here's a PostScript program to give us the answer:

```
%!PS-Adobe-3.0 EPSF-3.0
%%Creator: Gene Michael Stover
%%Title: $Id: rand0.eps,v 1.1 2006/05/14 02:23:11 gene Exp $
```

```
%%CreationDate: 2006 May 13
%%DocumentData: Clean7Bit
%%LanguageLevel: 1
%%Pages: 1
%%BoundingBox: 72 72 242 242
%%EndComments
%%BeginProlog
```

```
100 dict begin
```

```
%
% There are 480/17 points in a centimeter.
% That's about 28.24 points in a centimeter.
%
/cm 480 17 div def
```

```
%
% The picture is square, six centimeters on a side.
% 6 cm
% is  $6 * 480 / 17$  point
% is  $2880 / 17$  point
% is about 169.4 point.
%
/Size 6 cm mul def
```

```
%
% Minimum coordinates in user space.
%
/Lower Size neg def
```

```
%%EndProlog
%%Page: 1 1
```

```
%
% Translate user space
% The lower left corner is at (72, 72).
% So the top right corner is at (72 + 169.4, 72 + 169.4),
% which is (241.4, 241.4).
%
Size 72 add Size 72 add translate
```

```
%
% Draw our diagonal line
% from user space's minimum (which is negative)
% to user space's upper right, which is (0, 0).
%
newpath
  Lower Lower moveto
  0 0 lineto
3 setlinewidth
```

```

0 setgray
stroke

%
% Draw markers at the lower left corner.
% Each of the two marker lines is one
% centimeter long.
%
newpath
  Lower      Lower moveto
  Lower cm add Lower lineto      % horizontal
  Lower      Lower moveto
  Lower Lower cm add lineto      % vertical
1 setlinewidth
0 setgray
stroke

%
% Draw markers at the upper right corner.
% Each of the two marker lines is one
% centimeter long.
%
newpath
  0 0 moveto
  cm neg 0 lineto      % horizontal
  0 0 moveto
  0 cm neg lineto      % vertical
1 setlinewidth
0 setgray
stroke

% showpage
%%Trailer
end
%%EOF

```

The program is also available at <http://cybertiggyr.com/gene/fps/xlate0.tex>.

Figure 2 shows the output of that PostScript program.

Looks like negative coordinates in user space are okay, if you have offset user space so that the negative coordinates are on the page.

I notice that the upper right end of the line protrudes past the cornering marks, whereas the lower left end fits into them nicely. I suspect the lower left end is clipped & the upper right end isn't. If I placed the upper right end more carefully, it would benefit from clipping, too. I'll leave that for another experiment.

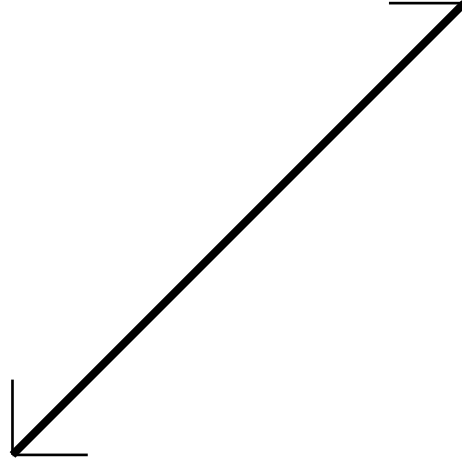


Figure 2: The output of the `xlate0.eps` Encapsulated PostScript program

4 Simple 2-D graphs

In Appendix A is a Lisp program which eats a list of two-dimensional data points & excretes a PostScript program which graphs those points. Figure 3 and Figure ?? show two graphs which resulted from running that Lisp program.

To make Figure 3, I ran the `demo0000` function from `graph1.lisp`, which does this:

```
(graph0 '((10 9) (8 7) (6 5) (4 3)) "demo0000.eps")
```

In that expression, you can see the the `graph0` function does the work of generating a PostScript program. You give it a list of 2-D points. In this case, I've hard-coded some arbitrary points. `graph0` also needs the pathname for an output file; in this case, I've hard-coded `demo0000.eps`.

To make Figure 4, I ran the `demo0001` function from `graph1.lisp`, which does this:

```
(graph0 (mapcar #'(lambda (x) (list x (expt x 3)))  
'(-1 -1/2 -1/3 -1/7 0 1/33 1/7 1/5 1/3 1/2 1))  
"demo0001.eps")
```

Here, we have a list of X values, & we're generating the $Y = X^3$ values to go with the X values.

As an example, the PostScript source code which produced Figure 4 is in Appendix B.

If you look at the Lisp program & the PostScript program it produced, you'll notice that the Lisp program is mostly an ugly collection of `format` calls. The Lisp program mostly substitutes its arguments into the PostScript program.

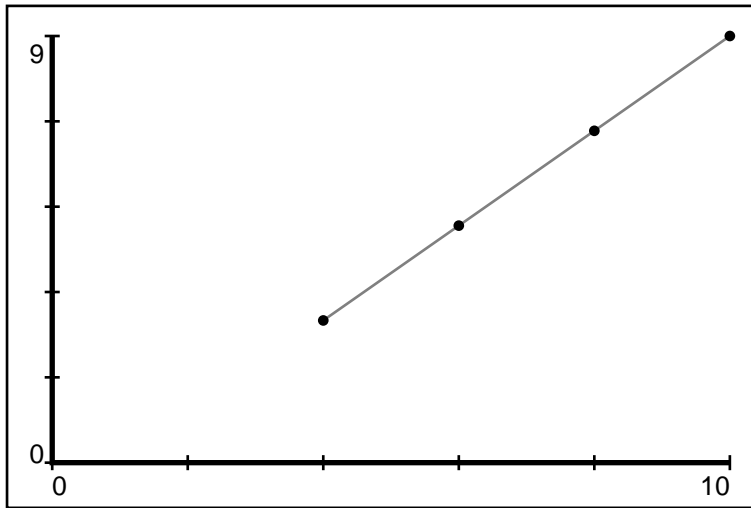


Figure 3: The output from my COM.CYBERTIGGYR.GENE.FPS.GRAPH1::DEMO0000 Lisp program

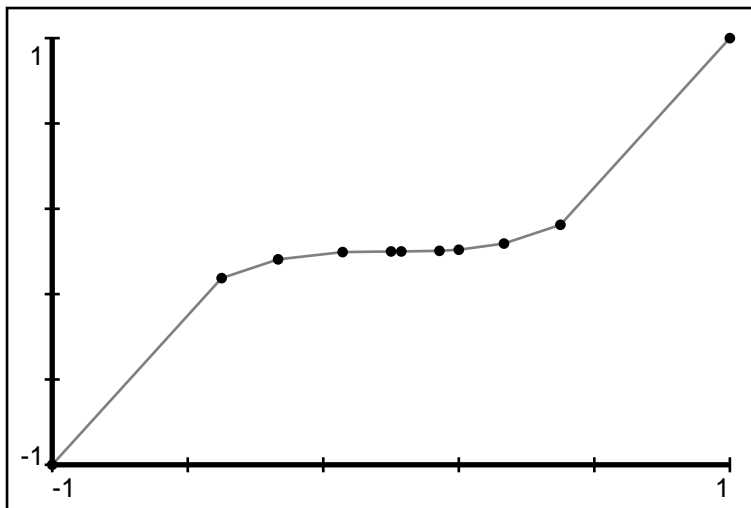


Figure 4: The output from my COM.CYBERTIGGYR.GENE.FPS.GRAPH1::DEMO0001 Lisp program

About the only data processing it does it sorting the list of points. All the work happens in the PostScript program.

I did it that way because I suspect that most PostScript printer drivers do most of the computation, then produce PostScript programs which are just lists of “move here, draw this”. I don’t know for sure that they do this; but I suspect they do. I wanted to see what happened if the PostScript program did most of the work. So instead of a driver-generated program which says “move here, draw this”, this PostScript program says “figure out where to move, move there, then draw this”. See the big difference? *grin*

I like the resulting PostScript program, but the Lisp program is kind of ugly. At least, it’s super basic. It’s so basic that writing it in C or Bourne shell wouldn’t have required more effort. Maybe that’s a good thing. If I ever think of a prettier way to accomplish the same thing in Lisp, I might re-write that program. It will still just substitute its arguments into the PostScript program template & let the PostScript program do the rest of the work.

5 Doodling: yin & yang

Figure 5 shows a familiar icon. Here is the source code for that icon. The source code is also at

<http://cybertiggyr.com/gene/fps/yinyang.eps>.

```
%!PS-Adobe-3.0 EPSF-3.0
%%Creator: Gene Michael Stover
%%Title: $Id: yinyang.eps,v 395.1 2008/04/20 17:25:46 gene Exp $
%%CreationDate: 2006-05-20
%%DocumentData: Clean7Bit
%%LanguageLevel: 1
%%Pages: 1
%%BoundingBox: 72 250 542 720
%%EndComments
%%BeginProlog
100 dict begin

    % BOUNDING BOX ~~~~~
    % That bounding box up there is for a radius of 3 cm,
    % a margin of 3 mm around the image, & a margin of 72 points
    % between the image & the paper’s edge.
    %
    % A radius of 3 cm is a diameter of 6 cm.
    % With a border of 3 mm, the width of 6.3 cm.
    % 6.3 cm is 177.88235 PostScript points.
    %
    % So the right margin is 72 + 177.88235 = 249.88235.
    %
    % The top margin is at 720 because the paper is 11
    % inches high & we have a margin of 72.
    %
```

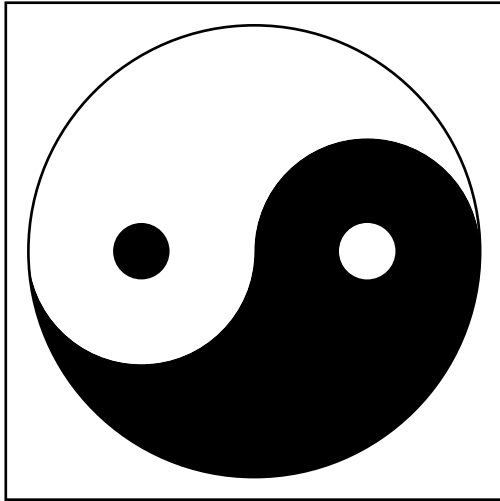


Figure 5: You've seen it before

```

% The bottom margin is 720 - 177.88235 = 542.1177.
%

%%%
%%% Library 0: Does not depend on any parameters
%%% or arguments.
%%%
%
% Given inches, return points.
% Stack: inches => points
%
/inch {
    72 mul
} def

%
% Given centimeters, return points.
% There are 480/17 points in a centimeter.
% I multiply, then divide, by integers in the
% hope of taking advantage of the rare case
% in which we have an integral number of
% points.
%
/cm {
    480 mul 17 div
} def

%
% Given millimeters, return points.
% There are 48/17 points in a millimeter.
%
/mm {
    48 mul 17 div
} def

%
% Make a box path.  Creates a new path for it.
% Does not stroke or fill it.
% Stack: min-x min-y max-x max-y => -
%
/newpath-box {
    4 dict begin
        /max-y exch def
        /max-x exch def
        /min-y exch def
        /min-x exch def

    newpath
        min-x min-y moveto
        max-x min-y lineto

```

```

        max-x max-y lineto
        min-x max-y lineto
        closepath
    end
} def

%%%
%%% Parameters & arguments
%%%
%
% Paper's height in PostScript points. We assume the
% page is 11 inches high; that's common in the
% United States.
% The units are PostScript points.
%
/PageHeight 11 inch def

%
% Width of the margin between the edge of the paper &
% the edge of the image. Any edge.
% The unit is PostScript points.
%
/Margin 1 inch def

%
% Border between the image's edge & the graphing
% area.
%
/Border 3 mm def

%
% The radius of the circle.
%
/FullRadius 3 cm def

%
% The thickness of all the lines.
%
/LineThickness 1 def

%%%
%%% Library 1: Depends on Parameters & Arguments
%%%

%
% The left edge of the entire image
%
/ImageMinX Margin def

%
```

```

% The right edge of the image
%
/ImageMaxX FullRadius Border add 2 mul ImageMinX add def

%
% The top edge of the whole image
%
/ImageMaxY PageHeight Margin sub def

%
% The bottom edge of the whole image
%
/ImageMinY ImageMaxY FullRadius Border add 2 mul sub def

%
% The location of the center of the circle. The X
% center is obvious. The Y center is less obvious
% because we want it to be near the top of the page.
% That location doesn't matter at all when this
% Encapsulated PostScript program is being encapsulated,
% but it's useful for debugging.
%
/FullCenterX ImageMinX Border add FullRadius add def
/FullCenterY ImageMaxY Border sub FullRadius sub def

%
% The radius of the two semi-circles. They have the same
% radius, & that radius is 1/2 that of the full circle.
%
/SemiRadius FullRadius 2 div def

%
% The center of the right semi-circle.
%
/RightCenterX FullCenterX SemiRadius add def
/RightCenterY FullCenterY def

%
% The center of the left semi-circle.
%
/LeftCenterX FullCenterX SemiRadius sub def
/LeftCenterY FullCenterY def

%
% The radius of the dots within the two semi-circles.
% The two dots have the same radius.
%
/DotRadius SemiRadius 4 div def

%%EndProlog

```

%%Page: 1 1

```
%
% Draw a box around the entire image
%
ImageMinX ImageMinY ImageMaxX ImageMaxY newpath-box
LineThickness setlinewidth
0 setgray
stroke

%
% Make a path around the right (black) spermatazoa.
% This requires drawing both the semi-circles & half
% of the full circle (which is literally a semi-circle).
% And we have to fill it.
%
newpath
  % Draw the right semi-circle.
  RightCenterX RightCenterY SemiRadius 0 180 arc
  % Draw the left semi-circle.
  LeftCenterX LeftCenterY SemiRadius 0 180 arc
  % Draw the bottom semi- of the full-circle.
  FullCenterX FullCenterY FullRadius 180 360 arc
0 setgray
fill

%
% Draw the white dot within the right, black,
% spermatazoa.
%
newpath
  RightCenterX RightCenterY DotRadius 0 360 arc
1 setgray
fill

%
% Make a path around the left, white spermatazoa.
% This requires drawing both the semi-circles & half
% of the full circle (which is literally a semi-circle).
% And we have to fill it.
% On white paper, this is unnecessary, but it's necessary
% for viewing on the web. The program which "latex2html"
% uses to convert Encapsulated PostScript to an image
% interprets non-drawn areas in a light gray. To force
% the white spermatazoa to be white in the resulting
% image on the web, we must draw it as white.
%
newpath
  % Draw the right semi-circle.
  RightCenterX RightCenterY SemiRadius 0 180 arc
```

```

        % Draw the left semi-circle.
        LeftCenterX LeftCenterY SemiRadius 0 180 arcn
        % Draw the top semi- of the full-circle.
        FullCenterX FullCenterY FullRadius 180 360 arcn
1 setgray
fill

%
% Draw the black dot within the left, white
% spermatazoa.
%
newpath
    LeftCenterX LeftCenterY DotRadius 0 360 arc
0 setgray
fill

%
% Draw the full circle to avoid an impressionistic
% white spermatazoa.
%
newpath
    FullCenterX FullCenterY FullRadius 0 360 arc
0 setgray
stroke

%%Trailer
end
%%EOF

```

A Lisp program to graph 2-D points in PostScript

This Lisp program is also at <http://cybertiggyr.com/gene/fps/graph1.lisp>.

```

;;; -*- Mode: Lisp -*-
;;;
;;; $Header: /home/gene/library/website/docsrc/fps/RCS/graph1.lisp,v 395.1 2008/04/20 17:25:46 gene Ex
;;;
;;; Copyright (c) 2006 Gene Michael Stover. All rights reserved.
;;;
;;; This program is free software; you can redistribute it and/or modify
;;; it under the terms of the GNU Lesser General Public License as
;;; published by the Free Software Foundation; either version 2 of the
;;; License, or (at your option) any later version.
;;;
;;; This program is distributed in the hope that it will be useful,
;;; but WITHOUT ANY WARRANTY; without even the implied warranty of
;;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

```

```

;;; GNU Lesser General Public License for more details.
;;;
;;; You should have received a copy of the GNU Lesser General Public
;;; License along with this program; if not, write to the Free Software
;;; Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301
;;; USA
;;;

(defpackage "COM.CYBERTIGGYR.GENE.FPS.GRAPH1"
  (:use "COMMON-LISP"))
(in-package "COM.CYBERTIGGYR.GENE.FPS.GRAPH1")

(export 'graph0)

(defstruct (point2 (:type list)) x y)

(defun load-points (pn)
  (with-open-file (strm pn)
    (do ((lst nil (cons x lst))
        (x (read strm nil strm) (read strm nil strm)))
        ((eq x strm) lst))))

(defun min-x (points)
  "Return the minimum X value for the graph."
  (min 0 (reduce #'min points :key #'point2-x)))

(defun max-x (points)
  "Return the maximum X value for the graph."
  (max 0 (reduce #'max points :key #'point2-x)))

(defun min-y (points)
  "Return the minimum X value for the graph."
  (min 0 (reduce #'min points :key #'point2-y)))

(defun max-y (points)
  "Return the maximum X value for the graph."
  (max 0 (reduce #'max points :key #'point2-y)))

(defun inch (n)
  "Given a number of inches, return the number of PostScript
points. There are 72 PostScript points in an inch."
  (* 72 n))

(defun mm (n)
  "Given millimeters, return PostScript points. There are 72
PostScript points in an inch. There are 51/144 PostScript
points in a millimeter."
  (inch (/ n 25.5)))

(defun cm (n)

```

```

"Given a number of centimeters, return the number of
PostScript points."
(inch (/ n 2.55)))

(defun kBoxLowX ()
  "Return the coordinate of the left side of the image
area in PostScript user space points."
  72)

(defun kBoxUppX (width)
  "Return the location of the right edge of the image in
PostScript user space points. WIDTH is centimeters."
  ;; To get the right edge, we start with the left edge
  ;; and add the width. Remember to convert the width
  ;; from centimeters to PostScript user space points.
  (first
   (multiple-value-list
    (round (+ (kBoxLowX) (cm width))))))

(defun kBoxUppY ()
  "Return the coordinate of the upper side of the image
area in PostScript user space points."
  ;; We assume the paper is 11 inches tall, & we leave
  ;; a 1-inch border between the top of the paper &
  ;; the top of the image.
  (inch 10))

(defun kBoxLowY (height)
  "Return the coordinates of the bottom side of the image
area in PostScript user space points. HEIGHT is in
centimeters."
  (first
   (multiple-value-list
    (round (- (kBoxUppY) (cm height))))))

(defvar *border-around-graph* 4
  "Border between graphing area & image area, in millimeters")

(defun ps-number (n)
  "Give this function a number & it gives you a string which
you print into a PostScript program. The string you get will
always encode an integer or a real number, whichever is appropriate.
Even if N is a ratio (which Lisp can do but PostScript can't),
you will get an encoded real number."
  (cond ((integerp n) (format nil "~D" n))
        ;; This next case is an attempt to recognize a non-integer
        ;; which can be treated as an integer. I think this will
        ;; always work, but if it doesn't, no harm done because it
        ;; won't hurt to treat the number as a real.
        ((equalp n (round n)) (format nil "~D" n))
        ))

```

```

;; Finally, we don't have an integer, & we don't have a
;; ratio or real which happens to have an integral value.
;; So we bite the bullet & encode a real number.
(t (format nil "~,10E" n))))

```

```

(defun print-library0 (strm)
  "To STRM, print a library of PostScript functions. This is
  Library 0, the lowest level library, because it does not depend
  on any of the parameters or arguments which are specific to
  drawing the graph."
  (format strm "~%   %")
  (format strm "~%   % Given inches, return points.")
  (format strm "~%   % Stack: inches => points")
  (format strm "~%   %")
  (format strm "~%   /inch {")
  (format strm "~%           72 mul")
  (format strm "~%   } def")
  (format strm "~%")
  (format strm "~%   %")
  (format strm "~%   % Given centimeters, return points.")
  (format strm "~%   % There are 480/17 points in a centimeter.")
  (format strm "~%   % I multiply, then divide, by integers in the")
  (format strm "~%   % hope of taking advantage of the rare case")
  (format strm "~%   % in which we have an integral number of")
  (format strm "~%   % points.")
  (format strm "~%   %")
  (format strm "~%   /cm {")
  (format strm "~%           480 mul 17 div")
  (format strm "~%   } def")
  (format strm "~%")
  (format strm "~%   %")
  (format strm "~%   % Given millimeters, return points.")
  (format strm "~%   % There are 48/17 points in a millimeter.")
  (format strm "~%   %")
  (format strm "~%   /mm {")
  (format strm "~%           48 mul 17 div")
  (format strm "~%   } def")
  (format strm "~%")
  (format strm "~%   %")
  (format strm "~%   % Make a box path. Creates a new path for it.")
  (format strm "~%   % Does not stroke or fill it.")
  (format strm "~%   % Stack: min-x min-y max-x max-y => -")
  (format strm "~%   %")
  (format strm "~%   /newpath-box {")
  (format strm "~%           4 dict begin")
  (format strm "~%               /max-y exch def")
  (format strm "~%               /max-x exch def")
  (format strm "~%               /min-y exch def")
  (format strm "~%               /min-x exch def")
  (format strm "~%   }")

```

```

(format strm "~%          newpath")
(format strm "~%          min-x min-y moveto")
(format strm "~%          max-x min-y lineto")
(format strm "~%          max-x max-y lineto")
(format strm "~%          min-x max-y lineto")
(format strm "~%          closepath")
(format strm "~%          end")
(format strm "~%      } def")
strm)

```

```
(defun print-parameters (strm)
```

"Print parameter for the graph. The parameters rarely change. In fact, this implementation of the function outputs a hard-coded string. So these parameters never change unless I change this function."

```

(format strm "~%      %")
(format strm "~%      % Paper's height in PostScript points. We assume the")
(format strm "~%      % page is 11 inches high; that's common in the")
(format strm "~%      % United States.")
(format strm "~%      % The units are PostScript points.")
(format strm "~%      %")
(format strm "~%      /PageHeight 11 inch def")
(format strm "~%")
(format strm "~%      %")
(format strm "~%      % Width of the margin between the edge of the paper &")
(format strm "~%      % the edge of the image. Any edge.")
(format strm "~%      % The unit is PostScript points.")
(format strm "~%      %")
(format strm "~%      /Margin 1 inch def")
(format strm "~%")
(format strm "~%      %")
(format strm "~%      % I draw a box around the entire image. This is the")
(format strm "~%      % thickness of the line which forms the box.")
(format strm "~%      % The unit is PostScript points.")
(format strm "~%      %")
(format strm "~%      /BoxLineWidth 1 def")
(format strm "~%")
(format strm "~%      %")
(format strm "~%      % Width of the border between the image's edge & the")
(format strm "~%      % graphing area's AXIS edges. An \"axis edge\" is an")
(format strm "~%      % edge which has an axis. The left edge is an axis")
(format strm "~%      % edge. The bottom edge is an axis edge.")
(format strm "~%      % The unit is PostScript points.")
(format strm "~%      %")
(format strm "~%      /BorderAxis 6 mm def")
(format strm "~%")
(format strm "~%      %")
(format strm "~%      % Width of the border between the image's edge & the")
(format strm "~%      % graphing area's NON-axis edges. An \"axis edge\" is")
(format strm "~%      % an edge which has an axis. The top edge is a non-")

```

```

(format strm "% axis edge. The right edge is a non-axis edge.")
(format strm "% The unit is PostScript points.")
(format strm "%")
(format strm "% /BorderNonAxis 4 mm def")
(format strm "%")
(format strm "% The line width of the axes.")
(format strm "% The unit is PostScript points.")
(format strm "%")
(format strm "% /AxisLineWidth 2 def")
(format strm "%")
(format strm "% Number of tick marks on an axis.")
(format strm "% The unit is PostScript points.")
(format strm "%")
(format strm "% /TicksPerAxis 5 def")
(format strm "%")
(format strm "% ONE HALF of the length of a tick mark on an axis.")
(format strm "% So why is it HALF of the tick's length instead of")
(format strm "% the length? I originally used the tick length, but")
(format strm "% every time I used it, I had to divide it by two.")
(format strm "% So we save a few cycles & have a little less code")
(format strm "% just by redefining this constant from TickLength to")
(format strm "% TickHalfLength.")
(format strm "% The unit is PostScript points.")
(format strm "%")
(format strm "% /TickHalfLength 1 mm def")
(format strm "%")
(format strm "% The width of the line forming an axis.")
(format strm "% The unit is PostScript points.")
(format strm "%")
(format strm "% /TickLineWidth 1 def")
(format strm "%")
(format strm "% The width of the lines connecting the dots.")
(format strm "% The unit is PostScript points.")
(format strm "%")
(format strm "% /DotLineWidth 1 def")
(format strm "%")
(format strm "% The radius of the dots.")
(format strm "% The unit is PostScript points.")
(format strm "%")
(format strm "% /DotRadius 2 def")
(format strm "%")
(format strm "% The size of the font for labels.")
(format strm "%")

```

```

(format strm "~% /LabelFontSize 10 def")
(format strm "~%")
(format strm "~% %")
(format strm "~% % The font for the labels.")
(format strm "~% %")
(format strm "~% /LabelFont")
(format strm "~% /Helvetica findfont")
(format strm "~% LabelFontSize scalefont")
(format strm "~% def")
strm)

(defun print-arguments (lst height width min-x min-y max-x max-y strm)
  "Print the PostScript definitions of the graph's arguments.
The arguments probably change from one graph to another, so
they are not like the parameters, which rarely change."
  (format strm "~% %")
  (format strm "~% % The width of the image.")
  (format strm "~% %")
  (format strm "~% /ImageWidth ~A cm def" (ps-number width))
  (format strm "~%")
  (format strm "~% %")
  (format strm "~% % The height of the image.")
  (format strm "~% %")
  (format strm "~% /ImageHeight ~A cm def" (ps-number height))
  (format strm "~%")
  (format strm "~% %")
  (format strm "~% % The list of data points. It is sorted from lowest X")
  (format strm "~% % to highest X. It assumes that each value of X is")
  (format strm "~% % unique. It's a list (array). There is one element")
  (format strm "~% % for each data point. Each data point element has")
  (format strm "~% % exactly two elements: X & Y, in that order. The X")
  (format strm "~% % is at index 0; the Y is at index 1.")
  (format strm "~% %")
  (format strm "~% /Data [ ~{[ ~A ~A ]~~& ~} ] def"
    (mapcar #'ps-number
      (reduce #'append
        (sort (copy-list lst) #'< :key #'point2-x))))
  (format strm "~%")
  (format strm "~% %")
  (format strm "~% % The lowest value of the X axis.")
  (format strm "~% %")
  (format strm "~% /DataMinX ~A def" (ps-number min-x))
  (format strm "~%")
  (format strm "~% %")
  (format strm "~% % The highest value of the X axis.")
  (format strm "~% %")
  (format strm "~% /DataMaxX ~A def" (ps-number max-x))
  (format strm "~%")
  (format strm "~% %")
  (format strm "~% % The lowest value of the Y axis.")

```

```

(format strm "~%  %")
(format strm "~%  /DataMinY ~A def" (ps-number min-y))
(format strm "~%")
(format strm "~%  %")
(format strm "~%  % The highest value of the Y axis.")
(format strm "~%  %")
(format strm "~%  /DataMaxY ~A def" (ps-number max-y))
strm)

(defun print-library1 (strm)
  "Print a library of PostScript programs like I did with
PRINT-LIBRARY0. Unlike Library 0, this library depends on
parameters & arguments for the graph."
  (format strm "~%  %")
  (format strm "~%  % The left edge of the image. It's easy to compute.")
  (format strm "~%  % It's just the width of the Margin.")
  (format strm "~%  %")
  (format strm "~%  /ImageMinX Margin def")

  (format strm "~%  %")
  (format strm "~%  % The right edge of the image. It is Width cm from")
  (format strm "~%  % the left edge.")
  (format strm "~%  %")
  (format strm "~%  /ImageMaxX ImageMinX ImageWidth add def")

  (format strm "~%  %")
  (format strm "~%  % The upper edge of the image. We figure this before")
  (format strm "~%  % the lower limit on Y because we want to place the")
  (format strm "~%  % image at the top of the page. Since this PostScript")
  (format strm "~%  % program is encapsulated, the actual position on the")
  (format strm "~%  % page is meaningless when we generate the production")
  (format strm "~%  % document, but it's easier for on-screen debugging if")
  (format strm "~%  % the image is higher on the page.")
  (format strm "~%  %")
  (format strm "~%  /ImageMaxY PageHeight Margin sub def")

  (format strm "~%  %")
  (format strm "~%  % The lower edge of the image. It is the upper edge")
  (format strm "~%  % less the height.")
  (format strm "~%  %")
  (format strm "~%  /ImageMinY ImageMaxY ImageHeight sub def")

  (format strm "~%  %")
  (format strm "~%  % The left-most edge of the graphing area on the")
  (format strm "~%  % paper. It is Border points to the right of the")
  (format strm "~%  % left-most edge of the image area. The graphing")
  (format strm "~%  % area is contained within the image area.")
  (format strm "~%  %")
  (format strm "~%  /GraphMinX ImageMinX BorderAxis add def")
  (format strm "~%  %")

```

```

(format strm "% % The right-most edge of the graphing area on the")
(format strm "% % paper. It is Border points to the left of the")
(format strm "% % right-most edge of the image area. The graphing")
(format strm "% % area is contained within the image area.")
(format strm "% %")
(format strm "% % /GraphMaxX ImageMaxX BorderNonAxis sub def")
(format strm "% %")
(format strm "% % The top edge of the graphing area on the")
(format strm "% % paper. It is Border points below the")
(format strm "% % top edge of the image area. The graphing")
(format strm "% % area is contained within the image area.")
(format strm "% %")
(format strm "% % /GraphMaxY ImageMaxY BorderNonAxis sub def")
(format strm "% %")
(format strm "% % The bottom edge of the graphing area on the")
(format strm "% % paper. It is Border points above the")
(format strm "% % bottom edge of the image area. The graphing")
(format strm "% % area is contained within the image area.")
(format strm "% %")
(format strm "% % /GraphMinY ImageMinY BorderAxis add def")
(format strm "% %")
(format strm "% %")
(format strm "% % Draw both axes.")
(format strm "% % Creates its own, new path. Strokes that path.")
(format strm "% % Uses some of the parameters to determine the")
(format strm "% % thickness of the axis, the thickness of the ticks,")
(format strm "% % & the length of the ticks.")
(format strm "% % Does not save the graphics state. (Should it save")
(format strm "% % the graphics state?)"")
(format strm "% % Stack: min-x max-x min-y max-y => -")
(format strm "% %")
(format strm "% % /draw-axis {"")
(format strm "% %     15 dict begin")
(format strm "% %         /max-y exch def")
(format strm "% %         /min-y exch def")
(format strm "% %         /max-x exch def")
(format strm "% %         /min-x exch def")
(format strm "% %     newpath")
(format strm "% %         % Draw the two lines for the axes. The X")
(format strm "% %         % axis has one line, & the Y axis has")
(format strm "% %         % another. These lines get their own path,"")
(format strm "% %         % separate from that of the tick marks,"")
(format strm "% %         % because their line width color could")
(format strm "% %         % differ from those of the tick marks.")
(format strm "% %         min-x max-y moveto")
(format strm "% %         min-x min-y lineto")
(format strm "% %         max-x min-y lineto")
(format strm "% %         AxisLineWidth setlinewidth")
(format strm "% %         0 setgray")

```



```

(format strm "~%      end")
(format strm "~%    } def")
(format strm "~%")
(format strm "~%    %")
(format strm "~%    % Given a Data point, return its coordinates in")
(format strm "~%    % PostScript usre space.")
(format strm "~%    % The Data point is specified as a two-element")
(format strm "~%    % array.")
(format strm "~%    % Stack: [x y] => -")
(format strm "~%    %")
(format strm "~%    /xlate-data-to-user {"")
(format strm "~%        aload pop % stack: x y")
(format strm "~%        % Convert the Y value.")
(format strm "~%        DataMinY DataMaxY GraphMinY GraphMaxY")
(format strm "~%        xlate-data-to-user-1")
(format strm "~%        % Convert the X value.")
(format strm "~%        exch          % stack: y x")
(format strm "~%        DataMinX DataMaxX GraphMinX GraphMaxX")
(format strm "~%        xlate-data-to-user-1")
(format strm "~%        exch          % stack: x y")
(format strm "~%    } def")
(format strm "~%")
(format strm "~%    %")
(format strm "~%    % Move the drawing location to the data point.")
(format strm "~%    % The data points are in their own coordinate space;")
(format strm "~%    % they are not in PostScript user space.")
(format strm "~%    % The Data point is specified as a two-element")
(format strm "~%    % array.")
(format strm "~%    % Stack: [x y] => -")
(format strm "~%    %")
(format strm "~%    /moveto-data {"")
(format strm "~%        xlate-data-to-user moveto")
(format strm "~%    } def");
(format strm "~%")
(format strm "~%    %")
(format strm "~%    % Given a Data point, draw it.")
(format strm "~%    % Stack: [x y] => -")
(format strm "~%    %")
(format strm "~%    /draw-data {"")
(format strm "~%        newpath")
(format strm "~%        xlate-data-to-user DotRadius 0 360 arc")
(format strm "~%        0 setgray")
(format strm "~%        fill")
(format strm "~%    } def")
(format strm "~%")
(format strm "~%    %")
(format strm "~%    % Draw the curve between the data points.")
(format strm "~%    % Stack: [ [x0 y0] [x1 y1] ... [xn yn] ] => -")
(format strm "~%    %")
(format strm "~%    /draw-data-curve {"")

```

```

(format strm "%           newpath")
(format strm "%           % Move the drawing location to the first")
(format strm "%           % point.")
(format strm "%           dup 0 get xlate-data-to-user moveto")
(format strm "%           % Draw the line segments.")
(format strm "%           dup { xlate-data-to-user lineto } forall")
(format strm "%           DotLineWidth setlinewidth")
(format strm "%           0.5 setgray")
(format strm "%           stroke")
(format strm "%       } def")
strm)

(defun print-fixed (strm)
  "Print the unchanging part of the PostScript program.  It
  doesn't change because it uses the parameters & arguments
  which we've already output.
  I could also have called this function PRINT-MAIN."
  (format strm "%           %")
  (format strm "%           % ~A begins" 'print-fixed)
  (format strm "%           %")

  (format strm "%           % Draw a box around the entire image")
  (format strm "%           ImageMinX ImageMinY ImageMaxX ImageMaxY newpath-box")
  (format strm "%           BoxLineWidth setlinewidth")
  (format strm "%           0 setgray")
  (format strm "%           stroke")

  (format strm "%           % Draw both axes.")
  (format strm "%           GraphMinX GraphMaxX GraphMinY GraphMaxY draw-axis")

  (format strm "%           %")
  (format strm "%           %% Draw the labels on the X axis.")
  (format strm "%           %")
  (format strm "%           % Label X at the origin.")
  (format strm "%           DataMinX 10 string cvs")
  (format strm "%           LabelFont setfont")
  (format strm "%           dup stringwidth")
  (format strm "%           /wy exch def      % string's height")
  (format strm "%           /wx exch def      % string's width")
  (format strm "%           GraphMinX GraphMinY LabelFontSize sub 2 sub moveto")
  (format strm "%           show")
  (format strm "%           % Label X at its highest value.")
  (format strm "%           DataMaxX 10 string cvs")
  (format strm "%           LabelFont setfont")
  (format strm "%           dup stringwidth")
  (format strm "%           /wy exch def      % string's height")
  (format strm "%           /wx exch def      % string's width")
  (format strm "%           GraphMaxX wx sub GraphMinY LabelFontSize sub 2 sub")
  (format strm "%           moveto")
  (format strm "%           show")

```

```

(format strm "~%      %")
(format strm "~%      %% Draw the labels on the Y axis.")
(format strm "~%      %")
(format strm "~%      % Label Y at the origin.")
(format strm "~%      DataMinY 10 string cvs")
(format strm "~%      LabelFont setfont")
(format strm "~%      dup stringwidth")
(format strm "~%      /wy exch def      % string's height, ignored")
(format strm "~%      /wx exch def      % string's width")
(format strm "~%      GraphMinX wx sub 3 sub GraphMinY moveto")
(format strm "~%      show")
(format strm "~%      % Label Y at its highest value.")
(format strm "~%      DataMaxY 10 string cvs")
(format strm "~%      LabelFont setfont")
(format strm "~%      dup stringwidth")
(format strm "~%      /wy exch def      % string's height, ignored")
(format strm "~%      /wx exch def      % string's width")
(format strm "~%      GraphMinX wx sub 3 sub GraphMaxY LabelFontSize sub")
(format strm "~%      moveto")
(format strm "~%      show")

(format strm "~%      Data draw-data-curve")
(format strm "~%      Data { dup draw-data } forall")

(format strm "~%      % ~A ends" 'print-fixed)
strm)

(defvar *default-width* 10
  "Width, in centimeters.  By default, it is 16 centimeters which is
  about the width of the printable area of a 8.5 inch by 11 inch piece
  of paper, which is a common paper size in the United States.")

(defvar *default-height* (/ (* 2 *default-width*) 3)
  "Height, in centimeters.  By default, it is 2/3 of the width.")

(defun graph0 (lst pathname &key (height *default-height*)
              (width *default-width*)
              (min-x (min-x lst)) (min-y (min-y lst))
              (max-x (max-x lst)) (max-y (max-y lst)))
  (with-open-file (strm pathname :direction :output :if-exists :rename)
    (format strm "%!PS-Adobe-3.0 EPSF-3.0")
    (format strm "%%%Creator: Gene Michael Stover")
    (format strm "%%%Title: ~A" (namestring pathname))
    (multiple-value-bind
      (ss mm hh dd mo yy) (decode-universal-time
                           (get-universal-time))
      (format strm "%%%CreationDate: ~D~2,'OD~2,'ODT~2,'OD:~2,'OD:~2,'OD"
                yy mo dd hh mm ss))
    (format strm "%%%DocumentData: Clean7Bit")
  )
)

```

```

(format strm "%%%LanguageLevel: 1")
(format strm "%%%Pages: 1")
(format strm "%%%BoundingBox: ~D ~D ~D ~D"
      (kBoxLowX) (kBoxLowY height)
      (kBoxUppX width) (kBoxUppY))
(format strm "%%%EndComments")
(format strm "%%%BeginProlog")
(format strm "%100 dict begin")
(print-library0 strm)
(print-parameters strm)
(print-arguments lst height width min-x min-y max-x max-y strm)
(print-library1 strm)
(format strm "%%%EndProlog")
(format strm "%%%Page: 1 1")
(print-fixed strm)
(format strm "%   % Is \"showpage\" necessary in an Encapsulated")
(format strm "%   % PostScript program?")
(format strm "%   % showpage")
(format strm "%%%Trailer")
(format strm "%end")
(format strm "%%%EOF")
(format strm "%")
(truename strm))

(defun demo0000 ()
  (graph0 '((10 9) (8 7) (6 5) (4 3)) "demo0000.eps"))

(defun demo0001 ()
  (graph0 (mapcar #'(lambda (x) (list x (expt x 3)))
                '(-1 -1/2 -1/3 -1/7 0 1/33 1/7 1/5 1/3 1/2 1))
          "demo0001.eps"))

;;; --- end of file ---

```

B Example 2-D graphing PostScript program

This is the PostScript source code which produced Figure 4. This source code is also at <http://cybertiggyr.com/gene/fps/demo0001.eps>.

```

%!PS-Adobe-3.0 EPSF-3.0
%%Creator: Gene Michael Stover
%%Title: demo0001.eps

%%CreationDate: 2006-05-19T22:46:02
%%DocumentData: Clean7Bit
%%LanguageLevel: 1
%%Pages: 1
%%BoundingBox: 72 532 354 720
%%EndComments

```

```

%%BeginProlog
100 dict begin
  %
  % Given inches, return points.
  % Stack: inches => points
  %
  /inch {
    72 mul
  } def

  %
  % Given centimeters, return points.
  % There are 480/17 points in a centimeter.
  % I multiply, then divide, by integers in the
  % hope of taking advantage of the rare case
  % in which we have an integral number of
  % points.
  %
  /cm {
    480 mul 17 div
  } def

  %
  % Given millimeters, return points.
  % There are 48/17 points in a millimeter.
  %
  /mm {
    48 mul 17 div
  } def

  %
  % Make a box path.  Creates a new path for it.
  % Does not stroke or fill it.
  % Stack: min-x min-y max-x max-y => -
  %
  /newpath-box {
    4 dict begin
      /max-y exch def
      /max-x exch def
      /min-y exch def
      /min-x exch def

      newpath
      min-x min-y moveto
      max-x min-y lineto
      max-x max-y lineto
      min-x max-y lineto
      closepath
    end
  } def

```

```

%
% Paper's height in PostScript points. We assume the
% page is 11 inches high; that's common in the
% United States.
% The units are PostScript points.
%
/PageHeight 11 inch def

%
% Width of the margin between the edge of the paper &
% the edge of the image. Any edge.
% The unit is PostScript points.
%
/Margin 1 inch def

%
% I draw a box around the entire image. This is the
% thickness of the line which forms the box.
% The unit is PostScript points.
%
/BoxLineWidth 1 def

%
% Width of the border between the image's edge & the
% graphing area's AXIS edges. An "axis edge" is an
% edge which has an axis. The left edge is an axis
% edge. The bottom edge is an axis edge.
% The unit is PostScript points.
%
/BorderAxis 6 mm def

%
% Width of the border between the image's edge & the
% graphing area's NON-axis edges. An "axis edge" is
% an edge which has an axis. The top edge is a non-
% axis edge. The right edge is a non-axis edge.
% The unit is PostScript points.
%
/BorderNonAxis 4 mm def

%
% The line width of the axes.
% The unit is PostScript points.
%
/AxisLineWidth 2 def

%
% Number of tick marks on an axis.
% The unit is PostScript points.
%

```

```

/TicksPerAxis 5 def

%
% ONE HALF of the length of a tick mark on an axis.
% So why is it HALF of the tick's length instead of
% the length? I originally used the tick length, but
% every time I used it, I had to divide it by two.
% So we save a few cycles & have a little less code
% just by redefining this constant from TickLength to
% TickHalfLength.
% The unit is PostScript points.
%
/TickHalfLength 1 mm def

%
% The width of the line forming an axis.
% The unit is PostScript points.
%
/TickLineWidth 1 def

%
% The width of the lines connecting the dots.
% The unit is PostScript points.
%
/DotLineWidth 1 def

%
% The radius of the dots.
% The unit is PostScript points.
%
/DotRadius 2 def

%
% The size of the font for labels.
%
/LabelFontSize 10 def

%
% The font for the labels.
%
/LabelFont
  /Helvetica findfont
  LabelFontSize scalefont
def

%
% The width of the image.
%
/ImageWidth 10 cm def

%

```

```

% The height of the image.
%
/ImageHeight 6.6666660000e+0 cm def

%
% The list of data points. It is sorted from lowest X
% to highest X. It assumes that each value of X is
% unique. It's a list (array). There is one element
% for each data point. Each data point element has
% exactly two elements: X & Y, in that order. The X
% is at index 0; the Y is at index 1.
%
/Data [ [ -1 -1 ]
        [ -5.0000000000e-1 -1.2500000000e-1 ]
        [ -3.3333334000e-1 -3.7037040000e-2 ]
        [ -1.4285715000e-1 -2.9154520000e-3 ]
        [ 0 0 ]
        [ 3.0303030000e-2 2.7826473000e-5 ]
        [ 1.4285715000e-1 2.9154520000e-3 ]
        [ 2.0000000000e-1 8.0000000000e-3 ]
        [ 3.3333334000e-1 3.7037040000e-2 ]
        [ 5.0000000000e-1 1.2500000000e-1 ]
        [ 1 1 ] ] def

%
% The lowest value of the X axis.
%
/DataMinX -1 def

%
% The highest value of the X axis.
%
/DataMaxX 1 def

%
% The lowest value of the Y axis.
%
/DataMinY -1 def

%
% The highest value of the Y axis.
%
/DataMaxY 1 def

%
% The left edge of the image. It's easy to compute.
% It's just the width of the Margin.
%
/ImageMinX Margin def

%
% The right edge of the image. It is Width cm from

```

```

% the left edge.
%
/ImageMaxX ImageMinX ImageWidth add def
%
% The upper edge of the image. We figure this before
% the lower limit on Y because we want to place the
% image at the top of the page. Since this PostScript
% program is encapsulated, the actual position on the
% page is meaningless when we generate the production
% document, but it's easier for on-screen debugging if
% the image is higher on the page.
%
/ImageMaxY PageHeight Margin sub def
%
% The lower edge of the image. It is the upper edge
% less the height.
%
/ImageMinY ImageMaxY ImageHeight sub def
%
% The left-most edge of the graphing area on the
% paper. It is Border points to the right of the
% left-most edge of the image area. The graphing
% area is contained within the image area.
%
/GraphMinX ImageMinX BorderAxis add def
%
% The right-most edge of the graphing area on the
% paper. It is Border points to the left of the
% right-most edge of the image area. The graphing
% area is contained within the image area.
%
/GraphMaxX ImageMaxX BorderNonAxis sub def
%
% The top edge of the graphing area on the
% paper. It is Border points below the
% top edge of the image area. The graphing
% area is contained within the image area.
%
/GraphMaxY ImageMaxY BorderNonAxis sub def
%
% The bottom edge of the graphing area on the
% paper. It is Border points above the
% bottom edge of the image area. The graphing
% area is contained within the image area.
%
/GraphMinY ImageMinY BorderAxis add def

%
% Draw both axes.
% Creates its own, new path. Strokes that path.

```

```

% Uses some of the parameters to determine the
% thickness of the axis, the thickness of the ticks,
% & the length of the ticks.
% Does not save the graphics state. (Should it save
% the graphics state?)
% Stack: min-x max-x min-y max-y => -
%
/draw-axis {
  15 dict begin
    /max-y exch def
    /min-y exch def
    /max-x exch def
    /min-x exch def

    newpath
      % Draw the two lines for the axes. The X
      % axis has one line, & the Y axis has
      % another. These lines get their own path,
      % separate from that of the tick marks,
      % because their line width color could
      % differ from those of the tick marks.
      min-x max-y moveto
      min-x min-y lineto
      max-x min-y lineto
      AxisLineWidth setlinewidth
    0 setgray
    stroke

    % Draw the tick marks on both axes. There
    % are TicksPerAxis of them on each axis.
    /x min-x def
    /delta-x max-x min-x sub TicksPerAxis div def
    /y min-y def
    /delta-y max-y min-y sub TicksPerAxis div def
    newpath
    0 1 TicksPerAxis {
      % Draw the tick mark on the X axis.
      x min-y TickHalfLength add moveto
      x min-y TickHalfLength sub lineto
      % Draw the tick mark on the Y axis.
      min-x TickHalfLength sub y moveto
      min-x TickHalfLength add y lineto
      % Increment X & Y for the next time.
      /x x delta-x add def
      /y y delta-y add def
    } for
    TickLineWidth setlinewidth
    0 setgray
    stroke
  end
end

```

```

} def

%
% Given a coordinate in Data space, the range for
% Data space, & the range for Graph space, return
% the coordinate in Graph space.
% Stack: Value DataMin DataMax GraphMin GraphMax =>
%       Graph
%
/xlate-data-to-user-1 {
  10 dict begin
    /GraphMax exch def
    /GraphMin exch def
    /DataMax exch def
    /DataMin exch def
    /Value exch def

    % Get relative location in Data space.
    Value DataMin sub
    DataMax DataMin sub
    div
    % Convert to offset within Graph space.
    GraphMax GraphMin sub mul
    % Add GraphMin to convert it to an actual
    % in Graph space.
    GraphMin add
  end
} def

%
% Given a Data point, return its coordinates in
% PostScript usre space.
% The Data point is specified as a two-element
% array.
% Stack: [x y] => -
%
/xlate-data-to-user {
  aload pop % stack: x y
  % Convert the Y value.
  DataMinY DataMaxY GraphMinY GraphMaxY
  xlate-data-to-user-1
  % Convert the X value.
  exch      % stack: y x
  DataMinX DataMaxX GraphMinX GraphMaxX
  xlate-data-to-user-1
  exch      % stack: x y
} def

%
% Move the drawing location to the data point.

```

```

% The data points are in their own coordinate space;
% they are not in PostScript user space.
% The Data point is specified as a two-element
% array.
% Stack: [x y] => -
%
/moveto-data {
    xlate-data-to-user moveto
} def

%
% Given a Data point, draw it.
% Stack: [x y] => -
%
/draw-data {
    newpath
        xlate-data-to-user DotRadius 0 360 arc
    0 setgray
    fill
} def

%
% Draw the curve between the data points.
% Stack: [ [x0 y0] [x1 y1] ... [xn yn] ] => -
%
/draw-data-curve {
    newpath
        % Move the drawing location to the first
        % point.
        dup 0 get xlate-data-to-user moveto
        % Draw the line segments.
        dup { xlate-data-to-user lineto } forall
    DotLineWidth setlinewidth
    0.5 setgray
    stroke
} def
%%EndProlog
%%Page: 1 1
%
% PRINT-FIXED begins
%
% Draw a box around the entire image
ImageMinX ImageMinY ImageMaxX ImageMaxY newpath-box
BoxLineWidth setlinewidth
0 setgray
stroke
% Draw both axes.
GraphMinX GraphMaxX GraphMinY GraphMaxY draw-axis
%%
%% Draw the labels on the X axis.

```

```

%%
% Label X at the origin.
DataMinX 10 string cvs
LabelFont setfont
dup stringwidth
/wy exch def      % string's height
/wx exch def      % string's width
GraphMinX GraphMinY LabelFontSize sub 2 sub moveto
show
% Label X at its highest value.
DataMaxX 10 string cvs
LabelFont setfont
dup stringwidth
/wy exch def      % string's height
/wx exch def      % string's width
GraphMaxX wx sub GraphMinY LabelFontSize sub 2 sub
moveto
show
%%
%% Draw the labels on the Y axis.
%%
% Label Y at the origin.
DataMinY 10 string cvs
LabelFont setfont
dup stringwidth
/wy exch def      % string's height, ignored
/wx exch def      % string's width
GraphMinX wx sub 3 sub GraphMinY moveto
show
% Label Y at its highest value.
DataMaxY 10 string cvs
LabelFont setfont
dup stringwidth
/wy exch def      % string's height, ignored
/wx exch def      % string's width
GraphMinX wx sub 3 sub GraphMaxY LabelFontSize sub
moveto
show
Data draw-data-curve
Data { dup draw-data } forall
% PRINT-FIXED ends
% Is "showpage" necessary in an Encapsulated
% PostScript program?
% showpage
%%Trailer
end
%%EOF

```

C Other File Formats

- This document is available in multi-file HTML format at <http://cybertiggyr.com/gene/fps/>.
- This document is available in Pointless Document Format (PDF) at <http://cybertiggyr.com/gene/fps/fps>.

I write almost all of my documents in \LaTeX ([7], [3]). I compile to PDF with `latex`, `dvips`, & `ps2pdf`. I compile to HTML with `latex2html` ([2], [4]).

References

- [1] How to use adobe PostScript language files properly. <http://www-cdf.fnal.gov/offline/PostScript/AdobePS.html>.
- [2] Nikos Drakos. `latex2html`.
- [3] Michel Goossens and Frank Mittelbach. *The \LaTeX Companion*. Addison Wesley Longman, Inc., 1993. ISBN 0201541998.
- [4] Michel Goossens and Sebastian Rahtz. *The \LaTeX Web Companion: Integrating \TeX , HTML, and XML*. Addison Wesley Longman, Inc., 1999. ISBN 020143317.
- [5] Adobe Systems Incorporated, editor. *PostScript Language Tutorial and Cookbook*. Adobe Systems Incorporated, 1985. ISBN 0-201-10179-3.
- [6] Adobe Systems Incorporated, editor. *PostScript Language Reference*. Addison-Wesley, third edition, 1999. ISBN 0201379228.
- [7] Leslie Lamport. *\LaTeX : A Document Preparation System*. Addison-Wesley Publishing Company, Inc., 1986. ISBN 0-201-15790-X.