

HP-GL Plotter Simulator for the HP-50g

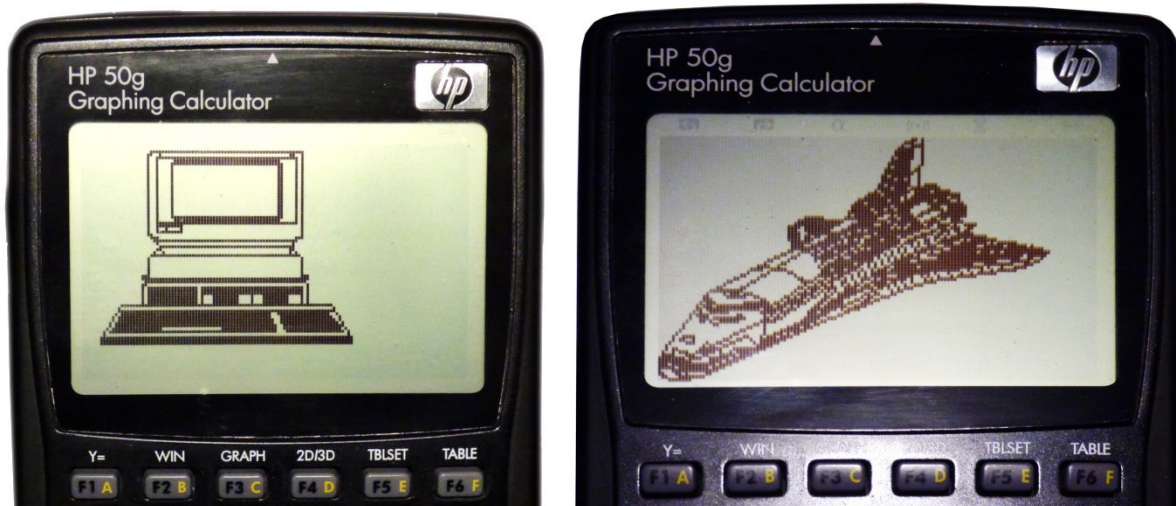
Martin Hepperle, 2024

This program represents a simple, limited subset HP-GL interpreter for the HP-50g and similar RPL calculators. The following commands mnemonics are understood: PA, PU, PD, SP. All other mnemonics are silently skipped. Files in HP-GL format can be exported by many commercial and free programs like Corel Draw, Inkscape and others. It might be necessary to modify these files with a text editor to match the requirements of this program.

The main routine is named “GLPLOT”. The required subroutines start with the lower case prefix “gl”.

A typical HP-GL file looks like this:

```
IN;SC;  
PU;SP1;LT;VS36;PA3917,4498;  
PD;PA3809,4405;  
PU;PA3692,4176;  
PU;PA3563,4800;  
PD;PA3459,4716,3436,4677,3428,4621;PA3564,4094;  
PU;PA3584,4075;  
...  
PU;PA1386,993;  
PD;PA1181,900,1182,902,1046,993,1233,1140,1318,1085,1386,993;  
PU;SP0;
```



Output of processing a vector graphic drawn in and exported from Corel Draw.

This is my second iteration on the theme. My first solution was only plotting the coordinate points without connecting lines, rather complex and not very robust. The proposed solution is a relatively simple, but rather robust proof of concept program.

It's execution speed can surely be increased, maybe in a first step by converting the core routines to SysRPL?

Main Routine “GLPLOT”

Purpose: Read a string of HP-GL commands and plot the graphics on the LCD screen.

Input: A HPGL string of arbitrary length on level 1 of the stack.

```
1: "IN;SC;PU;PA100,200;PD1100,200,1100,1200,100,1200;"
```

Subroutines: gLEVAL

Notes: The string must be composed of one or more *elements*.

Each *element* must start with a two-character HP-GL command mnemonic, followed by numeric arguments and a trailing semi-colon ‘;’ terminator. Multiple numeric arguments must be separated by a comma ‘,’ or a space ‘ ’ character.

The string can have line breaks (ASCII character codes 10 and 13). A single element cannot be longer than 14999 characters. An Element cannot be broken into lines.

Mnemonics immediately following each other like “PAPU100,100;” are not allowed. They must be separated into “PA;PU100,100;”. Such modifications can be accomplished easily with the search/replace function of a text editor.

The coordinate system is unscaled i.e. not considering “SC” or “IP” commands. Its origin should be placed in the lower left corner of the paper and it covers the rectangle (0,0)–(10900,7650) in plotter units, corresponding to the HP 7470.

However, it is possible to adjust the scaling by adapting the parameters given to the XRNG resp. YRNG functions. Alternatively, the “IP” and “SC” commands could be implemented, defining new XRNG and YRNG limits.

```
«
0 10900 XRNG
0 7650 YRNG
ERASE
0 PPAR 2 GET C→R SWAP DROP R→C PVIEW
(0.,0.) 'XYPT' STO
0 'PEN' STO
10 CHR ";" SREPL DROP
13 CHR ";" SREPL DROP
";;" ";" SREPL DROP
1 → I
«
  DUP SIZE
  1 SWAP
  FOR J
    DUP
    J J SUB
    ";" == IF
    THEN
      DUP I J 1 - SUB
      gLEVAL
      J 1 + 'I' STO
    END
  NEXT
  DROP
  1600 .3 BEEP {} PVIEW
»
'PEN' PURGE
'XYPT' PURGE
'PPAR' PURGE
»
```

```
define x range (plotter units, 7470, A4)
define y range (plotter units, 7470, A4)
clear PICT
show PICT with (0,7650) at upper left
define current point as global variable
define pen as global variable
replace any line feed characters
replace any carriage return characters
replace any doubled terminators
I = start index of current token
local variable I used only here
J = length of HPGL string
search string on stack for ';' character
FOR

  get character at index J
  IF this is a ';'
  THEN
    get substring[I:J-1] (without ';')
    evaluate this element
    new start index I=J+1 (behind ';')
  END
NEXT
drop HPGL string from stack
display PICT until Cancel/ON is pressed

remove temporary variables
```

Routine “gLEVAL”

Purpose: Parse one HPGL element and handle it.

Input: A HP-GL element on the stack, starting with a two-character mnemonics, followed by optional parameters, without the terminating semi-colon ‘;’:

```
1: "PA1100,200,1100,1200,100,1200,100,200"
```

Output: Nothing left on the stack.

Notes: Sets the ‘PEN’ variable as follows:

- 0, if a “PU” or “SP” command is found (pen is up).
- 1, if a “PD” command is found (pen is down).

Limitations: The input string must be shorter than 15000 characters and have less than 5000 numeric entries.

Subroutines: g1NUM, g1MOVE.

<pre>« DUP 1 2 SUB CASE DUP "PA" == THEN g1NUM g1MOVE END DUP "PD" == THEN 1 'PEN' STO g1NUM g1MOVE END DUP "PU" == THEN 0 'PEN' STO g1NUM g1MOVE END DUP "SP" == THEN 0 'PEN' STO g1NUM DROP END DROP2 END »</pre>	<p>extract the two-character mnemonics</p> <p>PA: move pen (may be up or down)</p> <p>PD: pen down, then move</p> <p>PU: pen up, then move</p> <p>SP: pen up, drop pen number</p> <p>else: drop string and mnemonics</p>
---	--

Note: one could move the call of the g1NUM subroutine out of the CASE construct, but I found the resulting code less clear and not really smaller.

Routine “gI NUM”

Purpose: Converts a string of HPGL mnemonics (starting with two characters, followed by optional numeric parameters) into a list of numeric values. If the HPGL command is not recognized or if no numeric arguments are given, the returned list is empty.

Input: Two entries on the stack:

```
2: "PA1100,200,1100,1200,100,1200,100,200"  
1: "PA"
```

Output: A list of parameters on level 1 of the stack:

```
1: {1100, 200, 1100, 1200, 100, 1200, 100, 200}
```

Notes: On input, the mnemonics identifier on level 1 is left by the caller and is silently dropped here, to avoid having to drop it in each branch of the caller’s gI EVAL CASE statement.

Limitations: The string must be shorter than 15000 characters and have less than 5000 numeric entries.

```
«  
  DROP  
  DUP SIZE  
  
  2 > IF  
  THEN  
    3 14999 SUB  
    "{" SWAP + "}" +  
    OBJ→  
  ELSE  
    DROP  
    {}  
  END  
»
```

drop the unused mnemonic string
get the length of the string

IF any parameters follow the mnemonic
THEN
 extract the parameter list
 convert to a list-like string
 convert to a list object
ELSE
 drop the string
 return an empty list
END

Implementation note: the OBJ→ function is very convenient as it accepts a comma or a space separated list-like string.

Routine “glMOVE”

Purpose: Split a list of numbers into coordinate pairs and move the pen to each point. If the pen is down ‘PEN’ == 1) a line is drawn from the previous to the new position. The current pen position is always updated.

Input: A list of numbers on level 1.

1: {1100, 200, 1100, 1200, 100, 1200, 100, 200}

Output: nothing on the stack, updated variable XYPT.

Notes: The list can be empty. In this case nothing happens.

Limitations: The input list must have less than 5000 entries.

```
«
  WHILE DUP SIZE 2 / IP
  REPEAT
    DUP
    3 4999 SUB
    SWAP
    1 2 SUB
    OBJ→ DROP
    R→C

    'PEN' RCL
    1 == IF
    THEN
      DUP 'XYPT' RCL LINE
    END

    'XYPT' STO
  END
  DROP
»
```

```
WHILE the list contains more pairs

  get list of elements from 3 to the end

  get list of the first two elements
  convert to two numbers and drop count
  convert to coordinate pair

  IF the pen is down
  THEN
    get previous point and draw line
  END

  update current point
END
drop repeat count (0 or 1)
```