# Long Live The HP-71B !

**Valentín Albillo (HPCC #1075)**

As much as I loved RPN, back in the early 80's, my first encounter with a handheld BASIC-programmable machine, the awesome **SHARP PC-1211**, left me deeply convinced that BASIC was a much easier and powerful environment to develop and run custom programs, as told in my previous Datafile article ***"Know Thy Foe"***. The *caveat*, however, was that the extensibility and capabilities of the flagship RPN machine, the **HP-41C**, certainly more than made up for its more primitive 'language' and so, in the end, it ultimately came up as the more powerful model, if not the one with the most convenient programming paradigm.

The end line, however, was clear. What I longed for was a BASIC-programmable handheld with an enhanced BASIC dialect (the **PC-1211**'s, while a brave first attempt, was certainly primitive), greater speed (the **PC-1211** was only as fast as the **HP-41C**, i.e., rather on the slow side), larger RAM (1.8 Kb for the **PC-1211**, about 7 Kb for an **HP-41CV** with two EMMs), and increased expandability (the **HP-41C** won this one hands down). Then the **SHARP PC-1500** came out and I gave it serious thought but, while interesting and certainly more powerful than the pioneer **PC-1211**, it still wasn't *exactly* what I wanted, something was still *missing*. Enter the **HP-71B**.

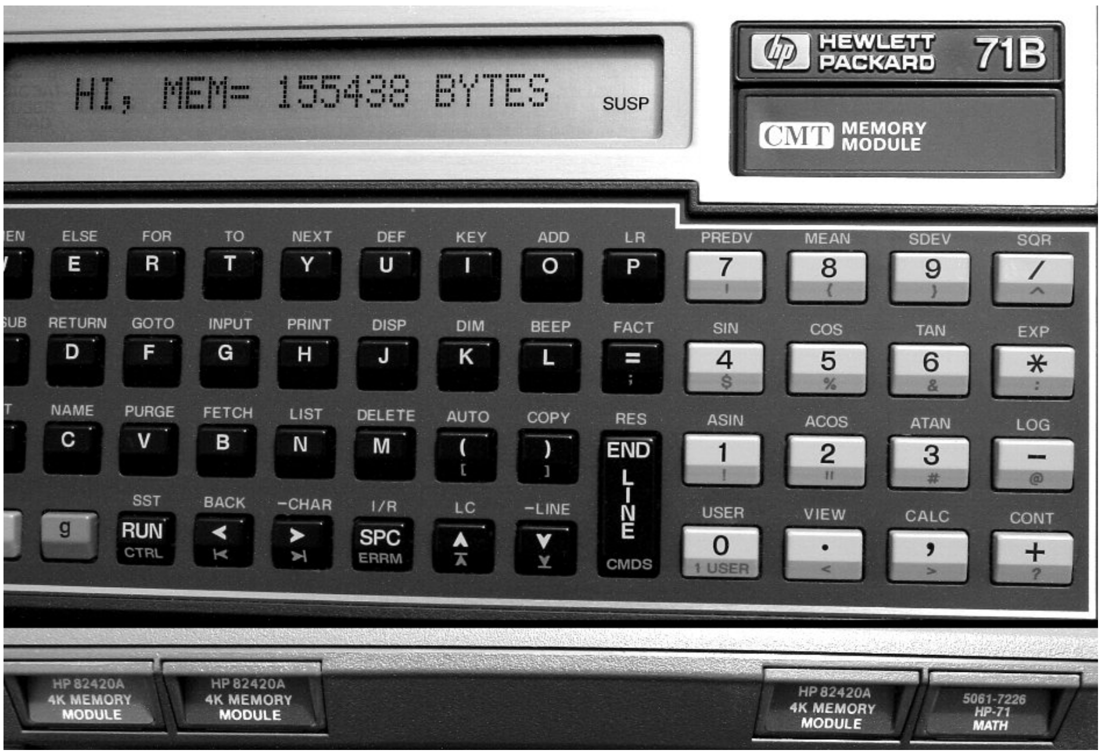
## A dream come true

It was February, 1984, and HP introduced yet another landmark model, the **HP-71B**. After hearing some rumours here and there, I was truly *itching* to lay my hands on one, and Pedro Casado, a friend of mine working at HP at the time, was kind enough to lend me one for just a weekend. Awesome. It was awesome, I was absolutely awed by it, by its feel, by its looks, by its speed, by its features. Everything was state-of-the-art quality, surpassing anything out there in spades, and letting the **SHARP PC-1500** in the dust. In fact, the mere thought of comparing them felt almost like utter blasphemy.

Regrettably, the same 'scaling factor' applied to its price. It was so *horrendously high* (something like 6-7 times the price of an **HP-41C**, even at the bare-bones configuration!!), that purchasing one was simply out of the question. I didn't have that kind of money, and even if I did, I could not commit it to purchase this machine however much I desired it. So, after returning the **HP-71B** to Pedro, I was left with the definite impression that it was curtains for my personal relationship with it. But, providentially, I was working at the time for an engineering firm which, as a byproduct, also developed architectural software exclusively for HP microcomputers such as the **HP-85/86/87**, etc. These were much more expensive machines than even the **HP-71B** was, but fitted with a Math ROM and extra RAM I found out that the **HP-71B** would be more or less capable of duplicating their performances in our fields of interest, with the added advantage of portability.

To make a long story short, I managed to convince my boss to get an **HP-71B** for me to try and see how could we port the programs to run on it, and lo and behold, after a few long weeks I was given a wonderful box with a brand-new, shiny **HP-71B** inside (plus card reader) for me and me alone ! I quickly went to work with it, learned its secrets, and quite soon I got also a Math ROM (this one paid for with *my very own money*) and the bulky 3-volume **IDS**, which I promptly (!?) devoured from first to last page. But what was so exciting about it ?. Let's briefly enumerate the (for me) most important and innovative aspects:

- Very small (**HP-75** anyone?), solid-as-a-brick, well built, nice looking machine, with a *dot-matrix* addressable display, a fantastic QWERTY keyboard, and a new, much faster CPU (the *very first*, 0.6 Mhz Saturn!).
- ***Tremendous ROM/RAM adressing capabilities***, up to nearly 512 Kb (!!), which was absolutely unheard of for such a small portable model, and even surpassed the amount of RAM typically available in the micros of the era, which was 16/32 Kb for the **HP-85**, about 128 Kb for the **HP-87XM**, and 64 Kb for most other models, **IBM PC** included. And of course, all RAM would be retained upon turn off and the operating system, being in ROM, would always be instantly available and use up very little of it.
- 4 built-in frontal expansion ports (see picture below) which would accept ROM and RAM modules in most any combination (you can see three 4K RAM modules plus the Math ROM pictured), plus an additional rear port for the HP-IL interface ROM (but which would accept any other RAM/ROM module if needed), plus and additional upper port for the card reader, which could also accept high-capacity third-party RAM/ROM/EPROM modules, such as the 128 Kb **CMT** RAM module pictured here.



- *Awesome expandability*, with a large number of plug-in ROMs available ( both **HP**-produced and third-party ones), which included such gems as the Math ROM (32 Kb of *truly state-of-the-art*, fully optimized assembly-language math routines in the convenient form of BASIC keywords), the Forth/Assembler ROM, and the HP-41C Translator/Forth ROM, not to mention the HP-IL ROM, which could handle dataflows much faster than the HP-41C equivalent, and was even able to control up to *three* independent IL loops at a time. Thanks to its extensive HP-IL capabilities, the **HP-71B** could control and use a vast number of peripherals, including various kinds of mass storage, printers, plotters, data acquisition devices, whatever.
- An incredibly powerful version of **HP**'s technical BASIC, inheriting many of the characteristics of its larger siblings' (such as the **HP-85/87**), but enhancing it to new heights with such revolutionary capabilities as a full RAM-based *filesystem*, independent *subprograms* with their own local environments and parameters passing by value or reference, *multi-line user-defined functions*, full support for *recursion*, programmable timers which could even *wake up* the turned-off machine and run programs at specified times, full **IEEE** compliancy with NaNs, infinities, and denormalization, and (with the Math ROM plugged in), real and complex-valued matrix operations, full support for complex variables and operations, *nested* Solve and Integrate, polynomial root-finding, and even Fast Fourier Transforms !
- *Open system concept*: the whole 64 Kb built-in operating system was listed and explained in detail in the three **Internal Design Specification** volumes, so that developers would be able to understand *everything* done in the machine and even use the many officially supported entry points in their own assembly-language routines.
- ***Multi-language programming***: the powerful BASIC language wasn't the only choice. You could further extend its capabilities with assembly-language routines, or you could program in **Forth**, a powerful threaded language faster (if less friendly) than BASIC. Running your **HP-41C** programs or even creating new ones in **HP-41C**'s own 'language' was easily made possible with the **HP-41** Translator/Forth ROM as well.

Regrettably, the **HP-71B** was marred by some poor (impossed ?) decissions, such as the *vastly undersized*, 1-line x 22-character dot matrix display, which made programming life really difficult, the vast amounts of System ROM space *wasted* in such undeserving features as CALC mode and IEEE compliancy which resulted in the initially built-in, ultra-useful Math ROM capabilities being *removed* from the System ROMs to make space for them, the *impossibly high* price, and the *inept*, virtually non-existing marketing, which combined resulted in this wonderful machine selling *very poorly*, thus failing to reach **HP**'s expectations and being the last of its kind. But that's another story !

## Sample program: Stereograms in 6 lines !

Just to show off the **HP-71B**'s versatility, this *very small*, 6-line program I've written specifically for this article will compute and print 3D random-text stereograms starting form a user-specified Z-map which records the heights for each location in a 32-row by 64-column random-character arrangement.

The user just needs to provide a *subprogram* which generates and returns the desired Z-map, and this program does the rest, quickly and efficiently. The resulting random-text stereogram can be viewed with the usual techniques to result in a genuine 3D rendering of your map.

Simple as the program is, it nevertheless demonstrates a number of very important features available in the powerful **HP-71B**'s BASIC dialect, as we'll see right now.

## Program listing

```
10 DESTROY ALL @ OPTION BASE 1 @ RANDOMIZE 7 @ DIM C,D,L,M,R,T,V,X,Y
20 M=24 @ C=64 @ R=32 @ T=12 @ DIM Z$(R)[C],P$[C+T] @ INPUT "SUB=";S$
30 CALL S$(Z$,(R),(C)) @ FOR Y=1 TO R @ L=0 @ D=T @ P$=""
40 FOR I=1 TO T @ P$=P$&CHR$(RND*16+32) @ NEXT I
50 FOR X=1 TO C @ V=VAL(Z$(Y)[X,X]) @ IF V#L THEN D=M-V @ L=V
60 P$=P$&P$[LEN(P$)-D+1][1,1] @ NEXT X @ PRINT P$ @ NEXT Y
```

## Notes:

- In order to actually produce the stereogram in an immediately viewable format, you must use a suitable output device, which typically will be either the 80-column display emulation available in **Emu71**, a physical HP-IL 80-column display device, or an HP-IL printer, which you must define as your `PRINTER IS` device. Obviously, for the purposes of seeing the 3D image, the built-in one-line display won't do unless you painstakingly type the output in some PC editor, by hand.
- Though the program code itself requires very little RAM, the generated stereogram will take some 2K to generate and store, plus the RAM that your Z-map generator subprogram uses. All in all, a bare-bones **HP-71B** with some 4 Kb free RAM should be perfectly adequate.
- This program uses only built-in BASIC commands, no external keywords taken from ROMs or LEX files (though some of the subprogram examples below do use common LEX-provided string operations).
- If desired, you can easily modify it to generate grids larger than 32x64 by simply changing the values assigned to `R` (rows) and `C` (columns) at line 20, to suit that 80- or 128-column line printer of yours. Also, you can use a larger character set by increasing the value `16` at line 40 (up to 96 max). I find these 16 characters to be easier on the eyes but you may find it easier still to 3D-visualize the image when using more varied characters.

## Program description

Let's point out some of the powerful BASIC statements mostly unique to the **HP-71B** dialect that are being used here, on a line-by-line basis:

```
10 DESTROY ALL @ OPTION BASE 1 @ RANDOMIZE 7 @ DIM C,D,L,M,R,T,V,X,Y
```

- **HP-71B's** BASIC does allow *dynamic creation and destruction of variables* as a fully-programmable feature, and further it shares calculator-mode variables with BASIC programs. So for this sample program, to start from a known variable state we use a **`DESTROY ALL`** statement which wipes out all calculator-mode variables, freeing RAM and avoiding type conflicts.

- We then specify the lower bound for arrays to start at 1 by using the standard **`OPTION BASE 1`** statement, and initialize the pseudo-random generator seed from a specific value, so that you can exactly duplicate the examples below to test that the program was typed in correctly. Once this is checked, you can simply use **`RANDOMIZE`** instead of **`RANDOMIZE 7`**, for simplicity.

- Once this is done, we reserve space for the variables by using a **`DIM`** statement. This isn't mandatory, but it's good programming practice.

```
20 M=24 @ C=64 @ R=32 @ T=12 @ DIM Z$(R)[C],P$[C+T] @ INPUT "SUB=";S$
```

- As mentioned above, unlike many other BASIC dialects, dynamic variable creation is allowed, and this also applies to vectors and matrices, which can have their dimensions specified at runtime. In this case, we are creating a vector of strings, **`Z$`**, which will have **`R`** elements of length **`C`**, as well as a single string variable **`P$`**, which will be able to hold up to **`C+T`** characters. The program then asks for your Z-map generating subprogram's name.

```
30 CALL S$(Z$,(R),(C)) @ FOR Y=1 TO R @ L=0 @ D=T @ P$=""
```

- Yet another very important feature of **HP-71B** BASIC, almost unique at its time, is the ability to define and use *independent subprograms*, which will be called and passed parameters either by value or by reference, and can return results to the calling program, while having their own environment with their own variables, etc.

- Here we're calling a subprogram which itself has a *variable name* (!), specified in the string variable **`S$`**, and which gets passed three parameters, namely **`Z$`**, the string array where the user-specified Z-map will be returned, passed *by reference* so that it will be returned to our calling program, plus the number of rows and columns, which are passed *by value* (by enclosing them in parentheses), so that if your subprogram modifies them, the original variables in the calling program won't be affected at all.

```
40 FOR I=1 TO T @ P$=P$&CHR$(RND*16+32) @ NEXT I
```

- This line shows a character being randomly generated in a given range, then getting appended to form a full line of output stored in string variable **`P$`**.

```
50 FOR X=1 TO C @ V=VAL(Z$(Y)[X,X]) @ IF V#L THEN D=M-V @ L=V
60 P$=P$&P$[LEN(P$)-D+1][1,1] @ NEXT X @ PRINT P$ @ NEXT Y
```

- Two powerful capabilities are demonstrated here, namely the *substring operator*, **`[]`**, which can *extract* a specified portion from any string expression (the **`Y`**–th element of string array **`Z$`** in the first case; it can also *assign* a substring within a string variable), and the string evaluation function, **`VAL`** which will compute and return the *numeric* value of any string expression which can be interpreted as a numeric expression.

- The **`[]`** substring operator can be used repeatedly in a daisy-chained fashion if necessary (such as **`P$[LEN(P$)-D+1][1,1]`** at line 60), and combined with numerical evaluation of logical operators can bring the user amazing substring-slicing and parsing capabilities, specially since it can be used at both sides of the **`=`** assignment operator to either extract or assign a given substring.

- On the other hand, while **`VAL`** is a standard function present in all respectable BASIC dialects, it's usually strictly limited to string expressions than can be interpreted as a *number*, say **`"-1.234E7"`**, while **HP-71B**'s implementation will evaluate *any expression which returns a numeric value*, from simple ones like **`"2+SIN(X)*EXP(X)"`** to really complex ones making use of external ROM keywords, such as **`"INTEGRAL(0,Z,1E-5,EXP(-IVAR^2))"`**, which will promptly return the numeric value of the integral as long as the Math ROM is present and **`Z`** is a numeric variable containing the upper limit. This is incredibly powerful as a program can accept funtion definitions or mathematical expressions from the user on the fly.

- Finally, the generated stereogram is output by issuing a **`PRINT P$`** statement, which, simple as it seems, actually uses *any device* specified by the current **`PRINTER IS`** setting, so it can be an HP-IL thermal printer, an RS-232 line printer, a 80x24 HP-IL LCD display, an emulated 80-colum display as the one provided by **Emu71**, or even some instrument which can be specified as the "printing" device and can accept output sent by the **`PRINT`** statement. How's that for utmost flexibility ?

## Usage instructions

Once you've keyed the program in, you must also enter *your own subprogram* which generates the desired Z-map and must meet these requirements:

- It can be in the same file as the main program itself, or in a completely separate file in RAM, either on its own or as part of some other code.
- It can have any legal name as long as it is *unique*. If not, then it must be included in the same file as the main program in order to avoid ambiguity.
- It *must* accept the following parameters, in this order, which can have any legal name as long as their types are as follows:
    - **`Z$`**, a one-dimensional string array where your subprogram must store the Z-map. As it is passed *by reference*, whatever your subprogram puts in it will be returned to the calling main program.
    - **`R`**, a real numeric value representing the number of rows in the map, which is the number of string elements in **`Z$`**. As it is passed *by value*, any changes to it will not alter the original value in the caller.
    - **`C`**, a real numeric value representing the number of columns in the map, which is the length in characters of each string element in **`Z$`**. It's also passed by value.

Your subprogram must fill up the **`Z$`** array with the proper heights for each row and column on the map, as follows (see also the examples below):

- Each *element* in **`Z$`** represents a map *row*, from top (**`Z$(1)`**) to bottom (**`Z$(R)`**). Each *position* in an element represents the number of a *column*, so that the character stored at position 3 in **`Z$(7)`** (i.e., **`Z$(7)[3,3]`**) records the *height* of the location at row 7, column 3 in your Z-map.

- The heights themselves are represented by single characters from the set ”**`0`**”, ”**`1`**”, ”**`2`**”, ”**`3`**”, ... , where ”**`0`**” represents the *base plane* (height = 0), ”**`1`**” represents the next plane above it (height=1), and so on. Theoretically you could have up to 10 different heights, from 0 to 9, but in practice that many different heights can be somewhat difficult to discern so you'd do well to limit your maps to heights from 0 to 4 for best viewing results.

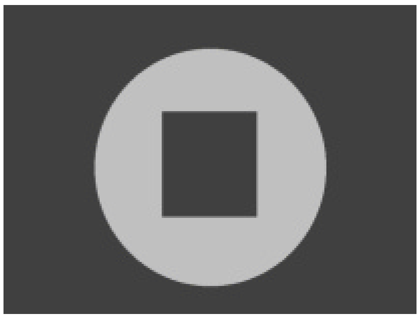
Once your subprogram has been written and entered, simply run the main program:

```
>RUN
   SUB= (key in the name of your subprogram, then press [ENTER])
```

The random-text stereogram is then generated and output.

## Examples

### 1. Try and generate a random-text stereogram for this 3D image



First of all, we need to write and enter a subprogram **`CIRCLE`** which will generate the required Z-map for this stereogram, assuming the darker area is the base plane (height=0), and the lighter circle-with-square-hole is above it, at height 1. The subprogram which will then fill **`Z$`** with properly placed ”**`0`**”s and ”**`1`**” is:

```
500 SUB CIRCLE(Z$(),R,C) @ FOR I=1 TO R @ Z$(I)=RPT$("0",C) @ NEXT I
510 FOR I=6 TO 28@T$=RPT$("1",7+SQR(121-(I-17)^2))@Z$(I)[1,LEN(T$)]=T$
520 Z$(I)=RPT$("0",12)&REV$(Z$(I)[1,20])&Z$(I)[1,20]&RPT$("0",12)
530 IF I>11 AND I<23 THEN Z$(I)[26,39]=RPT$("0",14)
540 NEXT I
```

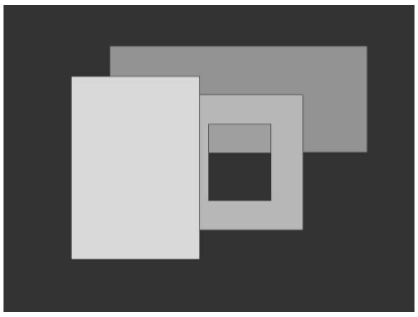
Let’s run it all and generate our stereogram:

```
>RUN
 SUB=CIRCLE [ENTER]
```

```
     /#-.!'&* ".+/#-.!'&* ".+/#-.!'&* ".+/#-.!'&* ".+/#-.!'&* ".+/#-.!'&* ".+/#-.
     $&%"(!#0/()$$&%"(!#0/()$$&%"(!#0/()$$&%"(!#0/()$$&%"(!#0/()$$&%"(!#0/()$$&%"
     %#!-,+ ""&)0%#!-,+ ""&)0%#!-,+ ""&)0%#!-,+ ""&)0%#!-,+ ""&)0%#!-,+ ""&)0%#!-
     ,$.')//')0&*,$.')//')0&*,$.')//')0&*,$.')//')0&*,$.')//')0&*,$.')//')0&*,$.'
     )*&).(-,*!0 )*&).(-,*!0 )*&).(-,*!0 )*&).(-,*!0 )*&).(-,*!0 )*&).(-,*!0 )*&)
     *&&(,.-")&+"*&&(,.-")&+"*&&(,.-")&+"*&(,.-")&+"*&&((,.-")&+"*&(,.-")&+"*&&((
     $)"/-&"*+"-.$)"/-&"*+"-.$)"/-&"*"-.$)"/-&"*+"-.$)"/-&"*""-.$)"/-&"*+"-.$)"/-
     &."(+%$*%(%%&."(+%$*%(%%&."(+%$%(%%&."(+%$*%(%%&."(+%$%(%%%&."(+%$*%(%%&."(+
     ($,*%$" ('%'($,*%$" ('%'($,*%" ('%'($,*%$" ('%'($,*%" ('%'(($,*%$" ('%'($,*%
     ($.0%,(..#(-($.0%,(..#(-($.0%(..#(-($.0%,(..#(-($.0%(..#(-(($.0%,(..#(-($.0%
     +$&& #$"!*0"+$&& #$"!*0"+$&&#$"!*0"+$&& #$"!*0"+$&&#$"!*0"+$$&& #$"!*0"+$&&#
     +,$))(")!"'(+,$))(")!"'(+,$)(")!"'(+,,$))(")!"'(+,$(")!"'(+,,,$))(")!"'(+,$(
     .!!''+(#"+0!.!!''+(#"+0!.!!'+(#"+0!.!!!''+(#"+0!.!!+(#"+0!.!!!!''+(#"+0!.!!+
     +!!(-'"$#,"*+!!(-'"$#,"*+!(-'"$#,"*+!!!(-'"$#,"*+!('"$#,"*+!!!!(-'"$#,"*+!('
     )'+*(!0)%*)/)'+*(!0)%*)/)'*(!0)%*)/)''+*(!0)%*)/)'*!0)%*)/)''++*(!0)%*)/)'*!
     +,('.*" /.!-+,('.*" /.!-+,'.*" /.!-+,,('.*" /.!-+,'*" /.!-+,,(('.*" /.!-+,'*
     -/%!,&*(,*-%-/%!,&*(,*-%-/!,&*(,*-%-//%!,&*(,*-%-/!&*(,*-%-//%%!,&*(,*-%-/!&
     ,$*+!%!'"'"*,$*+!%!'"'"*,$+!%!'"'"*,$$*+!%!'"'"*,$+%!'"'"*,$$**+!%!'"'"*,$+%
      0-/.* +-')"&0-/.* +-')"&0-.* +-')"&0--/.* +-')"&0-. +-')"&0--//.* +-')"&0-.
      ,0+/$ ,(0"!$,0+/$ ,(0"!$,0/$ ,(0"!$,00+/$ ,(0"!$,0/ ,(0"!$,00++/$ ,(0"!$,0/
     .$&'0-$($)*/.$&'0-$($)*/.$&0-$($)*/.$$&'0-$($)*/.$&-$($)*/.$$$&'0-$($)*/.$&-
     '#"#,('*"'+.'#"#,('*"'+.'#",('*"'+.'##"#,('*"'+.'#"('*"'+.'###"#,('*"'+.'#"(
     "",&.,+&# ,+"",&.,+&# ,+"",&,+&# ,+"",&.,+&# ,+"",&,+&# ,+""",&.,+&# ,+"",&,
     -'#*-//*%,-$-'#*-//*%,-$-'#*-/*%,-$-'#*-//*%,-$-'#*-/*%,-$--'#*-//*%,-$-'#*-
     -"*-.&&#'##(-"*-.&&#'##(-"*-.&#'##(-"*-.&&#'##(-"*-.&#'##(--"*-.&&#'##(-"*-.
     .0.%&*)'*0*,.0.%&*)'*0*,.0.%&*)*0*,.0.%&*)'*0*,.0.%&*)*0**,.0.%&*)'*0*,.0.%&
     , /,-$++'/"&, /,-$++'/"&, /,-$++/"&, /,-$++'/"&, /,-$++//"&, /,-$++'/"&, /,-
     )) #*"/),$,')) #*"/),$,')) #*"/),$,') #*"/),$,')) ##*"/),$,') #*"/),$,')) ##
     +*-$%).-)''++*-$%).-)''++*-$%).-)''++*-$%).-)''++*-$%).-)''++*-$%).-)''++*-$
      &)!+%&'$$!% &)!+%&'$$!% &)!+%&'$$!% &)!+%&'$$!% &)!+%&'$$!% &)!+%&'$$!% &)!
     $+,0&+*/,'-"$+,0&+*/,'-"$+,0&+*/,'-"$+,0&+*/,'-"$+,0&+*/,'-"$+,0&+*/,'-"$+,0
     #+-!%((*, %##+-!%((*, %##+-!%((*, %##+-!%((*, %##+-!%((*, %##+-!%((*, %##+-!
```

which you can try and see in full 3D with the usual techniques, either from the printed page or, probably better, from the **Emu71** output in your monitor.

## 2. Now do the proper thing with this multi-level image:



We’ll write and enter another subprogram, **RECT3D**, to generate the proper Z-map for this 3D image where again the darkest area is the base plane (height=0), and the lighter rectangles are at increasing heights above it. The subprogram which will fill **Z$** with properly placed “**0**”s, “**1**”s, “**2**”s, and “**3**”s is as follows:

```
430 SUB RECT3D(Z$(),R,C) @ FOR I=1 TO R @ Z$(I)=RPT$("0",C) @ NEXT I
440 FOR I= 5 TO 15 @ Z$(I)[23,62]=RPT$("1",40) @ NEXT I
450 FOR I=10 TO 23 @ Z$(I)[33,52]=RPT$("2",20) @ NEXT I
460 FOR I= 8 TO 26 @ Z$(I)[17,36]=RPT$("3",20) @ NEXT I
470 FOR I=16 TO 20 @ Z$(I)[38,47]=RPT$("0",10) @ NEXT I
480 FOR I=13 TO 15 @ Z$(I)[38,47]=RPT$("1",10) @ NEXT I
```

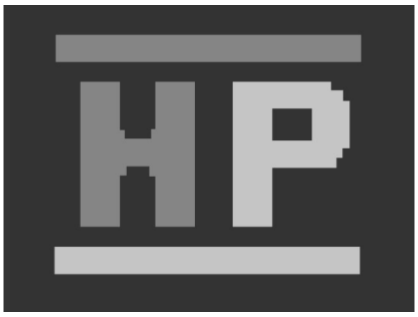
Let’s run it:

```
>RUN
 SUB=RECT3D  [ENTER]
```

```
/#-.!'&* ".+/#-.!'&* ".+/#-.!'&* ".+/#-.!'&* ".+/#-.!'&* ".+/#-.!'&* ".+/#-.
$&%"(!#0/()$$&%"(!#0/()$$&%"(!#0/()$$&%"(!#0/()$$&%"(!#0/()$$&%"(!#0/()$$&%"
%#!-,+ ""&)0%#!-,+ ""&)0%#!-,+ ""&)0%#!-,+ ""&)0%#!-,+ ""&)0%#!-,+ ""&)0%#!-
,$.')//')0&*,$.')//')0&*,$.')//')0&*,$.')//')0&*,$.')//')0&*,$.')//')0&*,$.'
)*&).(-,*!0 )*&).(-,*!0 )*&).(-,*! )*&).(-,*!0 )*&).(-,*! )*&).(-,*!0 )*&)).
*&&(,.-")&+"*&&(,.-")&+"*&&(,.-")&"*&&(,.-")&+"*&&(,.-")&"*&&(,.-")&+"*&&((,
$)"/-&"*+"-.$)"/-&"*+"-.$)"/-&"*+".$)"/-&"*+"-.$)"/-&"*+".$)"/-&"*+"-.$)"//-
&."(+%$*%(%%&."(+%$*%(%%&."(*%(%%&."(+%$*%(%%&."."(*%(%%&."(+%$*%(%%&."."((*
 ($,*%$" ('%'($,*%$" ('%'($,* ('%'($,*%$" ('%'($,$,* ('%'($,*%$" ('%'($,$,**
($.0%,(..#(-($.0%,(..#(-($.0..#(-($.0%,(..#(-($..0..#(-($.0%,(...#(-($..0...
+$&& #$"!*0"+$&& #$"!*0"+$&&"!*0"+$&& #$"!*0"+$&&&"!*0"+$&& #$"!!*0"+$&&&""!
+,$))(")!"'(+,$))(")!"'(+,$))!"'(+,$))(")!"'(+,$$))!"'(+,$))(")!!"'(+,$$)))!
.!!''+(#"+0!.!!''+(#"+0!.!!'#"+0!.!!''+(#"+0!.!!!!'#"+0!.!!'+(#""+0!.!!!!''#
+!!(-'"$#,"*+!!(-'"$#,"*+!!($#,"*+!!(-'"$#,"*+!!!!($#,"*+!!-'"$##,"*+!!!!(($
)'+*(!0)%*)/)'+*(!0)%*)/)'+*)%*)/)'+*(!0)%*)/)'+++*)%*)/)'+(!0)%%*)/)'+++**)
+,('.*" /.!-+,('.*" /.!-+,(' /.!-+,('.*" /.!-+,((,(' /.!-+,.*" / /.!-+,((,('
-/%!,&*(,*-%-/%!,&*(,*-%-/%!(,*-%-/%!,&*(,*-%-/%%/%!(,*-%-/,&*(,(,*-%-/%%/%!
,$*+!%!'"'"*,$*+!%!'"'"*,$*+'"'"*,$*+!%!'"'"*,$**$*+'"'"*,$!%!'"'"'"*,$**$*+
0-/.* +-')"&0-/.* +-')"&0-/.-')"&0-/.* +-')"&0-//-/.-')"&0-* +-'-')"&0-//-/.
,0+/$ ,(0"!$,0+/$ ,(0"!$,0+/(0"!$,0+/$ ,(0"!$,0++0+/(0"!$,0$ ,(0(0"!$,0++0+/
.$&'0-$($)*/.$&'0-$($)*/.$&'($)*/.$&'0-$($)*/.$&&'($)*/.$&'0-$($($)*/.$&&'($
'#"#,('*"'+.'#"#,('*"'+.'#"#*"'+.'#"#,('*"'+.'#""#*"'+.'#"#,('*"*"'+.'#""#*"
"",&.,+&# ,+"",&.,+&# ,+"",&&# ,+"",&.,+&# ,+"",,&&# ,+"",&.,+&#&# ,+"",,&&#
-'#*-//*%,-$-'#*-//*%,-$-'#**%,-$-'#*-//*%,-$-'#-'#**%,-$-'#*-//*%,-$-'#-'#*
-"*-.&&#'##(-"*-.&&#'##(-"*-#'##(-"*-.&&#'##(-"*-"*-#'##(-"*-.&&#'##(-"*-"*-
.0.%&*)'*0*,.0.%&*)'*0*,.0.%'*0*,.0.%&*)'*0*,.0..0.%'*0*,.0.%&*)'*0*,.0..0.%
, /,-$++'/"&, /,-$++'/"&, /,-$++'/"&, /,-$++'/"&, /,-$++'/"&, /,-$++'/"&, /,
)) #*"/),$,')) #*"/),$,')) #*"/),$,')) #*"/),$,')) #*"/),$,')) #*"/),$,')) #
+*-$%).-)''++*-$%).-)''++*-$%).-)''++*-$%).-)''++*-$%).-)''++*-$%).-)''++*-$
 &)!+%&'$$!% &)!+%&'$$!% &)!+%&'$$!% &)!+%&'$$!% &)!+%&'$$!% &)!+%&'$$!% &)!
$+,0&+*/,'-"$+,0&+*/,'-"$+,0&+*/,'-"$+,0&+*/,'-"$+,0&+*/,'-"$+,0&+*/,'-"$+,0
#+-!%((*, %##+-!%((*, %##+-!%((*, %##+-!%((*, %##+-!%((*, %##+-!%((*, %##+-!
```

Amazing, isn’t it ? Note the height-2, holed rectangle, which allows you to see *both* the height-1 rectangle immediately below it *and* the height-0 base plane far below. If desired, you can *display the generated Z-map* by executing this sentence from the command line: **FOR I=1 TO R @ Z$(I) @ NEXT I**

**3. Finally, let's make a pretty 3D logo of HP, like this:**



This time the shapes are too complicated to try and write a subprogram which attempts to create them *geometrically*. Instead, we'll resort to a more RAM-consuming approach, which nevertheless makes the Z-map that much *easier* to create and is completely general in nature, valid for any image whatsoever: we'll simply use a **DATA** grid, like this:

```
90 SUB HP(Z$(),R,C) @ RESTORE @ FOR I=1 TO R @ READ Z$(I) @ NEXT I
100 DATA 0000000000000000000000000000000000000000000000000000000000000000
110 DATA 0000000000000000000000000000000000000000000000000000000000000000
120 DATA 0000000000000000000000000000000000000000000000000000000000000000
130 DATA 0000000000001111111111111111111111111111111111111111111111110000
140 DATA 0000000000001111111111111111111111111111111111111111111111110000
150 DATA 0000000000001111111111111111111111111111111111111111111111110000
160 DATA 0000000000000000000000000000000000000000000000000000000000000000
170 DATA 0000000000000000000000000000000000000000000000000000000000000000
180 DATA 0000000000000000111111000000111111000000222222222222222200000000
190 DATA 0000000000000000111111000000111111000000222222222222222220000000
200 DATA 0000000000000000111111000000111111000000222222222222222222000000
210 DATA 0000000000000000111111000000111111000000222220000000222222000000
220 DATA 0000000000000000111111000000111111000000222220000000222222000000
230 DATA 0000000000000000111111100001111111000000222220000000222222000000
240 DATA 0000000000000000111111111111111111000000222222222222222222000000
250 DATA 0000000000000000111111111111111111000000222222222222222220000000
260 DATA 0000000000000000111111111111111111000000222222222222222200000000
270 DATA 0000000000000000111111100001111111000000222220000000000000000000
280 DATA 0000000000000000111111000000111111000000222220000000000000000000
290 DATA 0000000000000000111111000000111111000000222220000000000000000000
300 DATA 0000000000000000111111000000111111000000222220000000000000000000
310 DATA 0000000000000000111111000000111111000000222220000000000000000000
320 DATA 0000000000000000111111000000111111000000222220000000000000000000
330 DATA 0000000000000000000000000000000000000000000000000000000000000000
340 DATA 0000000000000000000000000000000000000000000000000000000000000000
350 DATA 0000000000002222222222222222222222222222222222222222222222220000
360 DATA 0000000000002222222222222222222222222222222222222222222222220000
370 DATA 0000000000002222222222222222222222222222222222222222222222220000
380 DATA 0000000000000000000000000000000000000000000000000000000000000000
390 DATA 0000000000000000000000000000000000000000000000000000000000000000
400 DATA 0000000000000000000000000000000000000000000000000000000000000000
410 DATA 0000000000000000000000000000000000000000000000000000000000000000
```

As you can see, any Z-map can be defined this way, with utmost ease and requiring very little time to generate, as it simply reads and assigns each string element, i.e.: 64 locations at a time. The drawback is of course the time required to enter the **DATA** statements into program memory, plus the extra 2 Kb of RAM they'll take, in addition to the 2 Kb which the **Z$** string array itself requires. On the other hand, by using this simpleminded approach you'll *save* the non-negligible time and effort required to write a *complicated* subprogram to fill up the Z-map which, as you can see in the previous examples (**CIRCLE** and **RECT3D)**, might be significant.

Let's run it !:

```
>RUN
 SUB=HP  [ENTER]

    /#-.!'&* ".+/#-.!'&* ".+/#-.!'&* ".+/#-.!'&* ".+/#-.!'&* ".+/#-.!'&* ".+/#-.
    $&%"(!#0/()$$&%"(!#0/()$$&%"(!#0/()$$&%"(!#0/()$$&%"(!#0/()$$&%"(!#0/()$$&%"
    %#!-,+ ""&)0%#!-,+ ""&)0%#!-,+ ""&)0%#!-,+ ""&)0%#!-,+ ""&)0%#!-,+ ""&)0%#!-
    ,$.')//')0&*,$.')//')0&*$.')//')0&*,$.')//')0&*$.')//')0&*,$.')//')0&*$..')/
    )*&).(-,*!0 )*&).(-,*!0 *&).(-,*!0 )*&).(-,*!0 *&).(-,*!0 )*&).(-,*!0 *&&).(
    *&&(,.-")&+"*&&(,.-")&+"&&(,.-")&+"*&&(,.-")&+"&&(,.-")&+"*&&(,.-")&+"&&&(,.
    $)"/-&"*+"-.$)"/-&"*+"-.$)"/-&"*+"-.$)"/-&"*+"-.$)"/-&"*+"-.$)"/-&"*+"-.$)"/
    &."(+%$*%(%%&."(+%$*%(%%&."(+%$*%(%%&."(+%$*%(%%&."(+%$*%(%%&."(+%$*%(%%&."(
    ($,*%$" ('%'($,*%$" ('%'($,*$" ('%%'($,*$" ('%%'($,* ('%%'($,*$" ('%'%%'($,*
    ($.0%,(..#(-($.0%,(..#(-($.0,(..#((-($.0,(..#((-($.0..#((-($.0,(..#((((-($.0
    +$&& #$"!*0"+$&& #$"!*0"+$&&#$"!*00"+$&&#$"!*00"+$&&"!*00"+$&&#$"!*00"0"+$&&
    +,$))(")!"'(+,$))(")!"'(+,$)(")!"''(+,$)(")!"''(+,$))!"''('(+,$))!"''('(+,$)
    .!!''+(#"+0!.!!''+(#"+0!.!!'+(#"+00!.!!'+(#"+00!.!!'#"+00!0!.!!'#"+00!0!.!!'
    +!!(-'"$#,"*+!!(-'"$#,"*+!!('"$#,"**+!!-'"$#,""*+!!($#,"****+!!-$#,""*"*+!!(
    )'+*(!0)%*)/)'+*(!0)%*)/)'+*!0)%*)/)'+*(!0)%*))/)'+*)%*)/)'+*(!0)%*))/)/)'+*
    +,('.*" /.!-+,('.*" /.!-+,('*" /.!-+,('.*" /.!!-+,(' /.!-+,('.*" /.!!!!-+,('
    -/%!,&*(,*-%-/%!,&*(,*-%-/%!&*(,*-%-/%!,&*(,*--%-/%!(,*-%-/%!,&*(,*-*--%-/%!
    ,$*+!%!'"'"*,$*+!%!'"'"*,$*+%!'"'"**,$*!%!'"'""*,$*+'"'"****,$*!%!'"'""*,$*+
    0-/.* +-')"&0-/.* +-')"&0-/. +-')""&0-/. +-')""&0-/.-')""&"&0-/. +-')""&0-/.
    ,0+/$ ,(0"!$,0+/$ ,(0"!$,0+/ ,(0"!!$,0+/ ,(0"!!$,0+/(0"!!$!$,0+/ ,(0"!!$,0+/
    .$&'0-$($)*/.$&'0-$($)*/.$&'-$($)**/.$&'-$($)**/.$&'($)**/*/.$&'-$($)**/.$&'
    '#"#,('*"'+.'#"#,('*"'+.'#"#('*"'++.'#"#('*"'++.'#"#*"'++.+.'#"#('*"'++.'#"#
    "",&.,+&# ,+"",&.,+&# ,+"",&,+&# ,,+"",&,+&# ,,+"",&&# ,,+,+"",&,+&# ,,+"",&
    -'#*-//*%,-$-'#*-//*%,-$-'#*-//*%,-$-'#*-//*%,-$-'#*-//*%,-$-'#*-//*%,-$-'#*
    -"*-.&&#'##(-"*-.&&#'##(-"*-.&&#'##(-"*-.&&#'##(-"*-.&&#'##(-"*-.&&#'##(-"*-
    .0.%&*)'*0*,.0.%&*)'*0*,.%&*)'*0*,.0.%&*)'*0*,.%&*)'*0*,.0.%&*)'*0*,.%&*&*)'
    , /,-$++'/"&, /,-$++'/"&/,-$++'/"&, /,-$++'/"&/,-$++'/"&, /,-$++'/"&/,-$-$++
    )) #*"/),$,')) #*"/),$,' #*"/),$,')) #*"/),$,' #*"/),$,')) #*"/),$,' #*"*"/)
    +*-$%).-)''++*-$%).-)''++*-$%).-)''++*-$%).-)''++*-$%).-)''++*-$%).-)''++*-$
     &)!+%&'$$!% &)!+%&'$$!% &)!+%&'$$!% &)!+%&'$$!% &)!+%&'$$!% &)!+%&'$$!% &)!
    $+,0&+*/,'-"$+,0&+*/,'-"$+,0&+*/,'-"$+,0&+*/,'-"$+,0&+*/,'-"$+,0&+*/,'-"$+,0
    #+-!%((*, %##+-!%((*, %##+-!%((*, %##+-!%((*, %##+-!%((*, %##+-!%((*, %##+-!
```

Nice, right ?

## Seeing stereograms

People who've never seen an stereogram before, either random-text- or random-dot-based, usually find it difficult to even *believe* that there's something to be seen amidst that ungainly-looking bunch of characters/dots, let alone a nice full 3D rendering of some well-defined shapes or objects. But once they succeed in mastering the technique, they'll be able to enjoy the awesome feeling of looking at a mere 2-dimensional, plane surface full of weird characters and seeing it gain real, authentic tridimensional depth where none should physically be possible, and where perfect geometrical shapes float by at various depths.

If you find it difficult to see stereograms in general, you'd do well to "google" for the required procedures, there are many excellent tutorials for free on the web, as well as many excellent books on the subject.

If you do manage to visualize the above examples in glorious 3D but you don't see the expected shapes, your eyes are at the *wrong distance* from the printout or else the printout is *too large* or *too small*. Correct and you'll eventually succeed.

## Final remarks

The **HP-71B** was and still *is* indeed a superb computing device, truly without equal (*nonpareil*, as they say ...). In its physical form, you've got a most solid, elegant, well-engineered gem which oozes *quality* and *reliability*, all in a rather small, convenient package which you can take with you without much trouble and negligible weight. The keyboard's a real pleasure to use, and the "continuous memory" feature coupled with the fact that you can have really large amounts of available RAM, plus the convenient file system, means that you'll be able to have *all* your programs and data ready to use without ever needing a mass storage device while doing field work in surveying, say, or geophysical engineering.

In its emulated form (**Emu71**, for instance) you get as much emulated RAM/ROM as you care for, a large 80-column x 40 (say) rows emulated display, and *incredible speed acceleration* (350x or more) in a typical 2.4 Ghz PC. This means it's got speed enough to do very cumbersome calculations in a flash which, coupled with its powerful and easy-to-use BASIC, the advanced complex-number handling, matrix commands, and the **`SOLVE`**/**`INTEGRATE`** capabilities available in the emulated Math ROM, it all amounts to real computational power even nowadays, despite your having the usual suite of productivity software at hand. I can, for instance solve a complex-domain contour integration problem in the emulated **HP-71B** in a fraction of the time it would take to even think how to enter the problem in Excel, say. It's interactivity, sheer power, and above all, ease of use, means that it's become the essential tool for most of my number crunching needs, simple or advanced, for fun or for blood. So I can confidently say ... **Long Live the HP-71B!**