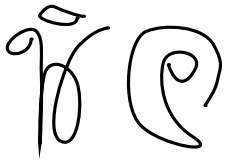
DIMFILM Revived

An Adventure in Software Archæology

NICK GLAZZARD

Highlands Cottage Computer Center



This book was created in LaTeX using MiKTeX 2.7 and TeXnicCenter 1β7.50. Previews were done with Foxit Reader V2.1 (for PDF), GSview V4.9 (PS,EPS) and GPL Ghostscript 8.63. All this software was run under Microsoft Windows XP Service Pack 3.

All DIMFILM output was generated on an emulated CDC Cyber 173 running NOS 1.4 L552. The emulator was Tom Hunter's Desktop Cyber V2.0β1 with extensive local modifications. It was hosted on CentOS 5 Linux.

The following ISBN number is not valid and is just there to show off barcodes.

ISBN 3-86541-114-2

9 783865 411143 >

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 What is DIMFILM?

DIMFILM was the primary 2D graphics package available at U.L.C.C. (University of London Computer Centre) between 1973 and around 1993. It was written by John Gilbert, and takes the form of a substantial library written in Fortran-77 and intended for use with programs written in that language.

The following history should not be considered definitive, but I believe it is reasonably accurate. The initial version of DIMFILM was designed specifically for CDC 6000 series machines with a Calcomp 1620 microfilm plotter as the only output device. This version was used throughout U.L.C.C.'s 'CDC era', which began with the installation of a 6600 in 1969 (DIMFILM was first released in 1973) and ended around 1984. During that period, DIMFILM was also available on the CDC 7600.

This original DIMFILM was written in the CDC dialect of Fortran-IV (which included seven letter variable names) and was never intended to be ported to other machines. In spite of that, it was successfully ported to VAX/VMS by Nigel Arnot and Adrian Clark and used for many years on that platform by the Physics Dept. of Queen Elizabeth College (University of London) and later by Kings College.

By 1982 or so, it was clear that the CDC machines at U.L.C.C. were going to be replaced by 'something else'. That turned out to be an Amdahl V470 and a Cray-1/S[1]. The transition occurred in 1983-84. DIMFILM was re-written by John Gilbert in portable Fortran-77 so that it could continue to be used in that new environment. The primary output device then was a Dicomed D-48C film recorder. This supported colour and raster image output, and the new DIMFILM allowed access to these new facilities. It could also be more easily modified to support other output devices (although the QEC port of the original DIMFILM had successfully done this too). It eventually supported many output devices, including various display terminals.

This 'new DIMFILM' (perhaps sometimes called DIMFILM-77) remained the primary 2D graphics

[1] Three Cray machines over the years, in fact. Cray-1S/1000 Serial 28, Cray-1S/2200 Serial 44 and Cray X-MP/28 Serial 412

package at U.L.C.C. until the 'Unix era' dawned with Convex machines being installed in the early 1990s. It seems that DIMFILM fell out of use at U.L.C.C. somewhere around 1993-94.

DIMFILM was installed at the Rutherford Appleton Laboratory Central Computing Department in early 1988 according to: `http://www.chilton-computing.org.uk/ccd/literature/ccd_newsletters/arclight/p010.htm` [2] This was intended to allow Cray X-MP/48 users at that site to generate output on the U.L.C.C Dicomed device. Presumably, this ended when the RAL Central Computing Department closed in 1994.

DIMFILM-77 was used by Adrian Clark at the University of Essex until at least 1990. However, by the mid 1990's, it seems to have pretty much disappeared. It never made it in to the 'Internet era', and hence there are virtually no references to it on the Web.

1.2 Why revive DIMFILM?

I was personally interested in 'reviving' DIMFILM for a number of reasons:

- As a user of DIMFILM at U.L.C.C. and QEC in the 1979-84 time period, I found it quite an inspiration! I regarded it as a very well thought out library that produced excellent quality output. This was a time when *any* form of computer graphics was rare – hard to believe now, but true. It was also the first large scale software product I had come across.
- Thanks to Tom Hunter's Desktop Cyber emulator, CDC mainframes continue to live, and I have been running an emulated Cyber 173 since 2003 (on and off!). This is a fascinating environment (for reasons I won't dwell on) and a major part of my interest in restoring DIMFILM was to have something substantial and interesting to run on it.
- It is interesting to compare examples of the best of the past with more current, mainstream, equivalents – to see how far we have come – *or not*, as the case may be. Since I have earned my living from writing graphics software since 1985, I am particularly interested in the progress and history of computer graphics.

Unfortunately (from a strict historical point of view) the original CDC DIMFILM has been lost. DIMFILM-77 never actually ran on CDC hardware in its time, so a CDC version is actually a new port.

While getting DIMFILM to run on CDC machines (under NOS specifically) was the main aim, getting a version that would build and run using the GNU G77 compiler was also part of the plan. This would make DIMFILM available on Linux and Windows, which would make it much less likely to die completely in the future. It was also a useful stepping stone to getting the CDC version up and running.

[2] This is the most relevant of a mere handful of mentions of DIMFILM on the Internet according to Google. The others are various papers acknowledging the use of DIMFILM.

Chapter 2

Reviving DIMFILM

2.1 Timescales

I first obtained a copy of the DIMFILM-77 sources in December 2003 from John Gilbert via Adrian Clark. This included drivers Adrian had written for various devices in the mid to late 1980's. These would prove to be very useful.

For several reasons, I didn't really do anything with them until August 2008, I am ashamed to say! Lack of time was the main problem, of course. The source was also in a form that needed a small program written to get it in to a usable form – basically a very cut down version of CDC UPDATE, but different in detail. This meant a little effort was required to get going – enough to keep shelving the project, I am afraid.

The ports to both CDC NOS and G77 for Linux were completed in late November 2008. Various further modifications are of course possible, and not all of the DIMFILM API [1] has actually been tested, but there is reason to believe that it works to a useful extent.

2.2 Problems

Once the `updeck` program had been written to combine input 'deck' files in to compilable Fortran, all went smoothly. Until it was discovered that the font files for DIMFILM were not in the source. This proved to be the major problem in getting DIMFILM going.

The DIMFILM font files were based on the Hershey fonts, but significant additions and alterations had been made to them, I believe. This certainly included adding accent characters which did not exist in the original Hershey data.

The only way forward was to reconstruct DIMFILM compatible font data from Hershey font data. Hershey fonts are available on the Internet in various formats. The one I used is from: `http://local.wasp.uwa.edu.au/~pbourke/dataformats/hershey/` Fortunately, I had done some work with this data for another purpose. With the aid of various tools, this was processed into a

[1] Raster output is not supported by the currently implemented device drivers.

set of fonts which cover those described in the DIMFILM-77 documentation. This is *not* to say that these fonts are the same as DIMFILM font data – even for the non-accent characters (which are entirely missing in the reconstructed font data). Apart from many possible relatively subtle differences, the **marker** and **symbol** fonts will be completely different, with different symbols appearing for each character code in the 'new' fonts when compared with the 'old'. One additional issue is that I don't have any information on what the original DIMFILM fonts looked like – specifically the codes assigned to symbols and markers. The reconstructed fonts do basically work, however.

The other significant problem was the documentation. There is a single manual available for DIMFILM - the *DIMFILM Preliminary User Guide*. In spite of a serious effort by John Gilbert to preserve the documentation (he supplied it in no less than 4 different formats), it could not be straightforwardly used! Neither the Microsoft Word nor Word Star files could be opened using any of the conversion programs I could find. The `.txt` files had a mixture of line lengths and were difficult to read. The `.sg` format files were input to a **runoff** style formatter. But I never located any that understood them. Actually, this style of text formatter appears to have almost entirely disappeared. Only DSR [2] is still (barely) alive. In the end, I wrote a simple program that processed `.sg` files into HTML and from that to PDF via Easy Software Product's **HTMLDOC**. This formatting program – `sgformat` – is actually too simple and is basically incorrect in how it formats paragraphs. The result is usable, however, and I fear no time will be available to improve it.

The documentation problem is a reminder that digital data is actually very fragile and often relies on a surprisingly large ecosystem that might change (or evaporate entirely) surprisingly rapidly. This is perhaps the strongest argument for open source ...

2.3 DIMFILM Source Code Structure

There are two major elements to DIMFILM:

- DIMFILM proper. This presents the API accessed by user programs, contains all device independent functionality and calls on a 'device driver' level external to DIMFILM proper to get output on one or more devices (supplying normalized device coordinates, etc., to this level). This part of DIMFILM (much the largest) is written in highly portable Fortran-77 and caused no difficulties with either the G77 or CDC port.

- Device Driver and System Interfaces. This needs to provide one or more output devices, as well as abstracting some inevitably system dependent functions. This part needs to be coded to support the desired output device(s) (obviously!) and will be specific to the operating system being used as well.

In the G77 port, no changes were made to DIMFILM proper. However, the CDC port did expose some uninitialized variables, and where located, these were set to initial values by modifying the code. These extra initializations were not incorporated in the G77 port at all. By presetting core to all zeros, the uninitialized variables issue disappears, and, although all these issues may have

[2] Digital Standard Runoff

been tracked down and fixed, presetting to zero on the CDC system is recommended. Problems may yet manifest themselves under G77, where no modifications have been made.

In the G77 port, DIMFILM proper is in the file `dimfilm.f` The Device Driver and System Interface layer is in the file `dimsg77.f`. The font data for all fonts is in the file `DIMFONT.DIM`. This font file must currently be in the local directory in which programs using DIMFILM are run. It would be trivial to change this (see below), but I have no particular opinion on where DIMFILM should be installed, so I haven't done anything more sensible yet.

In the CDC port, all code is in a single file: `dimfilm.txt`, together with a job deck at the top to compile the code and build a library from it. To build the library, just drop this file in the card reader and stand well back!

Once I had processed the original source code through the `updeck` program, no more use was made of the originally supplied 'deck' files. If DIMFILM were to be under active development, these 'deck' files should definitely be used for edits and builds. However, I don't intend to add anything to DIMFILM.

To create DIMFILM proper, the decks starting with `*DECK,DIMFILM` in `ZZZZZZ.DK` need to be assembled. The `updeck` program does this by recursive file inclusion driven by `CALL` and `DECK` directives in the deck (`.DK`) files.

2.4 Device Drivers

Both the G77 and CDC ports have two devices available - DEPBIN and DEPCOL. Both output Encapsulated PostScript on G77 and something that can be easily processed into EPSF on the CDC (the actual processing to EPSF must be done on the host, since CDC machines do not support lower case characters — actually this is not *quite* true, but it very nearly is!)

Both of these drivers are based on Adrian Clark's EPSF driver for his VAX/VMS port of DIMFILM (c. 1988). This was very helpful, and had pretty much the same functionality as DEPBIN. Substantial modifications have been made, though, to remove VMS-isms and for other reasons.

`CALL DEPBIN` selects the binary EPSF output device. Any non-black output will result in a black line drawn on a white paper background (if this sounds backwards, it actually works out to be quite sensible). Note you only have two colours with DEPBIN: black and white!

`CALL DEPCOL` selects the colour EPSF output device. This supports colour vector graphics using any of the DIMFILM colour models (only RGB has been tried, though). There is one issue, though: if you plot everything in *black and nothing else* you will get *no output!* There is a hidden assumption in DIMFILM that the background is black, and if all the plotting is done in black too... why not optimize by doing nothing! The easy answer is to use a very dark colour — but not entirely black!

The size and position on the page of the output plot is controlled by:
`CALL EPSDIM( REAL XSIZE, REAL YSIZE, REAL XOFFST, REAL YOFFST )`
This specifies the size of the plot and its offset from the bottom left of the page. All dimensions

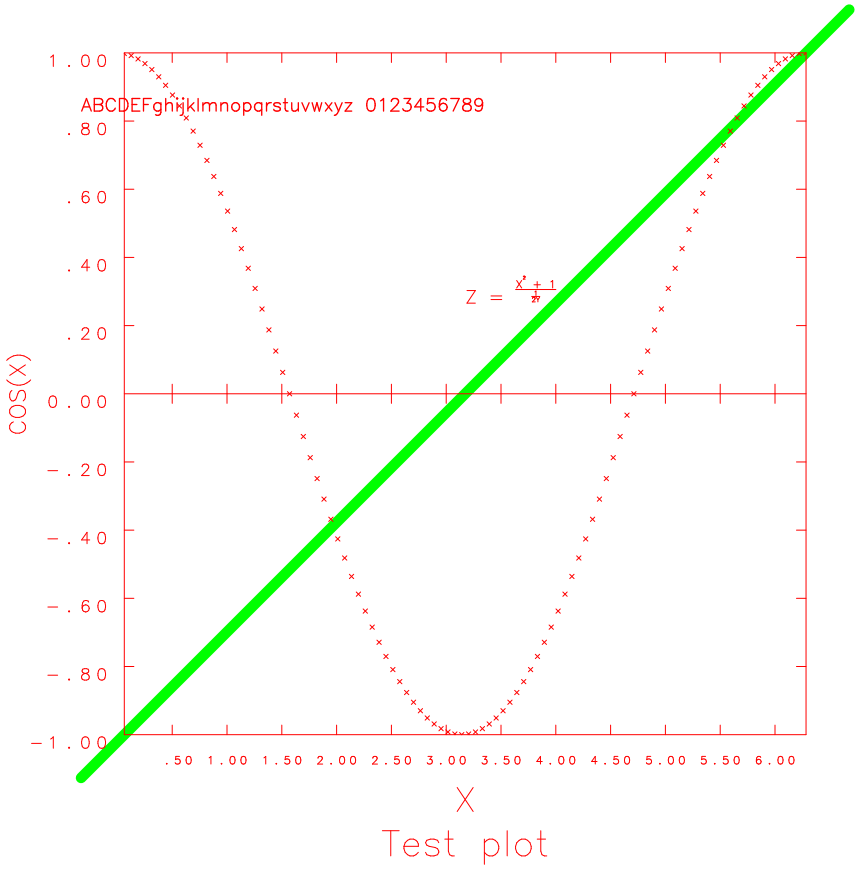


Figure 2.1: Example plot using the DEPCOL driver.

are in inches.

`CALL EPSNAM( CHARACTER*(*) NAME )`
specifies a name for the output files (on G77) on the single output file (for CDC). On G77, each frame is output in a separate file: `NAME001.eps` with the 3 digit sequence number being incremented for each frame.

In the CDC port, the output is written to a local file called NAME, which contains all the frames. Each frame begins with a `++PLOTFILE NAME001` and ends with `++EOF`. An ancilliary program called `cdcplot` splits this into separate EPS files, and inserts appropriate lower case PostScript definitions to make the output 'real' EPS. There is another assumption that the CDC plot file is output through a card punch (which results in all alphabetic characters appearing in lower case). The EPS files output from `cdcplot` are called `NAME001.eps`, etc., and can be used as any other EPS files.

2.5 CDC Plotfile Details

Below is an extract from an example multi-frame CDC plotfile.

```
 ++plotfile fon001
 ++epsheader
   420.645000    57.345000 m
      .100000      .100000      .100000 c
      .500000 w
   420.645000    57.660000   420.645000    57.345000 l
   ...
   339.060000   417.495000   339.375000   417.180000 l
   338.745000   418.125000   339.060000   417.495000 l
 ++epstrailer
 ++boundingbox     14    14   518   518
 ++eof
 ++plotfile fon002
 ...
   338.745000   418.125000   339.060000   417.495000 l
 ++epstrailer
 ++boundingbox     14    14   518   518
 ++eof
 ++plotfile fon025
 ++epsheader
~
}
```

The `cdcplot` program simply opens a new output file when it encounters `++plotfile` using the name following `++plotfile` as a basis and adding the `.eps` extension. It closes the output file when it encounters `++eof`. It inserts the following text in the output file when it encounters `++epsheader` (in place of `++epsheader`):

```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: CDC 6000 DIMFILM plot file.
%%BoundingBox: (atend)
%%EndComments
/l {moveto lineto currentpoint stroke moveto} def
/m {newpath moveto} def
/c {setrgbcolor} def
/w {setlinewidth} def
%%EndProlog
 1 setlinecap
```

When it encounters `++epstrailer`, it inserts in its place:

```
showpage
%%Trailer
```

It substitutes %%BoundingBox: in place of ++boundingbox, preserving the coordinates, of course. Note that the CDC plotfile will contain an incomplete last frame. The cdcplot program detects this and deletes the partial output file.

Note that many tools do not like %%BoundingBox: (atend) ...which is a shame because the G77 EPS files also use this! It is difficult to do otherwise, in fact.

2.6 What has been lost?

The original CDC version of DIMFILM is gone forever. From an historical point of view, that is a shame.

All the device drivers described in the *DIMFILM Preliminary User Guide* (PUG) are unavailable in the 'revived' version — needless to say. All the driver code was inevitably system and device specific. Hence D35, D35P, D35C, D16, D16P and D16C do not exist and cannot be used.

None of the Raster Plotting facilities described in Part 2, Chapter 3 of the PUG have been implemented, although, as far as I know, they could be by adding some more functionality to DEPCOL (it probably wouldn't be useful to try to do this to DEPBIN — although there is no reason it couldn't be done).

Everything in Part 5, Appendix 1 (Using DIMFILM at ULCC) is obviously irrelevant — although it makes interesting reading. Likewise, Part 5, Appendix 3 (Device Specifications) is irrelevant, but interesting. It gives a good description of the Dicomed D48C film recorder (which I was familar with in a very different environment in the second half of the 1980's).

The accent characters in alphabetic fonts as well as the whole of the original symbol and marker fonts have been lost. See the next chapter for details of what is now available.

Chapter 3

DIMFILM Fonts

3.1 Summary

The fonts now available with DIMFILM are plotted out in the following pages. Something corresponding to each of the original DIMFILM fonts has been supplied. However — there are some problems, even after accepting that the original accent characters are missing entirely. In the alphabetic fonts, not all the printable codes have characters assigned to them. This is only in the region below ASCII 40 (DIMFILM code 9 – an open parenthesis). The contents of the symbol fonts are also somewhat random and will certainly *not* correspond to the original fonts. The worst case is the marker font, which has nothing assigned for DIMFILM codes 1 and 5! Oh well. . .

3.2 Font Selection

Fonts are selected in DIMFILM using:
`CALL LDABET( INTEGER ALPHABET, INTEGER FONTNUM )`
where `ALPHABET` is the memory resident alphabetic font (1, 2 or 3) that is to be loaded, and `FONTNUM` is a number associated with each available font. By default, text is plotted with alphabetic font 1. Which memory resident font is used is selected by `CALL ABET( ALPHABET )`.

In addition to the alphabetic fonts, there are separate memory resident fonts for *symbols* and *markers*, the latter being used to mark points on graphs and so on. Fonts can be loaded into the memory resident symbol font using `CALL LDSYM( FONTNUM )` and into the memory resident marker set using `CALL LDMARK( FONTNUM )`. However, only one supplied font is really suitable for markers!

3.3 Initial Memory Resident Fonts

DIMFILM preloads the memory resident fonts shown in Table 3.1. *Simplex* means drawn with a single stroke, *Complex* indicates that the character is drawn with two strokes to give it some ‘width’, and *Triplex* with three strokes to further increase its ‘bulk’.

In-memory Font	Font Number	Description
Alphabetic 1	11	Sans, English, Simplex
Alphabetic 2	15	Serif, English, Complex
Alphabetic 3	1015	Serif, Greek, Complex
Symbol	8001	Mathematical
Marker	9001	Principal Marker Set

Table 3.1: Default Memory Resident Fonts

Field	Description
1	Font number
2	Always 0
3	Always 0
4	Number of characters in the font
5	Minimum X relative to center in font
6	Maximum X relative to center in font
7	Baseline Y coordinate
8	Height (Y range) of font
9	Line spacing (Y step) for font

Table 3.2: Font Header Fields

3.4 Font File Format

The font file format now used with DIMFILM is intended to be portable and self contained rather than efficient. The font data for all fonts is in a single FORTRAN formatted file using only characters from the strictly standard FORTRAN-77 character set. Each font begins with a line naming the following font. When parsing the font file, the only significant thing about this is the Z in column 1 which marks the start of a font definition.

```
Z FONT FILE F.SIMPLE.DIM
   11   0   0  96   -15    15     0    32    33
       3   -10     9     0     0
      21    -5     5     0     0
      33    -8     8     0     0
      57   -10    11     0     0
     111   -10    10     0     0
     175   -12    12    -9    12
     245   -13    13    -9    12
     ...
```

The next line contains 10 integers, the meaning of which is shown in Table 3.2. This is followed by an ‘index’ with one entry for each character in the file. Each entry has 5 integers, the meaning of which is shown in Table 3.3.

Field	Description
1	Index of *end* of character in vector data
2	Minimum X for character
3	Maximum X for character
4	Minimum Y for character
5	Maximum Y for chatacter

Table 3.3: Font Index Entry Fields

```
    ...
    2521    -7     7   -16    16
    2569   -12    12    -3     3
    2573   -10     9     0     0
    2574
9C0013009C00000000009C0000000000000000000009C00000000009C00000000009C00
000000009C00000000009C00000000009C00000000009C00000000009C00000000009C00
0000000000000000000000000000000000000000000000000000000000000000000000
000000009C00090CF7F79C00FC0CFE0AFE08FD06FB05F905F707F709F80BFA0CFC0CFE0B
010A040A070B090C9C0005FE03FD02FB02F904F706F708F809FA09FC07FE05FE9C000A03
```

The first field of the last entry in the index table (2574 in the above example) is the total number of 8 bit bytes in the vector data for the font, N.

After the index comes the font vector data, encoded as hexadecimal characters, maximum of 72 characters per line. These encode 2's complement signed integers in the range -128 to +127. Each 8 bit byte of vector data requires 2 characters. The number of lines required to hold the vector data in the font file is:

$$(\mathrm{N}-1)/36+1$$

Below, we assume the vector data has been read into a zero based integer array, while the 'index' table has been read into a one based array (nothing like consistency, is there?). Pairs of hexadecimal characters are used to assemble unsigned integers, V_u in the range 0 to 255, which are converted to signed integers using:
$V = V_\mathrm{u} - 256$ if $V_\mathrm{u} > 127$, or
$V = V_\mathrm{u}$ otherwise.
My apologies if this is obvious, but such simple things are often confusing (at least to me).

To draw a character, first find its entry in the 'index' table I using its character number, C. If A is the ASCII code for the character, then

$$\mathrm{C} = \mathrm{A} - 31$$

DIMFILM defines a system dependent routine (`DFXM01`) which must convert an internal character representation to a DIMFILM character number, which is, in fact, C. Because of this, systems which do not use ASCII (e.g. the CDC systems) have no particular difficulties. The *end* of character's vector data is then given by the first field of the 'index' table entry $\mathrm{I}(\mathrm{C})$. The *start* of the character's vector data is given by $\mathrm{I}(\mathrm{C}-1)$ if $\mathrm{C} > 1$ or 0 if $\mathrm{C} = 1$. The character data is drawn using the vectors $\mathrm{I}(\mathrm{C}-1)+1 \rightarrow \mathrm{I}(\mathrm{C})$ (or $0 \rightarrow \mathrm{I}(\mathrm{C})$).

The range of coordinates in both x and y is -99 to +99. Values outside this range are used for control purposes. DIMFILM defines many of these (see subroutine `DFX202`), but in the 'reconstructed fonts' we only use one. The pair $(-100, 0)$ is used for *pen up*.

The font file is read by DIMFILM system dependent subroutine `DFXM11`.

3.5 Complete Font Tables

The full set of fonts available is shown in Table 3.4. Each of these fonts is shown in full on the following pages. The red digits around the edge of the grid in which the characters are plotted gives the ASCII code for the character (the idea here is that you can see what the character is 'supposed to be' by referring to a table of ASCII characters — it has no other significance). The pale digit beside each character is the DIMFILM character number C.

Font Number	Description
Sans English	
11	Simplex
12	Duplex
111	Mono / Simplex
Serif English	
14	Small Complex
15	Complex
16	Triplex
Serif Italic English	
24	Small Complex Italic
25	Complex Italic
26	Triplex Italic
Script	
31	Simplex Script
35	Complex Script
Gothic	
46	Gothic English Triplex
56	Gothic German Triplex
66	Gothic Italian Triplex
Sans Greek	
1011	Greek Simplex
Serif Greek	
1014	Greek Small Complex
1015	Greek Complex
Serif Cyrillic	
2015	Cyrillic Complex
Symbols	
8001	Mathematical
8002	Cartographic
8003	Astronomical
8004	Astrological
8005	Musical
Markers	
9001	Principal Marker Set

Table 3.4: DIMFILM Fonts

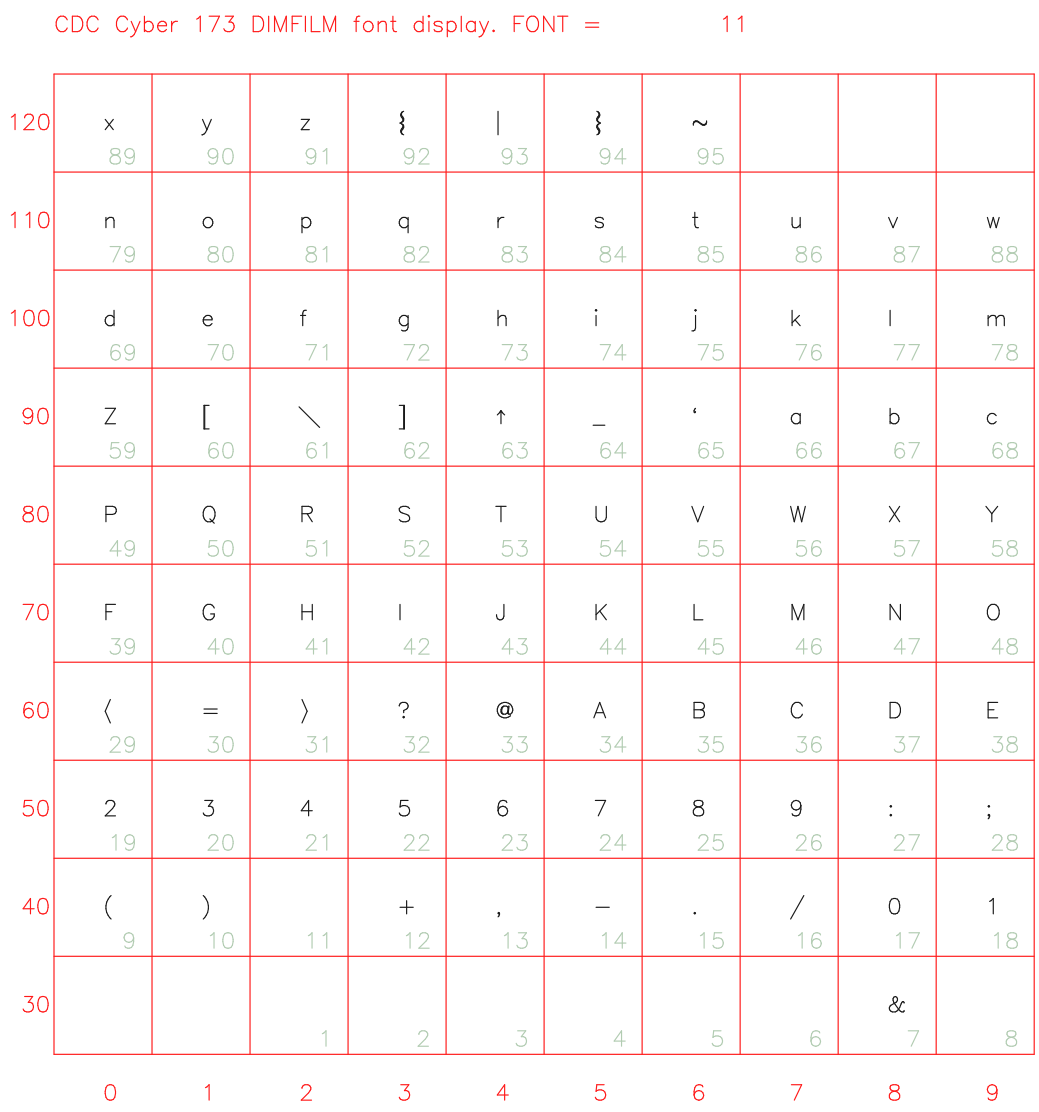


Figure 3.1: DIMFILM font 11 (Sans, English, Simplex, Alphabetic-1 default)

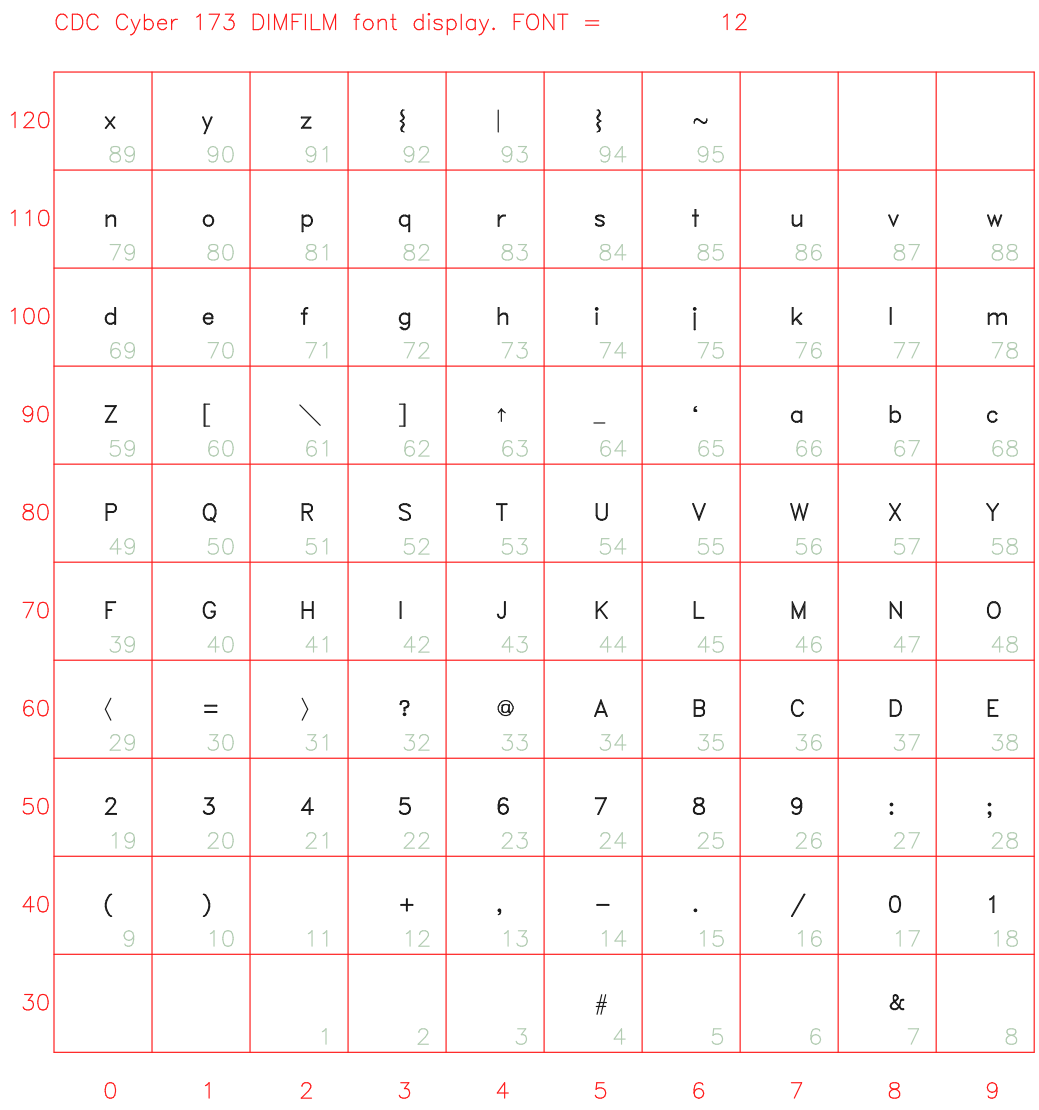


Figure 3.2: DIMFILM font 12 (Sans, English, Duplex)

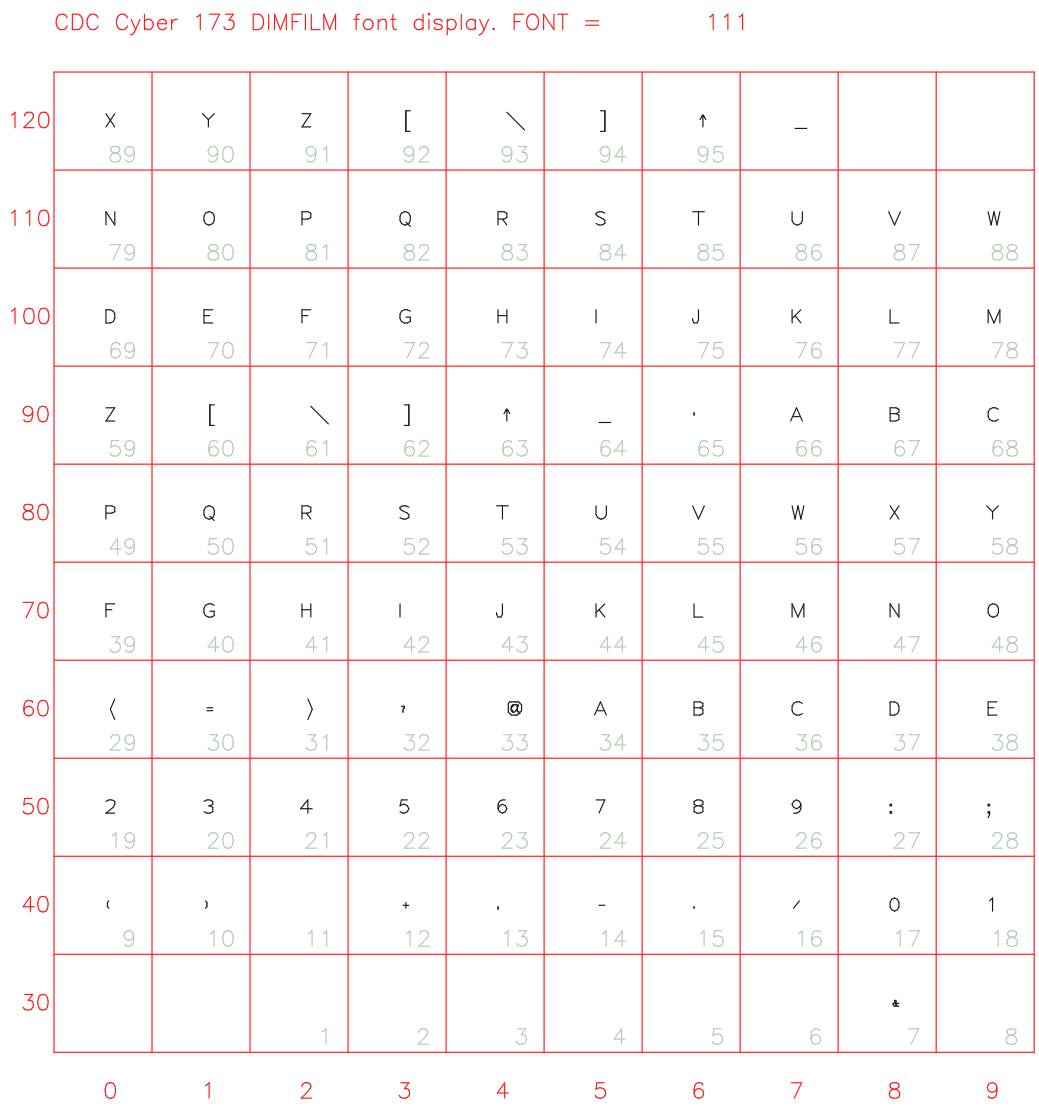


Figure 3.3: DIMFILM font 111 (Sans, English, Mono/Simplex)

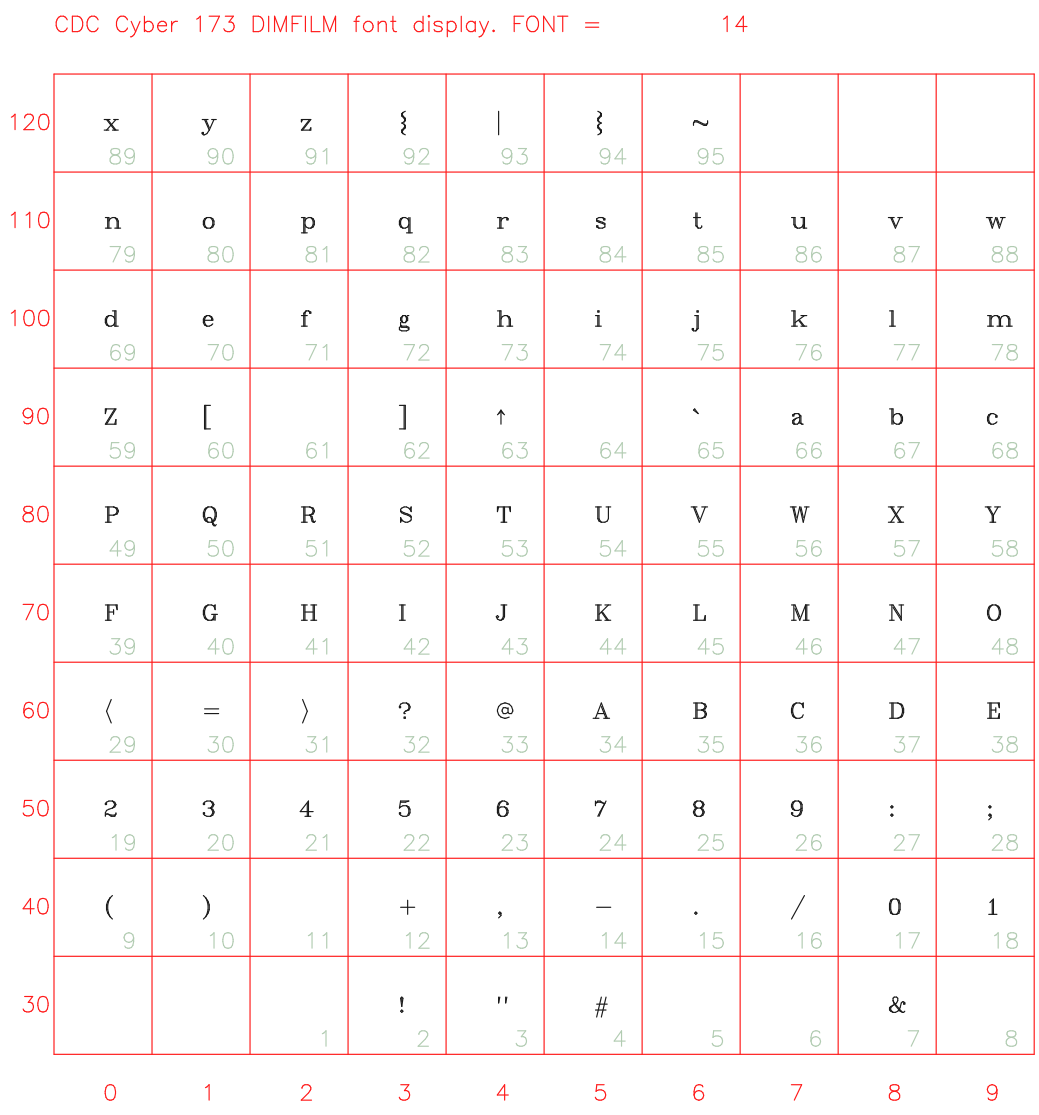


Figure 3.4: DIMFILM font 14 (Serif, English, Small Complex)

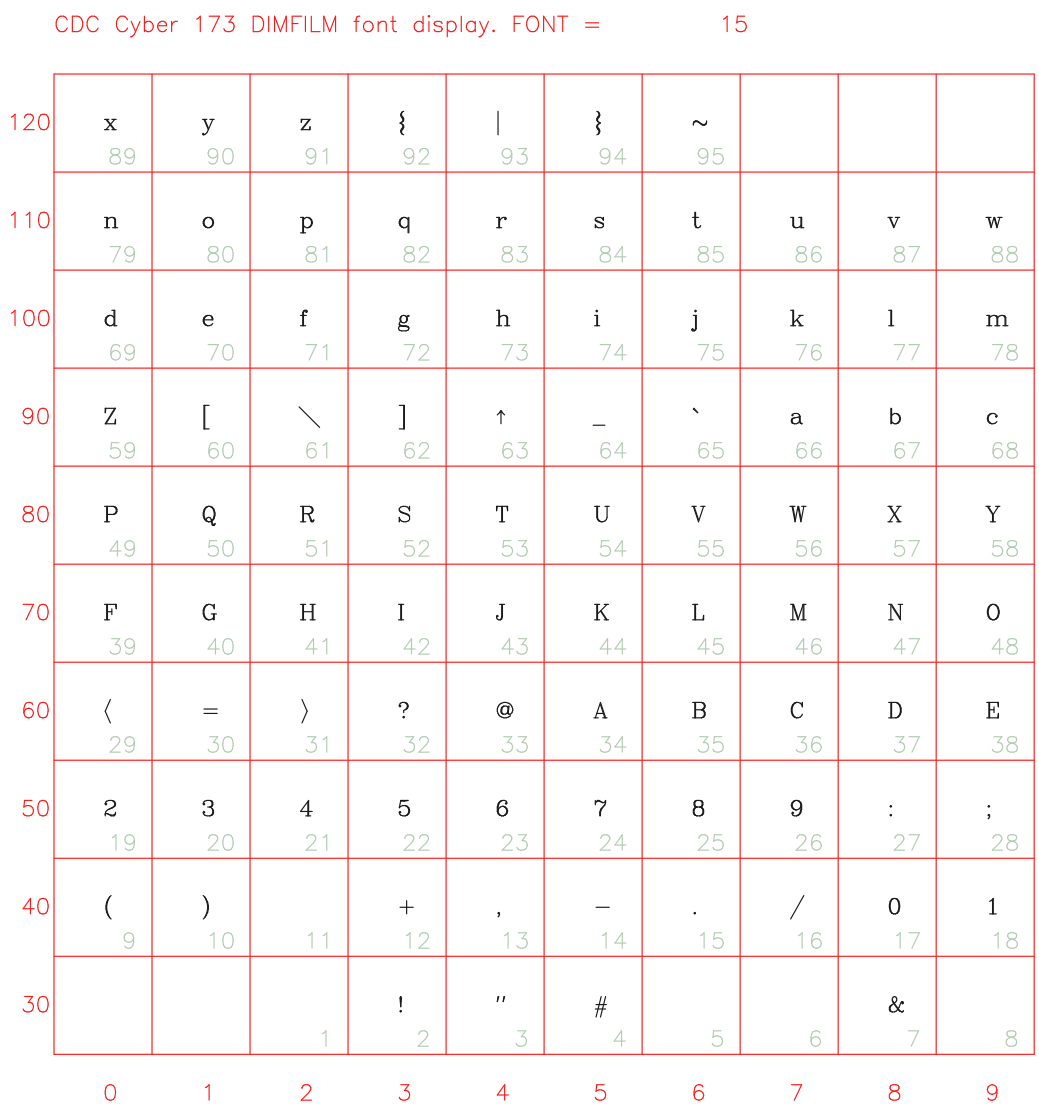


Figure 3.5: DIMFILM font 15 (Serif, English, Complex, Alphabetic-2 default)

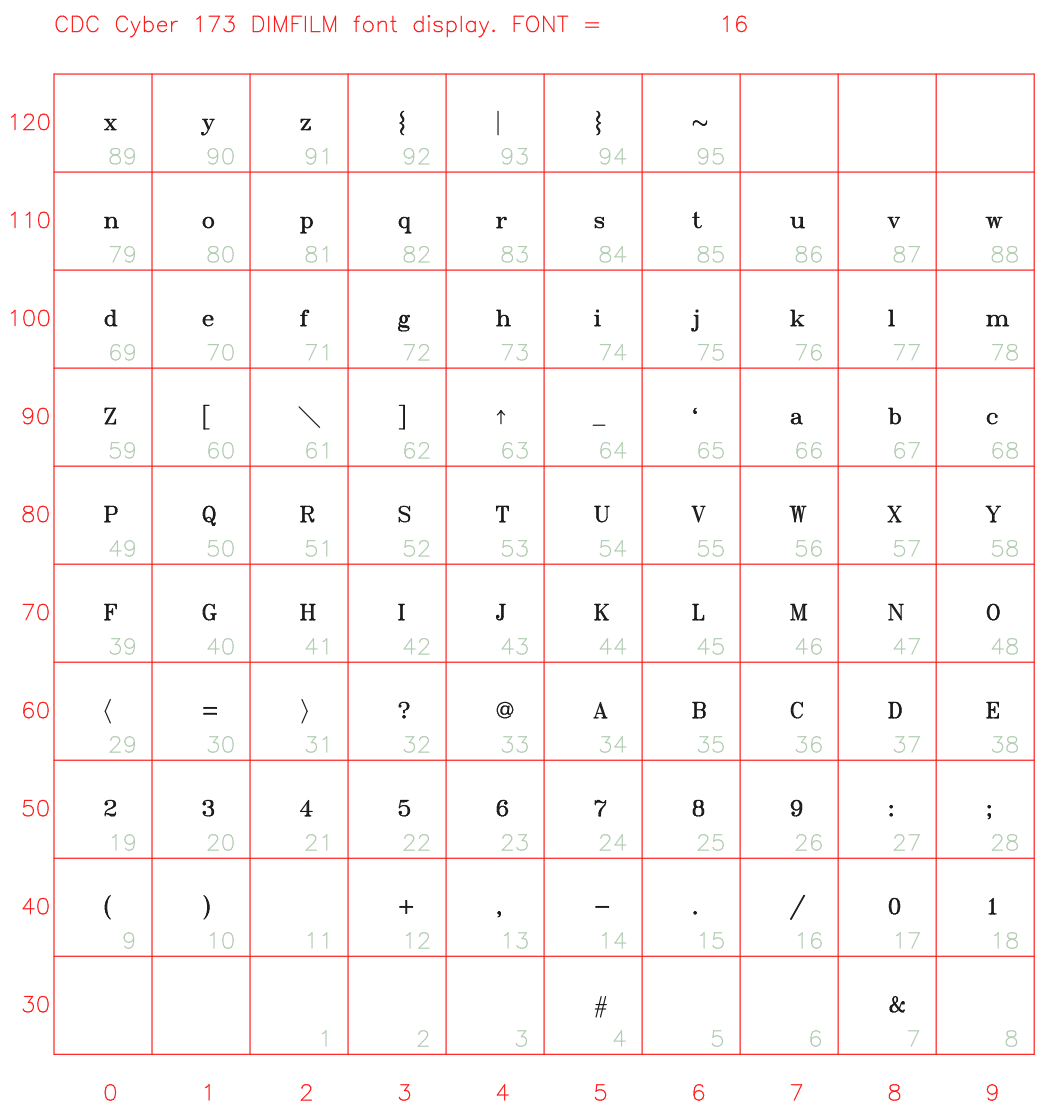


Figure 3.6: DIMFILM font 16 (Serif, English, Triplex)

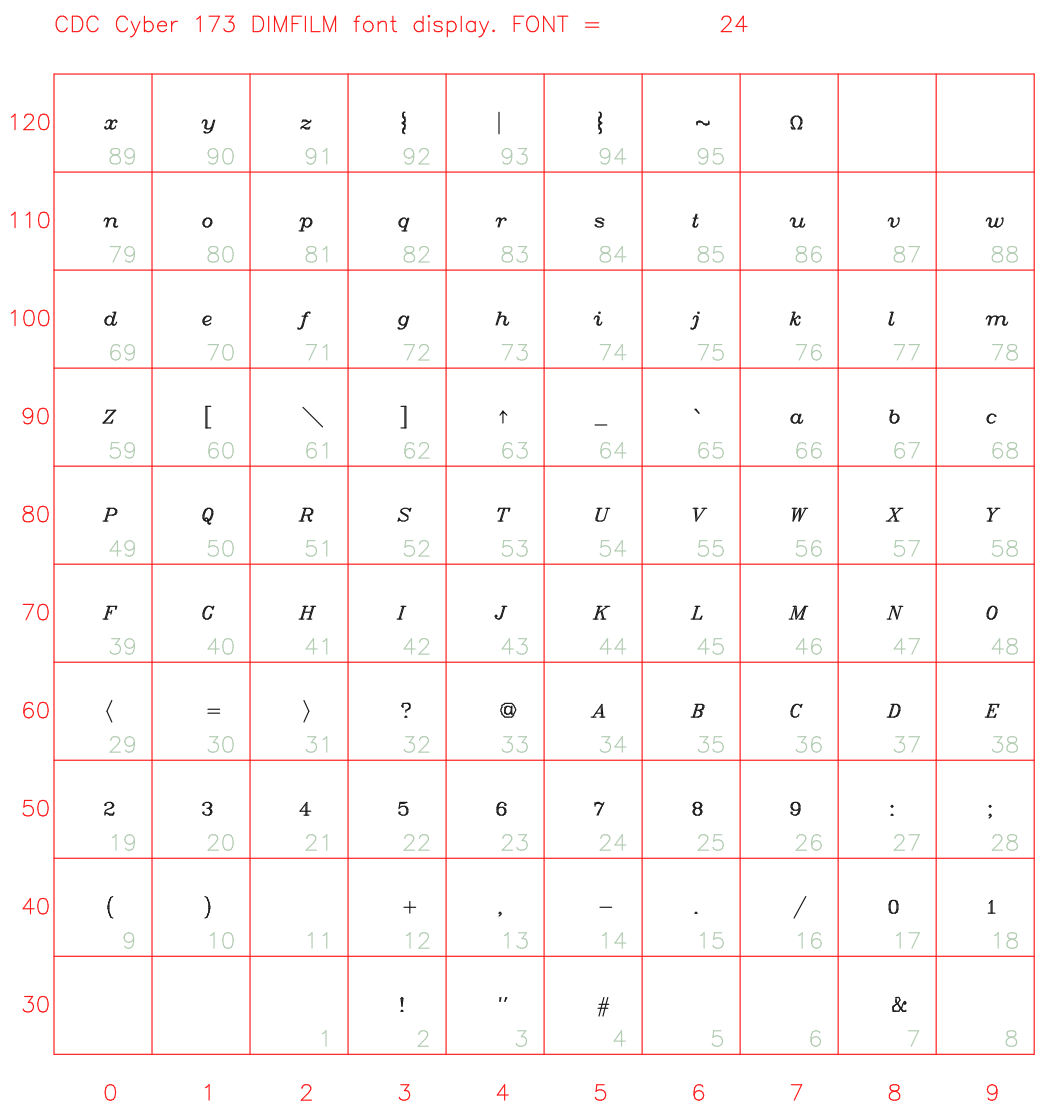


Figure 3.7: DIMFILM font 24 (Serif, English, Italic, Small Complex)

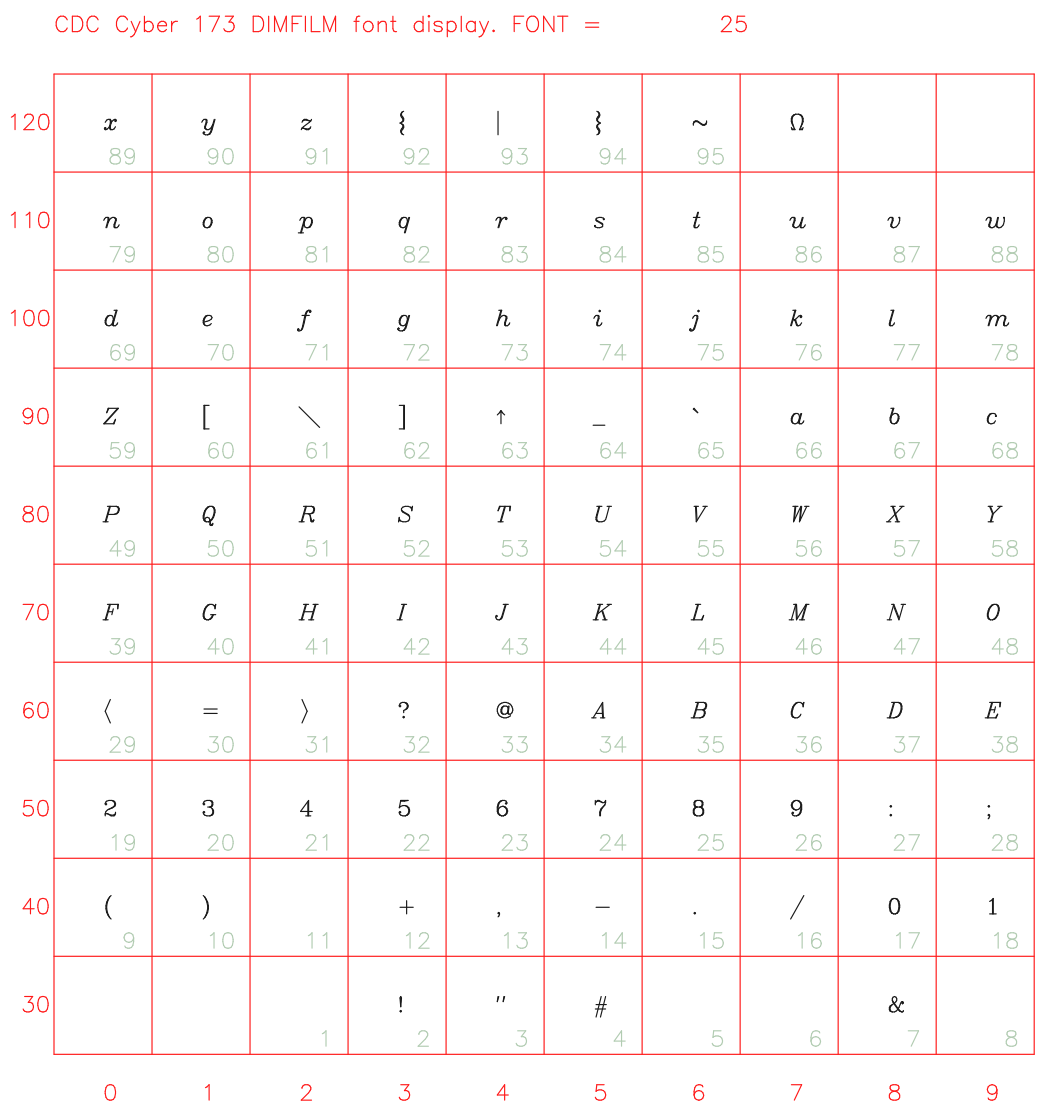


Figure 3.8: DIMFILM font 25 (Serif, English, Italic, Complex)

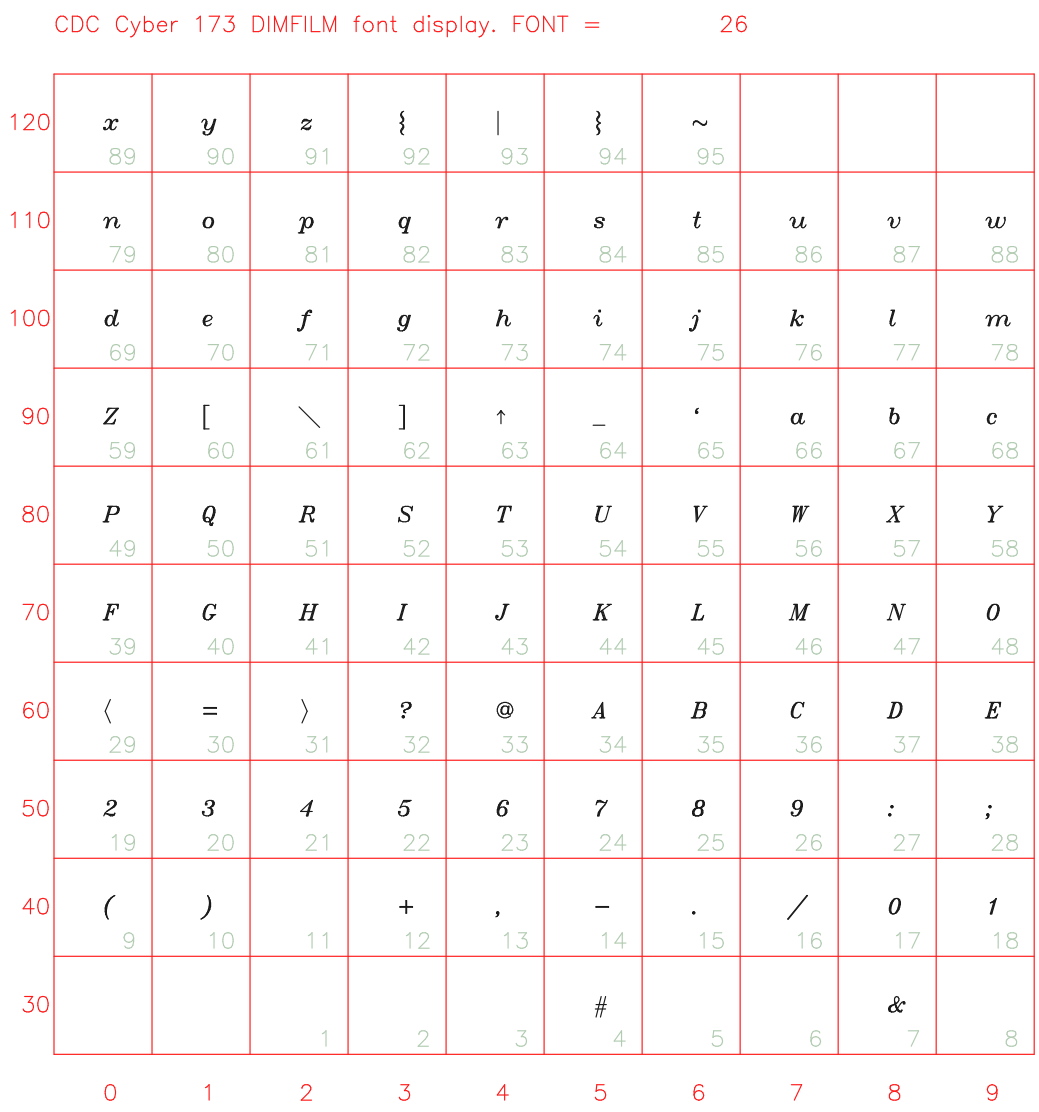


Figure 3.9: DIMFILM font 26 (Serif, English, Italic, Triplex)

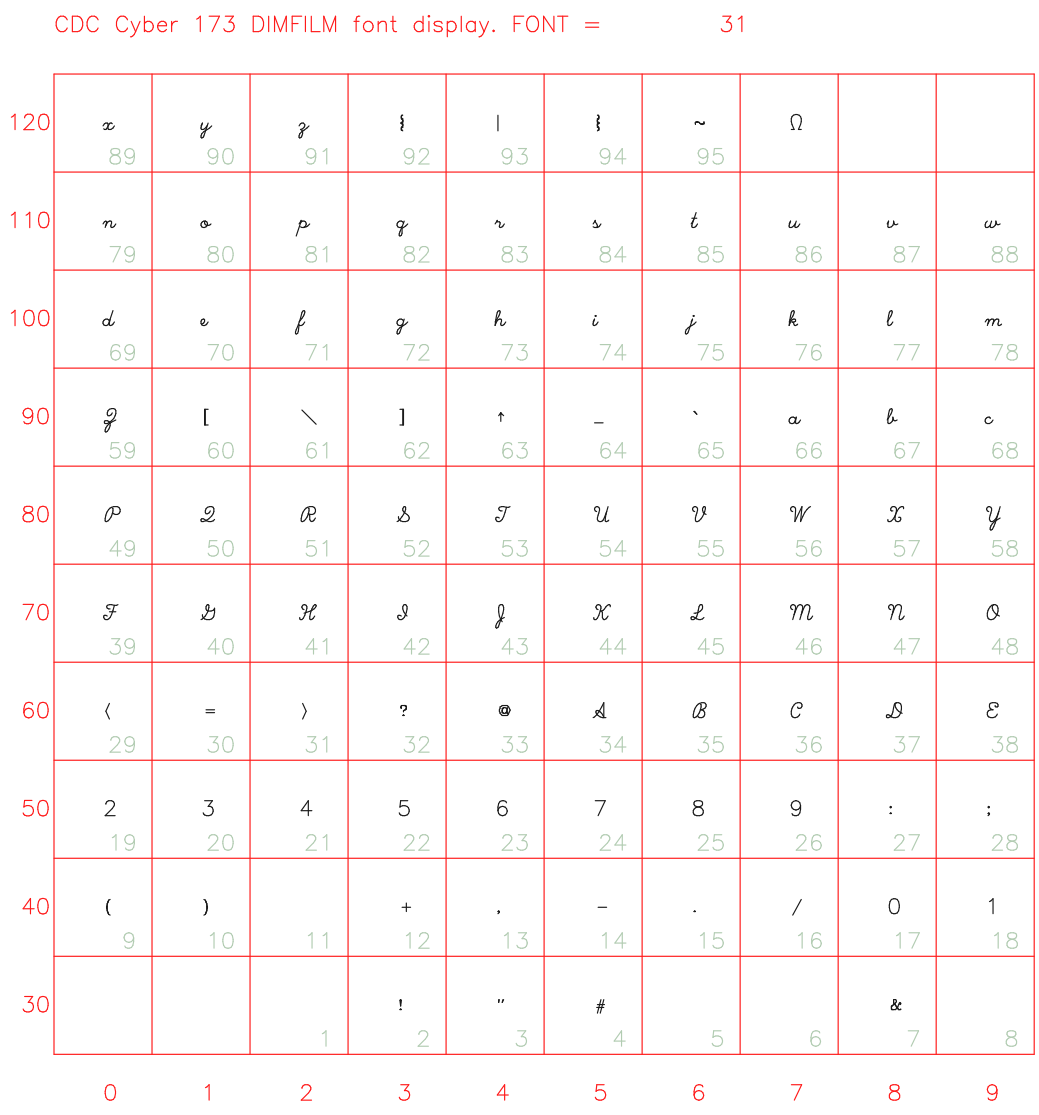


Figure 3.10: DIMFILM font 31 (Script, English, Simplex)

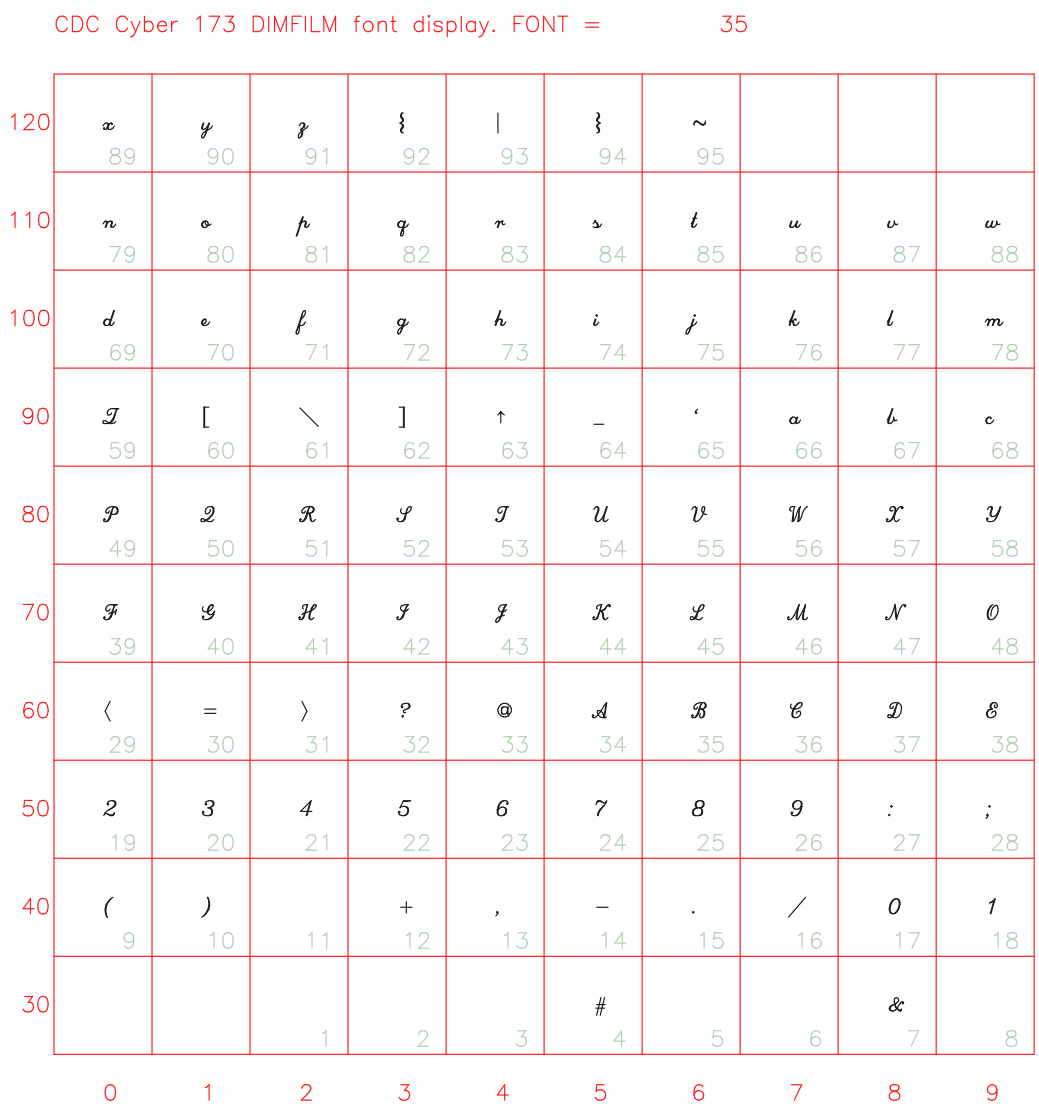


Figure 3.11: DIMFILM font 35 (Script, English, Complex)

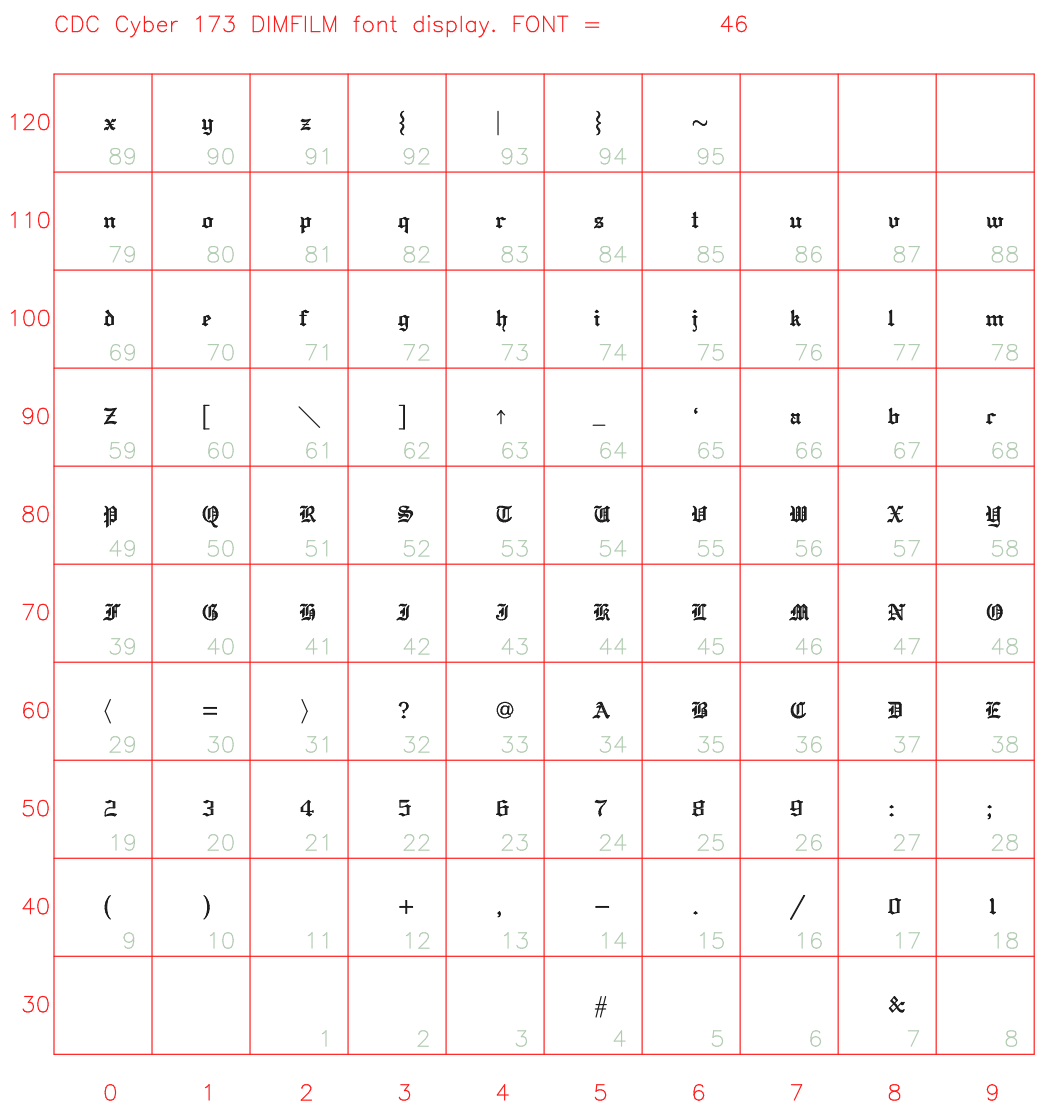


Figure 3.12: DIMFILM font 46 (Gothic, English, Triplex)

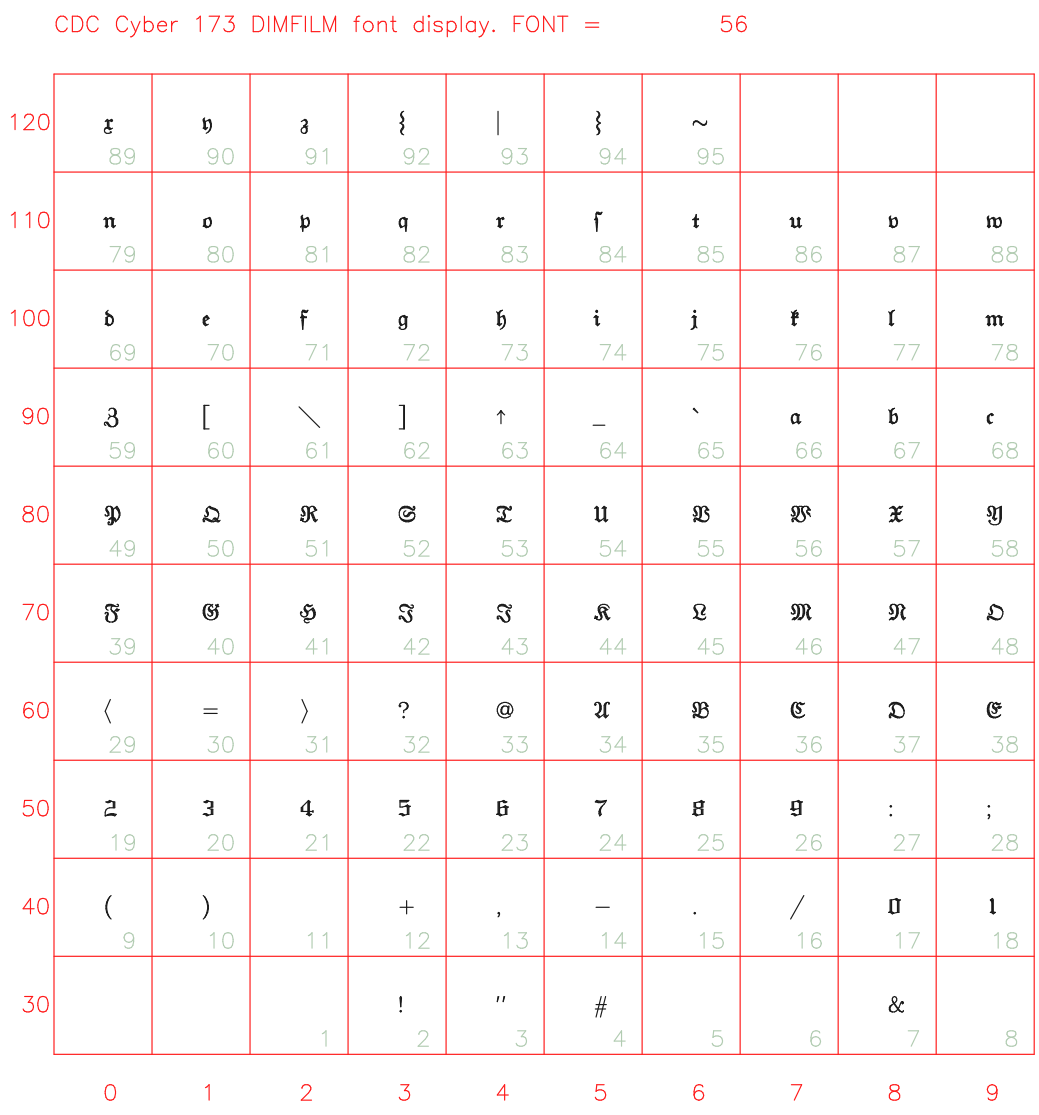


Figure 3.13: DIMFILM font 56 (Gothic, German, Triplex)

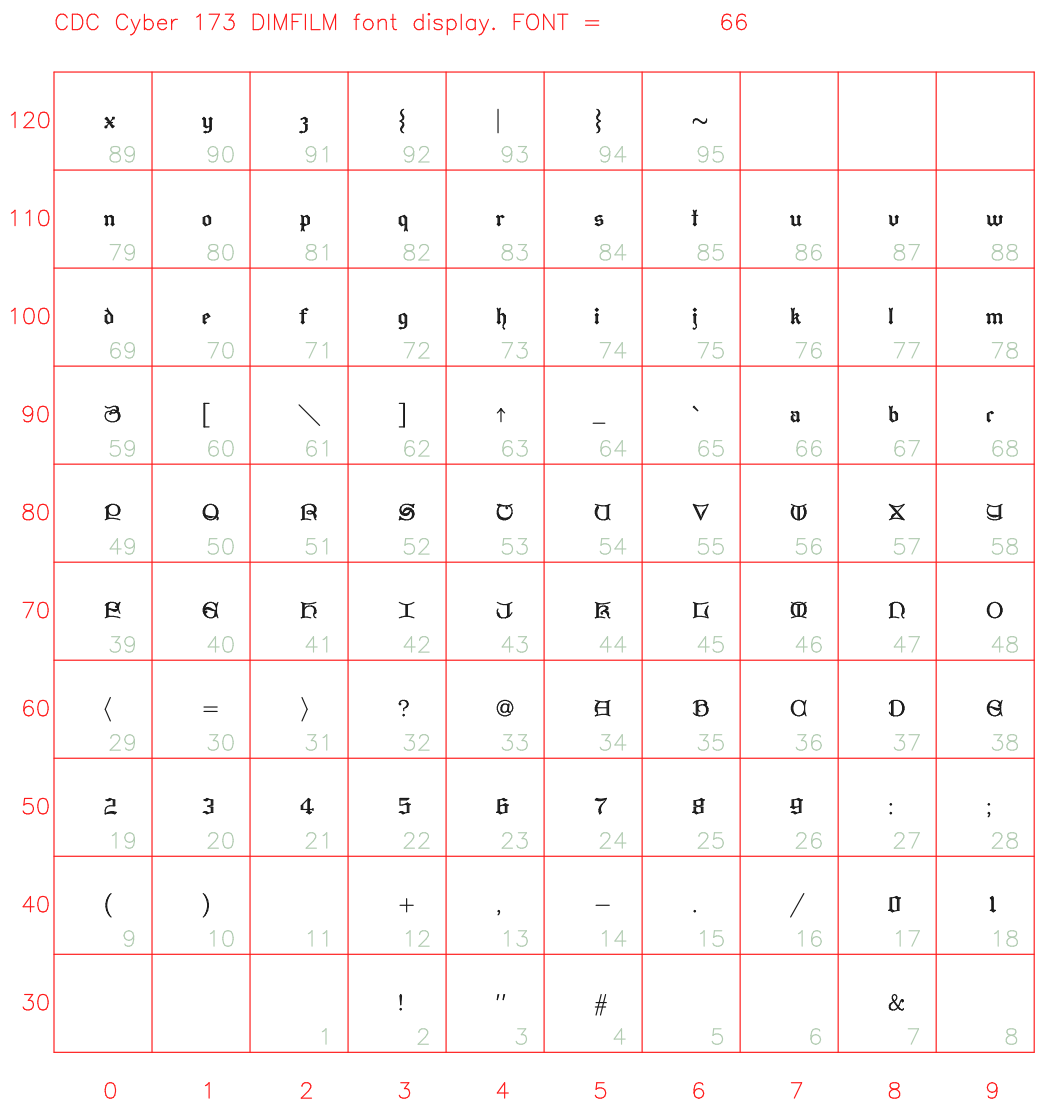


Figure 3.14: DIMFILM font 66 (Gothic, Italian, Triplex)

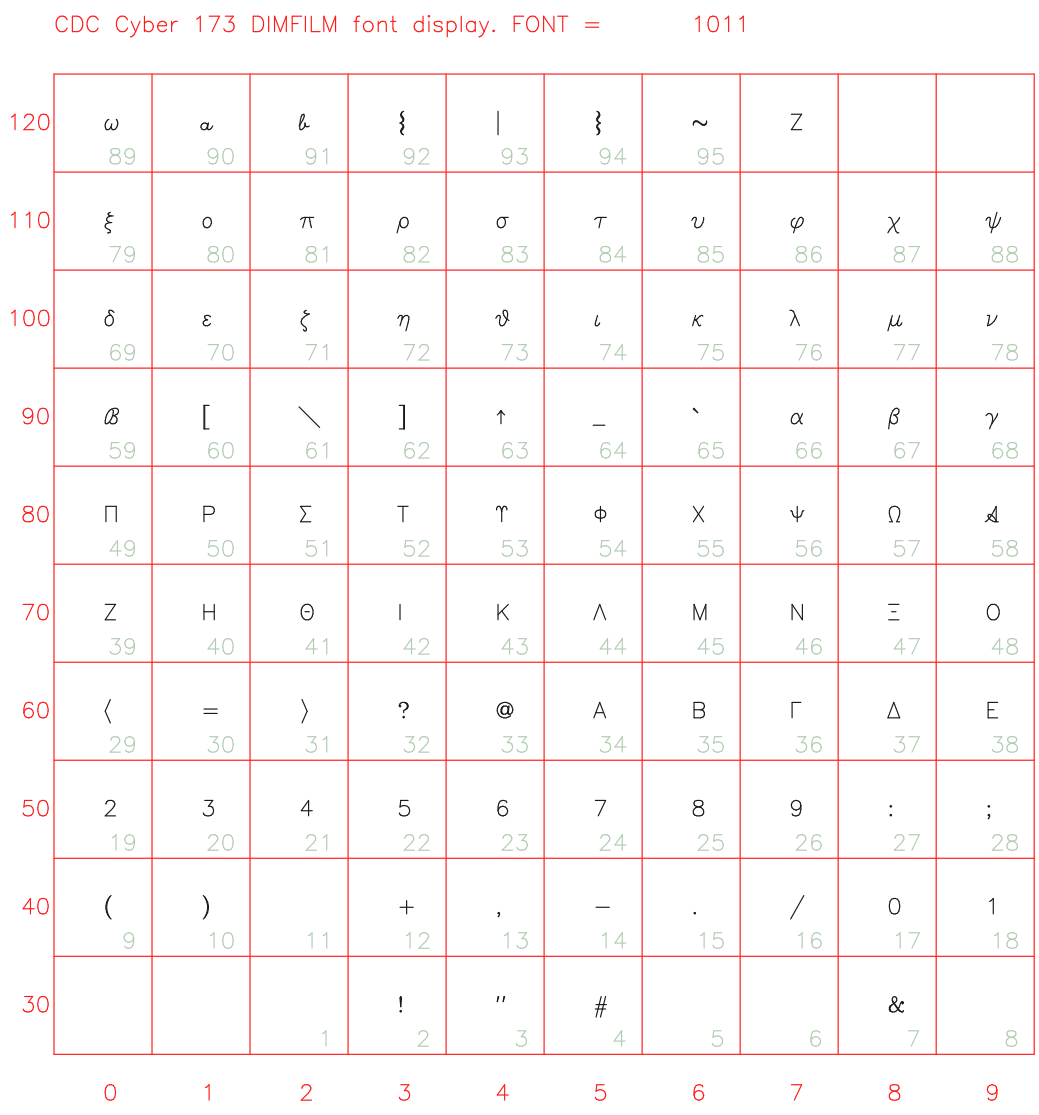


Figure 3.15: DIMFILM font 1011 (Sans, Greek, Simplex)

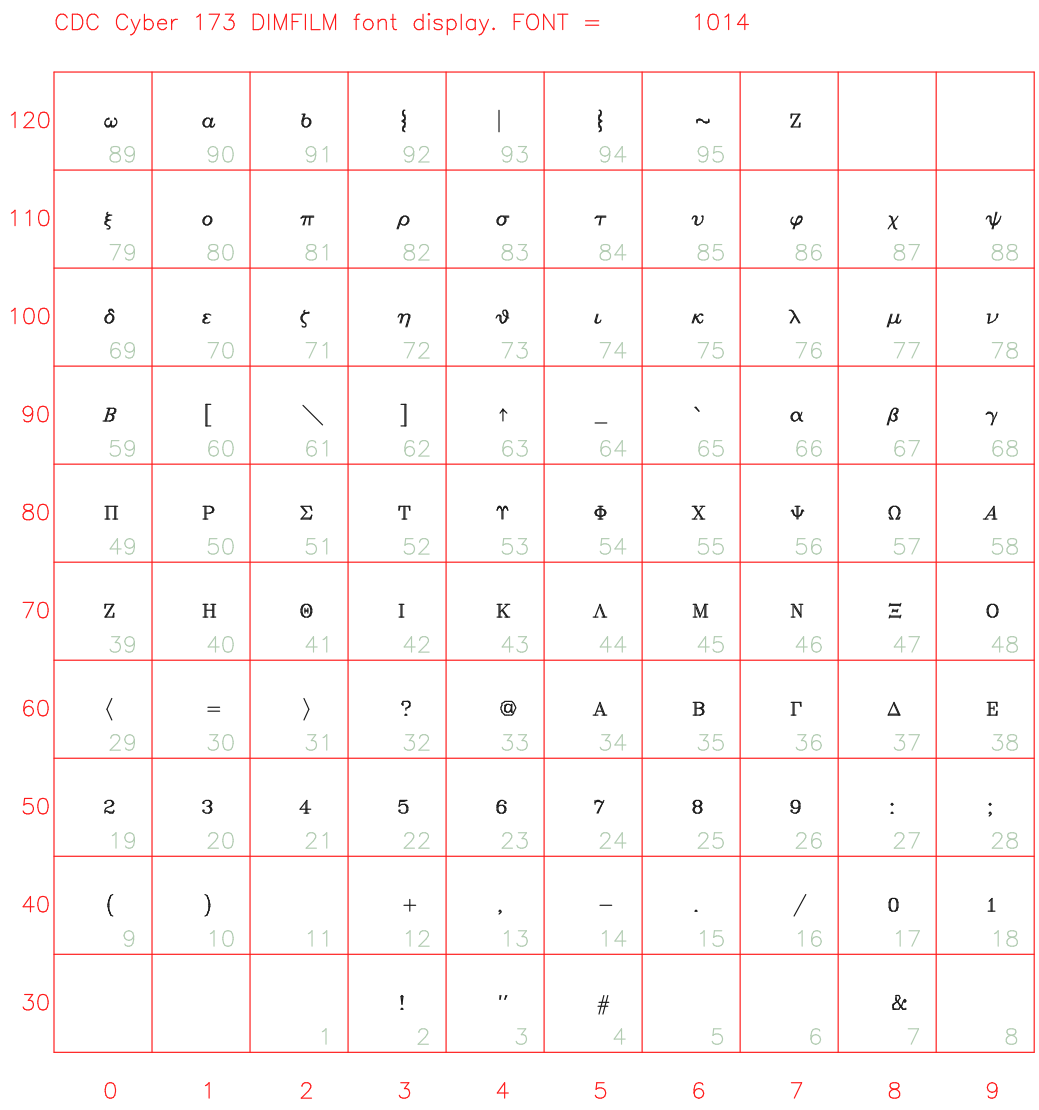


Figure 3.16: DIMFILM font 1014 (Serif, Greek, Small Complex)

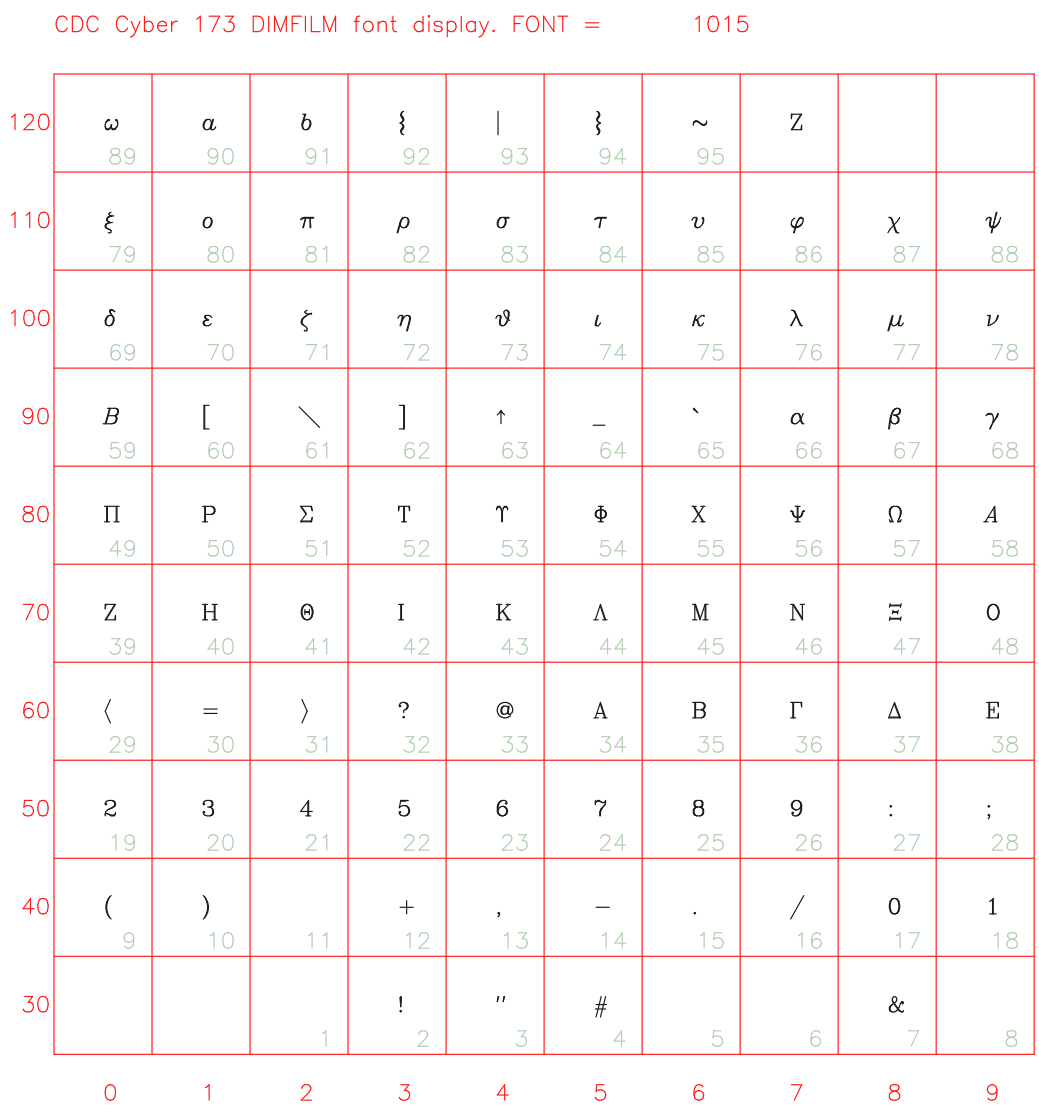


Figure 3.17: DIMFILM font 1015 (Serif, Greek, Complex, Alphabetic-3 default)

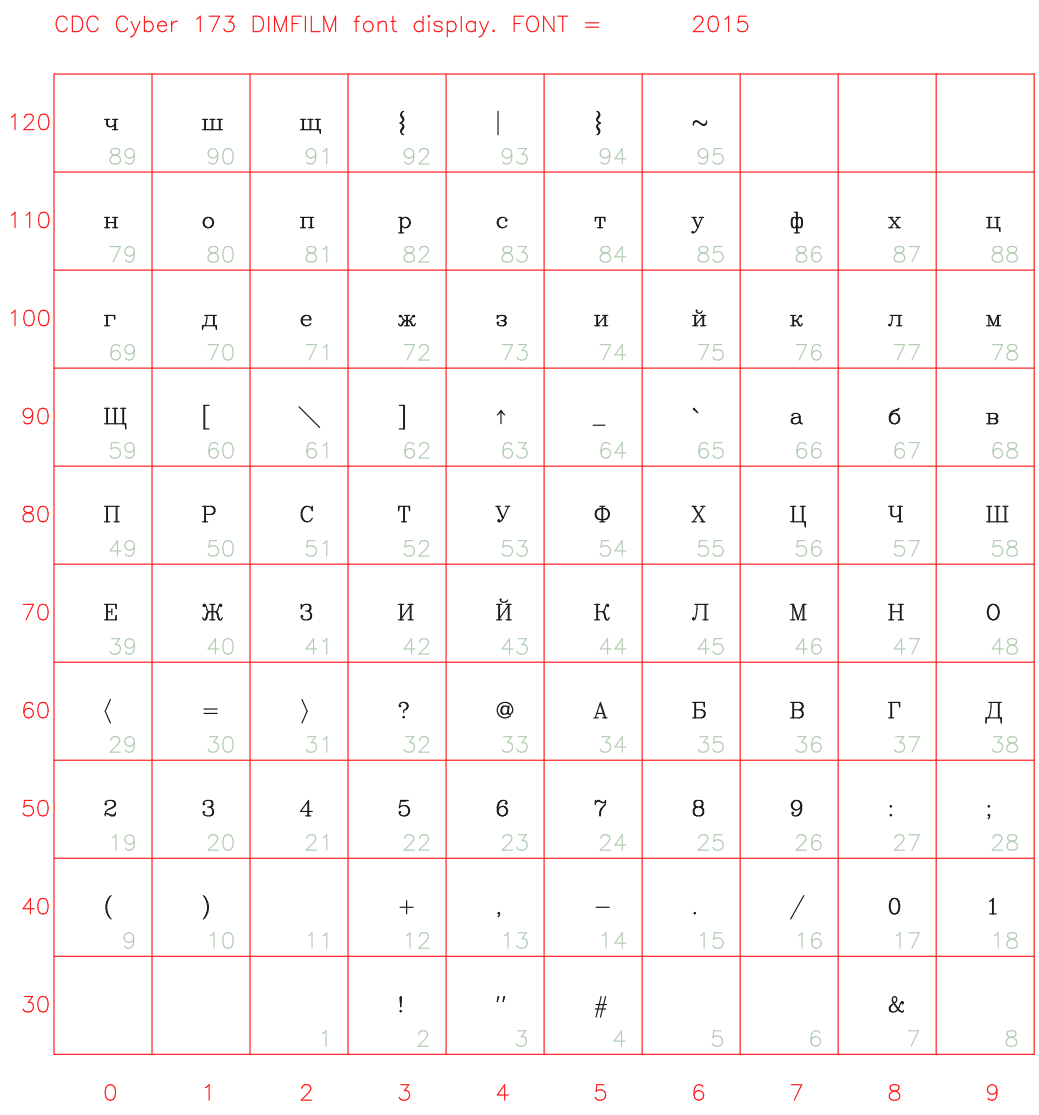


Figure 3.18: DIMFILM font 2015 (Serif, Cyrillic, Complex)

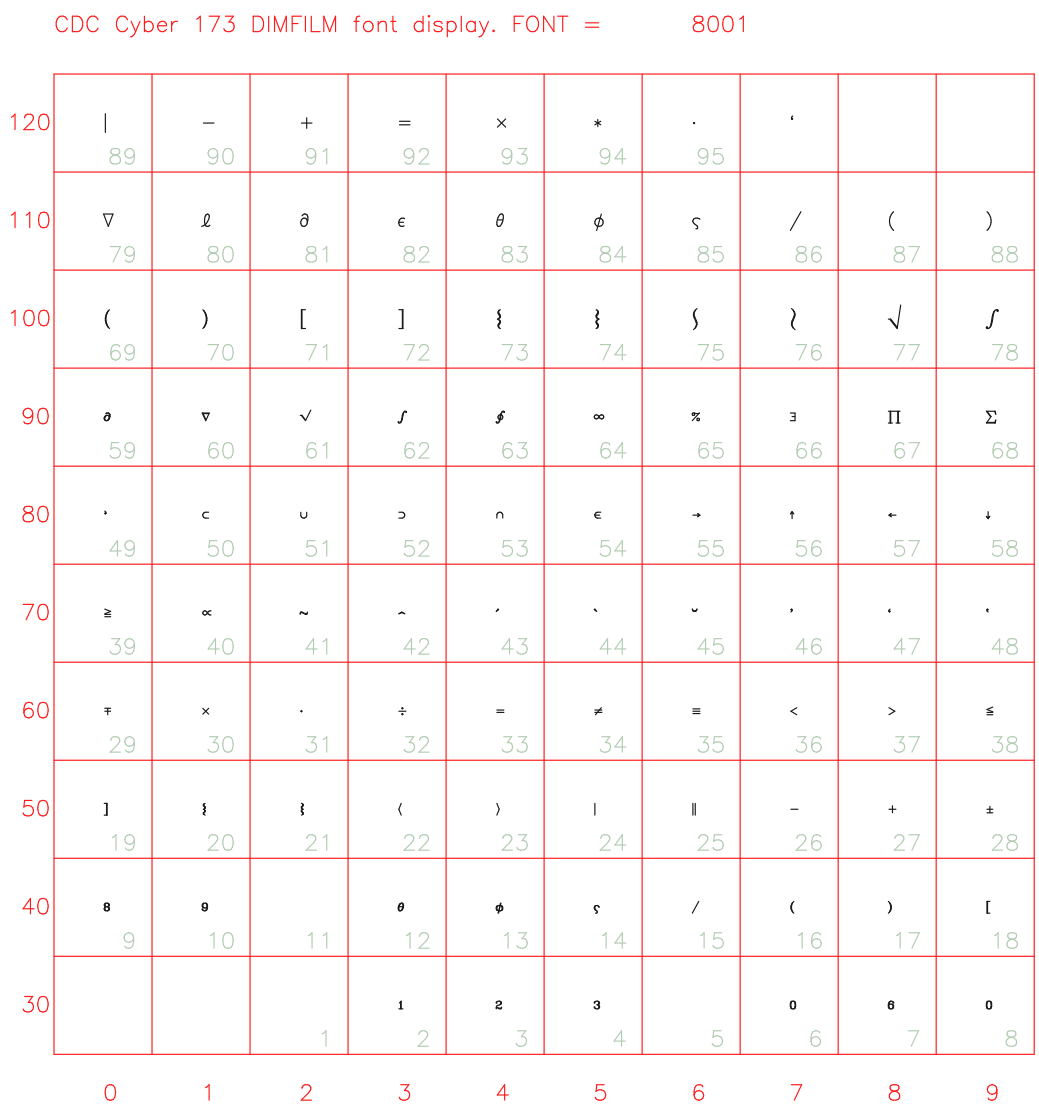


Figure 3.19: DIMFILM font 8001 (Symbols, Mathematical)

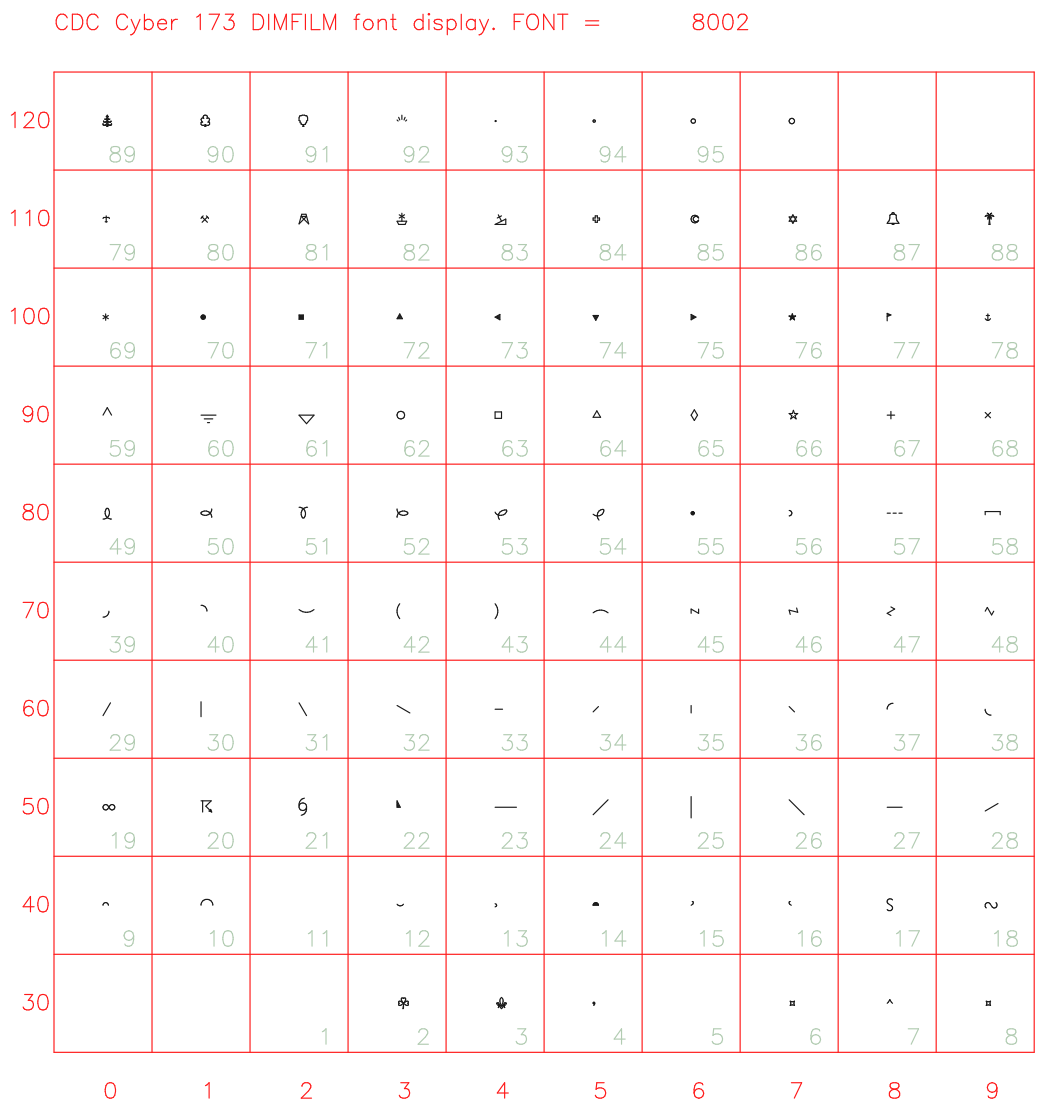


Figure 3.20: DIMFILM font 8002 (Symbols, Cartographic)

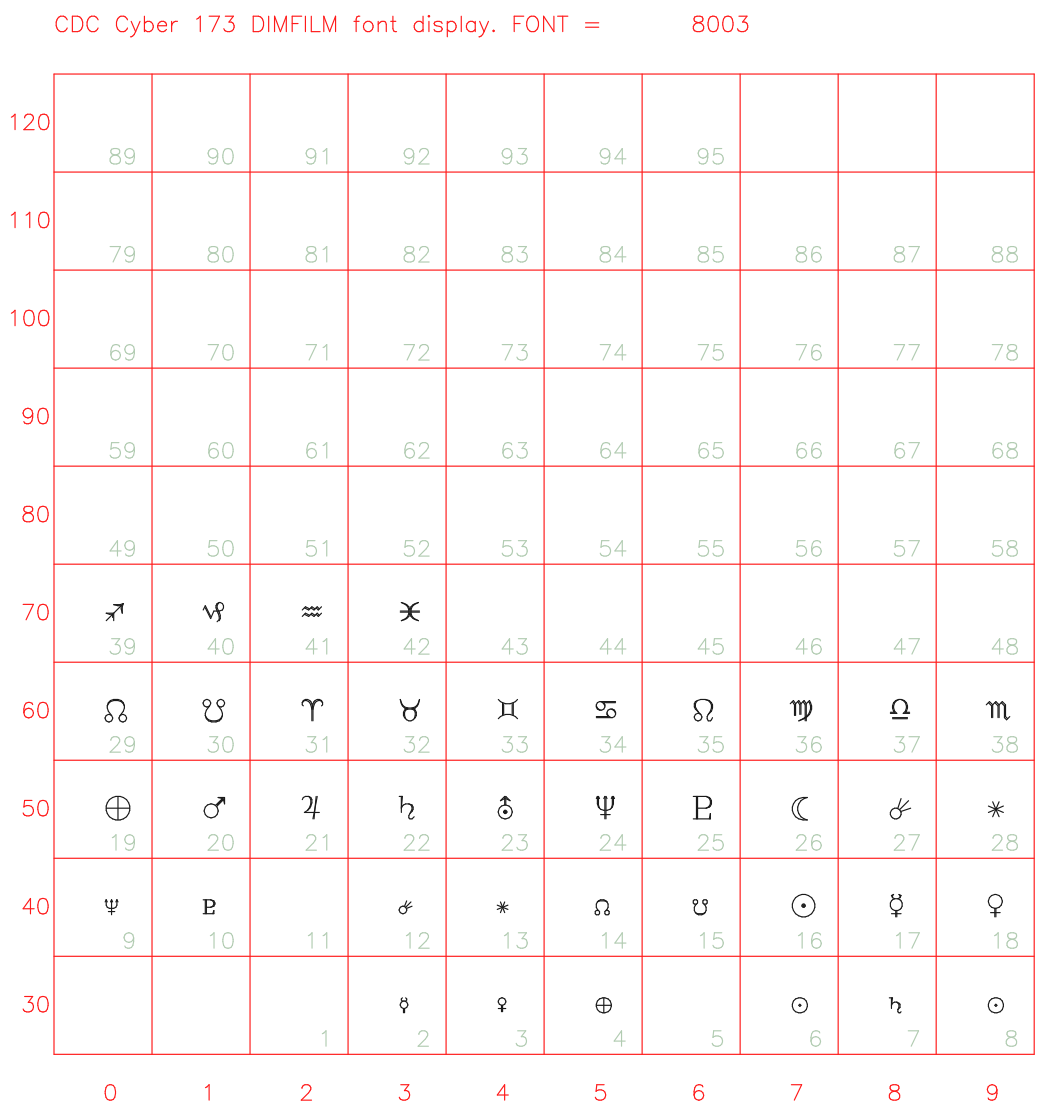


Figure 3.21: DIMFILM font 8003 (Symbols, Astronomical)

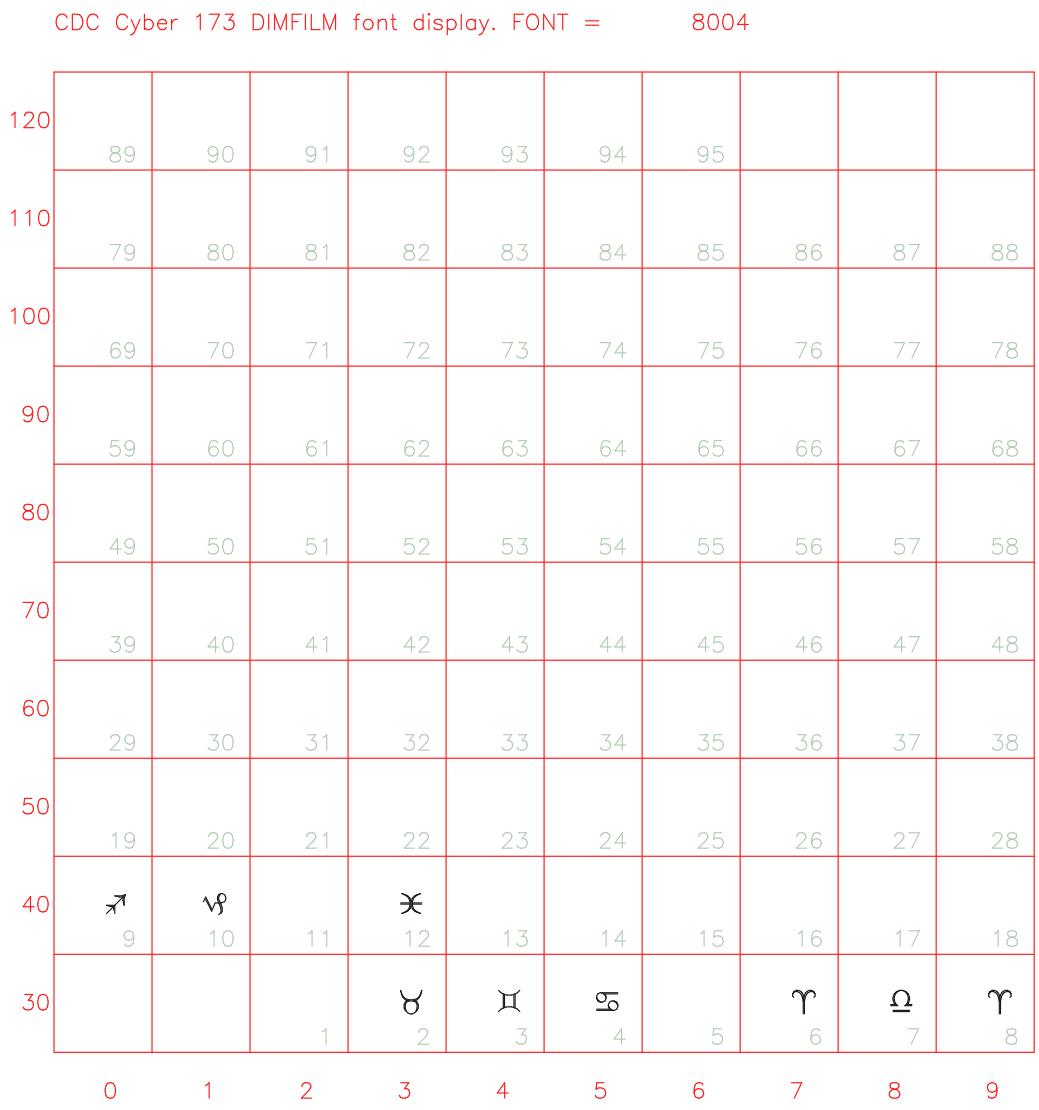


Figure 3.22: DIMFILM font 8004 (Symbols, Astrological)

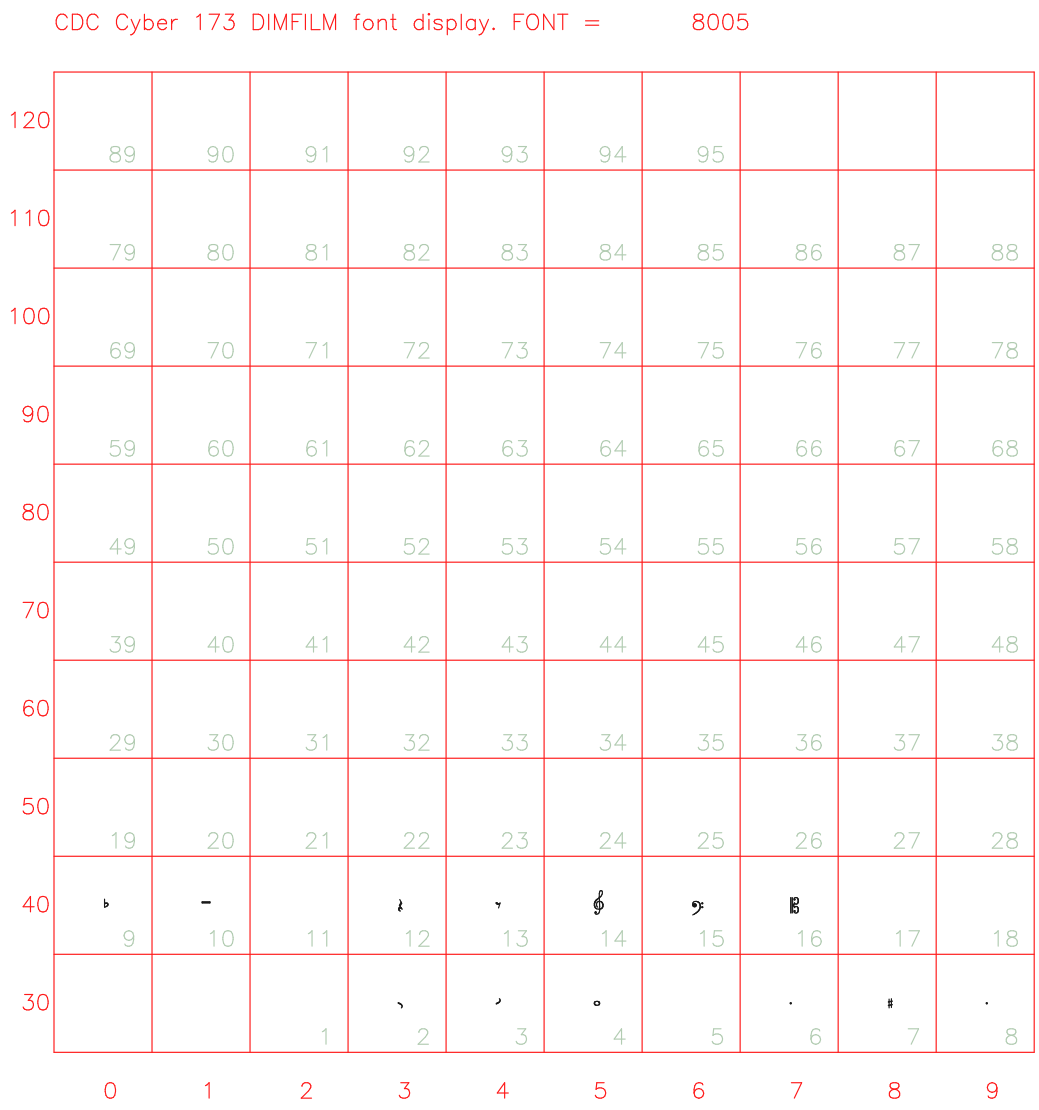


Figure 3.23: DIMFILM font 8005 (Symbols, Musical)

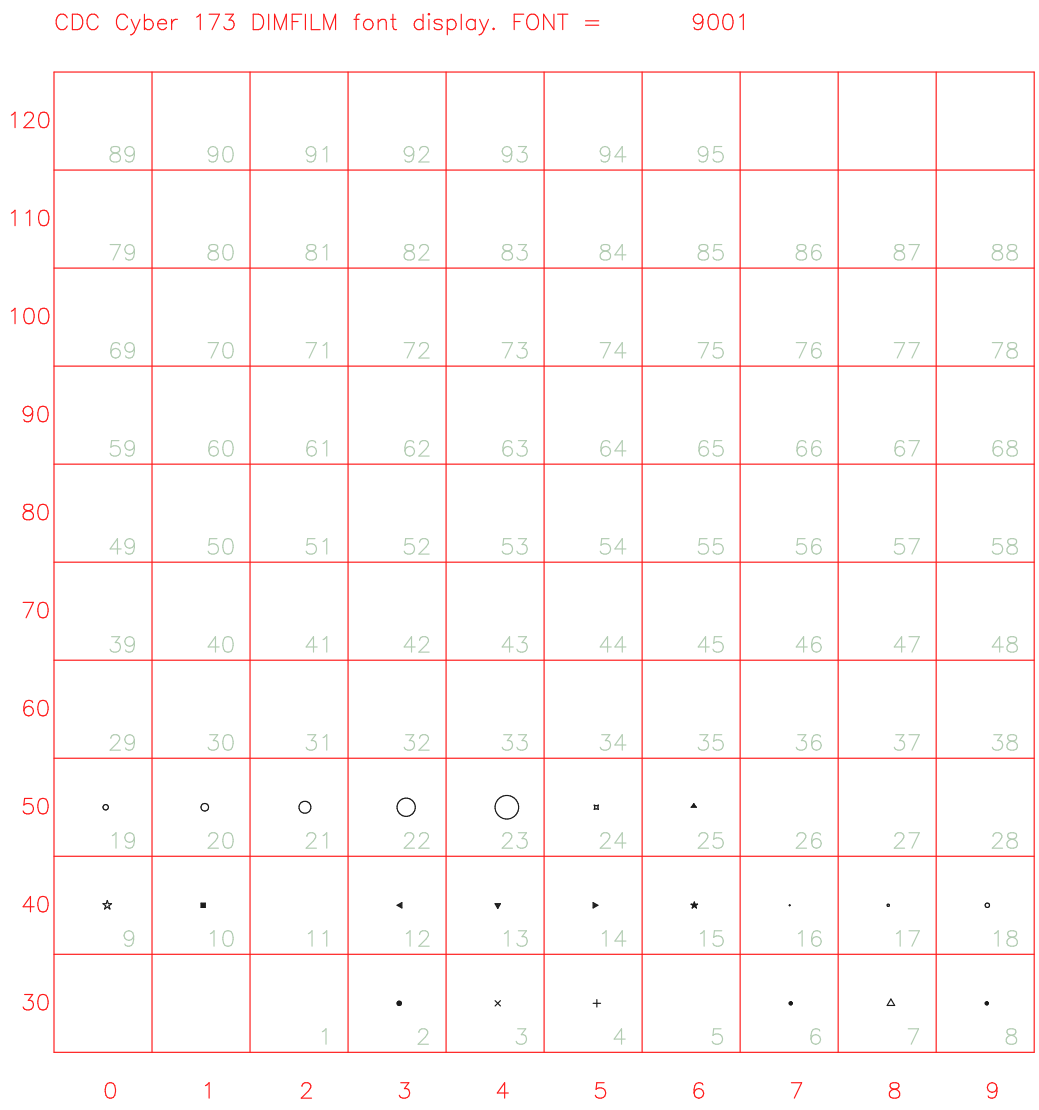


Figure 3.24: DIMFILM font 9001 (Markers)

Chapter 4

System Specific and Device Driver Functions

Porting DIMFILM requires the implementation of a relatively small number of system specific functions, as well as at least one device driver.

4.1 System Specific Functions

The functions in Table 4.1 open a device and send commands to the selected device.

DFXM04	Main subroutine.
DFXM06	Open a device entry point.
DFXM05	Send a command to a device entry point.

Table 4.1: Device Dispatch Functions

The functions in Table 4.2 provide system independent means of doing a number of things. DFXM01 insulates DIMFILM from the host system character set. DFXM3x do bit level manipulations on floating point numbers. This is used by the contouring functions to set, clear and test a flag associated with each value in the array being contoured without using any extra memory. For the CDC system, these were implemented in COMPASS assembler (a very pleasant experience, actually). For G77, intrinsic functions are available to do this (IBSET etc.).

DFXMSA	Convert an error code to a hexadecimal string.
DFXMS0	Write an error message to an output stream.
DFXM00	Stop on a fatal error.
DFXM01	Convert internal char code to DIMFILM char number.
DFXM31	Set least significant bit of REAL number.
DFXM32	Clear least significant bit of REAL number.
DFXM33	Test least significant bit of REAL number.

Table 4.2: System Support Functions

Four more functions allow DIMFILM to access font data, without specifying how this data is stored. For example, each font could be stored in a separate file if desired (this was done for the VAX/VMS port of DIMFILM by Nigel Arnot and Adrian Clark). These functions are described in Table 4.3.

DFXM10	Open the font file - if there is only one file.
DFXM11	Load font number NFONT into alphabet NBET.
DFXM12	Close the font file - if there is only one file.
DFXM20	Unpack a range of byte font data into integers.

Table 4.3: Font Access Functions

Finally, there are three routines which must be implemented at least as stubs. In the 'revived' DIMFILM they serve no useful purpose. See Table 4.4.

DFXUT1	Associate a 'workstation' with a magnetic tape of known VSN.
DFXMS7	See if running in batch or interactive. Always batch.
DFX007	Dummy device used if no device is selected.

Table 4.4: Stub Functions

4.2 Device Driver Functions

In order to do anything useful, DIMFILM needs to be able to output to at least one output device. Defining an output device requires three subroutines to be written – a *device selection* routine (the name of which can be freely chosen), a *vector output driver* function, and a *raster output driver* function. The vector output driver would be named DFDX01 for the first output device, DFDX02 for the second, and so on. Likewise, the raster output driver would be named DFX01F for the first output device, DFX02F for the second, etc. The vector driver actually accepts all output commands, and calls the raster driver as required.

The device selection routine tells DIMFILM which device is the current active device, as well as telling DIMFILM certain vital characteristics of that device. Each device is given a unique 'workstation' [1] number. DFXM05 must be implemented to call the appropriate DFDXnn based on the currently selected 'workstation', which is passed around in the COMMON variable NWS.

The vector driver subroutine takes the following form:

```
SUBROUTINE DFXDnn(IF,XARG,YARG,ZARG,NARG)
INTEGER IF     : Function code
REAL XARG(NARG): X coordinates (usually).
REAL YARG(NARG): Y coordinates (usually).
REAL ZARG(NARG): Third argument of a colour specification.
INTEGER NARG   : Number of elements in XARG, YARG, ZARG.
```

The function codes in Table 4.5 seem to be defined: No device specific functions (function codes

[1] This terminology comes from GKS – which was a *big thing* when DIMFILM-77 was new. Probably few remember it now. At least from my perspective, it was one of many doomed graphics standards of that time, all of which were overwhelmed by emerging *de facto* standards

1	Plot a point.
2	Move with the beam off to X,Y.
4	Move with the beam on to X,Y.
10	Raster operation. Data passed in COMMON
101	Set LUT with data passed in COMMON
102	Set LUT with data passed in arguments.
204	Relative move or draw.
-1	Open device.
-2	Close device.
-3	Frame advance.
-4	Reset device transformation.
< -2000	Device specific function.

Table 4.5: Driver Function Codes

< -2000) are required, and we do not specify any in the device drivers now implemented.

As explained in Chapter 2, the 'revived' DIMFILM implements two output devices — binary and full colour Encapsulated Postscript drivers, with the plot data going to disk files. Raster operations are not implemented in these drivers. This might be a future project! The *device nomination* routines for the two devices are shown in Table 4.6 Certain characteristics of these devices can

DEPBIN	Choose binary PostScript output.
DEPCOL	Choose colour PostScript output.

Table 4.6: Output Device Nomination Routines

be specfied by calling the routines shown in Table 4.7 *before* nominating the output device. This is an unconventional way of doing things from a DIMFILM perspective (as far as I can tell) but it seems sensible. Both device drivers are based on Adrian Clark's VAX/VMS DIMFILM PostScript

EPSDIM	Set the size and offset of the PostScript on the page.
EPSNAM	Set the output filename stem (frame numbers are appended).

Table 4.7: Setting Output Device Characteristics

driver, which in turn was based on John Gilbert's GENDEV deck, which contains a template device nomination subroutine and a template device driver.

In the 'revived' DIMFILM, both drivers perform output by calling a small set of PostScript output functions (which for the CDC implementation output something that can easily be converted to PostScript, but not actual PostScript). For completeness, these routines are listed in Table 4.8

4.3 Undocumented DIMFILM

When the first graphs were plotted with the 'revived' DIMFILM there was much rejoicing — but there was a subtle problem. The annotation (specifically the values next to axis ticks) was too

PSBEGN	Initialize PostScript output.
PSEND	End PostScript output.
PSCLR	End a PostScript page and start a new one.
PSMOVE	Move and draw lines.
PSRGBC	Set the current drawing colour.
PSWID	Set the current line width.

Table 4.8: PostScript Output Functions

small to be easily read. Reading the *Preliminary User Guide* revealed nothing that would help with this. Trying to use SYMHT to change the size of the value strings didn't work.

However, it turns out that there is a way to specify pretty much all aspects of the graph layout, namely the undocumented subroutine GRCON. Furthermore, there is another undocumented subroutine GRCONA which sets the graph layout to something that is much more readable! All the graphs shown in Chapter 5 were plotted with GRCONA in effect (i.e. CALL GRCONA was used just after device nomination).

The full specification of GRCON is:

```
SUBROUTINE GRCON(TOTAL,OUTER,XTITHT,VALHT,FRCVAL,AVAIL,
                 XYLBHT,FACTOR,FRCLAB,SPACE,ENDVAL)
```

Table 4.9 describes these, and shows the defaults and the values selected via CALL GRCONA. Apart from ENDVAL and OUTER, it is easy to guess what these mean (although the guesses may

Parameter	Description	Default	GRCONA
TOTAL	Fraction of pane width occupied by graph.	0.83	0.71
OUTER	Outer margin.	0.01	0.01
XTITHT	Height for title.	0.02	0.035
VALHT	Height for X and Y values.	0.01	0.02
FRCVAL	Frac of dist btw ticks usable for values.	0.8	0.8
AVAIL	Margin fraction.	0.07	0.13
XYLBHT	Axis label height ($\leqslant$ XTITHT).	0.015	0.025
FACTOR	Start position for common value factor.	0.85	0.78
FRCLAB	Fraction of length to use for axis label.	0.68	0.54
SPACE	Height from bottom to X axis label.	0.035	0.02
ENDVAL	Frac of length used by DFX306 extrema values.	0.125	0.175

Table 4.9: GRCON Graph Layout Control Parameters

not be correct!). The default values are established in internal routine DFX006, by the way. Some relationships are enforced between these parameters:

$$\texttt{XTITHT} \geqslant \texttt{XYLBHT} \tag{4.1}$$

$$0.5 < \texttt{FACTOR} < (1 + \texttt{TOTAL})/2 \tag{4.2}$$

$$\texttt{FRCLAB} < 2 \times \texttt{FACTOR} - 1 \tag{4.3}$$

$$\texttt{ENDVAL} < 0.5 \times \texttt{TOTAL} \tag{4.4}$$

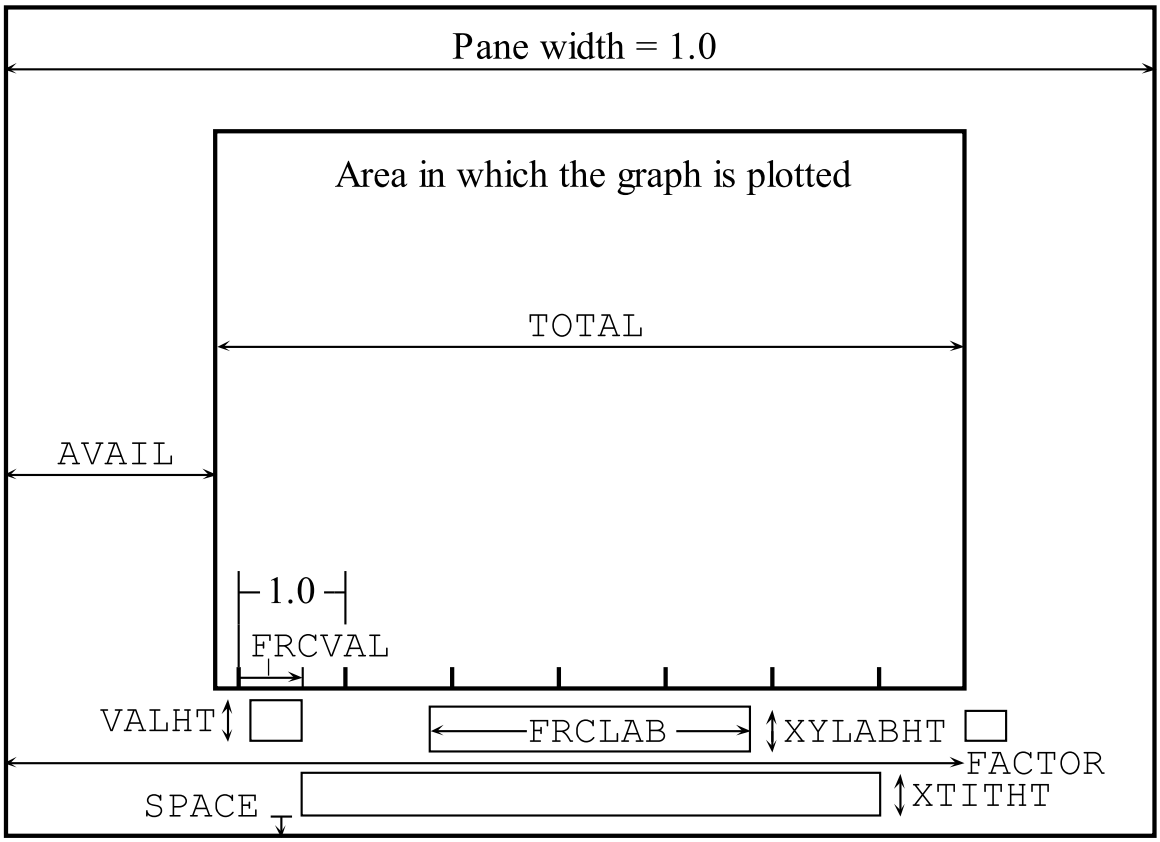


Figure 4.1: GRCON Parameters and Graph Layout

Figure 4.1 is a sketch showing how the `GRCON` parameters are related to aspects of the graph layout (perhaps!). All the parameters are measured in units of pane width (I think), except for `FRCVAL` which is measured as a fraction of the distance between axis ticks.

Chapter 5

A Gallery of DIMFILM Output

To give an indication of what DIMFILM can do, as well as to give the revived code a bit of a test, a short program called DIMTEST was written to generate a reasonably wide selection of plots and graphs.

The resulting graphics is shown in the following pages along with the code that generated it. We start by generating some data to plot:

```
      PROGRAM DIMTEST(OUTPUT,TAPE7=OUTPUT)
C
C--- TRY OUT A FEW DIMFILM FUNCTIONS.
C
      REAL X(100), Y(100), GRID(5,5), CVALS(9)
      REAL PERC(6), Y2(100), Y3(100), X2(100)
      REAL XE(8,3), YE(8,3)
      CHARACTER*3 CAPT(6)
C
C--- GENERATE SOME DATA
C--- COSINE.
C
      DO 1 I=1,100
         X(I) = FLOAT(I) / 50.0 * 3.14159
         Y(I) = COS(X(I))
         Y2(I) = 0.5 * Y(I) + 0.5
         Y3(I) = 10.0 ** ( 3.0 * Y(I) )
 1    CONTINUE
C
C--- SINE WITH ASYMMETRIC RANDOM ERROR BARS ON BOTH AXES.
C
      DO 11 I=1,8
         XE(I,1) = FLOAT(I) / 4.0 * 3.14159
         XE(I,2) = RANF() * 0.2
         XE(I,3) = RANF() * 0.2
         YE(I,1) = SIN(XE(I,1))
```

```
         YE(I,2) = RANF() * 0.2
         YE(I,3) = RANF() * 0.2
         PRINT *,XE(I,2),XE(I,3),YE(I,2),YE(I,3)
 11   CONTINUE
C
C--- GRID OF VALUES TO BE CONTOURED.
C
      DO 2 I=1,9
         CVALS(I) = 0.1 * I
 2       CONTINUE
      DO 3 J=1,5
         A = J - 2.5
         IF( A .LT. 0.0 )A = -A
         A = A / 2.5
         DO 4 I=1,5
            B = I - 2.5
            IF( B .LT. 0.0 )B = -B
            B = B / 2.5
            GRID(I,J) = A * B
 4          CONTINUE
 3       CONTINUE
C
C--- PIE CHART DATA
C
      PERC(1) = 10.0
      PERC(2) = 20.0
      PERC(3) = 40.0
      PERC(4) = 10.0
      PERC(5) =  5.0
      PERC(6) = 15.0
      CAPT(1) = 'A='
      CAPT(2) = 'B='
      CAPT(3) = 'C='
      CAPT(4) = 'D1='
      CAPT(5) = 'D2='
      CAPT(6) = 'E='
C
C--- OPEN THE OUTPUT DEVICE
C
C      CALL EPSDIM( 2.0, 2.0, 1.0, 1.0 )
      CALL EPSNAM( 'NW' )
      CALL DEPCOL
C
C--- SELECT MODIFIED GRAPH LAYOUT
C--- AND SET THE BOUNDS.
C
      CALL GRCONA
```

```
      CALL BOUNDS( 0.0, 1.0, 0.0, 1.0 )
```

The following code then generates the plot shown in Figure 5.1

```
C
C*** FRAME 1
C
C--- DO SOME BASIC PLOTTING
C
      CALL RGB( 0.0, 1.0, 0.0 )
      CALL LINSF( 8.0 )
      CALL OFF2( 0.1, 0.1 )
      CALL ON2( 0.9, 0.9 )
      CALL OFF2( 0.1, 0.8 )
      CALL RGB( 1.0, 0.0, 0.0 )
      CALL LINSF( 1.0 )
C
C--- DRAW SOME FANCY TEXT
C
      CALL SYMTXT( 'ABCDEF*LGHIJKLMNOPQ*LRSTUVWXYZ 0123456789' )
      CALL OFF2( 0.5, 0.6 )
      CALL LINSF( 0.5 )
      CALL SYMTXT('z = *,X*+2$+ + 1*.*,1*.2Y$.$.')
C
C--- DRAW A SIMPLE GRAPH WITH LINEAR AXES
C
      CALL AUTOXY
      CALL PTPLOT( X, Y, 100, 3 )
      CALL GRDEF( 'T*LEST PLOT*U', 'X', 'COS(X)', 3 )
      CALL FRAME
```

Figure 5.1 also shows the effect of setting line width using `LINSF`

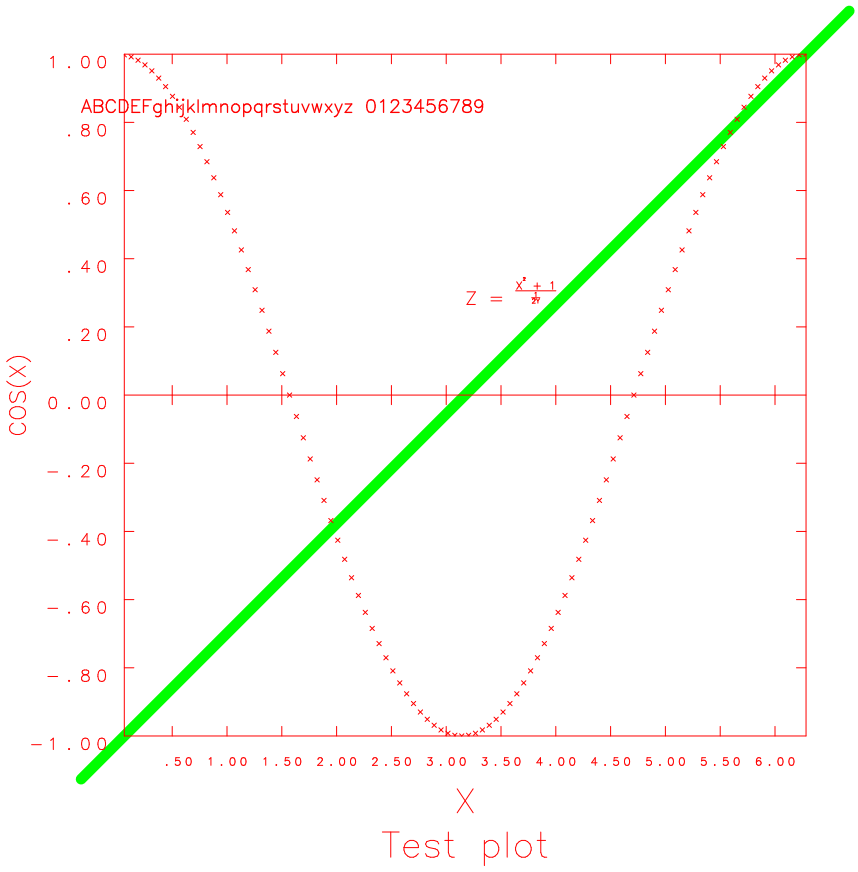


Figure 5.1: Simple plotting, text, and a simple graph

The following code will make a contour plot of a simple function defined on a 2D grid (shown in Figure 5.2):

```
C
C*** FRAME 2
C
C--- PLOT SOME CONTOURS
C
      CALL CINTER( 10 )
      CALL CLABHT( 0.015, 0.0 )
      CALL RGB( 1.0, 0.0, 0.0 )
      CALL CONTR( GRID, 5, 5, 9, CVALS )
      CALL XAXIS( -1.5, 2.5 )
      CALL YAXIS( -1.5, 2.5 )
      CALL RGB( 0.1, 0.1, 0.1 )
      CALL GRDEF( 'C*LONTOURS*U', 'X', 'Y', 3 )
      CALL FRAME
```

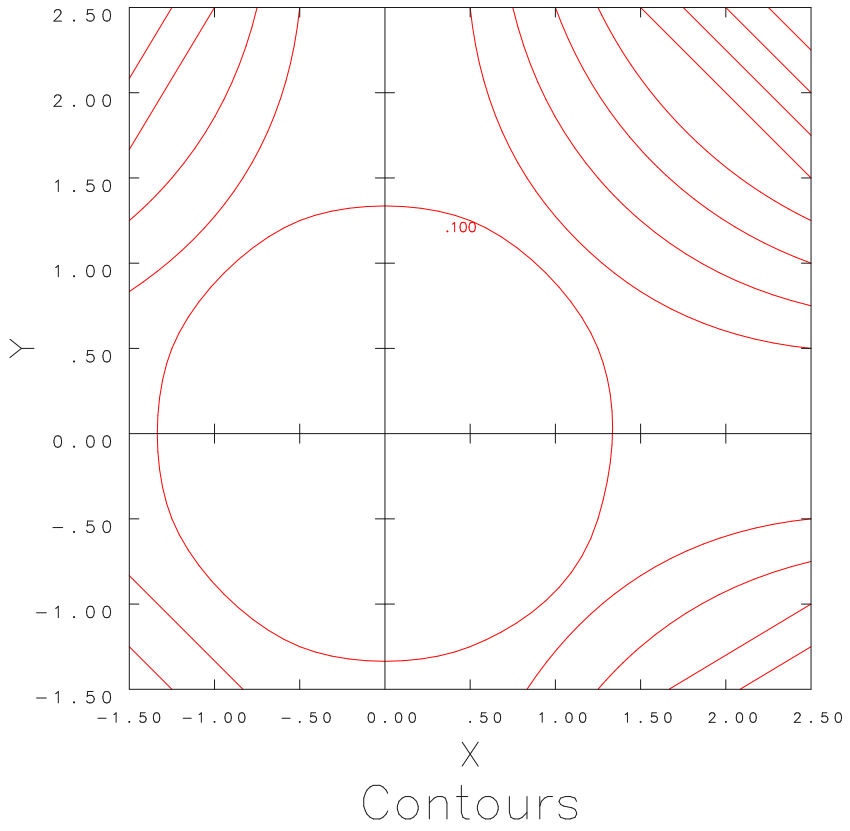


Figure 5.2: A contour plot

It is easy to plot using polar coordinates (shown in Figure 5.3):

```
C
C*** FRAME 3
C
C--- POLAR PLOT
C
      CALL RRANGE( 0.0, 1.0 )
      CALL TRANGE( 0.0, 6.283 )
      CALL RGB( 1.0, 0.0, 0.0 )
      CALL POLAR( X, Y, 100 )
      CALL RGB( 0.0, 1.0, 0.0 )
      CALL POLAR( Y2, X, 100 )
      CALL RGB( 0.1, 0.1, 0.1 )
      CALL POLDEF( 'COSINE', 'POLAR' )
      CALL POLOUT
      CALL POLGRD
      CALL FRAME
```

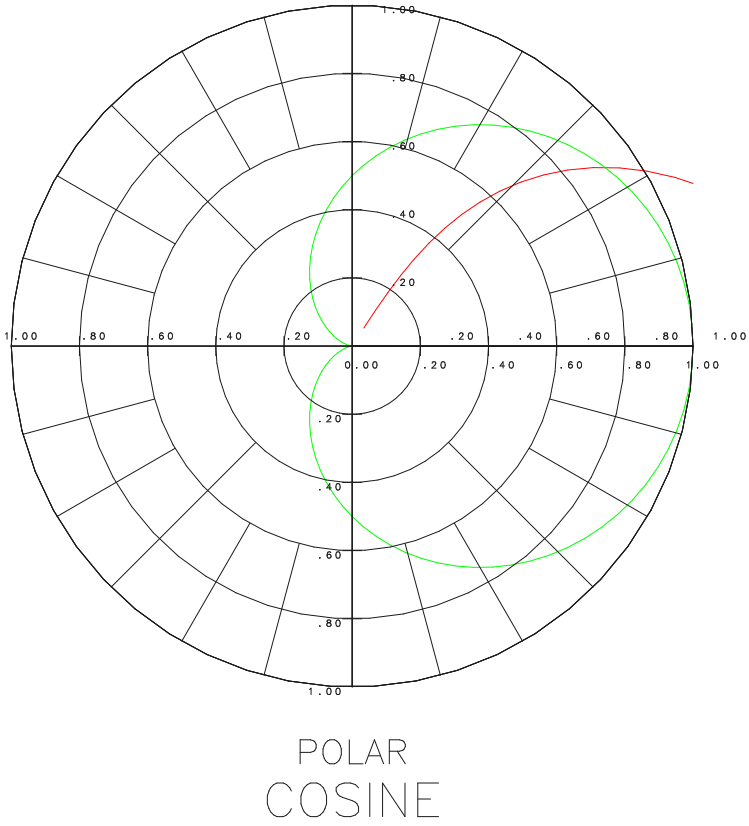


Figure 5.3: A polar plot

It is also possible to plot piecharts:

```
C
C*** FRAME 4
C
C--- PIE CHART
C
      CALL RGB( 0.1, 0.1, 0.1 )
      CALL PIECHT( 0.35, CAPT, PERC, 6, 2 )
      CALL UTITLE( 'A P*LIECHART' )
      CALL GRFRAM
      CALL FRAME
```

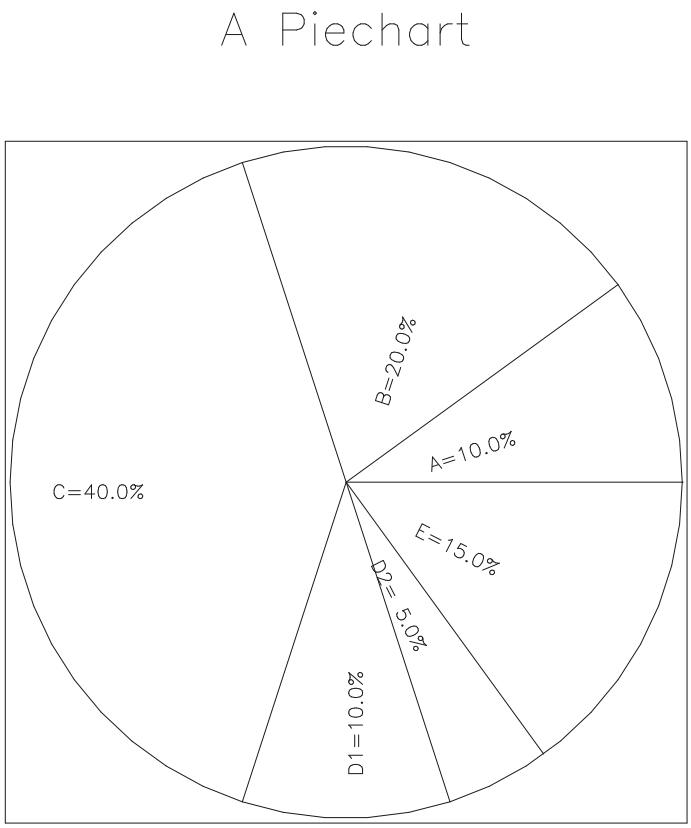


Figure 5.4: A piechart

Histograms are very straightforward:

```
C
C*** FRAME 5
C
C--- HISTOGRAM
C
      CALL AUTOXY
      CALL HISTGR( X, Y, 25, -1.0 )
      CALL GRDEF( 'A H*LISTOGRAM', 'X', 'Y', 3 )
      CALL FRAME
```

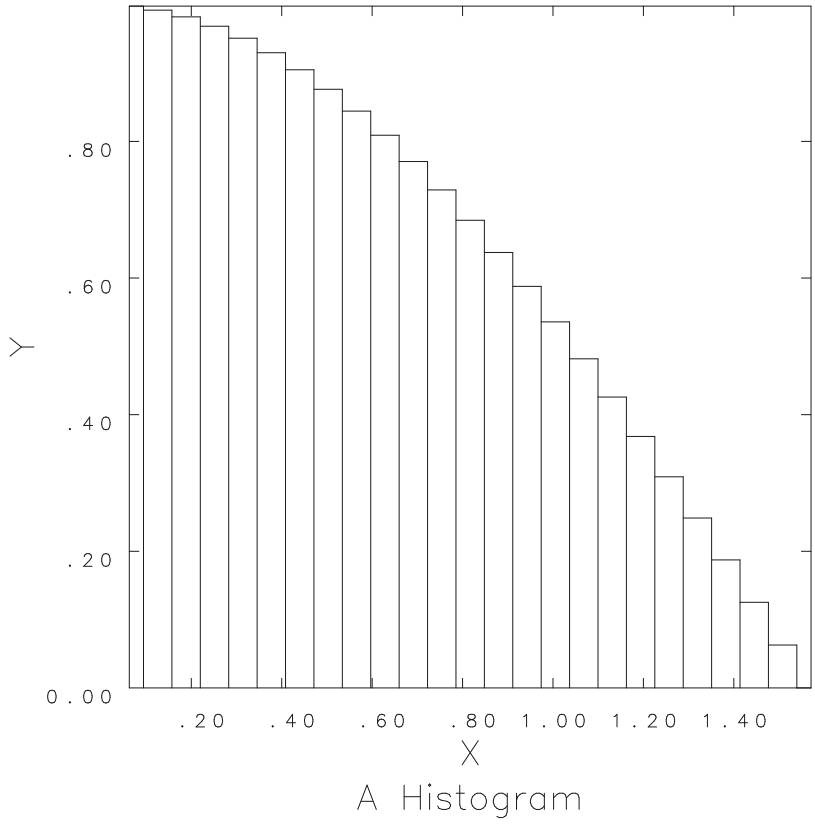


Figure 5.5: A simple histogram

Histograms with shaded bars are also possible:

```
C
C*** FRAME 6
C
C--- SHADED HISTOGRAM
C--- N.B. ENPANE BELOW IS *VITAL* OTHERWISE ALL THE
C--- ANNOTATION GETS PLOTTED INSIDE THE LAST HISTOGRAM
C--- BAR. THIS *MUST* BE A BUG.
C
      CALL AUTOXY
      CALL SHDEGR( X, Y, 25, 0.05, 33.0, 0.01 )
      CALL ENPANE
      CALL GRDEF( 'A*LNOTHER*U H*LISTOGRAM', 'X', 'Y', 3 )
      CALL FRAME
```

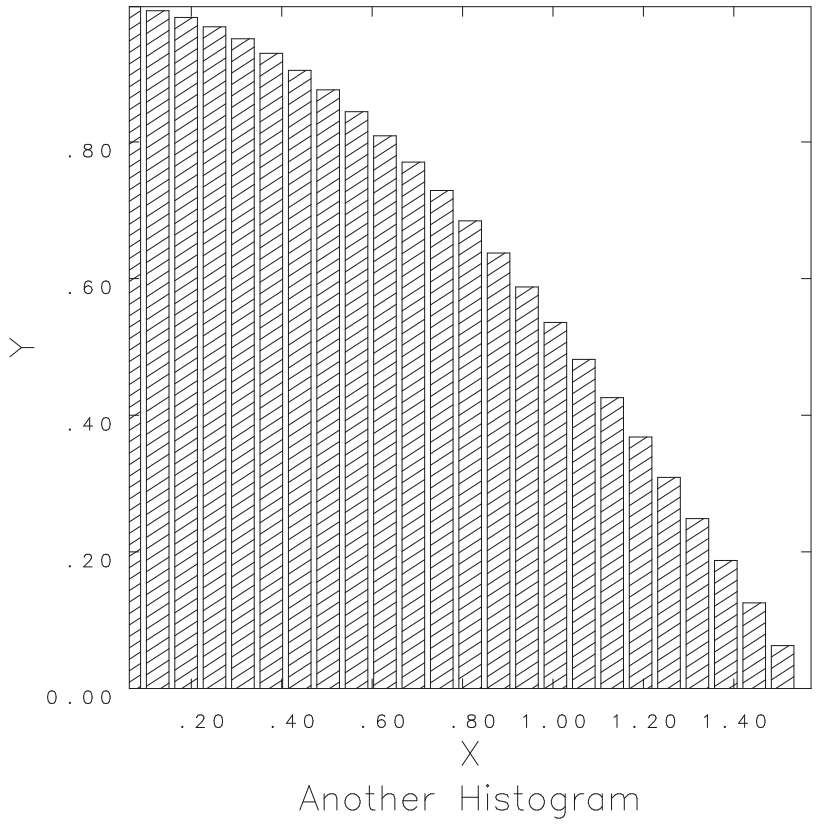


Figure 5.6: A shaded bar histogram

It is also possible to label the axes with months fro commercial applications:

```
C
C*** FRAME 7
C
C--- GRAPH WITH MONTH AXES.
C--- N.B. THE X2 ARRAY IS OVERWRITTEN IN THIS CASE BY GRAPH
C
      CALL MONTHX( 3.0 )
      CALL AUTOXY
      CALL GRAPH( X2, Y, 25 )
      CALL GRDEF( 'A C*LOMMERCIAL *UP*LLOT', 'DATE', 'Y', 3 )
      CALL FRAME
```

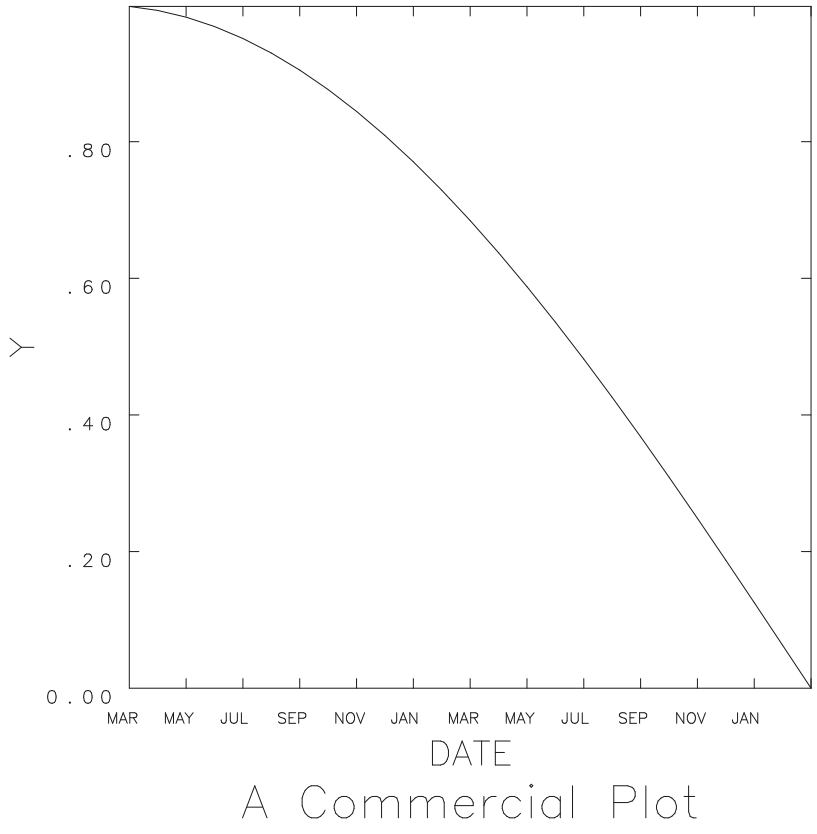


Figure 5.7: Plotting with Months on an axis

One or both axes may be logarithmic:

```
C
C*** FRAME 8
C
C--- LOG-LIN PLOT
C--- N.B. I CANNOT GET SUBDIVISIONS ON THE LOG AXIS.
C         NOTHING SEEMS TO MAKE THIS HAPPEN.
C
      CALL LINX
      CALL LOGY
      CALL LOGLIM( 2.0, 10000.0 )
      CALL LOGTIK( 1.0 )
      CALL XAXIS( 0.0, 6.284 )
      CALL YAXIS( 0.001, 1000.0 )
      CALL RGB( 0.1, 0.1, 1.0 )
      CALL GRAPH( X, Y3, 100 )
      CALL RGB( 0.1, 0.1, 0.1 )
      CALL GRDEF( 'A L*LOG-*UL*LIN *UP*LLOT', '*LX',
     1            '*LEXP10(3 COS(X))', 3 )
      CALL LYATIK
      CALL XYGRID
      CALL FRAME
```

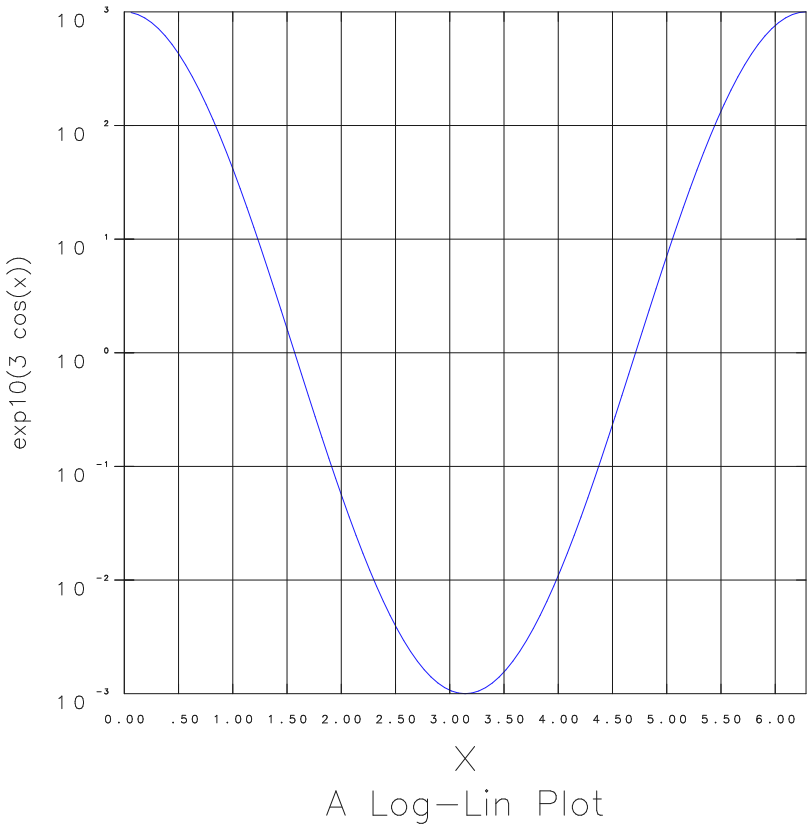


Figure 5.8: Plotting with a logarithmic axis

It is also easy to put multiple graphs on a single output frame using the PANE facility. We also illustrate how to prevent drawing in a specified box using the BLANK facility. This could be useful for adding a key annotation to a graph.

```
C
C*** FRAME 9
C
C--- MULTIPLE GRAPHS ON ONE FRAME
C
      CALL LINX
      CALL LINY
C
C--- DEFINE A BLANKED BOX.
C
      CALL BLANK( 0.8, 0.95, 0.85, 0.9 )
C
C--- BOTTOM LEFT PANE.
C
      CALL PANE( 0.0, 0.5, 0.0, 0.5 )
      CALL CINTER( 10 )
      CALL CLABHT( 0.009, 0.0 )
      CALL RGB( 1.0, 0.0, 0.0 )
      CALL CONTR( GRID, 5, 5, 9, CVALS )
      CALL XAXIS( -1.5, 2.5 )
      CALL YAXIS( -1.5, 2.5 )
      CALL RGB( 0.1, 0.1, 0.1 )
      CALL GRDEF( 'C*LONTOURS*U', 'X', 'Y', 3 )
C
C--- BOTTOM RIGHT PANE.
C
      CALL PANE( 0.5, 1.0, 0.0, 0.5 )
      CALL AUTOXY
      CALL PTPLOT( X, Y, 100, 3 )
      CALL GRDEF( 'T*LEST PLOT*U', 'X', 'COS(X)', 3 )
C
C--- TOP LEFT PANE.
C
      CALL PANE( 0.0, 0.5, 0.5, 1.0 )
      CALL RRANGE( 0.0, 1.0 )
      CALL TRANGE( 0.0, 6.283 )
      CALL RGB( 1.0, 0.0, 0.0 )
      CALL POLAR( X, Y, 100 )
      CALL RGB( 0.0, 1.0, 0.0 )
      CALL POLAR( Y2, X, 100 )
      CALL RGB( 0.1, 0.1, 0.1 )
      CALL POLDEF( 'COSINE', 'POLAR' )
      CALL POLOUT
```

```
      CALL POLGRD
C
C--- TOP RIGHT PANE.
C
      CALL PANE( 0.5, 1.0, 0.5, 1.0 )
      CALL LINX
      CALL LOGY
      CALL XAXIS( 0.0, 6.284 )
      CALL YAXIS( 0.001, 1000.0 )
      CALL RGB( 0.1, 0.1, 1.0 )
      CALL GRAPH( X, Y3, 100 )
      CALL RGB( 0.1, 0.1, 0.1 )
      CALL GRDEF( 'A L*LOG-*UL*LIN *UP*LLOT', '*LX',
     1            '*LEXP10(3 COS(X))', 3 )
      CALL XYGRID
C
C--- STOP CLIPPING OUT THE BLANKED BOX
C
      CALL OBLANK
      CALL ENBLNK
C
C--- NOW PUT A STRING CENTERED IN THE BLANKED BOX.
C
      SWID = STRING( 'BLANK' )
      DRWID = ( 0.9 * ( 0.95 - 0.8 ) )
      XS = 0.5 * ( ( 0.95 + 0.8 ) - DRWID )
      SHT = DRWID / SWID
      YS = 0.5 * ( 0.9 + 0.85 )
      CALL OFF2( XS, YS )
      CALL SYMHT( SHT )
      CALL SYMTXT( 'BLANK' )
C
C--- OUTPUT THE FRAME
C
      CALL FRAME
```

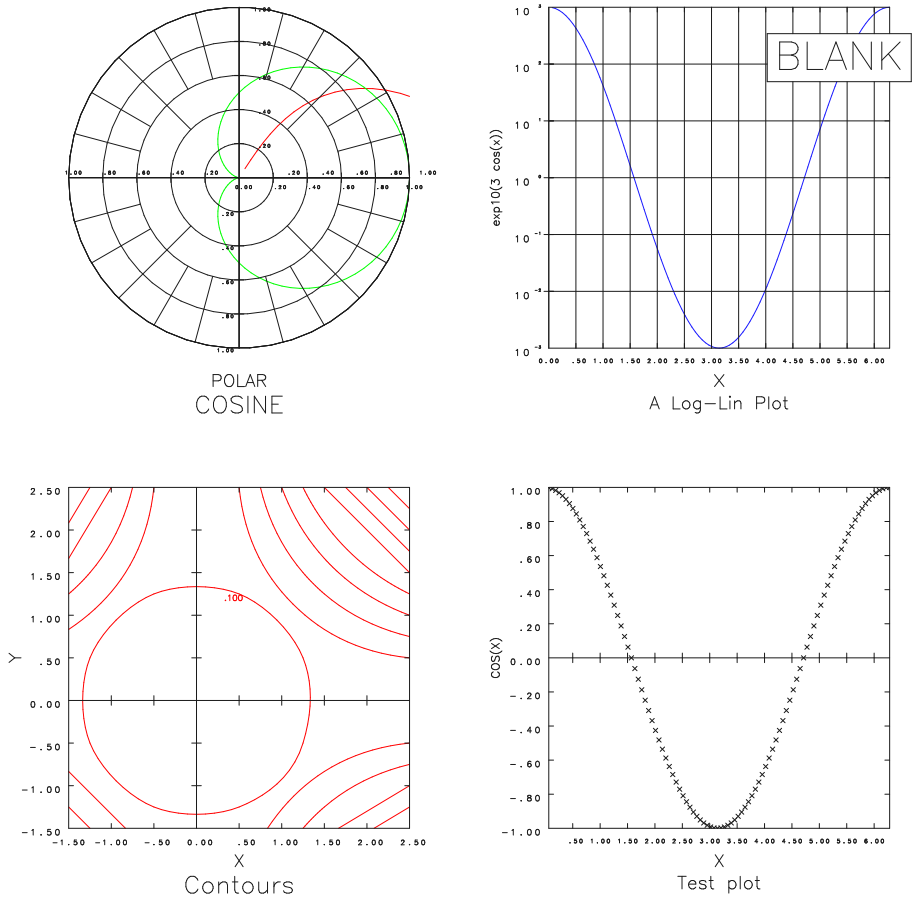


Figure 5.9: Multiple graphs on a single frame

Finally, it is possible to plot a line through data points with cubic or quintic polynomial interpolation, and to plot error bars — including asymettric error bars.

```
C
C*** FRAME 10
C
C--- PLOT WITH ERROR BARS AND POLYNOMIAL INTERPOLATION.
C--- N.B. POLY5E DOES NOT ACTUALLY SEEM TO PLOT THE
C---      ERROR BARS.
C
      CALL ENPANE
      CALL LINX
      CALL LINY
      CALL AUTOXY
      CALL RGB( 0.1, 0.1, 1.0 )
      CALL POLY5E( XE, YE, 8, 3, 3 )
      CALL SAMEXY
      CALL RGB( 1.0, 0.1, 0.1 )
      CALL PTPLTE( XE, YE, 8, 3, 3, 3 )
      CALL RGB( 0.1, 1.0, 0.1 )
      CALL GRAPH( X, Y, 100 )
      CALL RGB( 0.1, 0.1, 0.1 )
      CALL GRDEF( 'P*LOLYNOMIAL INTERPOLATION AND ERROR BARS',
     1            '*LX*U', '*LSIN(X)*U', 3 )
      CALL FRAME
```

To end the program we need to call `DIMEND`:

```
C
C--- ALL DONE
C
      CALL DIMEND
      STOP
      END
```

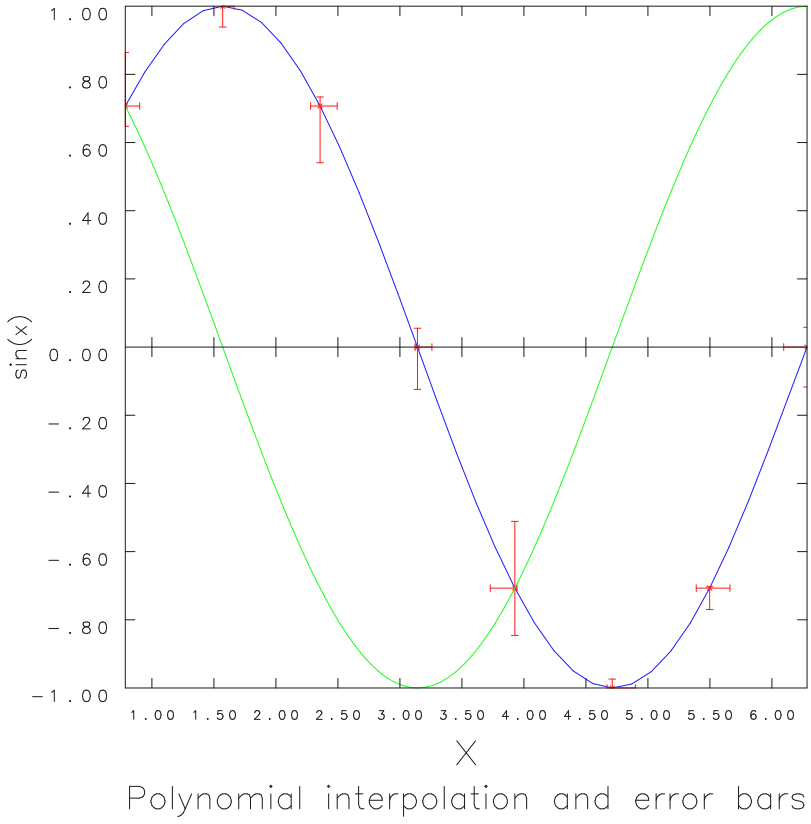


Figure 5.10: Polynomial interpolation and error bars

— empty —

ISBN 3-86541-114-2