Appendix F: Chart of Octal Codes for Characters

The characters and their octal codes in the reencoded standard character set are presented in the chart below. Gray areas signify codes reserved for control characters.

octal	0	1	2	3	4	5	6	7
\00x								
\01x								
\02x								
\03x								
\04x		!	"	#	$	%	&	'
\05x	(	)	*	+	,	-	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;	<	=	>	?
\10x	@	A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z	[	\	]	^	_
\14x	`	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z	{	|	}	~	
\20x								
\21x								
\22x								
\23x								
\24x		¡	¢	£	⁄	¥	ƒ	§
\25x	¤	'	“	«	‹	›	ﬁ	ﬂ
\26x	Á	–	†	‡	·	Â	¶	•
\27x	‚	„	”	»	…	‰	Ä	¿
\30x	À	`	´	ˆ	˜	¯	˘	˙
\31x	¨	É	˚	¸	Ê	˝	˛	ˇ
\32x	—	Ë	È	Í	Î	Ï	Ì	Ó
\33x	Ô	Ö	Ò	Ú	Û	Ü	Ù	á
\34x	â	Æ	ä	ª	à	é	ê	ë
\35x	è	Ø	Œ	º	í	î	ï	ì
\36x	ó	æ	ô	ö	ò	ı	ú	û
\37x	ü	ø	œ	ß	ù	Å	å	

Appendix G: PostScript fonts used by GMT

GMT uses the standard 34 fonts that come with most PostScript laserwriters. If your printer does not support all of these fonts, it should automatically substitute the default font (which is usually Courier). The following is a list of the standard fonts:

Font number:	Font name:
0	Helvetica
1	**Helvetica-Bold**
2	*Helvetica-Oblique*
3	***Helvetica-BoldOblique***
4	Times-Roman
5	**Times-Bold**
6	*Times-Italic*
7	***Times-BoldItalic***
8	Courier
9	**Courier-Bold**
10	*Courier-Oblique*
11	***Courier-BoldOblique***
12	Σψμβολ
13	AvantGarde-Book
14	*AvantGarde-BookOblique*
15	**AvantGarde-Demi**
16	***AvantGarde-DemiOblique***
17	Bookman-Demi
18	*Bookman-DemiItalic*
19	**Bookman-Light**
20	***Bookman-LightItalic***
21	Helvetica-Narrow
22	**Helvetica-Narrow-Bold**
23	*Helvetica-Narrow-Oblique*
24	***Helvetica-Narrow-BoldOblique***
25	NewCenturySchlbk-Roman
26	*NewCenturySchlbk-Italic*
27	**NewCenturySchlb k-Bold**
28	***NewCenturySchlbk-BoldItalic***
29	Palatino-Roman
30	*Palatino-Italic*
31	**Palatino-Bold**
32	***Palatino-BoldItalic***
33	*ZapfChancery-MediumItalic*

Appendix H: Hints and known bugs concerning display of GMT *PostScript*

GMT creates valid (so far as we know) Adobe *PostScript* Level I. It does not use operators specific to Level II and should therefore produce output that will print on old as well as new *PostScript* printers Sometimes unexpected things happen when GMT output is sent to certain printers or displays. This section lists some things we have learned from experience, and some work-arounds.

• *PostScript* driver bugs.

When you try to display a *PostScript* file on a device, such as a printer or your screen, then a program called a *PostScript* device driver has to compute which device pixels should receive which colors (black or white in the case of a simple laser printer) in order to display the file. At this stage, certain device-dependent things may happen. These are not limitations of GMT or *PostScript*, but of the particular display device. The following bugs are known to us based on our experiences:

Early versions of the Sun SPARCprinter software caused linewidth-dependent path displacement. We reported this bug and it has been fixed in newer versions of the software. Try using *psxy* to draw y = f(x) twice, once with a thin pen (–W1) and once with a fat pen (–W10); if they do not plot on top of each other, you have this kind of bug and need new software. The problem may also show up when you plot a mixture of solid and dashed (or dotted) lines of various pen thickness

The first version of the HP Laserjet 4M had bugs in the driver program. We reported it, and they have just released (we got ours Aug. 10 1993) a new one. The old one was *PostScript* SIMM, part number C2080-60001; the new one is called *PostScript* SIMM, part number C2080-60002. You need to get this one plugged into your printer if you have an HP LaserJet 4M.

Apple Laserwriters with the older versions of Apple's *PostScript* driver will give the error "limitcheck" and fail to plot when they encounter a path exceeding about 1500 points. Try to get a newer driver from Apple, but if you can't do that, set the parameter MAX_PATH to 1500 or even smaller in the file src/pslib_inc.h and recompile GMT. The number of points in a *PostScript* path can be arbitrarily large, in principle; GMT will only create paths longer than MAX_PATH if the path represents a filled polygon or clipping path. Line-drawings (no fill) will be split so that no segment exceeds MAX_PATH. This means *psxy* –G will issue a warning when you plot a polygon with more than MAX_PATH points in it. It is then your responsibility to split the large polygon into several smaller segments. If *pscoast* gives such warnings and the file fails to plot you may have to select one of the lower resolution databases The path limitation exemplified by these Apple printers is what makes the higher-resolution coastlines for *pscoast* non-trivial: such coastlines have to be organized so that fill operations do not generate excessively large paths. Some HP *PostScript* cartridges for the Laserjet III also have trouble with paths exceeding 1500 points; they may successfully print the file, but it can take all night!

8-bit color screen displays (and programs which use only 8-bits, even on 24-bit monitors, such as Sun's Pageview under OW3) may not dither cleverly, and so the color they show you may not resemble the color your *PostScript* file is asking for. Therefore, if you choose colors you like on the screen, you may be surprised to find that your plot looks different on the hardcopy printer or film writer. The only thing you can do is be aware of this, and make some test cases on your hardcopy devices and compare them with the screen, until you get used to this effect. (Each hardcopy device is also a little different, and so you will eventually find that you want to tune your color choices for each device.) The rgb color cube in example 11 may help.

Some versions of Sun's OW program Pageview have only a limited number of colors available; the number can be increased somewhat by starting openwin with the option "openwin -cubesize large". (Our SPARC-10 doesn't seem to need this anymore, but earlier machines did.)

Many color hardcopy devices use CMYK color systems. GMT *PostScript* uses RGB (even if your .cpt files are using HSV). The three coordinates of RGB space can be mapped into three coordinates in CMY space, and in theory K (black) is superfluous. But it is hard to get CMY inks to mix into a good black or gray, so these printers supply a black ink as well, hence CMYK. The *PostScript* driver for a CMYK printer should be smart enough to compute what portion of CMY can be drawn in K, and use K for this and remove it from CMY; however, some of them aren't.

In early releases of GMT we always used the *PostScript* command setrgbcolor(r,g,b) to specify colors, even if the color happened to be a shade of gray (r=g=b) or black (r=g=b=0). One of our users found that black came out muddy brown when he used FreedomOfPress to make a Versatec plot of a GMT map. He found that if he used the *PostScript* command setgray(g) (where g is a graylevel) then the problem went away. Apparently, his installation of FreedomOfPress uses only CMY with the command setrgbcolor, and so setrgbcolor(0,0,0) tries to make black out of CMY instead of K. To fix this, in release 2.1 of GMT we changed some routines in **pslib.c** to check if (r=g and r=b), in which case setgray(g), else setrgbcolor(r,g,b).

Recent experience with some Tektronix Phaser printers and with commercial printing shops has shown that this substitution creates problems precisely opposite of the problems our Versatec user has. The Tektronix and commercial (we think it was a Scitex) machines do not use K when you say setgray(0) but they do when you say setrgbcolor(0,0,0). We believe that these problems are likely to disappear as the various software developers make their codes more robust. Note that this is not a fault with GMT: r = g = b = 0 means black and should plot that way. Thus, the GMT source code as shipped to you checks whether r=g and r=b, in which case it uses setgray, else setrgbcolor. If your gray tones are not being drawn with K, you have two work-around options: (1) edit the source for **pslib.c** or (2) edit your *PostScript* file and try using setrgbcolor in all cases. The simplest way to do so is to redefine the setgray operator to use setrgbcolor. Insert the line

```
/setgray  {dup dup setrgbcolor} def
```

immediately following the first line in the file (starts with %!PS.)

Some color film writers are very sensitive to the brand of film. If black doesn't look black on your color slides, try a different film.

• Resolution and dots per inch.

The parameter DOTS_PR_INCH can be set by the user through the .gmtdefaults file or gmtset. By default it is equal to the value in the src/gmt_defaults.h file, which is supplied with 300 when you get GMT from us. This seems a good size for most applications, but should ideally reflect the resolution of your hardcopy device (most laserwriters have 300 dpi, hence our default value). GMT computes what the plot should look like in double precision floating point coordinates, and then converts these to integer coordinates at DOTS_PR_INCH resolution. This helps us find out that certain points in a path lie on top of other points, and we can remove these, making smaller paths. Small paths are important for the laserwriter bugs above, and also to make fill operations compute faster. Some users have set their DOTS_PR_INCH to very large numbers. This only makes the *PostScript* output bigger without affecting the appearence of the plot. However, if you want to make a plot which fits on a page at first, and then later magnify this same *PostScript* file to a huge size, the higher DPI is important. Your data may not have the higher resolution but on certain devices the edges of fonts will not look crisp if they are not drawn with an effective resolution of 300 dpi or so. Beware of making an excessively large path. Note that if you change dpi the linewidths produced by your -W options will change, unless you have used the p for linewidth in points.

- European Characters

Note for users of "pageview" in Sun OpenWindows: **GMT** now offers some octal escape sequences to load European alphabet characters in text strings (see section 4.15). When this feature is enabled, the header on **GMT** *PostScript* output includes a section defining special fonts. The definition is added to the header whether or not your plot actually uses the fonts.

Users who view their **GMT** *PostScript* output using "pageview" in OpenWindows on Sun computers or user older laserwriters may have difficulties with the European font definition. If your installation of OpenWindows followed a space-saving suggestion of Sun, you may have excluded the European fonts, in which case pageview will fail to show you anything when you try to view a plot.

Ask your system administrator about this, or run this simple test: (1) View a **GMT** *PostScript* file with "pageview". If it comes up OK, you will be fine. If it comes up blank, open the "Edit PostScript" button and examine the lower window for error messages. (The European font problem generates lots of error messages in this window). (2) Verify that the *PostScript* file is OK, by sending it to a laser printer and making sure it comes out. (3) If the *PostScript* file is OK but it chokes "pageview", then edit the PostScript file, cutting out everything between the lines:

%%%%% START OF EUROPEAN FONT DEFINITION %%%%%
<bunch of definitions
%%%%% END OF EUROPEAN FONT DEFINITION %%%%%

Now try "pageview" on the edited version. If it now comes up, you have a limited subset of OpenWindows installed. If you discover that these fonts cause you trouble, then you can edit your .gmtdefaults file to set WANT_EURO_FONT = FALSE, which will suppress the printing of this definition in the **GMT** *PostScript* header. With this set to FALSE, you can make output which will be viewable in pageview without any editing. However, you would have to reset this to TRUE before attempting to use European fonts, and then the output will become un-pageview-able again. If you try to concatenate segments of **GMT** *PostScript* made with and without the European fonts enabled, then you may find that you have problems, either with the definition, or because you ask for something not defined.

• Hints.

When making images and perspective views of large amounts of data, the GMT programs can take some time to run, the resulting *PostScript* files can be very large, and the time to display the plot can be long. Fine tuning a plot script can take lots of trial and error. We recommend using grdsample to make a low resolution version of the data files you are plotting, and practice with that, so it is faster; when the script is perfect, use the full-resolution data files. We often begin building a script using only psbasemap and/or pscoast to get the various plots oriented correctly on the page; once this works we replace the psbasemap calls with the actually desired GMT programs.

If you want to make color shaded relief images and you haven't had much experience with it, here is a good first cut at the problem: Set your COLOR_MODEL to HSV using gmtset. Use *makecpt* or *grd2cpt* to make a continuous color palette spanning the range of your data. Use the -Nt option on *grdgradient*. Try the result, and then play with the tuning of the .gmtdefaults, the .cpt file, and the gradient file.

Appendix I: Color Space – The final frontier

Beginning with GMT version 2.1.4, "Example 11" was included in the cookbook. The example makes an RGB color cube by a simple *nawk* script. We wrote a program to compute HSV grids for each face of this cube, and present a version of the cube with HSV contours on it as file contoured_cube.ps.

In this appendix, we are going to try to explain the relationship between the RGB and HSV color systems so as to (hopefully) make them more intuitive. GMT allows users to specify colors in .cpt files in either system (colors on command lines, such as pen colors in –W option, are in RGB). GMT uses the HSV system to achieve artificial illumination of colored images (e.g. –I option in grdimage) by changing the s and v coordinates of the color. When the intensity is zero, the data are colored according to the .cpt file. If the intensity is non-zero, the data are given a starting color from the .cptfile but this color (after conversion to HSV if necessary) is then changed by moving (s, v) toward HSV_MIN_SATURATION, HSV_MIN_VALUE if the intensity is negative, or toward HSV_MAX_SATURATION, HSV_MAX_VALUE if positive. These are defined in the .gmtdefaults file and are usually chosen so the corresponding points are nearly black (s = 1, v = 0) and white (s = 0, v = 1). The reason this works is that the HSV system allows movements in color space which correspond more closely to what we mean by "tint" and "shade"; an instruction like "add white" is easy in HSV and not so obvious in RGB.

We are going to try to give you a geometric picture of color mixing in HSV from a tour of the RGB cube. The geometric picture is helpful, we think, since HSV are not orthogonal coordinates and not found from RGB by an algebraic transformation. But before we begin traveling on the RGB cube, let us give two formulae, since an equation is often worth a thousand words.

$$v = \max(r, g, b)$$
$$s = (\max(r, g, b) - \min(r, g, b)) / \max(r, g, b)$$

Note that when $r = g = b = 0$ (black), the expression for s gives 0/0; black is a singular point for s. The expression for h is not easily given without lots of "if" tests, but has a simple geometric explanation. So here goes: Look at the cube face with black, red, magenta, and blue corners. This is the $g = 0$ face. Orient the cube so that you are looking at this face with black in the lower left corner. Now imagine a right-handed cartesian (r, g, b) coordinate system with origin at the black point; you are looking at the $g = 0$ plane with r increasing to your right, g increasing away from you, and b increasing up. Keep this sense of (r, g, b) as you look at the cube.

The RGB color cube has six faces. On three of these one of (r, g, b) is equal to 0. These three faces meet at the black corner, where $r = g = b = 0$. On these three faces saturation, the S in HSV, has its maximum value; $s = 1$ on these faces. (Accept this definition and ignore the s singularity at black for now). Therefore h and v are contoured on these faces; h in gray solid lines and v in white dashed lines (v ranges from 0 to 1 and is contoured in steps of 0.1).

On the other three faces one of (r, g, b) is equal to the maximum value. These three faces meet at the white corner, where $r = g = b = 255$. On these three faces value, the V in HSV, has its maximum value; $v = 1$ on these faces. Therefore h and s are contoured on these faces; h in gray solid lines and s in black dashed lines (s ranges from 0 to 1 with contours every 0.1).

The three faces where $v = 1$ meet the three faces where $s = 1$ in six edges where both $s = v = 1$ (and at least one of $(r, g, b) = 0$ and at least one of $(r, g, b) = 255$). Trace your finger around these edges, starting at the red point and moving to the yellow point, then on around. You will visit six of the eight corners of the cube, in this order: red (h = 0); yellow (h = 60); green (h = 120); cyan (h = 180); blue (h = 240); magenta (h = 300). Three of these are the RGB colors; the other three are the CMY colors which are the complement of RGB and are used in many color hardcopy devices (color monitors usually use RGB). The only cube corners you did not visit on this path are the black and white corners. Imagine an axis running through the black and white corners. If you project the RYGCBM edge path onto a plane perpendicular to the black-white axis, the path will look like a hexagon, with RYGCBM at the vertices, every 60° apart. Now we can make a geometric definition of hue: Take a vector from the origin (black point) to any point in the cube; project this vector onto the plane with the RYGCBM hexagon; then hue is the angle this projected vector makes with the R direction on the hexagon. Thus hue is an angle describing rotation around the black-white axis. Note that by this definition, if a point is on the black-white axis, its (r, g, b) vector will project as a point at the center of the hexagon, so its hue is undefined. Points on the black-white axis have $r = g = b$, and they are shades of gray; we will call the black-white axis the gray axis.

Let us call the points where $s = v = 1$ (the points on the RYGCBM path of cube edges) the "pure" colors. If we start at a pure color and we want to whiten it, we can keep h constant and $v = 1$ while decreasing s; this will move us along one of the cube faces toward the white point. If we start at a pure color and we want to blacken it, we can keep h constant and $s = 1$ while decreasing v; this will move us along one of the cube faces toward the black point. Any point in (r, g, b) space which can be thought of as a mixture of pure color + white, or pure color + black, is on a face of the cube.

The points in the interior of the cube are a little harder to describe. The definition for h above works at all points in (non-gray) (r, g, b) space, but so far we have only looked at (s, v) on the cube faces, not inside it. At interior points, none of (r, g, b) is equal to either 0 or 255. Choose such a point, not on the gray axis. Now draw a line through your point so that the line intersects the gray axis and also intersects the RYGCBM path of edges somewhere. It is always possible to construct this line, and all points on this line have the same hue. This construction shows that any point in RGB space can be thought of as a mixture of a pure color plus a shade of gray. If we move along this line away from the grayaxis toward the pure color, we are "purifying" the color by "removing gray"; this move increases the color's saturation. When we get to the point where we cannot remove any more gray, at least one of (r, g, b) will have become zero and the color is now fully saturated; $s = 1$. Conversely, any point on the gray axis is completely undersaturated, so that $s = 0$ there. Now we see that the black point is special, because it is the intersection of three planes on which $s = 1$, but it is on a line where $s = 0$; it is a singular point, and we get 0/0 in the above formula. We see also that saturation is a measure of "purity" or "vividness" of the color.

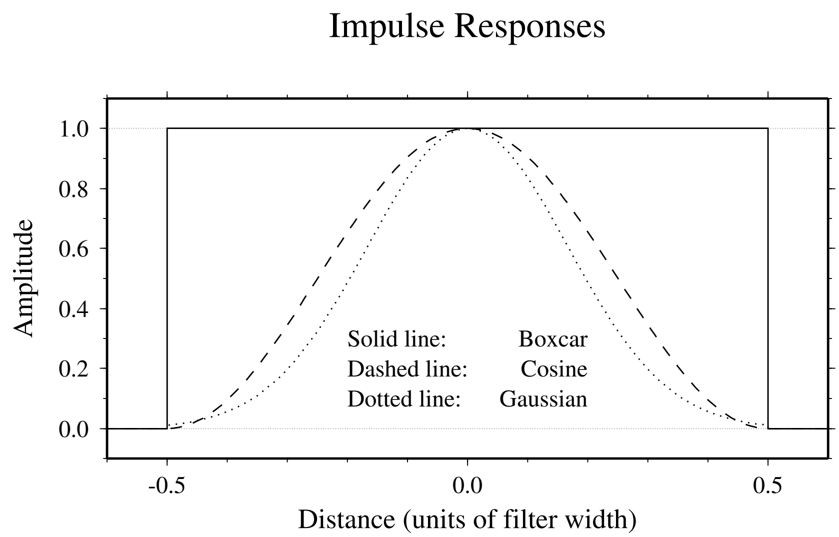
It remains to define value, and the formula above is really the best definition. But if you like our geometric constructions, try this: Take your point in RGB space and construct a line through it so that this line goes through the black point; produce this line from black past your point until it hits a face on which $v = 1$. All points on this line have the same hue. Note that this line and the line we made in the previous paragraph are both contained in the plane whose equation is hue = constant. These two lines meet at some arbitrary angle which varies depending on which point you chose. Thus HSV is not an orthogonal coordinate system. If the line you made in the previous paragraph happened to touch the gray axis at the black point, then these two lines are the same line, which is why the black point is special. Now, the line we made in this paragraph illustrates the following: If your chosen point is not already at the end of the line, where $v = 1$, then it is possible to move along the line in that direction so as to increase *(r, g, b)* while keeping the same hue. The effect this has on a color monitor is to make the color shine more brightly, but "brightness" has other meanings in color geometry, so let us say that if you can move in this way, you can make your hue "stronger"; if you are already on a plane where at least one of *(r, g, b)* = 255, then you cannot get a stronger version of the same hue. Thus, v measures strength. Note that it is not quite true to say that v measures distance away from the black point, because v is not equal to $\sqrt{r^2+g^2+b^2}/255$.

The RGB system is understandable because it is cartesian, and we all learned cartesian coordinates in school. But it doesn't help us create a tint or shade of a color; we cannot say, "We want orange, and a lighter shade of orange, or a less vivid orange". With HSV we can do this, by saying, "Orange must be between red and yellow, so its hue is about $h = 30$; a less vivid orange has a lesser s, a darker orange has a lesser v". On the other hand, the HSV system is a peculiar geometric construction, it is not an orthogonal coordinate system, and it is not found by a matrix transformation of RGB; these make it difficult in some cases too. Note that a move toward black or a move toward white will change both s and v, in the general case of an interior point in the cube. The HSV system also doesn't behave well for very dark colors, where the gray point is near black and the two lines we constructed above are almost parallel. If you are trying to create nice colors for drawing chocolates, for example, you may be better off guessing in RGB coordinates.

Well, there you have it, folks. We've been doing GMT for 5 years and all we know about color can be written in less than 3 pages. We hope we haven't told you any lies. For more details, you should consult a book about color systems. But as example 11 shows, a lot can be learned by experimenting with GMT tools. Our thanks to John Lillibridge for Example 11.

Appendix J: Filtering of data in GMT

The GMT programs *filter1d* and *grdfilter* allow low-pass filtering of data by convolution in the spatial domain. (To filter a grid by Fourier transform use *grdfft*). The filter type and width are specified by an argument -F<type><width>. The boxcar, cosine arch, and Gaussian filters are all linear operators and their effect on the frequency content of the data (the transfer function) can be calculated. The median and mode estimators are not linear operators, strictly speaking, and their effect on frequency content cannot be calculated. In *filter1d* the width is a length of the time or space ordinate axis, while in *grdfilter* it is the diameter of a circular area whose distance unit is related to the grid mesh via the -D option. The boxcar filter is a simple running average, while the cosine and gaussian filters are weighted running averages. The weight functions (impulse responses) and transfer functions of the linear filters are shown below.



There are many definitions of the gaussian impulse and its transfer function (e.g., see Bracewell). We define σ equal to 1/6 of the width, and then the impulse response as $\exp(-0.5 * (t/\sigma)^2)$. With this definition the transfer function is $\exp(-2(\pi\sigma f)^2)$, and the wavelength at which the transfer function equals 0.5 is about 5.34 σ.