XFree86 Video Timings HOWTO

Eric Steven Raymond

Thyrsus Enterprises

esr@thyrsus.com

This is version 5.0

$Date: 2000/08/22 12:35:07 $

Revision History

Revision 5.0	21 August 2000	Revised by: esr
First DocBook version.		

How to compose a mode line for your card/monitor combination under XFree86. The XFree86 distribution now includes good facilities for configuring most standard combinations; this document is mainly useful if you are tuning a custom mode line for a high−performance monitor or very unusual hardware. It may also help you in using kvideogen to generate mode lines, or xvidtune to tweak a standard mode that is not quite right for your monitor.

Table of Contents

1. Disclaimer

You use the material herein *solely at your own risk*. It is possible to harm both your monitor and yourself when driving it outside the manufacturer's specs. Read Overdriving Your Monitor for detailed cautions. Any damage to you or your monitor caused by overdriving it is your problem.

The most up−to−date version of this HOWTO can be found at the Linux Documentation Project web page.

Please direct comments, criticism, and suggestions for improvement to `<esr@snark.thyrsus.com>`. Please do *not* send email pleading for a magic solution to your special monitor problem, as doing so will only burn up my time and frustrate you −− everything I know about the subject is already in here.

2. Introduction

The XFree86 server allows users to configure their video subsystem and thus encourages best use of existing hardware. This document is intended to help you learn how to generate your own timing numbers to make optimum use of your video card and monitor.

We'll present a method for getting something that works, and then show you how you can experiment starting from that base to develop settings that optimize for your taste.

If you already have a mode that almost works (in particular, if one of predefined VESA modes gives you a stable display but one that's displaced right or left, or too small, or too large) you can go straight to the section on Fixing Problems with the Image. This will enlighten you on ways to tweak the timing numbers to achieve particular effects.

Don't assume that you need to get all the way into mode tuning just because your X comes up with a scrambled display first time after installation; it may be that most of the factory mode lines are OK and you just happened to default to one that doesn't fit your hardware. Instead, cycle through all your installed modes with CTRL−ALT−KP+. If some of the modes look OK, try commenting out all but a 640x480 and check that that mode works. If it does then uncomment a couple of other modes, e.g. an 800x600 and a 1024x768 at a frequency that your monitor should be able to handle.

More help is on the way. Many driver modules in the just−released XFree86 4.0 support DDC, the VESA Display Data Channel facility. When DDC is enabled, the monitor actually tells XFree86 what modelines it can support. So with 4.0 and any recently−manufactured monitor you are likely not to have to do any configuration at all.

3. Tools for Automatic Computation

If you have a relatively new monitor (1996 or later) that supports the PnP specification, there is a chance that you use the read−edid program to ask the monitor for its stastics and compute a mode line for you. See http://altern.org/vii/programs/linux/read−edid/.

Starting with XFree86 3.2, XFree86 provides an **XF86Setup** program that makes it easy to generate a working monitor mode interactively, without messing with video timing number directly. So you shouldn't actually need to calculate a base monitor mode in most cases. Unfortunately, **XF86Setup** has some limitations; it only knows about standard video modes up to 1280x1024. If you have a very high−performance monitor capable of 1600x1200 or more you will still have to compute your base monitor mode yourself.

There is a KDE tool called KVideoGen that computes modelines from basic monitor and card statistics. I've experimented with generating modelines from it, but haven't tried them in live use. Note that its horizontal and vertical `refresh rate' parameters are the same as the sync frequencies HSF and VSF we describe below. The `horizontal sync pulse' number seems to be a sync pulse width in microseconds, HSP (with the tool assuming fixed `front porch' HGT1 and `back porch' HGT2 values). If you don't know the `horizontal sync pulse' number it's safe to use the default.

Another XFree86 modeline generator lives here. You can either download the Python script or use the CGI form provided.

Recent versions of XFree86 provide a tool called **xvidtune** which you will probably find quite useful for testing and tuning monitor modes. It begins with a gruesome warning about the possible consequences of mistakes with it. If you pay careful attention to this document and learn what is behind the pretty numbers in xvidtune's boxes, you will become able to use xvidtune effectively and with confidence.

If you have **xvidtune**(1), you'll be able to test new modes on the fly, without modifying your X configuration files or even rebooting your X server. Otherwise, XFree86 allows you to hot−key between different modes defined in Xconfig (see XFree86.man for details). Use this capabilty to save yourself hassles! When you want to test a new mode, give it a unique mode label and add it to the *end* of your hot−key list. Leave a known−good mode as the default to fall back on if the test mode doesn't work.

Towards the end of this document, we include a `modeplot' script that you can use to produce an analog graph of available modes. This is not directly helpful for generating modelines, but it may help you better understand the relationships that define them.

---

4. How Video Displays Work

Knowing how the display works is essential to understanding what numbers to put in the various fields in the file Xconfig. Those values are used in the lowest levels of controlling the display by the XFree86 server.

The display generates a picture from what you could consider to be a series of raster dots. The dots are arranged from left to right to form lines. The lines are arranged from top to bottom to form the picture. The dots emit light when they are struck by the electron beams inside the display, one for each primary color. To make the beams strike each dot for an equal amount of time, the beams are swept across the display in a constant pattern, called a raster.

We say "what you could consider to be a series of dots" because these raster dots don't actually correspond to physical phosphor dots. The physical phosphor dots are much smaller than raster dots −− they have to be, otherwise the display would suffer from severe moiré−pattern effects. The raster dots are really samples of the analog driver signal, and display as a grid of dots only because the peaks and valleys in the signal are quite regularly and finely spaced.

The pattern starts at the top left of the screen, goes across the screen to the right in a straight line, moving ever so slightly "downhill" (the downhill slope is too small to be perceptible). Then the beams are swept back to the left side of the display, starting at a new line. The new line is swept from left to right just as the first line was. This pattern is repeated until the bottom line on the display has been swept. Then the beams are moved from the bottom right corner of the display (sweeping back and forth a few times) to the top left corner, and the pattern is started over again.

There is one variation of this scheme known as interlacing: here only every second line is swept during one half−frame and the others are filled in during a second half−frame.

Starting the beams at the top left of the display is called the beginning of a frame. The frame ends when the beams reach the the top left corner again as they come from the bottom right corner of the display. A frame is made up of all of the lines the beams traced from the top of the display to the bottom.

If the electron beams were on all of the time they were sweeping through the frame, all of the dots on the display would be illuminated. There would be no black border around the edges of the display. At the edges of the display the picture would become distorted because the beams are hard to control there. To reduce the distortion, the dots around the edges of the display are not illuminated by the beams (because they're turned off) even though the beams, if they were turned on, would be pointing at them. The viewable area of the display is reduced this way.

Another important thing to understand is what becomes of the beams when no spot is being painted on the visible area. The time the beams would have been illuminating the side borders of the display is used for sweeping the beams back from the right edge to the left. The time the beams would have been illuminating the top and bottom borders of the display is used for moving the beams from the bottom−right corner of the display to the top−left corner.

The adapter card generates the signals which cause the display to turn on the electron beams (according to the desired color) at each dot to generate a picture. The card also controls when the display moves the beams from the right side back to the left by generating a signal called the horizontal sync (for synchronization) pulse. One horizontal sync pulse occurs at the end of every line. The adapter also generates a vertical sync pulse which signals the display to move the beams to the top−left corner of the display. A vertical sync pulse is generated near the end of every frame.

The display requires that there be short time periods both before and after the horizontal and vertical sync pulses so that the position of the electron beams can stabilize. If the beams can't stabilize, the picture will not be steady.

For more information, see TV and Monitor Deflection Systems.

In a later section, we'll come back to these basics with definitions, formulas and examples to help you use them.

5. Basic Things to Know about your Display and Adapter

There are some fundamental things you need to know before hacking an Xconfig entry. These are:

- your monitor's horizontal and vertical sync frequency options
- your monitor's bandwidth
- your video adapter's driving clock frequencies, or "dot clocks"

5.1. The monitor sync frequencies

The horizontal sync frequency is just the number of times per second the monitor can write a horizontal scan line; it is the single most important statistic about your monitor. The vertical sync frequency is the number of times per second the monitor can traverse its beam vertically.

Sync frequencies are usually listed on the specifications page of your monitor manual. The vertical sync frequency number is typically calibrated in Hz (cycles per second), the horizontal one in KHz (kilocycles per second). The usual ranges are between 50 and 150Hz vertical, and between 31 and 135KHz horizontal.

If you have a multisync monitor, these frequencies will be given as ranges. Some monitors, especially lower−end ones, have multiple fixed frequencies. These can be configured too, but your options will be severely limited by the built−in monitor characteristics. Choose the highest frequency pair for best resolution. And be careful −−− trying to clock a fixed−frequency monitor at a higher speed than it's designed for can easily damage it.

Earlier versions of this guide were pretty cavalier about overdriving multisync monitors, pushing them past their nominal highest vertical sync frequency in order to get better performance. We have since had more reasons pointed out to us for caution on this score; we'll cover those under Overdriving Your Monitor below.

5.2. The monitor's video bandwidth

Your monitor's video bandwidth should be included on the manual's spec page. If it's not, look at the monitor's higest rated resolution. As a rule of thumb, here's how to translate these into bandwidth estimates (and thus into rough upper bounds for the dot clock you can use):

```
        640x480                 25
        800x600                 36
        1024x768                65
        1024x768 interlaced     45
        1280x1024               110
        1600x1200               185
```

BTW, there's nothing magic about this table; these numbers are just the lowest dot clocks per resolution in the standard XFree86 Modes database (except for the last, which I extrapolated). The bandwidth of your monitor may actually be higher than the minimum needed for its top resolution, so don't be afraid to try a dot clock a few MHz higher.

Also note that bandwidth is seldom an issue for dot clocks under 65MHz or so. With an SVGA card and most hi−res monitors, you can't get anywhere near the limit of your monitor's video bandwidth. The following are examples:

```
        Brand                           Video Bandwidth
        ----------                      ---------------
        NEC 4D                          75Mhz
        Nano 907a                       50Mhz
        Nano 9080i                      60Mhz
        Mitsubishi HL6615               110Mhz
        Mitsubishi Diamond Scan         100Mhz
        IDEK MF-5117                    65Mhz
        IOCOMM Thinksync-17 CM-7126     136Mhz
        HP D1188A                       100Mhz
        Philips SC-17AS                 110Mhz
        Swan SW617                      85Mhz
        Viewsonic 21PS                  185Mhz
        PanaSync/Pro P21                220Mhz
```

Even low−end monitors usually aren't terribly bandwidth−constrained for their rated resolutions. The NEC Multisync II makes a good example −−− it can't even display 800x600 per its spec. It can only display 800x560. For such low resolutions you don't need high dot clocks or a lot of bandwidth; probably the best you can do is 32Mhz or 36Mhz, both of them are still not too far from the monitor's rated video bandwidth of 30Mhz.

At these two driving frequencies, your screen image may not be as sharp as it should be, but definitely of tolerable quality. Of course it would be nicer if NEC Multisync II had a video bandwidth higher than, say, 36Mhz. But this is not critical for common tasks like text editing, as long as the difference is not so significant as to cause severe image distortion (your eyes would tell you right away if this were so).

5.3. The card's dot clock

Your video adapter manual's spec page will usually give you the card's maximum dot clock (that is, the total number of pixels per second it can write to the screen).

If you don't have this information, the X server will get it for you. Recent versions of the X servers all support a −−probeonly option that prints out this information and exits without actually starting up X or changing the video mode.

If you don't have −probeonly, don't depair. Even if your X locks up your monitor, it will emit a line of clock and other info to standard error. If you redirect this to a file, it should be saved even if you have to reboot to get your console back.

The probe result or startup message should look something like one of the following examples:

If you're using XFree86:

```
Xconfig: /usr/X11R6/lib/X11/Xconfig
(**) stands for supplied, (--) stands for probed/default values
(**) Mouse: type: MouseMan, device: /dev/ttyS1, baudrate: 9600
Warning: The directory "/usr/andrew/X11fonts" does not exist.
         Entry deleted from font path.
```

```
(**) FontPath set to "/usr/lib/X11/fonts/misc/,/usr/lib/X11/fonts/75dpi/"
(--) S3: card type: 386/486 localbus
(--) S3: chipset:   924
                    ---
    Chipset -- this is the exact chip type; an early mask of the 86C911

(--) S3: chipset driver: s3_generic
(--) S3: videoram:  1024k
                    -----
         Size of on-board frame-buffer RAM

(**) S3: clocks:  25.00  28.00  40.00   3.00  50.00  77.00  36.00  45.00
(**) S3: clocks:   0.00   0.00  79.00  31.00  94.00  65.00  75.00  71.00
                  ------------------------------------------------------
                              Possible driving frequencies in MHz

(--) S3: Maximum allowed dot-clock: 110MHz
                                    ------
                                   Bandwidth
(**) S3: Mode "1024x768": mode clock =  79.000, clock used =  79.000
(--) S3: Virtual resolution set to 1024x768
(--) S3: Using a banksize of 64k, line width of 1024
(--) S3: Pixmap cache:
(--) S3: Using 2 128-pixel 4 64-pixel and 8 32-pixel slots
(--) S3: Using 8 pages of 768x255 for font caching
```

If you're using SGCS or X/Inside X:

```
WGA: 86C911 (mem: 1024k clocks: 25 28 40 3 50 77 36 45 0 0 79 31 94 65 75 71)
---  ------       -----         --------------------------------------------
 |     |            |                 Possible driving frequencies in MHz
 |     |            +-- Size of on-board frame-buffer RAM
 |     +-- Chip type
 +-- Server type
```

Note: do this with your machine unloaded (if at all possible). Because X is an application, its timing loops can collide with disk activity, rendering the numbers above inaccurate. Do it several times and watch for the numbers to stabilize; if they don't, start killing processes until they do. Your mouse daemon process, if you have one, is particularly likely to trip you up (that's gpm for Linux users, mousemgr for SVr4 users).

In order to avoid the clock−probe inaccuracy, you should clip out the clock timings and put them in your Xconfig as the value of the Clocks property −−− this suppresses the timing loop and gives X an exact list of the clock values it can try. Using the data from the example above:

```
wga
        Clocks  25 28 40 3 50 77 36 45 0 0 79 31 94 65 75 71
```

On systems with a highly variable load, this may help you avoid mysterious X startup failures. It's possible for X to come up, get its timings wrong due to system load, and then not be able to find a matching dot clock in its config database −−− or find the wrong one!

5.4. What these basic statistics control

The sync frequency ranges of your monitor, together with your video adapter's dot clock, determine the ultimate resolution that you can use. But it's up to the driver to tap the potential of your hardware. A superior hardware combination without an equally competent device driver is a waste of money. On the other hand, with a versatile device driver but less capable hardware, you can push the hardware a little beyond its rated performance. This is the design philosophy of XFree86.

You should match the dot clock you use to the monitor's video bandwidth. There's a lot of give here, though −−− some monitors can run as much as 30% over their nominal bandwidth. The risks here have to do with exceeding the monitor's rated vertical−sync frequency; we'll discuss them in detail below.

Knowing the bandwidth will enable you to make more intelligent choices between possible configurations. It may affect your display's visual quality (especially sharpness for fine details).

6. Interpreting the Basic Specifications

This section explains what the specifications above mean, and some other things you'll need to know. First, some definitions. Next to each in parentheses is the variable name we'll use for it when doing calculations

horizontal sync frequency (HSF)

Horizontal scans per second (see above).

vertical sync frequency (VSF)

Vertical scans per second (see above). Mainly important as the upper limit on your refresh rate.

dot clock (DCF)

More formally, `driving clock frequency'; The frequency of the crystal or VCO on your adaptor −−− the maximum dots−per−second it can emit.

video bandwidth (VB)

The highest frequency you can feed into your monitor's video input and still expect to see anything discernible. If your adaptor produces an alternating on/off pattern (as in an interlaced mode), its lowest frequency is half the DCF, so in theory bandwidth starts making sense at DCF/2. For tolerately crisp display of fine details in the video image, however, you don't want it much below your highest DCF, and preferably higher.

frame length (HFL, VFL)

Horizontal frame length (HFL) is the number of dot−clock ticks needed for your monitor's electron gun to scan one horizontal line, *including the inactive left and right borders*. Vertical frame length (VFL) is the number of scan lines in the *entire* image, including the inactive top and bottom borders.

screen refresh rate (RR)

The number of times per second your screen is repainted (this is also called "frame rate"). Higher frequencies are better, as they reduce flicker. 60Hz is good, VESA−standard 72Hz is better. Compute it as

```
        RR = DCF / (HFL * VFL)
```

Note that the product in the denominator is *not* the same as the monitor's visible resolution, but typically somewhat larger. We'll get to the details of this below.

The rates for which interlaced modes are usually specified (like 87Hz interlaced) are actually the half−frame rates: an entire screen seems to have about that flicker frequency for typical displays, but every single line is refreshed only half as often.

For calculation purposes we reckon an interlaced display at its full−frame (refresh) rate, i.e. 43.5Hz. The quality of an interlaced mode is better than that of a non−interlaced mode with the same full−frame rate, but definitely worse than the non−interlaced one corresponding to the half−frame rate.

6.1. About Bandwidth

Monitor makers like to advertise high bandwidth because it constrains the sharpness of intensity and color changes on the screen. A high bandwidth means smaller visible details.

Your monitor uses electronic signals to present an image to your eyes. Such signals always come in in wave form once they are converted into analog form from digitized form. They can be considered as combinations of many simpler wave forms each one of which has a fixed frequency, many of them are in the Mhz range, eg, 20Mhz, 40Mhz, or even 70Mhz. Your monitor video bandwidth is, effectively, the highest−frequency analog signal it can handle without distortion.

For our purposes, video bandwidth is mainly important as an approximate cutoff point for the highest dot clock you can use.

6.2. Sync Frequencies and the Refresh Rate:

Each horizontal scan line on the display is just the visible portion of a frame−length scan. At any instant there is actually only one dot active on the screen, but with a fast enough refresh rate your eye's persistence of vision enables you to "see" the whole image.

Here are some pictures to help:

```
     _______________________
    |                       |     The horizontal sync frequency
    |->->->->->->->->->->-> |     is the number of times per
    |                      )|     second that the monitor's
    |<-----<-----<-----<--- |     electron beam can trace
    |                       |     a pattern like this
    |                       |
    |                       |
    |                       |
    |_______________________|
     _______________________
    |        ^              |     The vertical sync frequency
    |       ^ |             |     is the number of times per
    |       | v             |     second that the monitor's
    |       ^ |             |     electron beam can trace
    |       | |             |     a pattern like this
    |       ^ |             |
    |       | v             |
    |       ^ |             |
    |_______|_v_____________|
```

Remember that the actual raster scan is a very tight zigzag pattern; that is, the beam moves left−right and at the same time up−down.

Now we can see how the dot clock and frame size relates to refresh rate. By definition, one hertz (hz) is one cycle per second. So, if your horizontal frame length is HFL and your vertical frame length is VFL, then to cover the entire screen takes (HFL * VFL) ticks. Since your card emits DCF ticks per second by definition, then obviously your monitor's electron gun(s) can sweep the screen from left to right and back and from bottom to top and back DCF / (HFL * VFL) times/sec. This is your screen's refresh rate, because it's how many times your screen can be updated (thus *refreshed*) per second!

You need to understand this concept to design a configuration which trades off resolution against flicker in whatever way suits your needs.

For those of you who handle visuals better than text, here is one:

```
        RR                                      VB
         |   min HSF                     max HSF |
         |    |             R1        R2  |      |
max VSF -+----|------------/----------/---|------+----- max VSF
         |    |:::::::::::/::::::::::/:::::\     |
         |    \::::::::::/::::::::::/:::::::\    |
         |     |::::::::/::::::::::/:::::::::|   |
         |     |:::::::/::::::::::/::::::::::\   |
         |     \::::::/::::::::::/::::::::::::\  |
         |      \::::/::::::::::/::::::::::::::| |
         |       |::/::::::::::/:::::::::::::::| |
         |        \/::::::::::/:::::::::::::::::\|
         |        /\:::::::::/:::::::::::::::::::|
         |       /  \:::::::/::::::::::::::::::::|\
         |      /    |:::::/:::::::::::::::::::::| |
         |     /     \::::/::::::::::::::::::::::| \
min VSF -+----/-------\--/-----------------------|--\--- min VSF
         |   /         \/                        |   \
         +--/----------/\------------------------+----\- DCF
           R1        R2  \                       |     \
                          min HSF                |    max HSF
                                                 VB
```

This is a generic monitor mode diagram. The x axis of the diagram shows the clock rate (DCF), the y axis represents the refresh rate (RR). The filled region of the diagram describes the monitor's capabilities: every point within this region is a possible video mode.

The lines labeled `R1' and `R2' represent a fixed resolutions (such as 640x480); they are meant to illustrate how one resolution can be realized by many different combinations of dot clock and refresh rate. The R2 line would represent a higher resolution than R1.

The top and bottom boundaries of the permitted region are simply horizontal lines representing the limiting values for the vertical sync frequency. The video bandwidth is an upper limit to the clock rate and hence is represented by a vertical line bounding the capability region on the right.

Under Plotting Monitor Capabilities you'll find a program that will help you plot a diagram like this (but much nicer, with X graphics) for your individual monitor. That section also discusses the interesting part; the derivation of the boundaries resulting from the limits on the horizontal sync frequency.

7. Tradeoffs in Configuring your System

Another way to look at the formula we derived above is

```
        DCF = RR * HFL * VFL
```

That is, your dot clock is fixed. You can use those dots per second to buy either refresh rate, horizontal resolution, or vertical resolution. If one of those increases, one or both of the others must decrease.

Note, though, that your refresh rate cannot be greater than the maximum vertical sync frequency of your monitor. Thus, for any given monitor at a given dot clock, there is a minimum product of frame lengths below which you can't force it.

In choosing your settings, remember: if you set RR too low, you will get mugged by screen flicker. Keep it above 60Hz. 72Hz is the VESA ergonomic standard. 120Hz is the flicker rate of fluorescent lights; if you're sensitive to those, you need to keep it above that.

Flicker is very eye−fatiguing, though human eyes are adaptable and peoples' tolerance for it varies widely. If you face your monitor at a 90% viewing angle, are using a dark background and a good contrasting color for foreground, and stick with low to medium intensity, you *may* be comfortable at as little as 45Hz.

The acid test is this: open a xterm with pure white back−ground and black foreground using **xterm −bg white −fg black** and make it so large as to cover the entire viewable area. Now turn your monitor's intensity to 3/4 of its maximum setting, and turn your face away from the monitor. Try peeking at your monitor sideways (bringing the more sensitive peripheral−vision cells into play). If you don't sense any flicker or if you feel the flickering is tolerable, then that refresh rate is fine with you. Otherwise you better configure a higher refresh rate, because that semi−invisible flicker is going to fatigue your eyes like crazy and give you headaches, even if the screen looks OK to normal vision.

For interlaced modes, the amount of flicker depends on more factors such as the current vertical resolution and the actual screen contents. So just experiment. You won't want to go much below about 85Hz half frame rate, though.

So let's say you've picked a minimum acceptable refresh rate. In choosing your HFL and VFL, you'll have some room for maneuver.

8. Memory Requirements

Available frame−buffer RAM may limit the resolution you can achieve on color or gray−scale displays. It probably isn't a factor on displays that have only two colors, white and black with no shades of gray in between.

For 256−color displays, a byte of video memory is required for each visible dot to be shown. This byte contains the information that determines what mix of red, green, and blue is generated for its dot. To get the amount of memory required, multiply the number of visible dots per line by the number of visible lines. For a display with a resolution of 1024x768, this would be 1024 x 768 = 786432, which is the number of visible dots on the display. This is also, at one byte per dot, the number of bytes of video memory that will be necessary on your adapter card.

Thus, your memory requirement will typically be (HR * VR)/1024 Kbytes of VRAM, rounded up (it would come to 768K exactly in this example). If you have more memory than strictly required, you'll have extra for virtual−screen panning.

However, if you only have 512K on board yor video card, then you won't be able to use this resolution. Even if you have a good monitor, without enough video RAM, you can't take advantage of your monitor's potential. On the other hand, if your SVGA has one meg, but your monitor can display at most 800x600, then high resolution is beyond your reach anyway (see Using Interlaced Modes for a possible remedy).

Don't worry if you have more memory than required; XFree86 will make use of it by allowing you to scroll your viewable area (see the Xconfig file documentation on the virtual screen size parameter). Remember also that a card with 512K bytes of memory really doesn't have 512,000 bytes installed, it has 512 x 1024 = 524,288 bytes.

If you're running X/Inside using an S3 card, and are willing to live with 16 colors (4 bits per pixel), you can set depth 4 in Xconfig and effectively double the resolution your card can handle. S3 cards, for example, normally do 1024x768x256. You can make them do 1280x1024x16 with depth 4.

9. Computing Frame Sizes

Warning: this method was developed for multisync monitors. It will probably work with fixed−frequency monitors as well, but no guarantees!

Start by dividing DCF by your highest available HSF to get a horizontal frame length.

For example; suppose you have a Sigma Legend SVGA with a 65MHz dot clock, and your monitor has a 55KHz horizontal scan frequency. The quantity (DCF / HSF) is then 1181 (65MHz = 65000KHz; 65000/55 = 1181).

Now for our first bit of black magic. You need to round this figure to the nearest multiple of 8. This has to do with the VGA hardware controller used by SVGA and S3 cards; it uses an 8−bit register, left−shifted 3 bits, for what's really an 11−bit quantity. Other card types such as ATI 8514/A may not have this requirement, but we don't know and the correction can't hurt. So round the usable horizontal frame length figure down to 1176.

This figure (DCF / HSF rounded to a multiple of 8) is the minimum HFL you can use. You can get longer HFLs (and thus, possibly, more horizontal dots on the screen) by setting the sync pulse to produce a lower HSF. But you'll pay with a slower and more visible flicker rate.

As a rule of thumb, 80% of the horizontal frame length is available for horizontal resolution, the visible part of the horizontal scan line (this allows, roughly, for borders and sweepback time −− that is, the time required for the beam to move from the right screen edge to the left edge of the next raster line). In this example, that's 944 ticks.

Now, to get the normal 4:3 screen aspect ratio, set your vertical resolution to 3/4ths of the horizontal resolution you just calculated. For this example, that's 708 ticks. To get your actual VFL, multiply that by 1.05 to get 743 ticks.

The 4:3 is not technically magic; nothing prevents you from using a different ratio if that will get the best use out of your screen real estate. It does make figuring frame height and frame width from the diagonal size convenient, you just multiply the diagonal by by 0.8 to get width and 0.6 to get height.

So, HFL=1176 and VFL=743. Dividing 65MHz by the product of the two gives us a nice, healthy 74.4Hz refresh rate. Excellent! Better than VESA standard! And you got 944x708 to boot, more than the 800 by 600 you were probably expecting. Not bad at all!

You can even improve the refresh rate further, to almost 76 Hz, by using the fact that monitors can often sync horizontally at 2khz or so higher than rated, and by lowering VFL somewhat (that is, taking less than 75% of 944 in the example above). But before you try this "overdriving" maneuver, if you do, make *sure* that your monitor electron guns can sync up to 76 Hz vertical. (the popular NEC 4D, for instance, cannot. It goes only up to 75 Hz VSF). (See Overdriving Your Monitor for more general discussion of this issue.)

So far, most of this is simple arithmetic and basic facts about raster displays. Hardly any black magic at all!

10. Black Magic and Sync Pulses

OK, now you've computed HFL/VFL numbers for your chosen dot clock, found the refresh rate acceptable, and checked that you have enough VRAM. Now for the real black magic −− you need to know when and where to place synchronization pulses.

The sync pulses actually control the horizontal and vertical scan frequencies of the monitor. The HSF and VSF you've pulled off the spec sheet are nominal, approximate maximum sync frequencies. The sync pulse in the signal from the adapter card tells the monitor how fast to actually run.

Recall the two pictures above? Only part of the time required for raster−scanning a frame is used for displaying viewable image (ie. your resolution).

10.1. Horizontal Sync:

By previous definition, it takes HFL ticks to trace the a horizontal scan line. Let's call the visible tick count (your horizontal screen resolution) HR. Then Obviously, HR < HFL by definition. For concreteness, let's assume both start at the same instant as shown below:

```
|___ __ __ __ __ __ __ __ __ __ __ __ __
|_ _ _ _ _ _ _ _ _ _ _ _ _              |
|_______________________|_______________|_____
0                       ^               ^     unit: ticks
                        |   ^       ^   |
                        HR  |       |  HFL
                        |   |<----->|   |
                        |<->|  HSP  |<->|
                        HGT1         HGT2
```

Now, we would like to place a sync pulse of length HSP as shown above, ie, between the end of clock ticks for display data and the end of clock ticks for the entire frame. Why so? because if we can achieve this, then your screen image won't shift to the right or to the left. It will be where it supposed to be on the screen, covering squarely the monitor's viewable area.

Furthermore, we want about 30 ticks of "guard time" on either side of the sync pulse. This is represented by HGT1 and HGT2. In a typical configuration HGT1 != HGT2, but if you're building a configuration from scratch, you want to start your experimentation with them equal (that is, with the sync pulse centered).

The symptom of a misplaced sync pulse is that the image is displaced on the screen, with one border excessively wide and the other side of the image wrapped around the screen edge, producing a white edge line and a band of "ghost image" on that side. A way−out−of−place vertical sync pulse can actually cause the image to roll like a TV with a mis−adjusted vertical hold (in fact, it's the same phenomenon at work).

If you're lucky, your monitor's sync pulse widths will be documented on its specification page. If not, here's where the real black magic starts...

You'll have to do a little trial and error for this part. But most of the time, we can safely assume that a sync pulse is about 3.5 to 4.0 microsecond in length.

For concretness again, let's take HSP to be 3.8 microseconds (which btw, is not a bad value to start with when experimenting).

Now, using the 65Mhz clock timing above, we know HSP is equivalent to 247 clock ticks (= 65 * 10**6 * 3.8 * 10^−6) [recall M=10^6, micro=10^−6]

Some vendors like to quote their horizontal framing parameters as timings rather than dot widths. You may see the following terms:

active time (HAT)

> Corresponds to HR, but in time units (usually microseconds). HAT * DCF = HR.

blanking time (HBT)

> Corresponds to (HFL − HR), but in time units (usually microseconds). HBT * DCF = (HFL − HR).

front porch (HFP)

> This is just HGT1.

sync time

> This is just HSP.

back porch (HBP)

> This is just HGT2.

10.2. Vertical Sync:

Going back to the picture above, how do we place the 247 clock ticks as shown in the picture?

Using our example, HR is 944 and HFL is 1176. The difference between the two is 1176 − 944=232 < 247! Obviously we have to do some adjustment here. What can we do?

The first thing is to raise 1176 to 1184, and lower 944 to 936. Now the difference = 1184−936= 248. Hmm, closer.

Next, instead using 3.8, we use 3.5 for calculating HSP; then, we have 65*3.5=227. Looks better. But 248 is not much higher than 227. It's normally necessary to have 30 or so clock ticks between HR and the start of SP, and the same for the end of SP and HFL. AND they have to be multiple of eight! Are we stuck?

No. Let's do this, 936 % 8 = 0, (936 + 32) % 8 = 0 too. But 936 + 32 = 968, 968 + 227 = 1195, 1195 + 32 = 1227. Hmm.. this looks not too bad. But it's not a multiple of 8, so let's round it up to 1232.

But now we have potential trouble, the sync pulse is no longer placed right in the middle between h and H any more. Happily, using our calculator we find 1232 − 32 = 1200 is also a multiple of 8 and (1232 − 32) − 968 = 232 corresponding using a sync pulse of 3.57 microseconds long, still reasonable.

In addition, 936/1232 ~ 0.76 or 76%, still not far from 80%, so it should be all right.

Furthermore, using the current horizontal frame length, we basically ask our monitor to sync at 52.7khz (= 65Mhz/1232) which is within its capability. No problems.

Using rules of thumb we mentioned before, 936*75%=702, This is our new vertical resolution. 702 * 1.05 = 737, our new vertical frame length.

Screen refresh rate = 65Mhz/(737*1232)=71.6 Hz. This is still excellent.

Figuring the vertical sync pulse layout is similar:

```
   |___ __ __ __ __ __ __ __ __ __ __ __ __
   |_ _ _ _ _ _ _ _ _ _ _ _                |
   |_______________________|_______________|______
   0                      VR              VFL     unit: ticks
                           ^   ^       ^
                           |   |       |
                           |<->|<----->|
                            VGT    VSP
```

We start the sync pulse just past the end of the vertical display data ticks. VGT is the vertical guard time required for the sync pulse. Most monitors are comfortable with a VGT of 0 (no guard time) and we'll use that in this example. A few need two or three ticks of guard time, and it usually doesn't hurt to add that.

Returning to the example: since by the defintion of frame length, a vertical tick is the time for tracing a complete HORIZONTAL frame, therefore in our example, it is 1232/65Mhz=18.95us.

Experience shows that a vertical sync pulse should be in the range of 50us and 300us. As an example let's use 150us, which translates into 8 vertical clock ticks (150us/18.95us~8).

Some makers like to quote their vertical framing parameters as timings rather than dot widths. You may see the following terms:

active time (VAT)

Corresponds to VR, but in milliseconds. VAT * VSF = VR.

blanking time (VBT)

Corresponds to (VFL – VR), but in milliseconds. VBT * VSF = (VFL – VR).

front porch (VFP)

This is just VGT.

sync time

This is just VSP.

back porch (VBP)

This is like a second guard time after the vertical sync pulse. It is often zero.

11. Putting it All Together

The Xconfig file Table of Video Modes contains lines of numbers, with each line being a complete specification for one mode of X−server operation. The fields are grouped into four sections, the name section, the clock frequency section, the horizontal section, and the vertical section.

The name section contains one field, the name of the video mode specified by the rest of the line. This name is referred to on the "Modes" line of the Graphics Driver Setup section of the Xconfig file. The name field may be omitted if the name of a previous line is the same as the current line.

The dot clock section contains only the dot clock (what we've called DCF) field of the video mode line. The number in this field specifies what dot clock was used to generate the numbers in the following sections.

The horizontal section consists of four fields which specify how each horizontal line on the display is to be generated. The first field of the section contains the number of dots per line which will be illuminated to form the picture (what we've called HR). The second field of the section (SH1) indicates at which dot the horizontal sync pulse will begin. The third field (SH2) indicates at which dot the horizontal sync pulse will end. The fourth field specifies the toal horzontal frame length (HFL).

The vertical section also contains four fields. The first field contains the number of visible lines which will appear on the display (VR). The second field (SV1) indicates the line number at which the vertical sync pulse will begin. The third field (SV2) specifies the line number at which the vertical sync pulse will end. The fourth field contains the total vertical frame length (VFL).

Example:

```
#Modename    clock  horizontal timing  vertical timing

"752x564"     40    752 784  944 1088  564 567 569 611
              44.5  752 792  976 1240  564 567 570 600
```

(Note: stock X11R5 doesn't support fractional dot clocks.)

For Xconfig, all of the numbers just mentioned − the number of illuminated dots on the line, the number of dots separating the illuminated dots from the beginning of the sync pulse, the number of dots representing the duration of the pulse, and the number of dots after the end of the sync pulse − are added to produce the number of dots per line. The number of horizontal dots must be evenly divisible by eight.

Example horizontal numbers: 800 864 1024 1088

This sample line has the number of illuminated dots (800) followed by the number of the dot when the sync pulse starts (864), followed by the number of the dot when the sync pulse ends (1024), followed by the number of the last dot on the horizontal line (1088).

Note again that all of the horizontal numbers (800, 864, 1024, and 1088) are divisible by eight! This is not required of the vertical numbers.

The number of lines from the top of the display to the bottom form the frame. The basic timing signal for a frame is the line. A number of lines will contain the picture. After the last illuminated line has been displayed, a delay of a number of lines will occur before the vertical sync pulse is generated. Then the sync pulse will last for a few lines, and finally the last lines in the frame, the delay required after the pulse, will be generated. The numbers that specify this mode of operation are entered in a manner similar to the following example.

Example vertical numbers: 600 603 609 630

This example indicates that there are 600 visible lines on the display, that the vertical sync pulse starts with the 603rd line and ends with the 609th, and that there are 630 total lines being used.

Note that the vertical numbers don't have to be divisible by eight!

Let's return to the example we've been working. According to the above, all we need to do from now on is to write our result into Xconfig as follows:

```
<name>   DCF     HR  SH1 SH2   HFL   VR  SV1 SV2 VFL
```

where SH1 is the start tick of the horizontal sync pulse and SH2 is its end tick; similarly, SV1 is the start tick of the vertical sync pulse and SV2 is its end tick.

To place these, recall the discussion of black magic and sync pulses given above. SH1 is the dot that begins the leading edge of the horiziontal sync pulse; thus, SH1 = HR + HGT1. SH2 is the trailing edge; thus, SH2 = SH1 + HSP. Similarly, SV1 = VR + VGT (but VGT is usually zero) and SV2 = SV1 + VSP.

```
#name    clock   horizontal timing   vertical timing    flag
936x702  65      936 968 1200 1232   702 702 710 737
```

No special flag necessary; this is a non−interlaced mode. Now we are really done.

12. Overdriving Your Monitor

You should absolutely *not* try exceeding your monitor's scan rates if it's a fixed−frequency type. You can smoke your hardware doing this! There are potentially subtler problems with overdriving a multisync monitor which you should be aware of.

Having a pixel clock higher than the monitor's maximum bandwidth is rather harmless, in contrast. It's exceeding the rated maximum sync frequencies that's problematic. Some modern monitors might have protection circuitry that shuts the monitor down at dangerous scan rates, but don't rely on it. In particular there are older multisync monitors (like the Multisync II) which use just one horizontal transformer. These monitors will not have much protection against overdriving them. While you necessarily have high voltage regulation circuitry (which can be absent in fixed frequency monitors), it will not necessarily cover every conceivable frequency range, especially in cheaper models. This not only implies more wear on the circuitry, it can also cause the screen phosphors to age faster, and cause more than the specified radiation (including X−rays) to be emitted from the monitor.

However, the basic problematic magnitude in question here is the slew rate (the steepness of the video signals) of the video output drivers, and that is usually independent of the actual pixel frequency, but (if your board manufacturer cares about such problems) related to the maximum pixel frequency of the board.

So be careful out there...

13. Using Interlaced Modes

(This section is largely due to David Kastrup `<dak@pool.informatik.rwth-aachen.de>`)

At a fixed dot clock, an interlaced display is going to have considerably less noticable flicker than a non−interlaced display, if the vertical circuitry of your monitor is able to support it stably. It is because of this that interlaced modes were invented in the first place.

Interlaced modes got their bad repute because they are inferior to their non−interlaced companions at the same vertical scan frequency, VSF (which is what is usually given in advertisements). But they are definitely superior at the same horizontal scan rate, and that's where the decisive limits of your monitor/graphics card usually lie.

At a fixed *refresh rate* (or half frame rate, or VSF) the interlaced display will flicker more: a 90Hz interlaced display will be inferior to a 90Hz non−interlaced display. It will, however, need only half the video bandwidth and half the horizontal scan rate. If you compared it to a non−interlaced mode with the same dot clock and the same scan rates, it would be vastly superior: 45Hz non−interlaced is intolerable. With 90Hz interlaced, I have worked for years with my Multisync 3D (at 1024x768) and am very satisfied. I'd guess you'd need at least a 70Hz non−interlaced display for similar comfort.

You have to watch a few points, though: use interlaced modes only at high resolutions, so that the alternately lighted lines are close together. You might want to play with sync pulse widths and positions to get the most stable line positions. If alternating lines are bright and dark, interlace will *jump* at you. I have one application that chooses such a dot pattern for a menu background (XCept, no other application I know does that, fortunately). I switch to 800x600 for using XCept because it really hurts my eyes otherwise.

For the same reason, use at least 100dpi fonts, or other fonts where horizontal beams are at least two lines thick (for high resolutions, nothing else will make sense anyhow).

And of course, never use an interlaced mode when your hardware would support a non−interlaced one with similar refresh rate.

If, however, you find that for some resolution you are pushing either monitor or graphics card to their upper limits, and getting dissatisfactorily flickery or outwashed (bandwidth exceeded) display, you might want to try tackling the same resolution using an interlaced mode. Of course this is useless if the VSF of your monitor is already close to its limits.

Design of interlaced modes is easy: do it like a non−interlaced mode. Just two more considerations are necessary: you need an odd total number of vertical lines (the last number in your mode line), and when you specify the "interlace" flag, the actual vertical frame rate for your monitor doubles. Your monitor needs to support a 90Hz frame rate if the mode you specified looks like a 45Hz mode apart from the "Interlace" flag.

As an example, here is my modeline for 1024x768 interlaced: my Multisync 3D will support up to 90Hz vertical and 38kHz horizontal.

```
ModeLine "1024x768" 45 1024 1048 1208 1248 768 768 776 807 Interlace
```

Both limits are pretty much exhausted with this mode. Specifying the same mode, just without the "Interlace" flag, still is almost at the limit of the monitor's horizontal capacity (and strictly speaking, a bit under the lower limit of vertical scan rate), but produces an intolerably flickery display.

Basic design rules: if you have designed a mode at less than half of your monitor's vertical capacity, make the vertical total of lines odd and add the "Interlace" flag. The display's quality should vastly improve in most cases.

If you have a non−interlaced mode otherwise exhausting your monitor's specs where the vertical scan rate lies about 30% or more under the maximum of your monitor, hand−designing an interlaced mode (probably with somewhat higher resolution) could deliver superior results, but I won't promise it.

14. Questions and Answers

Q: The example you gave is not a standard screen size, can I use it?
Q: It this the only resolution given the 65Mhz dot clock and 55Khz HSF?
Q: You just mentioned two standard resolutions. In Xconfig, there are many standard resolutions available, can you tell me whether there's any point in tinkering with timings?
Q: Can you summarize what we have discussed so far?

Q: The example you gave is not a standard screen size, can I use it?

A: Why not? There is NO reason whatsover why you have to use 640x480, 800x600, or even 1024x768. The XFree86 servers let you configure your hardware with a lot of freedom. It usually takes two to three tries to come up the right one. The important thing to shoot for is high refresh rate with reasonable viewing area. not high resolution at the price of eye−tearing flicker!

Q: It this the only resolution given the 65Mhz dot clock and 55Khz HSF?

A: Absolutely not! You are encouraged to follow the general procedure and do some trial−and−error to come up a setting that's really to your liking. Experimenting with this can be lots of fun. Most settings may just give you nasty video hash, but in practice a modern multi−sync monitor is usually not damaged easily. Be sure though, that your monitor can support the frame rates of your mode before using it for longer times.

Beware fixed−frequency monitors! This kind of hacking around can damage them rather quickly. Be sure you use valid refresh rates for *every* experiment on them.

Q: You just mentioned two standard resolutions. In Xconfig, there are many standard resolutions available, can you tell me whether there's any point in tinkering with timings?

A: Absolutely! Take, for example, the "standard" 640x480 listed in the current Xconfig. It employes 25Mhz driving frequency, frame lengths are 800 and 525 => refresh rate ~ 59.5Hz. Not too bad. But 28Mhz is a commonly available driving frequency from many SVGA boards. If we use it to drive 640x480, following the procedure we discussed above, you would get frame lengths like 812 (round down to 808) and 505. Now the refresh rate is raised to 68Hz, a quite significant improvement over the standard one.

Q: Can you summarize what we have discussed so far?

A: In a nutshell:

- for any fixed driving frequency, raising max resolution incurs the penalty of lowering refresh rate and thus introducing more flicker.
- if high resolution is desirable and your monitor supports it, try to get a SVGA card that provides a matching dot clock or DCF. The higher, the better!

---

15. Fixing Problems with the Image.

OK, so you've got your X configuration numbers. You put them in Xconfig with a test mode label. You fire up X, hot−key to the new mode, ... and the image doesn't look right. What do you do? Here's a list of common video image distortions and how to fix them.

(Fixing these minor distortions is where **xvidtune**(1) really shines.)

You *move* the image by changing the sync pulse timing. You *scale* it by changing the frame length (you need to move the sync pulse to keep it in the same relative position, otherwise scaling will move the image as well). Here are some more specific recipes:

The horizontal and vertical positions are independent. That is, moving the image horizontally doesn't affect placement vertically, or vice−versa. However, the same is not quite true of scaling. While changing the horizontal size does nothing to the vertical size or vice versa, the total change in both may be limited. In particular, if your image is too large in both dimensions you will probably have to go to a higher dot clock to fix it. Since this raises the usable resolution, it is seldom a problem!

15.1. The image is displaced to the left or right

To fix this, move the horizontal sync pulse. That is, increment or decrement (by a multiple of 8) the middle two numbers of the horizontal timing section that define the leading and trailing edge of the horizontal sync pulse.

If the image is shifted left (right border too large, you want to move the image to the right) decrement the numbers. If the image is shifted right (left border too large, you want it to move left) increment the sync pulse.

15.2. The image is displaced up or down

To fix this, move the vertical sync pulse. That is, increment or decrement the middle two numbers of the vertical timing section that define the leading and trailing edge of the vertical sync pulse.

If the image is shifted up (lower border too large, you want to move the image down) decrement the numbers. If the image is shifted down (top border too large, you want it to move up) increment the numbers.

15.3. The image is too large both horizontally and vertically

Switch to a higher card clock speed. If you have multiple modes in your clock file, possibly a lower−speed one is being activated by mistake.

15.4. The image is too wide (too narrow) horizontally

To fix this, increase (decrease) the horizontal frame length. That is, change the fourth number in the first timing section. To avoid moving the image, also move the sync pulse (second and third numbers) half as far, to keep it in the same relative position.

15.5. The image is too deep (too shallow) vertically

To fix this, increase (decrease) the vertical frame length. That is, change the fourth number in the second timing section. To avoid moving the image, also move the sync pulse (second and third numbers) half as far, to keep it in the same relative position.

Any distortion that can't be handled by combining these techniques is probably evidence of something more basically wrong, like a calculation mistake or a faster dot clock than the monitor can handle.

Finally, remember that increasing either frame length will decrease your refresh rate, and vice−versa.

Occasionally you can fix minor distortions by fiddling with the picture controls on your monitor. The disadvantage is that if you take your controls too far off the neutral (factory) setting to fix graphics−mode problems, you may end up with a wacky image in text mode. It's better to get your modeline right.

16. Plotting Monitor Capabilities

To plot a monitor mode diagram, you'll need the gnuplot package (a freeware plotting language for UNIX–like operating systems) and the tool **modeplot**, a shell/gnuplot script to plot the diagram from your monitor characteristics, entered as command–line options.

Here is a copy of **modeplot**:

```
#!/bin/sh
#
# modeplot -- generate X mode plot of available monitor modes
#
# Do `modeplot -?' to see the control options.
#

# Monitor description. Bandwidth in MHz, horizontal frequencies in kHz
# and vertical frequencies in Hz.
TITLE="Viewsonic 21PS"
BANDWIDTH=185
MINHSF=31
MAXHSF=85
MINVSF=50
MAXVSF=160
ASPECT="4/3"
vesa=72.5       # VESA-recommended minimum refresh rate

while [ "$1" != "" ]
do
        case $1 in
        -t) TITLE="$2"; shift;;
        -b) BANDWIDTH="$2"; shift;;
        -h) MINHSF="$2" MAXHSF="$3"; shift; shift;;
        -v) MINVSF="$2" MAXVSF="$3"; shift; shift;;
        -a) ASPECT="$2"; shift;;
        -g) GNUOPTS="$2"; shift;;
        -?) cat <<EOF
modeplot control switches:

-t "<description>"      name of monitor            defaults to "Viewsonic 21PS"
-b <nn>                 bandwidth in MHz           defaults to 185
-h <min> <max>          min 38; max HSF (kHz)        defaults to 31 85
-v <min> <max>          min 38; max VSF (Hz)         defaults to 50 160
-a <aspect ratio>       aspect ratio               defaults to 4/3
-g "<options>"          pass options to gnuplot

The -b, -h and -v options are required, -a, -t, -g optional.  You can
use -g to pass a device type to gnuplot so that (for example) modeplot's
output can be redirected to a printer.  See gnuplot(1) for  details.

The modeplot tool was created by Eric S. Raymond <esr@thyrsus.com> based on
analysis and scratch code by Martin Lottermoser <Martin.Lottermoser@mch.sni.de>

This is modeplot $Revision: 1.1 $
EOF
                exit;;
        esac
        shift
done
```

```
gnuplot $GNUOPTS <<EOF
set title "$TITLE Mode Plot"

# Magic numbers.  Unfortunately, the plot is quite sensitive to changes in
# these, and they may fail to represent reality on some monitors.  We need
# to fix values to get even an approximation of the mode diagram.  These come
# from looking at lots of values in the ModeDB database.
F1 = 1.30       # multiplier to convert horizontal resolution to frame width
F2 = 1.05       # multiplier to convert vertical resolution to frame height

# Function definitions (multiplication by 1.0 forces real-number arithmetic)
ac = (1.0*$ASPECT)*F1/F2
refresh(hsync, dcf) = ac * (hsync**2)/(1.0*dcf)
dotclock(hsync, rr) = ac * (hsync**2)/(1.0*rr)
resolution(hv, dcf) = dcf * (10**6)/(hv * F1 * F2)

# Put labels on the axes
set xlabel 'DCF (MHz)'
set ylabel 'RR (Hz)' 6  # Put it right over the Y axis

# Generate diagram
set grid
set label "VB" at $BANDWIDTH+1, ($MAXVSF + $MINVSF) / 2 left
set arrow from $BANDWIDTH, $MINVSF to $BANDWIDTH, $MAXVSF nohead
set label "max VSF" at 1, $MAXVSF-1.5
set arrow from 0, $MAXVSF to $BANDWIDTH, $MAXVSF nohead
set label "min VSF" at 1, $MINVSF-1.5
set arrow from 0, $MINVSF to $BANDWIDTH, $MINVSF nohead
set label "min HSF" at dotclock($MINHSF, $MAXVSF+17), $MAXVSF + 17 right
set label "max HSF" at dotclock($MAXHSF, $MAXVSF+17), $MAXVSF + 17 right
set label "VESA $vesa" at 1, $vesa-1.5
set arrow from 0, $vesa to $BANDWIDTH, $vesa nohead # style -1
plot [dcf=0:1.1*$BANDWIDTH] [$MINVSF-10:$MAXVSF+20] \
  refresh($MINHSF, dcf) notitle with lines 1, \
  refresh($MAXHSF, dcf) notitle with lines 1, \
  resolution(640*480,   dcf) title "640x480  " with points 2, \
  resolution(800*600,   dcf) title "800x600  " with points 3, \
  resolution(1024*768,  dcf) title "1024x768 " with points 4, \
  resolution(1280*1024, dcf) title "1280x1024" with points 5, \
  resolution(1600*1280, dcf) title "1600x1200" with points 6

pause 9999
EOF
```

Once you know you have **modeplot** and the gnuplot package in place, you'll need the following monitor characteristics:

- video bandwidth (VB)
- range of horizontal sync frequency (HSF)
- range of vertical sync frequency (VSF)

The plot program needs to make some simplifying assumptions which are not necessarily correct. This is the reason why the resulting diagram is only a rough description. These assumptions are:

- All resolutions have a single fixed aspect ratio AR = HR/VR. Standard resolutions have AR = 4/3 or AR = 5/4. The **modeplot** programs assumes 4/3 by default, but you can override this.
- For the modes considered, horizontal and vertical frame lengths are fixed multiples of horizontal and vertical resolutions, respectively:

```
        HFL = F1 * HR
        VFL = F2 * VR
```

As a rough guide, take F1 = 1.30 and F2 = 1.05 (see Computing Frame Sizes.

Now take a particular sync frequency, HSF. Given the assumptions just presented, every value for the clock rate DCF already determines the refresh rate RR, i.e. for every value of HSF there is a function RR(DCF). This can be derived as follows.

The refresh rate is equal to the clock rate divided by the product of the frame sizes:

```
        RR = DCF / (HFL * VFL)          (*)
```

On the other hand, the horizontal frame length is equal to the clock rate divided by the horizontal sync frequency:

```
        HFL = DCF / HSF                 (**)
```

VFL can be reduced to HFL be means of the two assumptions above:

```
        VFL = F2 * VR
            = F2 * (HR / AR)
            = (F2/F1) * HFL / AR        (***)
```

Inserting (**) and (***) into (*) we obtain:

```
        RR = DCF / ((F2/F1) * HFL**2 / AR)
           = (F1/F2) * AR * DCF * (HSF/DCF)**2
           = (F1/F2) * AR * HSF**2 / DCF
```

For fixed HSF, F1, F2 and AR, this is a hyperbola in our diagram. Drawing two such curves for minimum and maximum horizontal sync frequencies we have obtained the two remaining boundaries of the permitted region.

The straight lines crossing the capability region represent particular resolutions. This is based on (*) and the second assumption:

```
        RR = DCF / (HFL * VFL) = DCF / (F1 * HR * F2 * VR)
```

By drawing such lines for all resolutions one is interested in, one can immediately read off the possible relations between resolution, clock rate and refresh rate of which the monitor is capable. Note that these lines do not depend on monitor properties, but they do depend on the second assumption.

The **modeplot** tool provides you with an easy way to do this. Do **modeplot −?** to see its control options. A typical invocation looks like this:

```
        modeplot -t "Swan SW617" -b 85 -v 50 90 -h 31 58
```

The −b option specifies video bandwidth; −v and −h set horizontal and vertical sync frequency ranges.

When reading the output of **modeplot**, always bear in mind that it gives only an approximate description. For example, it disregards limitations on HFL resulting from a minimum required sync pulse width, and it can only be accurate as far as the assumptions are. It is therefore no substitute for a detailed calculation (involving some black magic) as presented in Putting it All Together. However, it should give you a better feeling for what is possible and which tradeoffs are involved.

17. Credits

The original ancestor of this document was by Chin Fang <fangchin@leland.stanford.edu>.

Eric S. Raymond <`esr@snark.thyrsus.com`>; reworked, reorganized, and massively rewrote Chin Fang's original in an attempt to understand it. In the process, he merged in most of a different how−to by Bob Crosson <crosson@cam.nist.gov>.

The material on interlaced modes is largely by David Kastrup <dak@pool.informatik.rwth−aachen.de>

Nicholas Bodley <nbodley@alumni.princeton.edu> corrected and clarified the section on how displays work.

Payne Freret <payne@freret.org> corrected some minor technical errors about monitor design.

Martin Lottermoser <Martin.Lottermoser@mch.sni.de> contributed the idea of using gnuplot to make mode diagrams and did the mathematical analysis behind **modeplot**. The distributed **modeplot** was redesigned and generalized by ESR from Martin's original gnuplot code for one case.