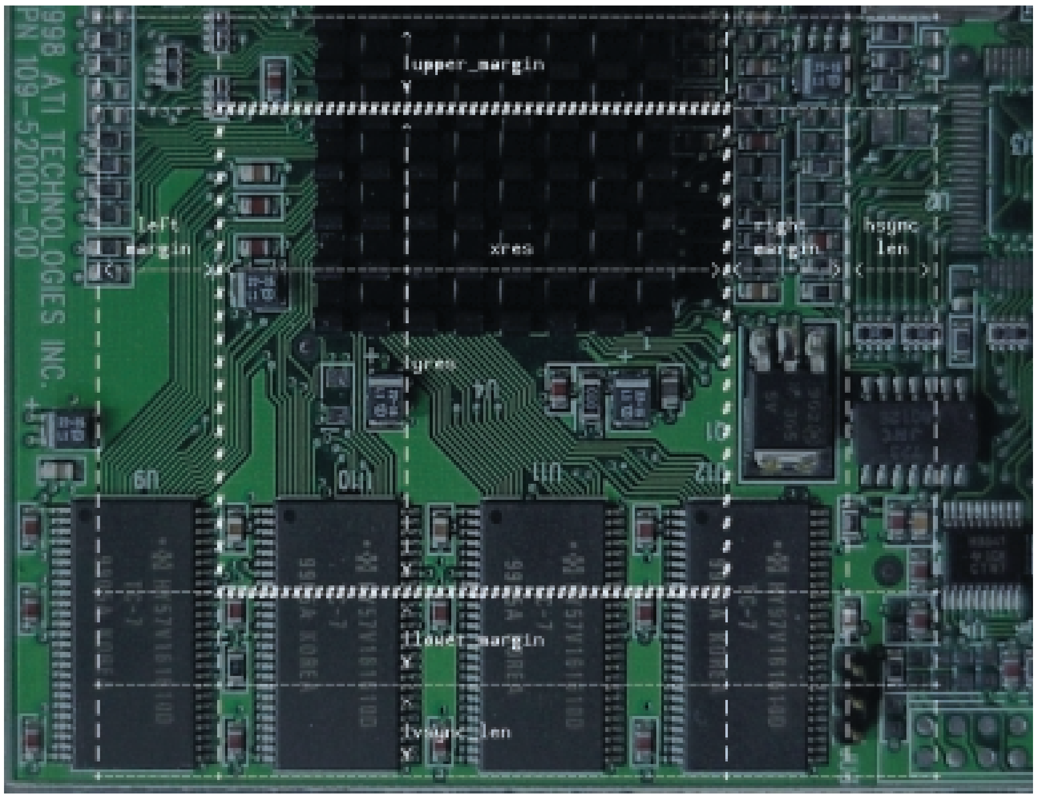
How to use framebuffer devices

SMALL IS BEAUTIFUL

DENNIS SCHÖN AND BERNHARD KUHN



Linux handhelds have become all the rage now that framebuffer graphics have become available as an alternative to the resource-hungry X Window System. In this article we look at how to use this display technology.

The X Window System has been around for more than a decade. It's a tried and tested graphics interface for Linux desktop computers and other Unix workstations. Resource usage isn't a great concern in that environment, but it has been the main obstacle to the development of Linux as an embedded OS for handheld computers (PDAs, organisers and so on). In these tiny computers every megabyte saved counts.

In the past few years several toolkits have been developed to help create applications with more efficient graphical user interfaces. Many of these are based on the "kernel framebuffer device" originally developed for Linux/M68K. The graphics subsystems of these platforms (Amiga, Atari, Macintosh) offer little in the way of hardware acceleration but share a very similar representation

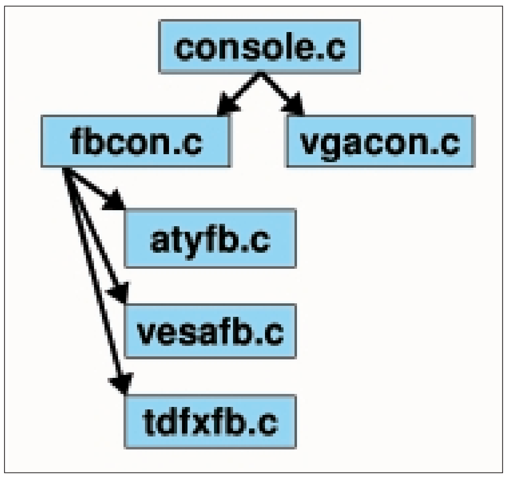


Fig. 1: One is not enough: the generic console driver allows several consoles per computer.

format. It seemed appropriate to produce a generic graphics driver.

From kernel 2.1.107 on, the framebuffer device for all platforms (x86, Alpha and so on) is integrated into the standard kernel. As in Figure 1, the generic driver for the text console runs as desired on either the ordinary VGA driver or on the *fbcon* for the underlying framebuffer devices (*fbdev*).

Configuration of the kernel

When using the framebuffer it is advisable to employ the latest stable kernel. The kernel 2.4.0-test* is currently experimental. As such, it should only be used if the kernel 2.2.x has no support for the desired graphics card. This situation should be an exception.

The framebuffer device for the VESA-BIOS is usually found in IBM-compatible PCs. Special drivers such as those for products from 3dfx, ATI or Matrox allow for higher resolutions and image repetition frequencies than the VESA driver.

After downloading, the kernel sources can be unpacked (while running with *root* privileges) under */usr/src* and then configured using *make menuconfig* or *make xconfig*.

You must activate the menu item "Prompt for development and/or incomplete code/drivers" under "Code maturity level options". If this isn't done, the option "Support for frame buffer devices" will not appear in the "Console Drivers" menu later on.

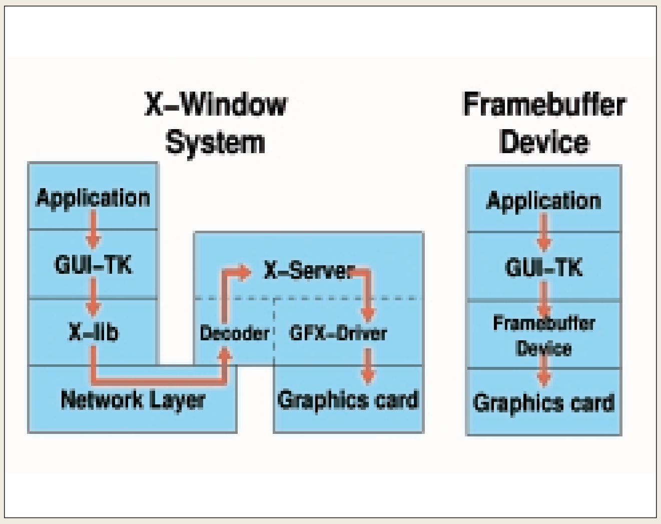
Fig. 2: Economical with resources: this static linked framebuffer version of a Tetris clone is just 11Kb.

Table 1: List of graphics chipsets supported.

Graphics cards chipset	Kernel	Kernel
	2.2.16	2.4.0-test1
VGA 16	X	X
VESA 2.0 compatible	X	X
Permedia2	X	X
Matrox	X	X
ATI Mach64	X	X
nVidia Riva		X
Cirrus Logic GD542x/543x		X
ATI Rage128		X
SiS630/540		X

Framebuffer vs. X Server

In the traditional X-Windows system, the application communicates with the X Server via the network layer. This then accesses the graphics hardware in the user-space. In the framebuffer device, the application accesses the graphics memory via the device files /dev/fb.*



X-Server: flexible but complex. It's simpler with a framebuffer device

Advantages of the framebuffer device subsystem

- The framebuffer is an powerful and efficient alternative to the X-Server;
- Many graphics cards are switched into graphics mode by firmware and therefore provide no hardware text mode. For such graphics cards, framebuffer-type drivers would nevertheless be necessary;
- The framebuffer allows very flexible use of the graphics card. It offers various resolutions, refresh rates, colour depths and type sizes, either with X-Windows (XF68_FBDev) or on the console. The console can be run at a resolution of 1600x1200 at 90 Hz (200x150 text lines) with powerful enough hardware. Even at 1024x768 / 75 Hz, economy is noticeably increased compared to standard VGA (normally used with 640x480 pixels at 60 Hz).
- With the framebuffer there is no limit for the console on the number of symbols in a line of text.
- There isn't a dedicated X-Server in existence for a framebuffer card. However, the device does have VESA-BIOS (established in the PC environment for about five years). With the aid of the framebuffer X-Server (or the corresponding XFree86-4.0 driver module) an X-Window system can be used.
- A kernel equipped with a framebuffer can display a "tux" or other logo (in multiprocessor machines) in the top part of the screen during the boot procedure, instead of a plain black screen.
- The framebuffer architecture is very simple. An experienced programmer can implement a new driver in a single afternoon.

The following options must be activated:

```
[*] VGA text console
[*] Video mode selection support
....
[*] Support for frame buffer devices
```

Extended options will follow. Apart from the driver for the graphics card (such as VESA 2.0, ATI Mach64, 3Dfx Banshee/Voodoo3) the following options can still be activated:

```
[*] Advanced low level driver options
....
 8 bpp packed pixels support
 16 bpp packed pixels support
 24 bpp packed pixels support
 32 bpp packed pixels support
....
[*] Select compiled-in fonts
....
[*] VGA 8x16 font
```

Using "x bpp packed pixel support", select the possible colour depths with which the framebuffer device can be operated. Under "Select compiled-in fonts" you can choose the fonts for the console. Activating more than one font makes it possible later on to select which one should be used for the console as a boot parameter.

Configuring the bootloader

Depending on the bootloader used, the files */etc/lilo.conf* for *lilo* or */boot/grub/menu.lst* for *grub* can be modified – once the kernel has been compiled and the module installed. This must be done in order to activate the framebuffer device at the next reboot. Here are two examples:

```
# LILO configuration file
boot = /dev/hda3
# Linux bootable partition config begins
image = /vmlinuz
append = "video=atyfb:1024x768-8@76,font:SU
N8x16"
root = /dev/hda3
label = Linux
read-only
# GRUB configuration file
# For booting Linux
title  GNU/Linux (experimental 2.4.0-test1 "
1024x768@76")
kernel (hd0,0)/vmlinuz video=atyfb:1024x768-
8@76,font:SUN8x16
root=/dev/sda1
```

Configuration of the X Server XF68_FBDev

One of the main applications for the framebuffer, apart from the graphical console is the X Server XF68_FBDev. It's name doesn't mean that this is a number cruncher – this X-server was originally developed for platforms with Motorola 68000 processors. XFree86 from version 4.0 on gives the driver module fbdev_drv.o of the generic X-server direct access to the graphics hardware. But now every current distribution includes the framebuffer server in pre-compiled form. After installing the software package for the respective distribution or compiling and installing the source package (only advisable in exceptional cases) a few small alterations have to be made to the configuration file for the X server (version 3.x) (/etc/X11/XF86Config or /etc/XF86Config). The "Screen" section might look like this:

```
Section "Screen"
  Driver     "FBDev"
  device     "Primary Card"
  Monitor    "Primary Monitor"
  SubSection "Display"
    Modes    "default"
  EndSubSection
EndSection
```

In XFree86 4.0 and later versions, only the driver module fbdev_drv.o has to be taken into account:

```
Section "device"
    Identifier  "3dfx"
    Driver      "fbdev"
EndSection
```

```
Section "Screen"
    Identifier  "Screen 1"
    device      "3dfx"
    ...
EndSection
```

With these configurations the X-server starts in the resolution at which the framebuffer has been most recently set (either by boot parameter or with the command line fbset). Resolutions can also be specified in the XF86Config file. This makes it possible to modify the resolution of the X server at run time. Unfortunately the timing values of the video mode have to be specified in a different format than in the normal XF86Config.

There are two ways to find out the correct values: Either convert the existing XF86Config values into the new framebuffer values. This can be performed using the formulas given in /usr/src/linux/Documentation/fb/framebuffer.txt Section 6. "Converting XFree86 timing values in frame buffer device timings", or by switching to the resolution desired for X-Windows using the command fbdev and the observing the timing values displayed when you run fbset -x.

```
# fbset -x

Mode "1024x768"
    # D: 84.991 MHz, H: 62.493 kHz, V: 75.933 Hz
    DotClock 84.992
    HTimings 1024 1032 1152 1360
    VTimings 768 784 787 823
    Flags    "-HSync" "-VSync"    # Warning: XFree86 doesn't s
upport
accel
EndMode
```

This mode specification can now be used easily by means of copy and paste in the "Monitor" section" of XF86Config. Apart from this the XF68FBDev server can be used like any other: using the "virtual" keyword, a virtual resolution can be set.

The meaning of the kernel parameter "video" should be self-explanatory. The ATI driver should be loaded at a resolution of 1024x768 pixels with a colour depth of 8 bits per pixel and a refresh rate of 76 Hz. The font compiled into the kernel is SUN 8x16.

Depending on the graphics card driver the kernel parameter may be omitted or look completely different. The VESA-BIOS and the Matrox driver, for example, have to be given a VESA mode number (such as *append = "video=matrox:vesa:440"*).

More details on these drivers can be found in the directory */usr/src/linux/Documentation/fb/*. This is also where special options for the drivers are explained (for example *ypan* and *ywrap* to increase scroll rate).

Until now almost every framebuffer driver has used its own video modes. A good way of avoiding this chaotic waste of resources can be found in the recent kernel series 2.4. Here, the drivers for Amiga (ami), ATI Mach64 (atyfb), ATI Rage128 (aty128fb) and Voodoo3 (tdfx) all share a single video mode database (modedb). The following self-explanatory format specifies a valid video mode:

```
"video=<driver>:<xres>x<yres>[-<bpp>][@<re➘
fresh>]"
```

Other configuration

If no device files have yet been created in the */dev* directory for the framebuffer devices this must be done by hand:

```
for i in 0 1 2 3 4 5 6 7; do
mknod /dev/fb$i c 29 $[$i * 32]
done
```

A restart a message, similar to the one below, should appear during the boot procedure:

```
atyfb: 3D RAGE PRO (BGA, AGP) [0x4742 rev 0x➘
7c] 8M SGRAM, 14.31818 MHz
XTAL,
230 MHz PLL, 100 MHz MCLK
Console: switching to colour frame buffer de➘
vice 128x48
fb0: ATY Mach64 frame buffer device on PCI
```

The graphics card immediately switches to graphics mode. The tux logo should appear during the operating system boot-up.

Troubleshooting

If problems arise, the first thing to do is to consult the documentation under */usr/src/linux/Documentation/fb/*. The texts on the various drivers used (aty128fb, tdfx, ...) are particularly good sources of information.

Another good place to start if there are problems is the Linux framebuffer project homepage (listed below). Here you'll find a link to

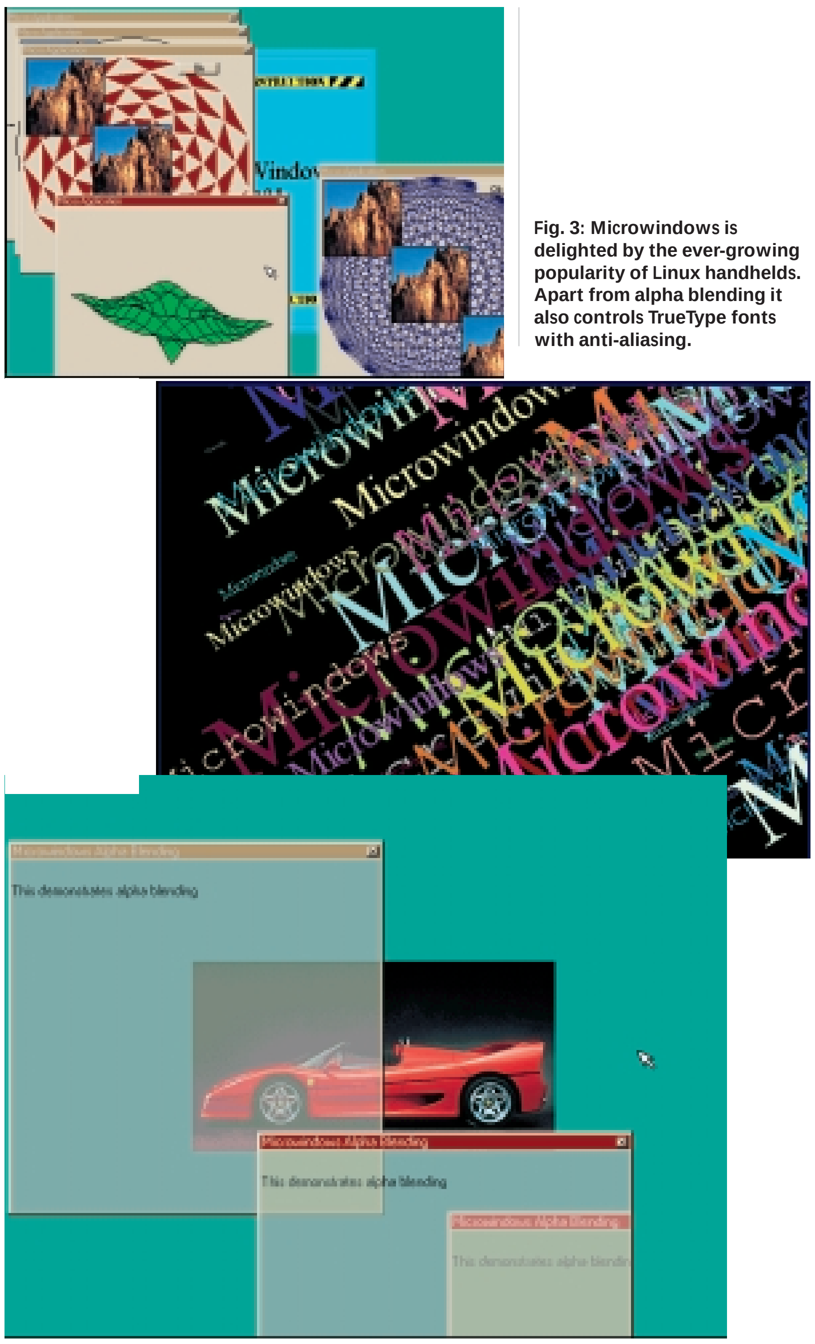

Microwindows Alpha Blending
This demonstrates alpha blending
Microwindows Alpha Blending
This demonstrates alpha blending
Microwindows Alpha Blending
This demonstrates alpha blendin
Microwindows

Fig. 3: Microwindows is delighted by the ever-growing popularity of Linux handhelds. Apart from alpha blending it also controls TrueType fonts with anti-aliasing.

the Linux framebuffer HOWTO (which unfortunately has not been updated for a year). There is also a mailing list, which can be found at *linux-fbdev@vu.union.edu*. There can be many different problems. Typical approaches to solutions are listed below:

- If the framebuffer does not start (Error messages in the boot procedure!) the wrong driver has probably been compiled or the card is not yet supported. It may be that the ESA-BIOS-framebuffer or the experimental kernel (2.4.0-text*) should be tried.
- If the framebuffer does start, but does not switch to the specified mode, all the options which are not vital (*ypan*, *ywrap*, *font*) should be deactivated and then a lower-resolution mode (for example 800x600) tried.

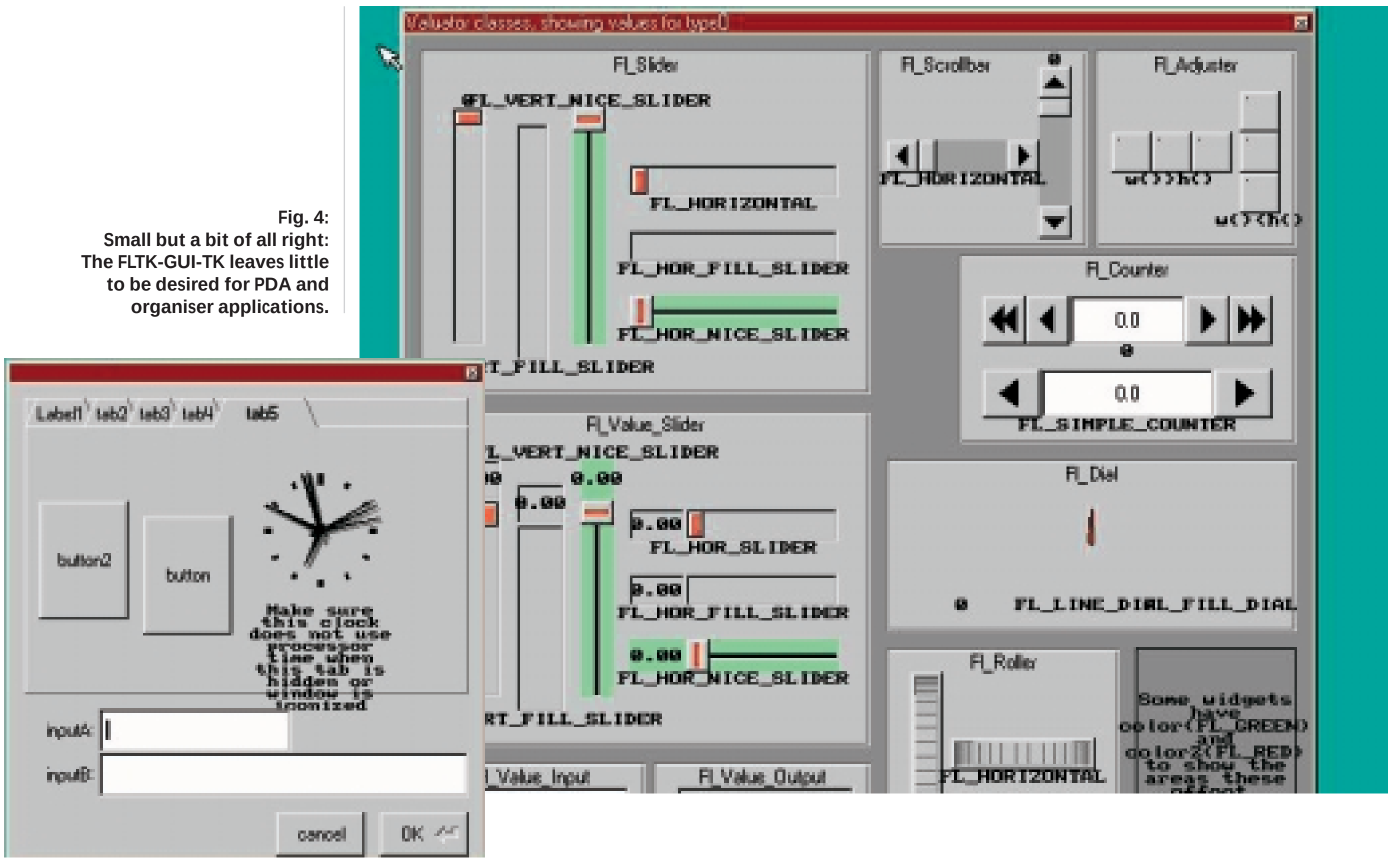


Fig. 4: Small but a bit of all right: The FLTK-GUI-TK leaves little to be desired for PDA and organiser applications.

Tools and applications

The *fbset* program by Geert Uytterhoeven makes it possible to alter the resolution of the framebuffer during operation. It manages its own database of video modes (in */etc/fb.modes*). This can obviously be extended as desired. A whole range of sample modes accompany the sources of the program, for example, for ATI graphics cards, Atari Falcon or video modes to control NTSC or PAL TV monitors. Another useful tool is *fbview*. Image files can be displayed on the console with this program. It now runs with almost all framebuffer drivers.

If you don't like seeing the penguin when booting, take a look at *fblogo*. Images in Tiff format can be converted into your own boot logo header files for the kernel with this program.

Tomas Berndtsson has begun a very interesting project with *Zen*. This is a web browser with various interfaces (plain text, oFBis, ncurses, GTK). The interface of interest here is the oFBis interface. This consists of a library of graphics routines for the framebuffer device. With the aid of this library, Zen is able to display images on the console. This project is still alpha software, but will be worth watching out for.

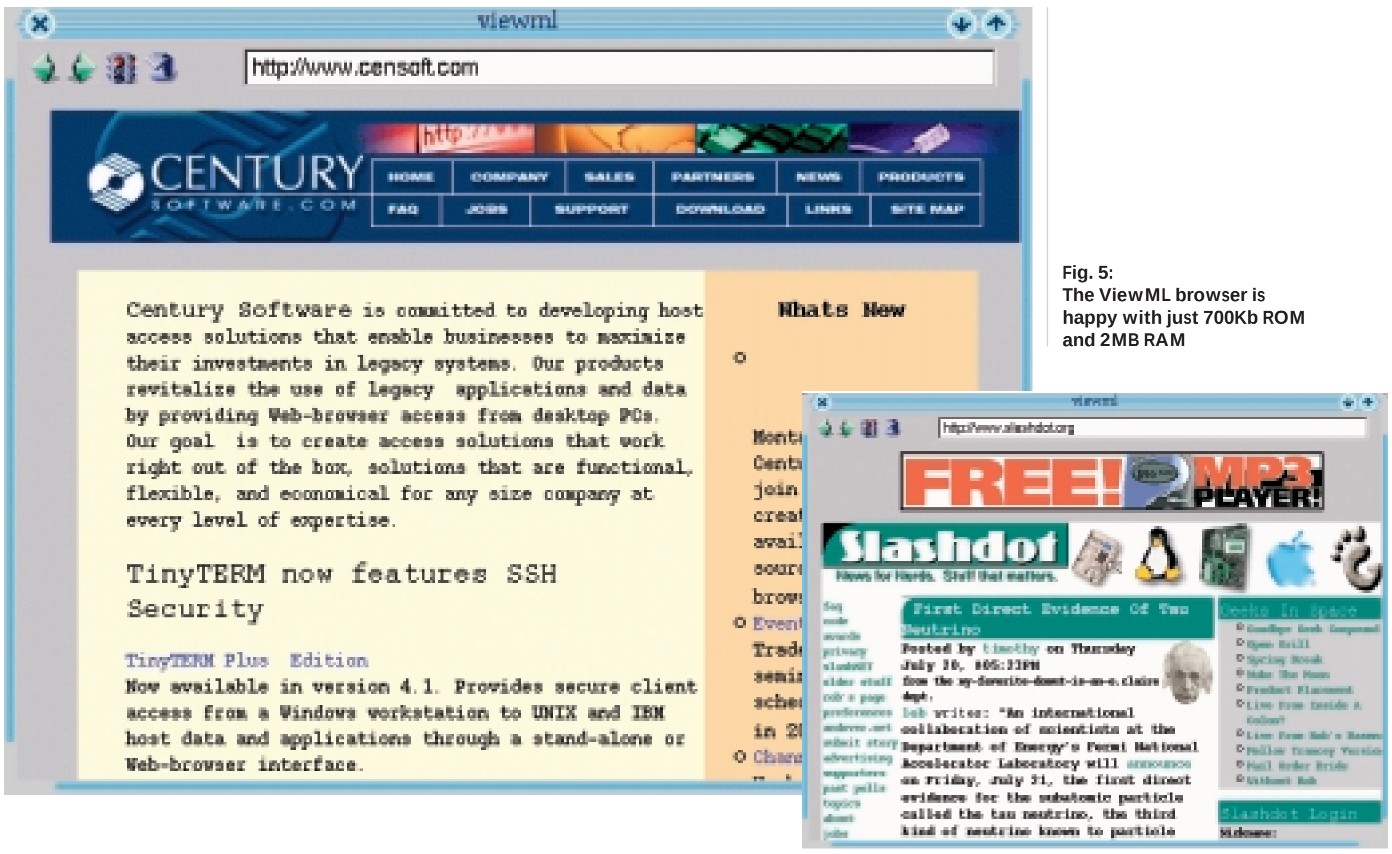
Microwindows is a portable and extremely efficient windowing system (see Figure 3) which runs on a whole range of hardware and software environments. It was developed for the handheld and pocket PC market (LinuxCE). Hence it needs, on a 16 Bit system (ELKS) for mouse, keyboard and screen driver, less than 64Kb of memory.

But Microwindows also runs on modern PC systems under Linux with a bit of help from the framebuffer devices, the SVGAlib library or under X-Windows in a separate window. The latter makes application development a great deal easier. With Nano-X Microwindows has an API very similar to that of the X-Window system. This means the resource-sparing yet powerful Fast Lightning Tool Kit (FLTK, see Figure 4) can be used and applications such as the web browser ViewML (see Figure 5) developed. The latter uses the highly refined HTML engine of *KFM* (the KDE file manager). Because of its extremely low resource requirements it is of particular interest for handhelds.

Last, but not least, "Qt Embedded" deserves a mention: The object-oriented GUI toolkit, already well known as the basis of the KDE desktop (in version 2.2.0) is also available in a variant which allows direct access to the graphics hardware. All class definitions are fully compatible with the X-Window version. This is why Qt/KDE applications can be used without any porting costs even without an X server.

Table 2: Framebuffer-based windowing systems and GUI toolkits

Microwindows	WindowsCE-look-alike API	www.microwindows.org
Nano-X	X-Window-look-alike network -transparent API. Based on Microwindows	-"-
FLTK	object-oriented GUI-TK for Nano-X	-"-
Tiny-X	Trimmed down X-Server with limited functionality	www.xfree86.org
Qt Embedded	Framebuffer variants of Qt	www.troll.no


viewml

http://www.censoft.com

CENTURY SOFTWARE.COM

HOME | COMPANY | SALES | PARTNERS | NEWS | PRODUCTS
FAQ | JOBS | SUPPORT | DOWNLOAD | LINKS | SITE MAP

Century Software is committed to developing host access solutions that enable businesses to maximize their investments in legacy systems. Our products revitalize the use of legacy applications and data by providing Web-browser access from desktop PCs. Our goal is to create access solutions that work right out of the box, solutions that are functional, flexible, and economical for any size company at every level of expertise.

TinyTERM now features SSH Security

TinyTERM Plus Edition
Now available in version 4.1. Provides secure client access from a Windows workstation to UNIX and IBM host data and applications through a stand-alone or Web-browser interface.

Whats New

viewml

http://www.slashdot.org

FREE! MP3 PLAYER!

Slashdot
News for Nerds. Stuff that matters.

First Direct Evidence Of Tau Neutrino
Posted by timothy on Thursday July 20, @05:33PM
from the my-favorite-doesn't-is-an-e.claire dept.
Lab writes: "An international collaboration of scientists at the Department of Energy's Fermi National Accelerator Laboratory will announce on Friday, July 21, the first direct evidence for the subatomic particle called the tau neutrino, the third kind of neutrino known to particle

Geeks In Space

Slashdot Login
Nickname:

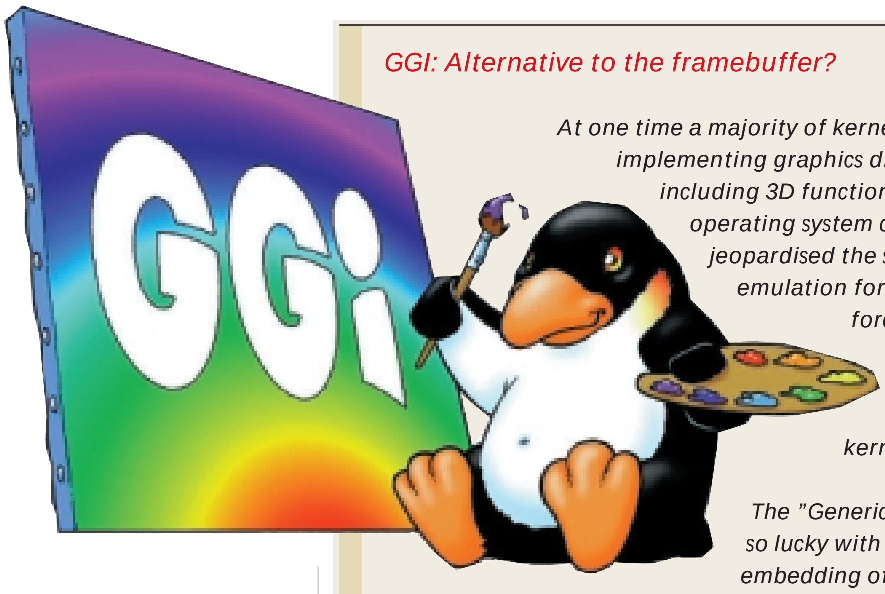
Fig. 5:
The ViewML browser is happy with just 700Kb ROM and 2MB RAM

Framebuffer device programming with C and C++

The framebuffer device of the first graphics card in the computer can be addressed via the device file /dev/fb0. For compatibility reasons there are often links from /dev/fb0current or /dev/fb to this file. With a command such as cat /dev/fb0 > /tmp/fbdump the complete memory (yes, all of it!) of the graphics card can be read out. The visible area of the "graphical text console" in that case always begins at the file position 0. Modification of the screen contents using seek, read and write is tedious and relatively slow in execution. Therefore, the framebuffer device can be embedded with the aid of mmap in the data memory segment of the application (see listing, line 25). Note, by the way, that other graphics cards can be controlled using the device files /dev/fb1 to /dev/fb7.

The following sample program colours the console blue (at eight bit colour depth the defaults for the colours 0 to 15 correspond to the usual 4-bit VGA palette). The program can be compiled using g++ -o fbdemo fbdemo.cpp.

```
01 // Read in header files
02 #include <sys/types.h>
03 #include <sys/stat.h>
04 #include <fcntl.h>
05 #include <linux/fb.h>
06 #include <unistd.h>
07 #include <sys/mman.h>
08 #include <sys/ioctl.h>
09
10 main() {
11
12  // open framebuffer device and read out info
13   int fd = open("/dev/fb0", O_RDWR);
14   struct fb_var_screeninfo screeninfo;
15   ioctl(fd, FBIOGET_VSCREENINFO, &screeni↵
nfo);
16
17   // continue only if 8 bit colour depth
18   if (screeninfo.bits_per_pixel == 8) {
19
20     // determine size
21     int width = screeninfo.xres;
22     int height = screeninfo.yres;
23
24     // embed framebuffer into memory
25     unsigned char *data = (unsigned char*)
26       mmap(0, width * height ,
27       PROT_READ|PROT_WRITE, MAP_SHARED, f↵
d, 0);
28
29     // process screen content line by line
30     for(int row=0; row<height; row++) {
31       for(int column=0; column<width; colu↵
mn++) {
32         data[column + row * width] = 0x01;
33       };
34     };
35
36     // mask framebuffer out of memory
37     munmap(data, width * height);
38
39   };
40 };
```



GGI: Alternative to the framebuffer?

At one time a majority of kernel developers were against the idea of implementing graphics drivers at kernel level. The driver software including 3D functions rapidly exceeded the complexity of the operating system core and as it increased in size it jeopardised the stability of the system. The necessary text emulation for the Atari, Amiga and friends, however, forced at least a rudimentary mechanism to be included, which was then steadily built up. The framebuffer device has thus achieved integration in the standard kernel through the back door.

The "Generic Graphics Interface" (GGI for short) was not so lucky with its "Kernel Graphics Interface" (KGI). The embedding of complex 2D and 3D interfaces, including multi-input and multi-head support was going too far in the view of the hard-line core developers. What was once considered "Not a Bad Idea" is now eking out a shadowy existence, although there is a big "fan club" for it and therefore also a whole range of impressive demos and applications (GGI X server, games).

Table 3: ioctl functions of the framebuffer device

FBIOGET_VSCREENINFO	determine variable dimensions of the framebuffer
FBIOPUT_VSCREENINFO	define variable dimensions of the framebuffer
FBIOGET_FSCREENINFO	determine fixed dimensions of the framebuffer
FBIOGETCMAP	determine colour palette
FBIOPUTCMAP	define colour palette
FBIOPAN_DISPLAY	move physical display within the virtual
FBIOGET_CON2FBMAP	salvage content of console
FBIOPUT_CON2FBMAP	restore content of console
FBIOBLANK	delete content of console
FBIOGET_VBLANK	determine current raster beam position
FBIO_ALLOC	allocate graphics memory for own purposes (e.g. dual buffer)
FBIO_FREEF	free graphics memory

As an especially neat and useful gimmick, Qt Embedded has a widget for inputting and recognition of handwriting (see our test in the previous issue). But for commercial use, one-off developer licence fees have to be paid for each workstation. These are roughly within the same price range as Windows variants. There are also costs for run-time licences ($2 per device,assuming high numbers of items).

Framebuffer goes into action

In comparison with the tried and true graphics version aided by a hardware-accelerated X server, applications based on framebuffer devices are noticeably slower at high desktop resolutions such as 1280x1024 or greater. The displays of pocket PCs and handhelds, however have considerably fewer pixels (320x240 or less). Windows are smaller and are represented by less data, so the lower performance is scarcely noticed.

Memory consumption is another matter. A few megabytes can be saved by doing without an X-server. Thanks to the framebuffer device and its associated GUI toolkits it seems likely that more mobile Linux PDAs will be coming on to the market in the near future. ■

Info

Framebuffer Home page: *http://www.linux-fbdev.org*
Framebuffer Howto: *http://www.linuxdoc.org/HOWTO/framebuffer-HOWTO.html*
Framebuffer Mailing List: *http://www.linux-fbdev.org/mlist.html*
XFree86 Homepage: *http://www.xfree86.org*
Various Framebuffer Tools: *home.tvd.be/cr26864/Linux/fbdev/*
fbview Home page: *http://www.nocrew.org/software/fbview/*
Zen Web-browser: *http://www.nocrew.org/software/zen/*
User-Space Graphics functions for the framebuffer device: *http://osis.nocrew.org/ofbis/*
fblogo Homepage: *http://home.sol.no/~dvedoy/*
Microwindows / Nano-X Homepage: *http://www.microwindows.org*
ViewML Website: *www.viewml.com*
Homepage of the GGI Project: *http://www.ggi-project.org/*
■