

EVERYTHING YOU EVER WANTED TO KNOW ABOUT **X**

Work started on the X Window System two decades ago. Can it still cut it at the heart of a modern Linux desktop? **Richard Drummond** gives X some scrutiny...

cover feature



When people discuss the suitability of Linux as a desktop platform, they generally focus their attentions on the features and capabilities of one of the major Open Source desktop projects such as KDE or GNOME. That's not surprising; it is these projects and the applications that are built upon them that provide the direct user experience for desktop Linux. Many desktop users don't know or don't care that layered beneath the KDE and GNOME application frameworks on every Linux installation is a vital piece of infrastructure that creates and manages the graphical, windowed environment in which these desktops run: the X Window System (or X for short). It's actually a testament to X's reliability and ubiquity that it can be safely ignored by the end user. X is middleware, plumbing. And who doesn't take their plumbing for granted, as long as it works? Events of earlier this year, however, have thrust X – and especially its most popular implementation, produced by the open-source XFree86 project – unwillingly into the limelight.

In late February, the release of XFree86 4.3.0 was announced, and all seemed well. This was the first XFree86 release in nearly six months and brought some long-awaited new features (see the box *What's new in XFree86 4.3*). Barely a month after, though, the XFree86 project's Core Team announced that one of their most prolific developers, Keith Packard, the man behind many of the advancements in the XFree86 4.x series, had been dismissed due to alleged misconduct. He had, it was claimed, been seeking support from the X community to fork the XFree86 project; that is, set up an independent project, led by himself, that would develop a separate X release based on XFree86. Shortly afterward Packard responded with a message to

a public XFree86 forum stating that he hadn't intended to undermine the XFree86 project but rather he was seeking support for a change in the way that XFree86 was governed. This post, *A Call for Open Governance of X Development*, cited XFree86 failings such as slow release schedules and lack of co-operation with other open-source projects. Packard said, 'Persistent problems in XFree86 development have become widely recognised within the X community... The key issue is that XFree86 is not a community-governed project... Decisions appear to be arbitrary and are not seen to reflect the will of the community.'

The controversy

The debate that has ensued on the mailing lists and forums that constitute the online Open Source community has often created more heat than light. The raised emotions are understandable: X is such a crucial piece of technology for Linux and Unix, and the XFree86 developers have expended considerable amounts of time and effort on creating and maintaining their implementation. But, amidst all the flaming and trolling, some serious questions have been raised concerning the way X is developed, the role the XFree86 project should play, and the technical problems which need to be solved for X to meet the demands of the contemporary desktop. A few have even asked whether X should be ditched entirely and replaced with something that's more modern.

In the light of all this controversy, this article will examine the current state of the X Windows System and, in particular, its most popular implementation: XFree86. Just what is X and what advantages does X offer to Linux's assault on the desktops of the world? Is X even necessary? Like any piece of engineering, X is not without its problems, so we'll look at



“A few have even asked whether X should be ditched entirely and replaced with something that's more modern.”

XWindowSystem



the technical challenges facing X and the solutions that have been proposed. We'll also discuss some of the management problems surrounding the XFree86 project. To understand all the issues at stake, however, we need to start at the beginning.

What is X?

Simply put, the X Window System is the *de facto* standard for providing a graphical environment on Unix and Unix-like platforms, including Linux. (How it became established as such is a story we'll look at shortly and is due largely to a couple of fortuitous design decisions and the availability of its source code, rather than any grand master plan. In this, the popularity of X mirrors the rise of Unix itself.)

Although X is superficially similar to

rival graphical platforms such as Windows and Mac OS, the environment that X creates differs in two important ways.

Firstly, X was designed from the ground up to be network transparent. That is, X allows you to run a program on one computer and interact with it via its graphical interface displayed on another. This magic is due to the client/server nature of X (see box *X architecture*, right). In X terminology, the display and input devices, such as the mouse and keyboard, are managed by a piece of software called an X server. The programs that you interact with via this X server – known as X clients – may be running on the same computer or on any other host on the network. An X client communicates with the X server via a mechanism called the X protocol, and it is this that lets the client remotely create its interface on the user's display and receive user input.

The other great difference with X is that X really does just provide the core infrastructure for a windowing system. X doesn't specify how applications should look and feel: it's the job of an X client to create the user interface elements such as menus and buttons, either directly via the primitives that the X protocol provides or via an X toolkit. X doesn't even provide a window management policy; that's up to a special X client called a window manager. All this flexibility can be a double-edged sword, however. One the one hand, this approach of 'mechanism not policy' – as the mantra goes – has been key to the endurance of X; on the other, the lack of a consistent look and feel is just one of the plethora of reasons why Linux is having such a hard time on the desktop now.

Before we can understand these differences and the pros and cons they bring to Linux on the desktop, we need to look at a little history.

The genesis of X

The X Window System began life in 1984 at Project Athena, a research project in distributed systems at MIT whose aim was to make computing resources easily accessible to students. The name X reflects that it drew inspiration from an earlier project called 'W' (presumably short

for 'window' – sorry, all you recursive acronym fans!) A platform-independent windowing system was required because MIT couldn't afford to buy all the workstations needed for the project themselves, nor was any one vendor willing to supply sufficient workstations. Thus from day one, X was designed to provide network-transparent windowing across a variety of heterogeneous hardware.

Initial X development was a joint effort by MIT and Digital Equipment Corporation (DEC at the time supplied Unix for their VAX and VAXstation hardware). Progress was rapid at first, with Version 8 being the first to support colour display hardware and Version 10 the first to be widely deployed. Development culminated with release of X version 11 (or X11) in 1987 which offered vastly better performance, an improved rendering model and support for 32-bit deep colour displays. Note that the X version number mentioned here refers to the version of the X protocol implemented, with different version numbers being incompatible with each other. For instance, an X10 client won't work with an X11 server and *vice versa*. However, the core X protocol has remained largely unchanged since 1987. Only occasional minor revisions have been made in the intervening period, the last significant one being X Version 11 Release 6 or X11R6 in 1994. X11 clients written sixteen years ago will still work on modern X servers, an impressive achievement in backwards compatibility.

DEC began shipping X with its flavour of Unix, Ultrix, and thanks to the portability of X11 and the availability of the source code, other Unix vendors followed suit. The rest, as they say, is history. The X Windows Systems was an open-source project long before that term was coined. Like the UC Berkeley version of Unix – BSD Unix – the X source code was made available under a very liberal license. And, as with the BSD license, the MIT X License allows the unrestricted use, modification and distribution of the X source code and the even the sale of commercial products based on it. However, to ensure that X development wouldn't splinter into incompatible factions – as had happened with Unix – MIT set

What's new in XFree86 4.3.0?

Significant changes and updates

XFree86 4.3.0 is arguably is the most significant XFree86 release since 4.0.3, which first introduced the *Render* extension. Besides the usual round of bug-fixes and driver improvements – in particular 4.3.0 includes updated Radeon drivers supporting the newer 9000 and 9100 models – it also includes some important technology updates.

Firstly, XFree86 4.3.0 includes the new *Xft2* and *fontconfig* libraries to improve font rendering and vastly ease font configuration, both for developers and users. *Xft2* is the X-based wrapper for the TrueType font rasterizer and is part of the new client-side font-handling system for X which replaces the old and inefficient font handling of the core X protocol. It was *Xft* that

introduced anti-aliased fonts, based on the alpha-blending capabilities of the *Render* extension. New in *Xft2* is support for X servers without the *Render* extension, so that it's no longer necessary to fall back on core font handling when *Render* isn't available.

Also new is a partial implementation of the new *X RandR* extension, which allows an X display to be dynamically re-sized, rotated and reflected without the need to restart the X server.

Finally, support for full-colour, alpha-blended, animated cursors (mouse pointers) is included for the first time (previously, only monochrome pointers could be used), and this is backed up by a new client-side library, *Xcursor*, for handling cursor configuration.



Much to the delight of all those that love to theme, XFree86 4.3.0 now supports full-colour alpha-blended, animated mouse pointers.

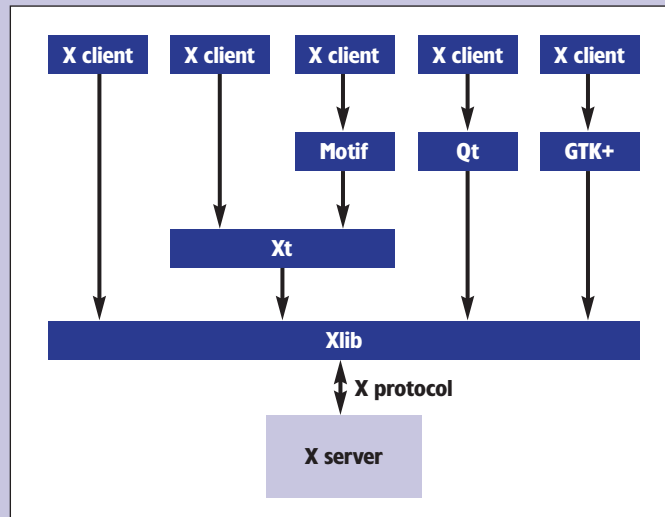
X architecture

Client/Server, the X protocol and X toolkits

The X Window System is a network-transparent windowing system based on a client-server architecture. The X server is the program which manages the display hardware, input devices and so on, while programs which a user interacts with via the X server are known as X clients. This terminology may seem confusing at first, but it makes sense if you look at things from the point of view of the X client. The client uses services provided by the X server to display a graphical interface and get input from the user.

An X client communicates with the server via a stream-based channel known as an X wire using a protocol called the X wire protocol or just X protocol. This is a device-independent protocol and enables the client to send commands to the server to open and close windows, draw geometric shapes, render text, receive inputs and so on without caring about the specific hardware upon which these requests take effect.

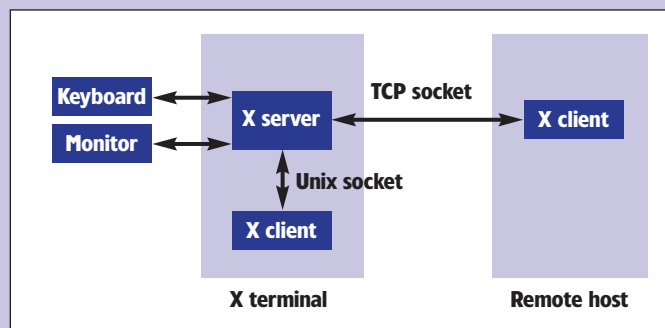
The X protocol can be transported over any suitable medium, and this is what gives X its network-transparency. Usually, any X clients you are using will be running on the same machine as the X server – on the computer or X terminal that you are sitting in front of – and some local method of IPC (Inter-Process Communication) will be employed to construct the X wire. For example, on Unix hosts this will commonly be the Unix socket. But, since the X protocol will work equally well (albeit more slowly) over a network link such as a TCP socket, you can run an X client on any machine on your local network (or even on the



Toolkits – Xt is better than Xlib for building a GUI as it's higher-level.

Internet, security issues notwithstanding, of course!) and still operate it via your X server. Moreover, since all communication takes place via the device-independent X protocol, the X client doesn't need to be running on the same type of host as the X server. It

can have a completely different CPU architecture, and the remote host upon which the client is running doesn't even need its own any display hardware. This network-transparency is a unique and compelling feature of X and a major reason for its popularity.



X architecture – device-independent protocol.

X toolkits

The X Windows System provides a shared library of functions called *Xlib* which can be used by X clients to communicate with an X server without actually having to understand the X protocol. The problem with *Xlib* is that it is very low-level. The functions it offers directly correspond to the requests supported by the X protocol, and thus it is an awful lot of hard work to create a user interface just using *Xlib* on its own.

The other problem is that there is no consistency of look and feel between applications that employ *Xlib* directly, since they have to implement the entire interface themselves. To overcome these difficulties, various GUI toolkits have been created over the years to ensure a measure of continuity in the user experience.

To make it easier to develop GUI toolkits, X includes an additional library called the *X Toolkit Intrinsics* or *Xt*. This provides an object-based infrastructure for implementing user interface elements (known as widgets) in C and C++ and contains many utility functions to ease development. Once popular toolkits based on *Xt* include *Xaw* (the *Athena Widget* library) and *Motif*.

Motif was once the industry standard for GUI programming on X, but as a proprietary system it has fallen out of favour in recent years to Open Source and more modern-looking toolkits such as *Qt* and *GTK+*. (Incidentally, as you can see in the diagram above left, *Qt* and *GTK+* themselves don't go through *Xt* but use *Xlib* directly to construct their widgets.)

up the X Consortium as a standards body to maintain the X11 protocol and to provide a reference implementation. Eventually, as interest in Project Athena at MIT waned, in 1988 the X Consortium was spun off as a separate not-for-profit company, which was finally merged under the wing of the Open Group around 1996 and renamed X.org.

Getting X for free

Unsurprisingly, as far as Linux is concerned, our story really begins with the rise in popularity of the PC. In the early 1990s, the increasing performance-to-price ratio of the average PC began to make it a

platform worth reckoning with. In terms of performance, it was fast catching up with the commodity hardware peddled by the proprietary Unix vendors. This, coupled with the internecine squabbling of the Unix vendors, enabled Windows and especially Windows NT to gain a foothold in the workstation and server marketplace – much to the detriment of Unix and the X Window System. (This was a trend that would continue throughout the 90s until the free Unices and Linux began to reclaim the ground lost to Microsoft.)

The first popular port of X Windows for PC versions of Unix was done in 1990 by Thomas Roell and

“MIT couldn't afford all the workstations, thus from day one X was designed to have network-transparent windowing across a variety of heterogeneous hardware.”

was released under the name X386. These changes were fed back to the X Consortium and became part of the reference distribution with X11R5. Unfortunately, or perhaps fortunately, this release was rather buggy. It prompted several developers – independently at first – to begin fixing the various problems with X386.





Initially, this group of four – David Wexelblatt, David Dawes, Glenn Lai and Jim Tsillas – released their changes under the name X386 1.2e, where the ‘e’ stood for ‘enhanced’, but in April 1992 they set up their own project, under the name XFree86 (a pun on X386 as Roell had taken X386 commercial). Their aim was to produce the best X implementation – commercial or otherwise – for the x86 architecture. Which they did, and then some.

XFree86 was ported to Linux in 1992, and the growth of the XFree86 project owes much to the growth of Linux. But XFree86 soon spread to other platforms, not just Unix, including OS/2 and Windows (the latter thanks to the *Cygwin* library, which implements a Unix-like API for windows) and to other architectures, not just the x86. Many consider today that XFree86 really is the best implementation of X for any platform; it's certainly the most popular.

When commercial Unix lost its direction in the second half of the 1990s, so interest in maintaining the X standard declined. As the X Consortium and then X.org became more and more ineffectual, it fell upon the reluctant shoulders of the XFree86 project to drive forward X

such claims be heeded?

Throughout its life, X's network transparency has been its killer feature. While this may not appeal to the home user running Linux on a single machine, that is no reason to drop such an important advantage. The network-transparency of X is crucial in the commercial sector, where it is put to good use in thin-client computing and for remote administration. Consider this: X's main competitors, Windows and Mac OS, have both recently had a remote access capability tacked on. X does it much more fluidly and naturally than either proprietary competitor.

Moreover, claims that any poor performance of X is due to its networked nature are generally unfounded. When you run client and server on the same Unix machine, the X protocol is transmitted via a Unix socket, which, on modern implementations such as Linux, is a very efficient means of Inter-Process Communication. In addition, when running X clients local to the server, there are techniques for by-passing the X-wire entirely for certain tasks and such techniques often find application in games programming. Complaints that X uses too much memory are often mistaken, too. Novices fail to recognise that the memory usage figures they quote for X also includes the on-board memory of their graphics card – which is routinely 32MB or more for modern cards.

Criticisms that X is old-fashioned are perhaps more substantive. But the fact that the core X technology has remained constant for sixteen years can also be seen as an advantage. X is a well-proven, reliable and widely-deployed system. The longevity of X and the fact that it runs on such a variety of hardware and operating systems means that there is a wealth of software out there written for the X platform. This would be foolish to carelessly throw away. As evidence, bear in mind that when any new graphics architecture is proposed for Linux – such as DirectFB or GGI, for instance – an X server built on that architecture is often quick to follow. Apple now even distributes an X server (based on XFree86) for Mac OS X. They would be unlikely to do that if there weren't demand!

The secret to X's longevity is its

flexibility. As we have seen, the approach of ‘mechanism not policy’ means that X doesn't present a barrier to creation of modern graphical toolkits, for instance. The other significant choice the original designers of the X11 protocol made was to allow for the protocol to be easily extended. So-called X extensions can be bolted on to an X server to implement optional new features and introduce new behaviour – without changing the core protocol. This mechanism is crucial to X's ability to adapt to changing requirements (see box *X Extensions* on the right).

The need for change

Back in 1987 when X11 was designed, true-colour display hardware was rare and scalable font technology – although used for printing (for example, in the PostScript page description language) – was not appropriate for on-screen fonts. These facts are reflected in the design of the core X protocol. While it's suitable for monochrome and low-colour displays, it doesn't match the abilities of today's accelerated, true-colour graphics hardware well or the demands of modern applications. The X protocol has poor support for handling images, and the way it handles fonts is inefficient especially for fonts with large numbers of glyphs (such as non-Latin scripts and Unicode fonts).

Part of the problem is X's old-fashioned rendering model, that is, the functions it provides for drawing (known as geometric primitives) and the way that it combines what you draw with what's already on screen. X's rendering model is inspired by PostScript's model – state of the art for the day – and provides a well-packed toolbox of primitives (but alas not PostScript's support for drawing splines or arbitrary curved paths). The result of drawing with each primitive is combined on-screen with a series of bit-wise logical operations, known as a raster operation or ROP. While this works well for monochrome and palette-mapped displays, it's cumbersome with true-colour displays and doesn't lend itself to techniques such as anti-aliasing (anti-aliasing is the process of blurring the edges between two shapes drawn on

“XFree86 was ported to Linux in 1992, and the growth of the XFree86 project owes much to the growth of Linux.”

development. XFree86 became in the eyes of many the real reference implementation of X and the testing ground for advancements.

The X advantage

As we have seen, X is the *de facto* graphical standard for Unix, but is it still relevant to the Linux desktop? Well, X does have its share of detractors. X is, they say, over-engineered, over complex and uses too many system resources. Those whose concern is only in Linux on the desktop naively demand that X's network-transparency be dropped; others say that X has had its day and should be replaced entirely. Should

screen by averaging out the colours along their boundary and thus making them look less jagged).

An *ad hoc* solution to such problems is for an application to perform any advanced rendering required at the client side and transport the result over the X-wire to the server as an image. Such an approach is inefficient, not only because it fails to take advantage of hardware acceleration but also, due to a lack of standardisation, it leads to the endless re-invention of the wheel.

The Render Extension

It was clear that what was needed was an extension to the X protocol to modernise X's rendering model, but the X community was forced to wait a long time for this. The demand was eventually met by Keith Packard's *X Rendering Extension* (*Render* extension for short), which he first proposed in 2000. The *Render* extension provides a compositing operator which allows any rendering to be blended with the screen using an optional level of translucency (known as an alpha channel); support for alpha-blending leads naturally to the implementation of anti-aliasing. In addition, the *Render* extension provides a much more basic set of geometric primitives, which map more closely to what can be accelerated by hardware in modern graphics chipsets. The *Render* protocol defines primitives for drawing only trapezoids and triangles, but this is not anywhere near as limiting as it might sound, since any other primitive can be decomposed into a sequence of trapezoids and triangles.

Not content with simply creating a new rendering model for X, Packard also revolutionised font-handling in X. Along with *Render*, he created a library for the client-side rasterization of scalable fonts, employing the popular TrueType font engine. This library, *Xft*, provides an X client much more power and flexibility in the handling of fonts, and uses the *Render* protocol's ability to upload rasterized font glyphs to the X server for rendering with optional anti-aliasing. At last a solution to how to support anti-aliased fonts in X!

The *Render* extension first appeared in XFree86 4.0.3, but it wasn't until the recent release of 4.3.0

that the new font system came of age. With *Xft*, when X clients were running on servers without the *Render* extension they had to fall back on the old core X protocol font-handling. *Xft2*, as included in XFree86 4.3.0, supports the new font model even on *Render*-less servers. Moreover, 4.3.0 also includes Packard's fontconfig library, which is a system-wide solution for the configuration and location of fonts – not just for X.

With the release of XFree86 4.3.0 most of the infrastructure is in place for a modern rendering system capable of meeting the demands of current application developers. The missing pieces of the puzzles are hardware acceleration for the *Render* extension and a fuller and more complete *Render*-based, client-side API for rendering. Little progress has been visible on the former. Currently only the Matrox XFree86 driver supports hardware-acceleration for the *Render* protocol and that's largely experimental. A full implementation will probably have to wait until the XFree86's acceleration architecture (XAA) is overhauled which is not expected until XFree86 5.0.

The other requirement, an advanced 2D-drawing library to take advantage of the *Render* extension, is much closer. Keith Packard and Carl Worth are developing the *Xr* library for just this purpose. Actually, *Xr* is device-independent rendering toolkit, which provides a rich set of graphics primitives similar to the PDF (Portable Document Format) and SVG (Structured Vector Graphics) models. Its supports the compositing abilities of the *Render* extension directly, but will also work on servers without the *Render* extension via a separate compositing toolkit, *Xc*, which Packard and Worth are also developing.

Development issues

Despite some of the debate that has flared up following the Packard incident, the technological challenges facing X have never really been an issue: any such problems are being solved or are solvable. More difficult questions to answer are how the XFree86 project should be managed and what role the project should play.

Many of the points that Keith Packard raised in his *Call for Open*

X Extensions

Some common extensions and what they do

DGA The XFree86 Direct Graphics Architecture extension gives an X client direct access to the X display's framebuffer. Naturally, this works only when the client runs locally to the server. The DGA extension is frequently used for games programming, where the performance gained by direct access is useful, but inherent security problems make it unpopular.

GLX The GLX protocol was created by SGI as means for sending OpenGL graphics primitives over an X wire and thus allowing X to support 3D graphics. XFree86 4.0 introduced hardware-accelerated 3D graphics with its Direct-Rendering Infrastructure, which boosts 3D performance by by-passing the X wire when the X client is local to the server.

MIT-SHM The MIT Shared Memory extension allows X images to be stored in memory shared between the X client and server and so avoids the need to send the image over the X wire offering a performance boost with large images. Client and server obviously must be on the same host.

RENDER The *Render* extension provides a new compositing render model for X. This is discussed in more detail on this page under the heading *The Render Extension*.

SHAPE The Non-rectangular Window Shape extension, as its unabbreviated name suggests without any ambiguity, permits X windows and borders to be non-rectangular.

VidModeExtension – An XFree86 extension which allows a client to modify monitor settings or change the display mode (unlike the *RandR* extension it doesn't change the size of the root window). It's used by the *xvidtune* tool to tweak monitor settings and is often employed in conjunction with the DGA extension by games software to give the appearance of running 'full screen'.

XVIDEO The XVideo extension allows X to make use of video-oriented hardware features supplied by a graphics chipset, such as hardware overlays, colour-space conversion, image scaling, and video capture.

Governance post are valid. But claims that XFree86 is not open are perhaps misleading. Maybe the XFree86 team could do more to encourage new developers, yes, but they have limited resources. Besides, important progress has been made, such as the opening up of the XFree86 CVS repository and mailing lists, and availability of regular source code snapshots. A critical problem for the XFree86 is a lack of developers, especially when it comes to driver development. Few new developers really have the expertise to tackle such a difficult task as writing and maintaining the code that drives the hardware of modern graphics chipsets.

Driver development is definitely an area that needs to be addressed. The XFree86 release schedules mean that end users often have to wait at least six months to be able to get updated drivers to support new hardware or fix bugs; this can be especially frustrating when the fixes appear much more quickly in CVS. A solution to speed up the distribution of drivers updates to the end user is to take advantage of XFree86's module-loading ability and finalise the hardware driver interface, so that binary drivers can be built and released independently of XFree86



XWindowSystem

“XFree86 was never intended to be a standards body, nor is its goal the global domination of Linux as a desktop system.”



releases. This would also be good news for hardware vendors, such as NVidia and ATI, who ship binary-only drivers for their cards. Although encouraging closed-source drivers

will not please Free Software advocates, the truth is the bulk of desktop computer users doesn't care about such issues; they just want their hardware to work. (This is why SciTech's SNAP driver architecture may do well, especially with distro vendors who have no problems shipping proprietary software.)

Some suggest that XFree86 such be made more modular throughout, to enable the separate release of its various components; others say the splitting off the driver development as a separate sub-project and promoting its use (without the full X distribution) as a hardware-layer for embedded systems would revitalise driver development. A simpler solution would be for the distro vendors to devote as many of their developers to the XFree86 project as they do to the Linux kernel at present. After all, it's just as critical.

Complaints that the XFree86 project is not doing enough to develop the technology and standards that the various Linux desktop projects need for interoperability are unfair. XFree86 was never intended to be a standards body, nor is its goal the global

domination of Linux as a desktop system. It simply aims to create the best X implementation on the platforms it supports. The XFree86 developers can be decidedly bullish in this regard: if you don't like the way we do things, do it yourself. As an Open Source undertaking, the XFree86 project can be forked. But a fork may not be a bad thing if it is done properly and with a definable aim – such as, for example, as a testing ground for experimental X extensions. The danger is a duplication of effort and splitting an already very shallow pool of developers. Notably, Keith Packard, appears not to be in favour of an XFree86 fork, despite this being the reason he was ousted from the XFree86 core team. If official standards are what are needed for desktop Linux to flourish – along the lines of the work that is being done unofficially by the freedesktop.org project – then this should really be done under the umbrella of a standards bodies like X.org or the Free Standards Group (who maintain the Linux Standard Base) and not put on the shoulder of XFree86.

The future

Whether you love or loathe the X Windows System, its role as a fundamental component of the Linux desktop is assured. As X itself dramatically proves, standards and APIs are key, not any one particular implementation. The X API is simply too important to disregard. It may be that one day XFree86 will prove to be not the right implementation for Linux on the desktop. Fine. Other solutions are available (see box, *X Alternatives*). But note that all these other graphics systems provide or intend to provide a means for X compatibility.

The future for XFree86 itself looks bright, so there's no need to replace it just yet, however. XFree86 5.0 will contain many of the developments we have talked about here, such as the powerful new Xr/Xc rendering toolkit, a new acceleration architecture, and perhaps even – at last – real support for translucent windows. Hopefully, some of the solutions proposed for improving driver development will be pursued, as this is the one area of XFree86 that can cause the desktop user real problems. **LXF**

X Alternatives

Some other graphical environments to try

DIRECTFB

www.directfb.org/

The DirectFB project provides an open-source, hardware-accelerated graphics system based on the kernel framebuffer layer and was originally designed for embedded use. It provides an advanced compositing model, much like the X *Render* extension. XFree86 has been ported, and XDirectFB now supports most X extensions, and, uniquely, translucent windows. Newly added to DirectFB is 3D hardware-acceleration, based on an embedded version of *Mesa* and a modified version of the kernel direct-rendering manager, and this can

be employed by XDirectFB's implementation of the GLX extension. The range of hardware supported by DirectFB is currently limited.

SCITECH SNAP

www.scitechsoft.com/

Scitech is in the process of porting its well-respected 2D graphics architecture, called *SNAP*, to Linux. Beta versions are currently available for the x86 architecture. This also include an XFree86 module to enable XFree86 to use *SNAP* drivers. Notable features are ease-of-use (*SNAP* drivers are completely plug-and-play), support for a wide range of hardware, and good 2D performance.

Hardware-accelerated 3D and the XVideo extension aren't currently implemented. *SNAP* is a proprietary system, but this means support for new chipsets can often be more complete than in the Open Source alternatives.

GGI

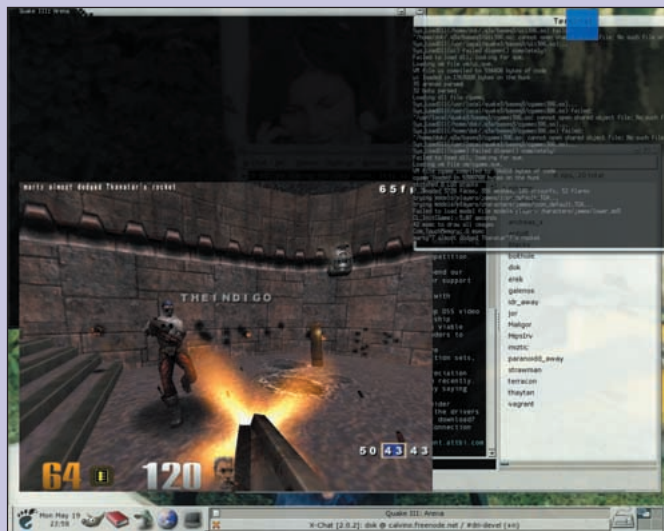
www.ggi-project.org/

The General Graphics Interface is a device-independent graphics system. It is cross-platform and can output to a variety of hardware abstraction layers including *X11*, *SVGAlib*, *fbdev* (the kernel framebuffer) and *libSDL*. Ultimately, on Linux, the GGI project aims to target the Kernel Graphic Interface (KGI), a kernel-based accelerated graphics framework, but little progress has been made on this project. GGI's core API provides a basic rendering model, but this is extended by various auxiliary libraries. The port of XFree86 to GGI is known as XGGI.

FRESCO

www.fresco.org/

Fresco is a long-established project to make a modern, network-transparent windowing system with a powerful structured graphics toolkit to replace X. It has changed names several times (it was previously called Berlin) and seems to have had problems with direction at times. It still very much at the experimental stage, but its interesting features include CORBA for an IPC system, an advanced rendering model, and the fact that it uses GGI as a hardware layer rather than supplying its own drivers.



The DirectFB project's XFree86 port is becoming a viable replacement for the vanilla XFree86, providing your graphics chipset is supported.