Metisse is not a 3D desktop!

Olivier Chapuis, Nicolas Roussel
Projet In Situ (CNRS, Université Paris-Sud & INRIA Futurs)
LRI, Bâtiment 490, Université Paris-Sud
91405 Orsay Cedex, France
chapuis | roussel @lri.fr

ABSTRACT

Twenty years after the general adoption of overlapping windows and the desktop metaphor, modern window systems differ mainly in minor details such as window decorations or mouse and keyboard bindings. While a number of innovative window management techniques have been proposed, few of them have been evaluated and fewer have made their way into real systems. We believe that one reason for this is that most of the proposed techniques have been designed using a low fidelity approach and were never made properly available. In this paper, we present Metisse, a fully functional window system specifically created to facilitate the design, the implementation and the evaluation of innovative window management techniques. We describe the architecture of the system, some of its implementation details and present several examples that illustrate its potential.

Categories and Subject Descriptors: D.4.9 [Systems Programs and Utilities]: Window managers. H.5.2 [User Interfaces]: Graphical user interfaces, Screen design, Windowing systems.

Additional Keywords and Phrases: application redirection, OpenGL, X Window system, VNC.

INTRODUCTION

Overlapping windows that users can freely move and resize have been described more than thirty years ago and have been available to the general public for more than twenty years [14]. Over the time, various interaction techniques have been proposed to control the placement, size and appearance of application windows. Yet, from a user perspective, the most popular window systems differ mainly in minor details such as window decorations or mouse and keyboard bindings, and not in their fundamental operation principles. As B. Myers already put it in 1988, “there is not a great deal of difference among different window managers” [13].

The growing range of activities supported by interactive computer applications makes it more and more difficult to remember these activities and to organize them. At the same time, recent advances in computer graphics and display technologies combined with decreasing costs are changing the nature of the problem. High performance graphics cards, high definition displays, big screens and multiple monitor systems are becoming common place. From the original time when window systems were too demanding and had to be carefully tuned for performance, we have now moved to a situation where a lot of software and hardware resources are available. The question is: how should these resources be used?

Little research has been performed on understanding people’s space management practices [9]. While a number of innovative window management techniques have been proposed by HCI researchers over the last few years [23], very few of these techniques have been formally evaluated and even fewer, if any, have made their way into current window systems. We believe that these two points are strongly related to the fact that most of the techniques proposed by the HCI community were designed using a low fidelity approach and were never made properly available in a real window system.

Building a whole new window system is a hard task, one that few HCI researchers are willing to do. At the same time, existing systems are either closed boxes, inaccessible to developers, too limited for the envisioned interaction techniques or too complex to program. How would you implement a zoomable window manager? One that would strengthen the paper and desktop metaphor? One that can be used on a tabletop display? One that would support bi-manual interaction?

In this paper, we present Metisse, a fully functional window system specifically created to facilitate the design, the implementation and the evaluation of innovative window management techniques. The paper is organized as follows. After introducing some related work, we describe Metisse by providing an overview of its design and architecture as well as some implementation details. We then present several examples that illustrate its potential for exploring new window management techniques. Finally, we conclude with a discussion and some directions for future research.

RELATED WORK

In this section, we briefly describe the current state of the three most popular window systems as well as several research projects related to the exploration of new window management techniques.

Apple Mac OS X, Microsoft Windows and X Window

The Apple Mac OS X graphics system is based on three different libraries : Quartz for 2D graphics (a rendering engine based on the PDF drawing model), OpenGL for 3D graphics and QuickTime for animated graphics and video. Windowing services are available through a software called *Quartz Compositor* [1]. This software handles the compositing of all visible content on the user's desktop: Quartz, OpenGL and QuickTime graphics are rendered into off-screen buffers that the compositor uses as textures to create the actual on-screen display.

Among other features, Quartz Compositor supports window transparency, drop shadows and animated window transformations, which are used to create various effects such as the *scale* and *genie* effects used for window (de)iconification, the *fast user switching* animation and the three *Exposé* modes. From a developer perspective, however, Quartz Compositor is a closed box. Most of its functionalities are available through a private, undocumented and probably unstable API that only a few highly-motivated developers are willing to use [1]. Gadget applications using this private API are interesting because they show that the compositor is much more powerful than it seems and that services such as *Exposé* are in fact the result of a careful selection of its features. At the same time, this is very frustrating since this compositing policy and the associated design space remain out of reach for the HCI researcher.

The window system of Microsoft Windows is tightly coupled with the operating system, which makes it difficult to access and modify. Several applications such as SphereXP[2] allow to replace the traditional desktop by a 3D space in which arbitrary objects can be painted with 2D images from application windows. However, the implementation details of these systems are not available. The next version of Microsoft Windows will most probably include a composite desktop based on DirectX 9 [3]. Details of what will be available to users and developers remain uncertain. However, one can reasonably imagine that the compositing policy will probably be out of reach for the average developer and HCI researchers.

A key feature of the X Window System [19] (or X) is that any application can act as a window manager. As a consequence, a large number of window managers have been developed for this system, providing a range of appearances and behaviors. Recent X extensions make it now possible for these window managers to use a compositing approach [8]: *Composite*, that allows windows to be rendered off-screen and accessed as images; *Damage*, that allows an application to be notified when window regions are updated; and *Event Interception*, that allows keyboard and mouse events to be pre-processed before being sent to their usual targets. Another extension, *Xfixes*, provides the data types and functions required by these extensions.

Experimental X window managers are slowly taking advantage of these extensions to provide eye candy effects similar to the ones proposed by Mac OS X. However, the numerous extensions required make it hard for developers unfamiliar with the X architecture to implement their own compositing window manager. Moreover, implementing a fully functional and standard-compliant X window manager requires much more than simple window image compositing and event preprocessing.

Window management research

Many of the window management solutions proposed by the HCI research community have been designed using a low-fidelity approach and have never been implemented as part of a real window system. *Elastic windows* [12], for example, were only implemented within custom applications. *Peeled back windows* have been demonstrated within specific Tcl/Tk and Java prototypes [2, 7]. Window shrinking operations and dynamic space management techniques have also been demonstrated within specific Java prototypes [10, 4].

A notable exception to the low-fidelity approach is Microsoft's Task Gallery [17], a system that uses input and output redirection mechanisms for hosting existing Windows applications in a 3D workspace. The redirection mechanisms require several modifications of the standard window manager of Windows 2000. They provide off-screen rendering and event pre-processing facilities similar to those becoming available in X [24]. However, as the Windows 2000 modifications have never been publicly released, researchers outside Microsoft have not been able to experiment with this high-fidelity approach.

Several recent projects have tried to move from low-fidelity prototypes to real functional systems. Scalable Fabric [16], mudibo [11] and WinCuts [22], for example, are implemented as "real" applications supplementing the legacy window manager of Windows XP. However the fact that these systems are developed outside the window system makes them unnecessary complex, potentially inefficient and harder to combine with other window or task management techniques. As an example, since they can't be notified of window content updates, the three mentioned systems resort to periodically calling a slow *PrintWindow* function to get window images, which is both inefficient and unsuitable for interactive manipulation of the window content.

Experimental desktop environments

Ametista [18] is a mini-toolkit designed to facilitate the exploration of new window management techniques. It supports the creation of new OpenGL-based desktop environments using both a low-fidelity approach, using placeholders, as well as a high-fidelity approach based on X application redirection through a custom implementation of the VNC [15] protocol. Several 3D environments such as Sun's Looking Glass[3] and Croquet [20] are also using the X extensions we already mentioned to host existing applications.

The main problem of these new environments is that although they provide the fundamental mechanisms for implementing compositing window managers, they implement only parts of the standard X protocols related to window management (e.g. ICCCM, EWMH) that define the interac-

[1]`http://cocoadev.com/index.pl?WildWindows`

[2]`http://www.hamar.sk/sphere/`

[3]`http://wwws.sun.com/software/looking_glass/`

tions between window managers, applications, and the various utilities that constitute traditional desktop environments such as GNOME or KDE. As a consequence, these environments are hardly usable on a daily basis, since they do not support common applications such as mail readers, Web browsers, media players or productivity tools.

METISSE

Metisse is an X-based window system designed with two goals in mind. First, it should make it easy for HCI researchers to design and implement innovative window management techniques. Second, it should conform to existing standards and be robust and efficient enough to be used on a daily basis, making it a suitable platform for the evaluation of the proposed techniques. Metisse is not focused on a particular kind of interaction (e.g. 3D) and should not be seen as a new desktop proposal. It is rather a tool for creating new desktops.

The design of Metisse follows the compositing approach and makes a clear distinction between the rendering and the interactive compositing process. The *Metisse server* is a modified X server that can render application windows off-screen. The default compositor is a combination of a slightly modified version of a standard X window manager, *FVWM*, with an interactive viewer called *FvwmCompositor*. As we will see, the use of FVWM provides a lot more flexibility and reliability than custom-made window managers such as those used by Ametista or Looking Glass. In the next section, we will also show that other compositors can be used in conjunction with the Metisse server.

Metisse is implemented in C and C++ and runs on the Linux and Mac OS X platforms. Figure 1 shows the communication links between the various software components. The Metisse server sends window-related information, including window images, to FvwmCompositor. FvwmCompositor displays these images with OpenGL, using arbitrarily transformed textured polygons, and forwards input device events to the server. FVWM can solely handle basic window operations such as move, resize or iconify, issuing the appropriate commands to the X server. It can also delegate these operations to FvwmCompositor. New window operations can be implemented either as FVWM functions, using a scripting language, or in FvwmCompositor.

The following subsections will provide more implementation details about the Metisse server, our modified FVWM and FvwmCompositor.

Metisse server

The Metisse server is a fully functional X Window server derived from Xserver[4], a software used by the X community for exploratory developments. It uses Xserver's rootless extension to provide off-screen rendering of application windows: each top-level window is rendered in a separate pixmap – an image stored in a single contiguous memory buffer – that is dynamically allocated when the window is created and reallocated when it is resized. The server stores along with each window image the coordinates of its upper-left corner

[4]http://www.freedesktop.org/

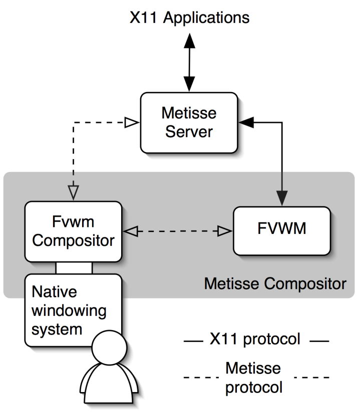


Figure 1: General overview of Metisse.

in the compositor's display. These coordinates are the ones reported to applications that issue geometry requests.

Each time an application updates a window content, the server sends an update notification to the compositor. The corresponding region of the pixmap can be transmitted to the compositor using a custom protocol similar to VNC. It can also be copied in a shared memory space if the two processes are running on the same machine. These off-screen rendering and update notification mechanisms are quite similar to the *Composite* and *Damage* X extensions. In fact, the main reason why we didn't use these extensions is that they were still in early development stages and heavily discussed when we started implementing Metisse.

The server sends various other notifications to the compositor to indicate the creation, destruction, mapping or unmapping of a window as well as geometry modifications, changes in the stacking order and cursor changes. It provides the compositor with the actual bounds of shaped (i.e. non rectangular) windows. It also indicates a possibly related window for all transient and override redirect windows (e.g. pop-up menus). All these notifications make it easy for the compositor to maintain a list of the windows to be displayed and to apply to pop-up menus the transformation used for the corresponding application window.

Window visibility is usually an important concern for X server implementations, since a window can be partially occluded by another one, or be partially off-screen. Traditional servers generate *Expose* events to notify applications of visibility changes and clip drawing commands to the visible regions. Since the Metisse server renders windows in separate pixmaps, partial occlusion never happens. Moreover, since the actual layout of windows is defined by the compositor, the notion of being partially off-screen doesn't make any sense in the server. As a consequence, the Metisse server never generates *Expose* events.

Traditional X servers receiving a mouse event use the pointer location and their knowledge of the screen layout to decide which window should receive the event. Again, in the case of Metisse, since the actual layout is defined by the compositor, the server cannot perform this computation. As a consequence, the mouse events transmitted by the compositor must explicitly specify the target window. When the screen layout changes, X servers usually look for the window under the pointer and, if it has changed, send *Leave* and *Enter* events to the appropriate windows. In the case of Metisse, this process is left to the compositor.

Metisse compositor: FVWM and FvwmCompositor

FVWM[5] is an X window manager created in 1993 and still actively developed. Originally designed to minimize memory consumption, it provides a number of interesting features such as GNOME and KDE compatibility, customizable window decorations, virtual desktops, keyboard accelerators, dynamic menus, mouse gesture recognition, as well as various focus policies. All these features can be dynamically configured at run-time using various scripting languages.

Scripted functions are a powerful and simple way of extending the window manager. As an example, one can easily define a new iconification function that raises the window, takes a screenshot of it with an external program, defines this image as the window icon and then calls the standard iconification command. Commands can be executed conditionally, depending on the nature and state of a window, and can be applied to a specific set of windows. Commands and scripted functions can be easily bound to a particular mouse or keyboard event on the desktop, a window or a decoration element. They can also be bound to higher-level events such as window creation or focus changes.

FVWM can also be extended by implementing *modules*, external applications spawned by the window manager with a two-way communication link. FvwmCompositor is an FVWM module implemented with the Núcleo[6] toolkit, which provides a simple OpenGL scenegraph and a basic asynchronous scheduler for multiplexing event sources. It uses the window images as textures that can be mapped on arbitrary polygons, these polygons being themselves arbitrarily transformed (Figure 2).

Implementing the Metisse compositor as an extension of FVWM has several advantages over developing one from scratch. First, almost nothing needs to be done to replicate the standard window operations of existing window systems: FvwmCompositor simply needs to display window images at the positions given by the Metisse server, and to forward input device events to it. Second, since FVWM reparents application windows in new ones containing the decorations, these decorations are automatically made available in the compositor through the server. This has proved to be much more convenient than writing OpenGL code to display and interact with title bars and borders, buttons and pull-down menus.

FvwmCompositor displays its composition in an OpenGL window of the native window system (a GLX window on Linux and a Carbon/AGL window on Mac OS X). Although not mandatory, this window is usually set to be full-screen, so that FvwmCompositor visually replaces the native window system. The current implementation uses a perspective projection. The third dimension (i.e. Z axis) of OpenGL is

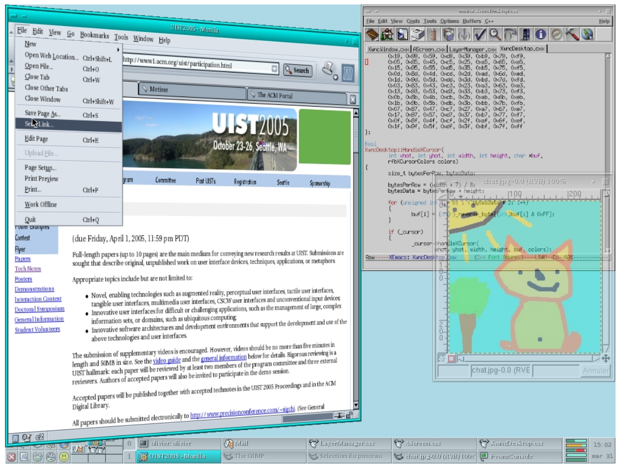


Figure 2: Basic composition showing a rotated Mozilla window with a pop-up menu, a downscaled xemacs window and a translucent GIMP window. Note that the same transformation is used for the pop-up menu and the Mozilla window.

used to enforce the stacking order of the windows defined in the server by FVWM. In order to avoid intersections between windows that would not be on parallel planes, large Z distances are used between windows, especially for the bottom and top ones. Consequently, all windows are rescaled to keep their original size despite their distance to the viewer and the perspective projection.

Keyboard events are simply forwarded to the Metisse server, the keyboard focus policy being handled by FVWM. When receiving a mouse event, FvwmCompositor uses OpenGL's selection mode and picking to find the window under the pointer. It then uses the transformation matrix associated to that window and its position on the server's virtual screen to transform the mouse coordinates into the server's coordinate system. The event is then forwarded to the server with these adjusted coordinates and additional information specifying the target window.

In some situations, FvwmCompositor needs to use the transformation matrix of a particular window even if the mouse pointer is not over it. This happens, for example, if the user is interactively resizing the window in a movement so fast that the pointer leaves the window. To avoid this particular situation, FvwmCompositor uses "infinite" polygons when drawing windows in selection mode.

EXAMPLES

In the previous section, we have described the architecture of Metisse and explained how this design provides a window system that is both fully functional and highly tailorable. In this section, we present several examples that illustrate how Metisse facilitates the implementation of innovative window management techniques.

Basic operations

The following code sample shows how FVWM can be configured to scale windows by clicking on one of the buttons of their title bar or pressing some keys:

[5] `http://www.fvwm.org/`
[6] `http://insitu.lri.fr/~roussel/projects/nucleo/`

```
Mouse 3 4 A SendToModule FvwmCompositor Scale 0.7

Key minus W C SendToModule FvwmCompositor Scale 0.9
Key plus W C SendToModule FvwmCompositor Scale 1.11
```

The first line requests FVWM to send the string “Scale 0.7” to FvwmCompositor when the user does a right mouse click (third button) on the minimize icon of the title bar (fourth icon), no matter the active keyboard modifiers (A is for “any modifier”). In addition to the specified string, FVWM will send the X id of the window that received the click. A simple parser implemented in FvwmCompositor will decode the message and perform a 30% reduction of the representation of the specified window. Similarly, the two other lines request FVWM to send a scale command to FvwmCompositor when the user presses Ctrl+ or Ctrl- while the mouse pointer is on the window.

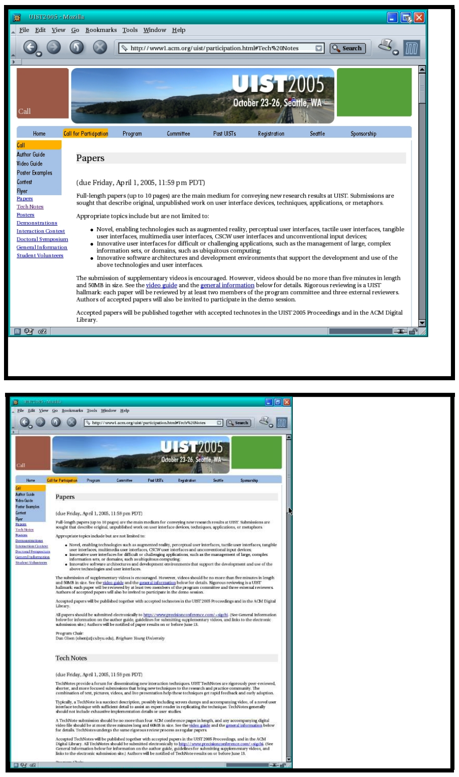
Other commands implemented in FvwmCompositor allow to rotate a window around different axes, to scale it non-uniformly and to set, lower and raise its opacity and brightness levels. Metisse provides a default configuration file for FVWM with menus and various bindings (mouse, keyboard or high-level events) for all these operations. These bindings make it possible, for example, to lower the brightness of all windows except those of the application having the keyboard focus. One can also specify that all windows of a certain type should be semi-transparent (e.g. those of an instant messaging application) and become fully opaque when the mouse pointer comes over them.

FvwmCompositor commands can also be combined with traditional window operations in interesting ways. As a first example, one can easily replace the usual maximizing operation by a function that zooms in the window so that it takes the whole screen space instead of resizing it. Another interesting combination is the ZoomOutAndMaximizeY function illustrated by Figure 3. This function zooms out a window uniformly and then resizes its height so that it takes the whole screen. It is available in Metisse as a toggle switch (i.e. calling the function a second time returns the window back to its previous state).

The ZoomOutAndMaximizeY function is particularly interesting when working on a large text document. When activated, it allows to see a larger part of the document which in turn makes it easier to navigate. Calling the function a second time restores the original scale and size of the window, providing a more detailed view of the selected part of the document. A notable fact about this function is that it has been designed and implemented in a few minutes by the authors on a laptop during a subway trip between Paris and Orsay. The final implementation is only a few lines long to be added to the configuration file of Metisse.

Interactive window manipulation

Interactive window manipulations such as the peel-back operation described in [2] (Figure 4) cannot be implemented with FVWM scripts. This kind of complex operations need to be handled directly by FvwmCompositor. In order to facilitate this, we have created a new FVWM command called MetisseInteractiveManip.



Papers

(due Friday, April 1, 2005, 11:59 pm PDT)

Full-length papers (up to 10 pages) are the main medium for conveying new research results at UIST. Submissions are sought that describe original, unpublished work on user interface devices, techniques, applications, or metaphors.

Appropriate topics include but are not limited to:

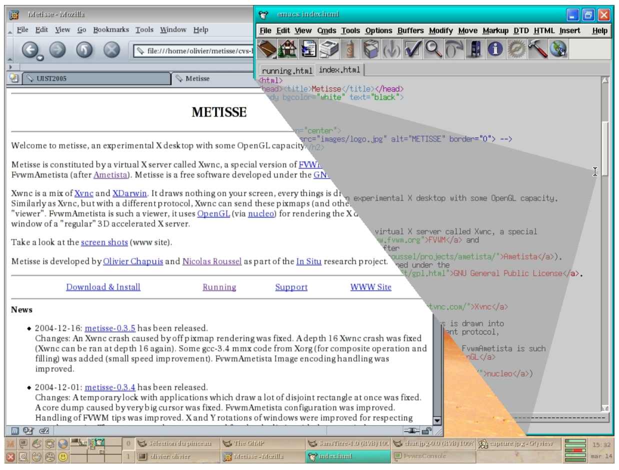
- Novel, enabling technologies such as augmented reality, perceptual user interfaces, tactile user interfaces, tangible user interfaces, multimedia user interfaces, CSCW user interfaces and unconventional input devices;
- Innovative user interfaces for difficult or challenging applications, such as the management of large, complex information sets, or domains, such as ubiquitous computing;
- Innovative software architectures and development environments that support the development and use of the above technologies and user interfaces.

The submission of supplementary videos is encouraged. However, videos should be no more than five minutes in length and 50MB in size. See the video guide and the general information below for details. Rigorous reviewing is a UIST hallmark: each paper will be reviewed by at least two members of the program committee and three external reviewers. Authors of accepted papers will also be invited to participate in the demo session.

Accepted papers will be published together with accepted technotes in the UIST 2005 Proceedings and in the ACM Digital Library.

Tech Notes

Figure 3: ZoomOutAndMaximizeY funtion applied on a Web browser showing a large document. The upper image shows the browser before calling the function, the lower one shows the resulting transformation.



METISSE

Welcome to metisse, an experimental X desktop with some OpenGL capacity.

Metisse is constituted by a virtual X server called Xwnc, a special version of FVWM
FvwmAmetista (after Ametista). Metisse is a free software developed under the GN

Xwnc is a mix of Xvnc and XDarwin. It draws nothing on your screen, every things is dr
Similarly as Xvnc, but with a different protocol, Xwnc can send these pixmaps (and othe
"viewer". FvwmAmetista is such a viewer, it uses OpenGL (via nucleo) for rendering the X d
window of a "regular" 3D accelerated X server.

Take a look at the screen shots (www site).

Metisse is developed by Olivier Chapuis and Nicolas Roussel as part of the In Situ research project.

Download & Install Running Support WWW Site

News

- 2004-12-16: metisse-0.3.5 has been released.
  Changes: An Xwnc crash caused by off pixmap rendering was fixed. A depth 16 Xwnc crash was fixed (Xwnc can be ran at depth 16 again). Some gcc-3.4 mmx code from Xorg (for composite operation and filling) was added (small speed improvement). FvwmAmetista Image encoding handling was improved.
- 2004-12-01: metisse-0.3.4 has been released.
  Changes: A temporary lock with applications which draw a lot of disjoint rectangle at once was fixed. A core dump caused by very big cursor was fixed. FvwmAmetista configuration was improved. Handling of FVWM tips was improved. X and Y rotations of windows were improved for respecting

```
<html>
head><title>Metisse</title></head>
dy bgcolor="white" text="black">
n="center">
src="images/logo.jpg" alt="METISSE" border="0"> -->
/h2>
n experimental X desktop with some OpenGL capacity.
virtual X server called Xwnc, a special
w.fvwm.org">FVWM</a> and
fter
oussel/projects/ametista/">Ametista</a>).
ned under the
t/gpl.html">GNU General Public License</a>.
tvnc.com/">Xvnc</a>
s is drawn into
ent protocol,
FvwmAmetista is such
nGL</a>
">nucleo</a>)
```

Figure 4: Xemacs window being peeled back.

MetisseInteractiveManip abstracts the concept of interactive manipulation from the FVWM point of view. It takes a window operation name and a cursor name as arguments. When executed, it grabs the mouse and the keyboard (i.e. forbids other applications to use them), changes the cursor for the one specified, and checks that the specified window is still valid. FVWM then sends to FvwmCompositor the operation name along with the cursor position and the target window, and enters a simple event loop, waiting for the operation to complete. Upon completion, FVWM releases the mouse and keyboard and reenters its main loop.

Interactive move, rotation and scaling are implemented in the same way, as FvwmCompositor operations called in response to an FVWM message sent by MetisseInteractiveManip. Here's how the folding operation might be configured in FVWM:

```
# Immediately (I) raise the window and start the
# fold operation (Fold) if the mouse is dragged (M)
# or hold (H)
AddToFunc FoldWindow
+ I Raise
+ M MetisseInteractiveManip Fold FOLD_CURSOR
+ H MetisseInteractiveManip Fold FOLD_CURSOR

# Bind the folding function to a right mouse button
# click (3) on the window border (F A)
Mouse 3 F A  FoldWindow
```

The interactive scale operation is in a certain way similar to the usual resize operation and can be bound to the manipulation of the borders of the windows with a given keyboard modifier. Rotation around the Y axis can be bound to a mouse drag on the left or right border of the window with a keyboard modifier. The top and bottom borders can be used for rotations around the X axis. The corners of the window might be used for rotations around the Z axis.

We believe that window scaling might offer some interesting new ways of managing overlapping windows. As opposed to the traditional resize operation, scaling a window reduces overlapping while preserving the layout of window contents. This has proved to be useful, for example, for checking the layout of a Web page in one or more browsers while editing it in a text editor. Temporarily scaling down two applications also allows to quickly perform a series of interactions between them, such as drag-and-drop or copy/paste operations. Note that when using a perspective projection, rotations around the X and Y axis produce a non-uniform scaling effect that can also be used to reduce overlapping.

Unlike low-fidelity environments usually used to implement innovative window management techniques, Metisse allows all these techniques to be used for real on a daily basis. This can help adjusting the details of a particular technique. The peel-back operation, for example, became much more interesting after we decided to make the back side of the peeled-back window translucent (Figure 4). Daily use also helped realize that the ability to put back windows into a "normal" state after some transformation was very important. As a consequence, the default Metisse configuration allows to cancel the transformations applied to a window by right-clicking on its title bar, a simple animation being used to ease the transition. We are also adding a history mechanism with an undo/redo mechanism that should make it easier to understand manipulation errors and to capture interaction sequences to create new commands.

Animations and temporary transformations
Animations have long been used in window managers to provide feedback of ongoing operations. Specifying animations in Metisse is quite simple. It doesn't require much programming skills and could probably be done by experienced users. The following code sample shows how to create an animated iconification function. It uses two lines of Perl to send multiple *Scale* commands to FvwmCompositor to produce the animation effect. Note that this animation is implemented as a script that can be parsed at run-time by FVWM. No modification of FvwmCompositor is necessary:

```
AddToFunc myIconify
+ I PipeRead 'for ((i=0; $i<20; i++)) ; do \
     echo "SendToModule FvwmCompositor Scale 0.9"; \
     done'
+ I State 1 True

AddToFunc myDeIconify
+ I PipeRead 'for ((i=0; $i<20; i++)) ; do \
     echo "SendToModule FvwmCompositor Scale 1.11"; \
     done'
+ I State 1 False

AddToFunc myToggleIconify
# deiconify if iconified
+ I ThisWindow (State 1) myDeIconify
# iconify if not iconified
+ I TestRc (NoMatch)  myIconify
```

The functions for moving, rotating and scaling windows have been implemented in two forms. The first one corresponds to the usual operation mode: the user presses a mouse button and moves the mouse to define the transformation. When the button is released, the window stays where it is, using the last transformation. The second form is a temporary one: when the mouse button is released, the window returns to its original position with an animation. This second form provides interesting alternatives to the folding operation. While experimenting with translucency effects, we also found out that temporary modifications of the opacity level also offers new interesting possibilities. As an example, when moving a window, making that window or the other ones translucent helps finding the right place to put it.

Position-dependent window transformations
Until now, we have presented techniques that put the user in total control of every detail of the transformations applied on windows. However, combining two or more elementary transformations, such as a move and a rotation, can be quite tedious. A simple and powerful way of solving this problem is to define the transformation to be applied on a window as dependent of its position. This way, the user will indirectly apply these transformations by simply moving the window. A position-dependent transformation can be described as a function

$$f : S^4 \longrightarrow \{\text{Finite sequence of OpenGL transformations}\}$$

where S is the screen (or a set of screens with its layout for a multi-monitor setting). Given a window we apply to it a sequence of OpenGL transformations $f(\bar{a}, \bar{b}, \bar{c}, \bar{d})$ where $\bar{a} = (a_1, a_2), \ldots \bar{d} = (d_1, d_2)$ are the coordinates of the corners of the window.

As a first example, we use this approach to scale down windows as they approach the left or right borders of the screen so that they remain fully visible instead of becoming partly off-screen (Figure 5). A minimum window size is imposed so that at some point, moving the window further towards the border of the screen has no effect. A special FvwmCompositor command allows to restore the original position and size of a window before it was moved and scaled. This new operation provides a simple and continuous way, as opposed to the usual iconification operation, of moving a window from the foreground to the background. Although different in spirit, it is in some ways similar to the window manipulation techniques provided by Scalable Fabric [16].

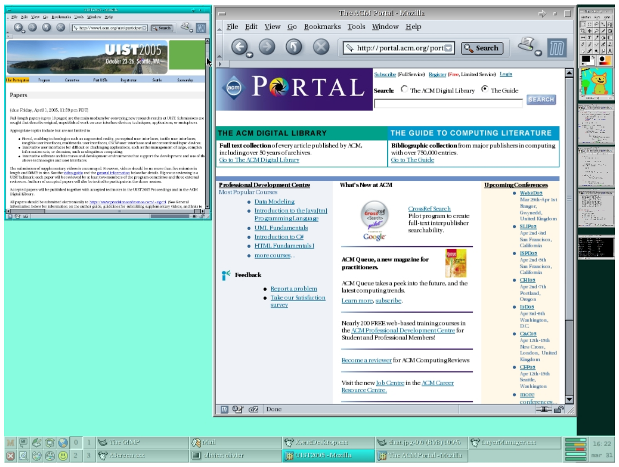


Figure 5: The top left window has been pushed against the left border of the screen and scaled down. The right window has the same size but is not scaled. The small windows on the right have been either dragged to the black region by hand or sent there by clicking on a button in their title bar.

Daily use of this move-and-scale operation also gave us the idea of implementing a second version of it, a temporary one following the approach described in the previous subsection. This version allows users to grab a window with the mouse, move it to the side of the screen (and thus reduce its size) and then simply release the mouse button to restore the window's original size and position. This new operation proved to be quite useful to see what's behind a window while keeping its content visible.

Our second example of position-dependent transformation has been designed for tabletop interactive displays [6]. In this example, we split the screen into two equal parts A (the bottom part) and B (the top part). When a window is totally contained in A no transformation is applied to it. When it is totally contained in part B, it is rotated around the Z axis by 180 degrees. When a window is between the A and B parts a rotation between 0 and 180 degree is applied to it depending on the distance to the splitting line. The rotation is applied clockwise if the center of the window is on the left part of the screen and counterclockwise if on the right. This way, if a window is moved from A to B, it is progressively rotated upside down (Figure 6).

One nice feature of FvwmCompositor is that it can duplicate a window. Users can interact with a duplicated window exactly as if it were the original one (see [21] for more details). This feature can be combined with the tabletop inter-

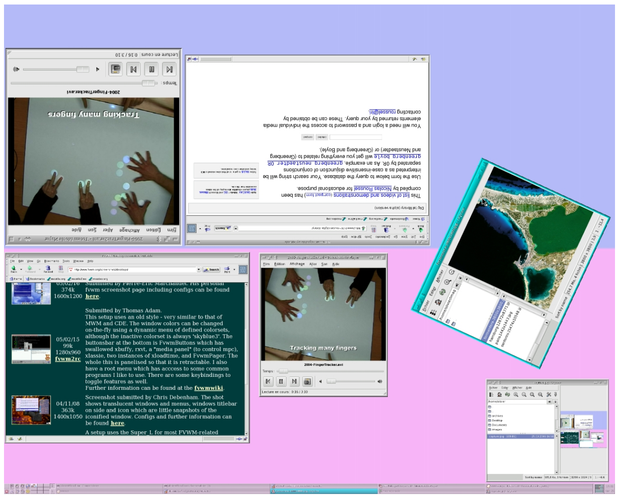


Figure 6: Tabletop interface featuring automatic window orientation and on-demand window duplication.

actions we just described: the top-left window of Figure 6 is a zoomed duplicate of the one in the middle of the lower part. Window duplication is also interesting in multiple-monitors configurations. Although the current implementation of Metisse does not support multiple simultaneous input, this example has already proved to be useful in situations where one user needs to show something to other people.

Interactive desktop manipulation

Global operations that transform all the windows can also be implemented in Metisse. As an example, we have implemented a zoomable desktop that allows to navigate in a virtual space nine times bigger than the physical one (nine virtual screens arranged in a 3x3 matrix). This desktop supports standard panning techniques that allow to move from one virtual screen to adjacent ones by simply moving the mouse towards the corresponding edge. It also supports continuous zooming with the mouse wheel, which provides an overview of several adjacent screens at the same time up to a complete bird's eye view of the virtual desktop (Figure 7).

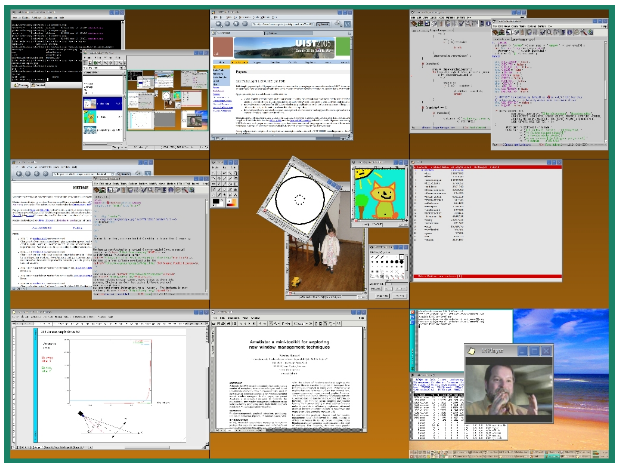


Figure 7: Bird's eye view of a virtual desktop made of nine virtual screens.

In the current implementation of this zoomable desktop, clicking on a particular virtual screen from an overview initiates an animated transition that zooms into it. Note that all applications remain accessible while in overview mode: they can be moved between virtual screens, resized or closed by the user who can also interact with them as usual. We have also implemented a simple version of Apple's *Exposé*, which scales down the windows on the screen and tiles them to make them all visible. Like our zoomable desktop and as opposed to *Exposé*, this version lets the user interact with applications as usual. The zoomable desktop and our *Exposé*-like could probably be combined, the latter allowing users to access windows otherwise unreachable.

DISCUSSION AND DIRECTIONS FOR FUTURE WORK

We believe that a well chosen subset of the techniques described above could significantly improve window management tasks. The main problem we have when we demonstrate a Metisse-based desktop to some people is that they tend to assume that Metisse is what they see (e.g. a 3D desktop). Metisse is not a 3D desktop! It is a highly tailorable, graphics-rich, out-of-the-box replacement for existing X desktops. Which makes it a perfect tool for rapid high-fidelity prototyping of window management techniques. As we already stated above, we believe this is an important point and a great improvement over other experimental desktops because it should make it possible to conduct longitudinal studies of new window management techniques.

Metisse raises some interesting questions. As an example, when the cursor comes over a transformed window, should the same transformation be applied to the cursor itself? Conversely, when a transformation is applied on a window and the cursor is on that window, should the cursor be automatically moved to stay at the same place? Should a pop-up menu or a transient window follow the transformation of its parent window? In some cases, this seems desirable but in some others not... Our experience indicates that a distinction between windows in the user's focus and peripheral windows might help finding answers to these questions.

Metisse allows to easily add rendering artifacts to windows, like shadows. Amon other things, we plan to investigate the design of new artifacts that would help users to understand the relations between the various windows. As an example, one could show the relation between a dialog box and its parent windows by drawing translucent lines between the corners of the two windows. This, like the other window management techniques described in this paper, should be evaluated as part of a future longitudinal study.

Performance and preliminary evaluation

One of the authors uses Metisse on a regular basis. In particular, a large part of FvwmCompositor has been developed inside FvwmCompositor itself (modify the code, compile and restart FvwmCompositor!). Metisse works perfectly well for day to day activities such as e-mail and instant messaging, digital pictures and web browsing, text and image editing, as well as small-sized videos. Rotations and scaling are often used to reduce overlapping. The overview mode of the zoomable virtual desktop is used for rearranging windows. One limitation of the current implementation is that OpenGL applications cannot be hardware-accelerated in Metisse, which makes them slower than usual. The current way of dealing with this is to switch from Metisse to the native desktop for OpenGL applications (and high-resolution videos).

The computer used by the author is a two years old Laptop running Linux, with a 2 GHz Pentium IV, 768 MB of memory and a Radeon Mobility M6 graphics card with 32 MB of memory. On that machine, applications making an extensive use of the X drawing API run at up to fifty frames per second while high-resolution videos, as we explained, can't be played at their nominal frame rate. As an example, a 720x576 Divx video is displayed at only twelve frames per second. As illustrated by Figure 7, many applications can run together without problem. The limited amount of video memory sometimes causes temporary performance problems. However, iconification of a few windows is usually enough to free some memory and return to the standard performance level. Several tests with a more recent graphics card, a Nvidia GeForce with 128 MB of memory, doubled the frame rate of drawing-based applications and allowed to view the Divx video at nominal frame rate.

A preliminary version of the Metisse source code has been publicly released in June 2004. A few maintenance releases have followed. The last release has been downloaded more than 4500 times and the Metisse web site serves around 800 pages by day. E-mail messages from about eighty people were sent to the authors asking for support, giving some feedback and reporting a few bugs. Most people who contacted us were very positive and a few of them actually use Metisse as their default desktop. We are currently preparing an evaluation survey that will be distributed with the next release. We are also working on an extension of Metisse that would allow the recording of all window operations for later replay and analysis.

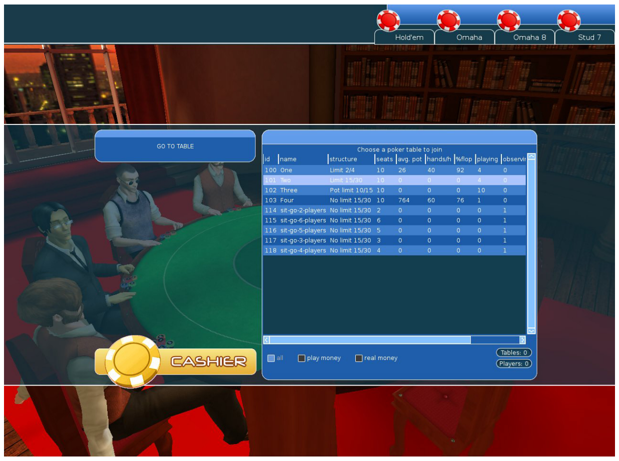
Towards a variety of compositors

The Metisse server is currently being used by a group of people from Mekensleep [7], a company developing an OpenGL-based on-line poker game. Their original motivation was to be able to integrate an external chat application in the game. Once they had implemented a basic Metisse compositor in their application, they realized that it could also be used to bring 2D interfaces built with traditional GUI toolkits such as GTK+ into their OpenGL scene (Figure 8).

This idea of using Metisse to integrate 2D interfaces in 3D environments seems very interesting to us. We hope that Metisse will be used by other researchers in similar ways. As a consequence, we are currently developing a library to facilitate the use of the Metisse server and the implementation of new compositors. FvwmCompositor has a now relatively long history. Some code clean-up will also be made, which will probably facilitate the implementation of new experimental window management techniques by other researchers.

As we said in the previous section, FvwmCompositor can

[7] `http://www.mekensleep.com/`



Choose a poker table to join

id	name	structure	seats	avg. pot	hands/h	%flop	playing	observi
100	One	Limit 2/4	10	26	40	92	4	0
101	Two	Limit 15/30	10	0	0	0	4	0
102	Three	Pot limit 10/15	10	0	0	0	10	0
103	Four	No limit 15/30	10	764	60	76	1	0
114	sit-go-2-players	No limit 15/30	2	0	0	0	0	1
115	sit-go-6-players	No limit 15/30	6	0	0	0	0	1
116	sit-go-5-players	No limit 15/30	5	0	0	0	0	1
117	sit-go-3-players	No limit 15/30	3	0	0	0	0	1
118	sit-go-4-players	No limit 15/30	4	0	0	0	0	1

Figure 8: Poker3D as a Metisse compositor: the blue windows contain GTK+ interface elements and are rendered by the Metisse server.

display multiple instances of a window. But it can do more: it can duplicate a window region (as described in [22]), create holes in a window (as suggested in [9]), create a new window from pieces of others, and embed part of a window into another one. In fact, FvwmCompositor should be seen as a *window region* compositor (details are available in [21]). The next natural step is to move to a *widget compositor*. We are currently investigating the use of accessibility APIs to get the widget tree associated to a particular window and use it to support new interaction techniques. As an example, since FvwmCompositor already handles all user input, this should make it possible to use semantic pointing [5] for window management.

CONCLUSION

In this paper, we have described Metisse, a window system created to facilitate the design, the implementation and the evaluation of innovative window management techniques. We have presented the general architecture of the system and described some of its implementation details. We have presented several examples of uses that illustrate its potential and several directions for future research.

Metisse is available from http://www.lri.fr/ chapuis/metisse/

REFERENCES

1. Mac OS X v10.2 Technologies: Quartz Extreme and Quartz 2D. Apple Technology brief, October 2002.
2. M. Beaudouin-Lafon. Novel interaction techniques for overlapping windows. In *Proceedings of UIST 2001*, pages 153–154. ACM Press, November 2001.
3. J. Beda. Avalon Graphics: 2-D, 3-D, Imaging And Composition. Presentation at the Windows Hardware Engineering Conference, May 2004.
4. B. Bell and S. Feiner. Dynamic space management for user interfaces. In *Proceedings of UIST 2000*, pages 239–248. ACM Press, 2000.
5. R. Blanch, Y. Guiard, and M. Beaudouin-Lafon. Semantic pointing: Improving target acquisition with control-display ratio adaptation. In *Proceedings of CHI 2004*, pages 519–526. ACM Press, April 2004.
6. P. Dietz and D. Leigh. DiamondTouch: a multi-user touch technology. In *Proceedings of UIST 2001*, pages 219–226, New York, NY, USA, 2001. ACM Press.
7. P. Dragicevic. Combining crossing-based and paper-based interaction paradigms for dragging and dropping between overlapping windows. In *Proceedings of UIST 2004*, pages 193–196. ACM Press, 2004.
8. J. Gettys and K. Packard. The (Re)Architecture of the X Window System. In *Proceedings of the Linux Symposium, Volume 1*, pages 227–237, July 2004.
9. D. Hutchings and J. Stasko. Revisiting Display Space Management: Understanding Current Practice to Inform Next-generation Design. In *Proceedings of GI 2004*, pages 127–134. Canadian Human-Computer Communications Society, June 2004.
10. D. Hutchings and J. Stasko. Shrinking window operations for expanding display space. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 350–353. ACM Press, 2004.
11. D. Hutchings and J. Stasko. mudibo: Multiple dialog boxes for multiple monitors. In *Extended abstracts of CHI 2005*. ACM Press, April 2005.
12. E. Kandogan and B. Shneiderman. Elastic Windows: evaluation of multi-window operations. In *Proceedings of CHI 1997*, pages 250–257. ACM Press, March 1997.
13. B. Myers. A taxonomy of window manager user interfaces. *IEEE Computer Graphics and Applications*, 8(5):65–84, sept/oct 1988.
14. B. Myers. A brief history of human-computer interaction technology. *ACM interactions*, 5(2):44–54, march/april 1998.
15. T. Richardson, Q. Stafford-Fraser, K.R. Wood, and A. Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1):33–38, Jan-Feb 1998.
16. G. Robertson, E. Horvitz, M. Czerwinski, P. Baudisch, D. Hutchings, B. Myers, D. Robbins, and G. Smith. Scalable Fabric: Flexible Task Management. In *Proc. of AVI 2004*, pages 85–89, May 2004.
17. G. Robertson, M. van Dantzich, D. Robbins, M. Czerwinski, K. Hinckley, K. Risden, D. Thiel, and V. Gorokhovsky. The Task Gallery: a 3D window manager. In *Proceedings of CHI 2000*, pages 494–501. ACM Press, April 2000.
18. N. Roussel. Ametista: a mini-toolkit for exploring new window management techniques. In *Proceedings of CLIHC 2003*, pages 117–124. ACM Press, August 2003.
19. R.W. Scheifler and J. Gettys. The X Window System. *ACM Transactions on Graphics*, 5(2):79–109, 1986.

20. D. Smith, A. Raab, D. Reed, and A. Kay. Croquet: A menagerie of new user interfaces. In *Proceedings of C5 2004*, pages 4–11. IEEE Computer Society, January 2004.

21. W. Stuerzlinger, O. Chapuis, and N. Roussel. User interface façades: towards fully adaptable user interfaces. Submitted for publication.

22. D. Tan, B. Meyers, and M. Czerwinski. Wincuts: manipulating arbitrary window regions for more effective use of screen space. In *Extended abstracts of CHI 2004*, pages 1525–1528. ACM Press, 2004.

23. M. Tomitsch. Trends and Evolution of Window Interfaces. Diploma thesis, University of Technology, Vienna, December 2003. 132 pages.

24. M. van Dantzich, G. Robertson, and V. Ghorokhovsky. Application Redirection: Hosting Windows Applications in 3D. In *Proceedings of NPIV99*, pages 87–91. ACM Press, 1999.