Scaled Window Support in DMX

Rickard E. Faith and Kevin E. Martin

15 October 2003 (created 19 September 2003)

Abstract

This document investigates the possibility of adding scaled window support to the DMX X server, thereby allowing a window or some selected part of the logical DMX area to be displayed using a scaling factor. For example, this might allow the contents of a window to be magnified for easier viewing. In particular, scaling for the VNC client is explored.

1. Introduction

1.1 DMX

The DMX X server (Xdmx) is a proxy server that is designed to allow X servers on multiple machines to be combined into a single multi-headed X server. Combined with Xinerama, these heads can appear as a single very high-resolution screen. Typical applications include the creation of a video wall with 16 1280x1024 displays arranged in a rectangle, for a total resolution of of 5120x4096.

1.2 Problem Statement

Applications displayed on a physically large video wall that provides high pixel-resolution may be difficult to see, especially if the application is designed for use on a typical desktop computer with a relatively small display located close to the human operator. The goal of this paper is to describe and discuss solutions to this problem.

The original driving problem for this work is to provide scaling for the `vncviewer` application when displayed using DMX (VNC scaling is currently available only with the Windows client, and there is no plan to extend that capability to other clients). While this specific problem will be addressed in this paper, the general solution space will also be explored, since this may lead to a good solution not only for `vncviewer` but also for other applications.

1.3 Task

For reference, here is the original description of the task this paper addresses:

- Scaled window support (for VNC)
    - Investigate possibility of implementing a "scaled window" extension:
        - Add XCreateScaledWindow call that could be used in place of XCreateWindow

    - All primitives drawn to scaled window would be scaled by appropriate (integral?) scaling factor
- Alternate approach: special case VNC support

2. Previous Work

This section reviews relevant previous work.

2.1 VNC

2.1.1 Scaling under VNC

When using the `vncviewer` program for Windows, it is possible to specify a scaling factor (as numerator and denominator). When scaling is in effect, the viewer software uses StretchBlt (instead of BitBlt) to display the pixels for the user. When this call is made, the viewer already has received all of the pixel information (at full unscaled resolution).

The scaling in VNC is primitive. It does not conserve bandwidth, it does not treat textual information differently (i.e., by using a suitably scaled font), and it does not provide any anti-aliasing other than that provided by the underlying (Windows-only) system library.

2.2 The X Video Extension

The X Video Extension is a widely-available extension to the X11 protocol that provides support for streaming video. Integral to this support is the ability to arbitrarily scale the output. In version 2.2 of the X Video specification, support for scaled still images was provided, using both shared memory and traditional transport. The API for this support uses calls that are quite similar to XCreateWindow, XPutImage, and XShmPutImage. Currently, most of the drivers implemented in XFree86 only support data in various YUV formats. However, several modern video adaptors support RGB as well.

Note, though, that the target output for this scaling is an overlay plane -- so X Video provides functionality that is fundamentally different from that provided by the Windows StrechBlt call.

3. Possible Solutions

This section briefly discusses possible solutions, including major advantages and disadvantages from both the implementation and the end-user programmer standpoint.

3.1 VNC-like Scaling

3.1.1 Software Scaling

The `vncviewer` application could be modified to provide software scaling. This is not a general solution, but it does solve one of the goals of this work.

A prototype of this solution was implemented and a patch against `vnc-3.3.7-unixsrc` is available in the `dmx/external` directory. Because of limited time available for this work, all of the edge cases were not considered and the solution works well mainly for integer scaling.

Currently, `vncviewer` writes to the X display with XPutImage, XCopyArea, and XFillRectangle. All instances of these calls have to be aware of scaling and must round correctly. In the prototype solution, rounding is incorrect and can cause artifacts.

A better solution would be to cache all updates to the desktop image in `vncviewer` and only send the damaged area to the X display with XPutImage. This would allow the damaged area to be computed so that rounding errors do not create artifacts. This method is probably similar to what is used in the Window client. (The whole VNC suite is being re-written in C++ and the forthcoming version 4 has not been evaluated.)

3.1.2 Scaling with the X Video Extension

The scaling in the Windows `vncviewer` application makes use of a scaled blit that is supplied by the underlying system library. Several video cards currently provide support for a scaled blit, and some X servers (including XFree86) expose this capability to applications via the XvPutImage interface of the X Video Extension. The capability exposed by XvPutImage results in the scaled image being drawn to an overlay plane. Most video cards also provide support for a scaled blit into the normal output planes, but this is not exposed via XvPutImage.

The `vncviewer` program could be modified to use the X Video Extension to provide scaling under X11 that is similar to the scaling currently provided under Windows. Unfortunately, Xdmx does not currently export the X Video Extension, so this would not provide an immediate solution usable with DMX.

A very early-stage proof-of-concept prototype was implemented and a preliminary patch against `vnc-3.3.7-unixsrc` is available in the `dmx/external` directory. This prototype was implemented to better understand the problems that must be solved to make this solution viable:

- As noted under the software scaling section above, `vncviewer` writes to the X display with several different calls. These calls write to the normal output planes and are compatible with XvPutImage, which writes to an overlay plane. To eliminate artifacts caused by this problem, `vncviewer` should be modified so that a cached copy of the desktop is available, either as a client-side image or a server-side off-screen pixmap, so that XvPutImage would be the only method for writing to the X display.
-

  Although several modern graphics adaptors support hardware scaling using an RGB format (e.g., ATI Radeon, nVidia, etc.), XFree86 drivers typically only implement YUV formats. YUV generally compress the pixel information in some way. For example, two commonly implemented formats, YUY2 and UYVY provide intensity information for every RGB pixel, but only provide chroma and luminance information for pairs of horizontal pixels. Since VNC uses pixel-resolution for communicating updates on the wire, additional artifacts are introduced (because there may not be enough information from the wire to update a pair of pixels).

  Further, the well-known problem with YUV encoding is even more evident when the image is a desktop instead of a movie. For example, consider a 1-pixel-wide vertical window border. If the border changes in color but not intensity (e.g., because a window manager uses color to indicate focus), there may or may not be a change in the YUY2 image, depending on the algorithm used for RGB to YUV conversion and on how the border pixel is ordered in the pair of pixels used by the algorithm.

  Many of these artifacts could be eliminated if `vncviewer` cached a complete RGB image of the desktop, and only did the conversion to YUV for properly aligned areas of damage. The remaining artifacts could be eliminated if an RGB format was used with X Video (which may require the extension of existing XFree86 drivers to support RGB).
- Most modern video cards support exactly one overlay plane that is suitable for use with X Video. Therefore, only one application can use X Video at any given time. This is a severe limitation in a desktop environment.

3.1.2.1 Implementing the X Video Extension for DMX

The user-level API for X Video is fairly simple, but the underlying support required for the full specification is large. However, since the API provides a method to query supported capabilities, a usable subset of X Video can be implemented that would support XvPutImage and little else. This would require support for the following:

- X Video Extension API calls, including the following:
  - XvQueryExtension
  - XvQueryAdaptors
  - XvQueryPortAttributes
  - XvFreeAdaptorInfo
  - XvListImageFormats
  - XvGrabPort
  - XvCreateImage
  - XvPutImage
  - XvShmCreateImage
  - XvShmPutImage
- Support for querying back-end X Video Extension capabilities.
- Support for sending the image to the back-ends. Because X Video requires sending full images, there may be a trade-off between bandwidth limitations and additional complexity to divide the image up such that is scales properly.
- Possible support for a software fall-back. For example, if all of the back-ends do not support the X Video Extension, software scaling can be implemented such that the image is sent to the back-end with XPutImage. This pathway would have poor performance.

3.1.2.2 Supporting RGB formats for the X Video Extension

Assuming an XFree86 driver already supports the X Video Extension, and assuming the target hardware supports an RGB format, then adding support for that format is relatively simple and straightforward.

3.1.3 Scaling with an XPutImageScaled Extension

Instead of (or in addition to) implementing the X Video Extension in DMX, one obvious solution would be to implement a new extension that provides access to hardware-assisted scaled blits, similar to the StretchBlt call available under Windows. This call would scale RGB images and would not use the overlay plane (unlike the X Video Extension).

This approach has many of the same advantages and disadvantages as the XCopyAreaScaled Extension, discussed in the next section. Discussion of XPutImageScaled is deferred in favor of XCopyAreaScaled for the following reasons:

- XPutImageScaled can be emulated with XCopyAreaScaled by first using XPutImage to copy the image to an off-screen pixmap, and then calling XCopyAreaScaled between that off-screen pixmap and the target drawable.
- Since XCopyAreaScaled would copy between two areas of on-screen or off-screen memory, it has additional uses and can be viewed as efficiently providing a superset of XPutImageScaled functionality.

3.1.4 Scaling with an XCopyAreaScaled Extension

As noted in the previous section, because XCopyAreaScaled provides a superset of the functionality provided by XPutImageScaled, we will consider this extension instead.

First, XCopyAreaScaled would provide for RGB scaling between pixmaps (i.e., on-screen or off-screen areas of memory that reside on the video card). Unlike the X Video Extension, which writes into an overlay plane, XCopyAreaScaled would write into the non-overlay areas of the screen. Key points to consider are as follows:

- Because different planes are involved, the two scaling operations are usually implemented in hardware differently, so an XCopyAreaScaled extension could be added in a manner that would neither conflict with nor interact with the X Video extension in any way.
- The XCopyAreaScaled extension provides new functionality that the X Video Extension does not provide. Based on anecdotal feedback, we believe that many people outside the DMX and VNC communities would be excited about this extension.
- The main drawback to this extension is that it is new and needs to be implemented at the driver level in XFree86 for each video card to be supported. At the present time, it is more likely that the X Video Extension will be implemented for a particular piece hardware because the X Video extension has multimedia uses. However, over time, we would expect the XCopyAreaScaled extension to be implemented along with the X Video extension, especially if it becomes popular.
- Another drawback is that not all modern cards provide support for a simple scaled blit operation. However, these cards usually do provide a 3D pipeline which could be used to provide this functionality in a manner that is transparent to the client application that is using the XCopyAreaScaled extension. However, this implementation pathway would make this extension somewhat more difficult to implement on certain cards.

3.1.5 Scaling with OpenGL

Another general solution to the scaling problem is to use the texture scaling found in all 3D hardware. This ability is already exposed through OpenGL and can be exploited by clients without X server modification (i.e., other than the ability to support OpenGL). An application using OpenGL would transmit the non-scaled image to the X server as a texture, and would then display a single non-transformed rect using that texture. This also works around the single overlay problem with the X Video Extension as well as the need to implement additional scaled primitive extensions.

The downside is that most OpenGL implementations require power of 2 texture sizes and this can be very wasteful of memory if, for example, the application needs to scale a 1025x1025 image, which would require a 2048x2048 texture area (even a 640x480 image would require a 1024x512 texture). Another downside is that some OpenGL implementations have a limited about of texture memory and cannot handle textures that are very large. For example, they might limit the texture size to 1024x1024.

3.2 Application-transparent Scaling for DMX

3.2.1 Back-end Scaling Without Disconnect/Reconnect

VNC does scaling on the client side (in the `vncviewer` application). Implementing a similar solution for DMX would require support in the back-end X servers and, therefore, is not a general solution.

XFree86 already implements some support for "scaling" that could be used with DMX: if, in the XF86Config file, multiple Modes are listed in the Display Subsection of the Screen Section, then pressing Ctrl-Alt-Plus and Ctrl-Alt-Minus can be used to iterate through the listed modes. The display dimensions will change to the dimensions in the Modes line, but the logical dimensions of the X server (i.e., the dimensions that Xdmx knows about) will not change.

Further, the dimensions of the XFree86 display are under software control (via the XFree86-VidModeExtension), so the Xdmx server could change the screen dimensions on a per-display basis, thereby scaling the information on part of that display.

However, this scaling appears to have limited use. For example, assume a 4 by 4 display wall consisting of 16 1280x1024 displays. If all of the back-end servers were simultaneously configured to display 640x480, the left hand corner of each display would be magnified, but the composite result would be unreadable. Magnifying one display at a time could be usable, but could have limited utility, since the result would still be no larger than a single display.

3.2.2 Back-end Scaling With Disconnect/Reconnect

Disconnect and reconnect features are not currently supported in DMX, but are scheduled to be implemented in the future. These features, combined with the XFree86-VidModeExtension Extension, would allow an application to do the following:

- Disconnect a specific back-end server (via the DMX Extension),
- reconfigure the XFree86 back-end server resolution, and
- reconnect the back-end server to DMX -- at a new origin with the new screen resolution.

For example, consider a display wall consisting of 16 1280x1024 displays with a total resolution of 5120x4096. All of the screens could be disconnected, repositioned, and reconnected each at a resolution of 640x480. The total resolution of the display wall would be 2560x1920, allowing a view of a selected area approximately one-fourth of the size of the DMX display. This change would be completely application independent (except, perhaps, for a DMX-aware window manager). When work at the increased resolution was completed, the back-end servers could be disconnected, reconfigured, and reconnected for the original 5120x4096 view.

Support for this type of scaling can be implemented in a DMX-aware X11 client assuming the DMX server support arbitrary disconnect and reconnect semantics. Because this application cannot be written before disconnect/reconnect is implemented, this solution will not be discussed further in this paper.

3.2.3 Server-side Scaling

In earlier versions of DMX, a frame buffer was maintained on the server side, and XPutImage was used to move the information from the server to the client (similar to some early VNC implementations). The use of a server-side frame buffer would allow the server to do scaling, but is not a recommended solution because of overall performance issues and server-side memory issues (i.e., the frame buffer would be very large for large display walls).

Exploration of this path is not recommended.

3.3 XCreateScaledWindow API

The implementation of X Video Extension in DMX, and the use of XvPutImage by applications requiring scaling requires significant changes in DMX Further, XvPutImage is, essentially a scaled blit, and it is only useful for applications which are already using (or can be modified to use) XPutImage. Therefore, a more general API will be discussed as another possibility.

X applications typically create windows with the XCreateWindow call. A new extension could provide an XCreateScaledWindow call that could be used in place of the XCreateWindow call and be otherwise transparent to the application. This would allow applications, even those that do not depend on XPutImage, to take advantage of window scaling. In this section we describe how the call would work, what transparency it provides, and how to solve the potential problems that transparency creates.

3.3.1 XCreateWindow

The XCreateWindow call takes width and height as parameters. An XCreateScaledWindow call could take all the same parameters, with the addition of a scaling factor.

3.3.2 XSetWindowAttributes

An X11 window has several attributes that would have to be scaled:

- Background and border pixmaps

- Border width
- Cursor

3.3.3 XGetWindowAttributes, XGetGeometry

For transparency, calls that query the window attributes should return unscaled information. This suggests that all unscaled pixmaps and window attributes should be cached.

Unfortunately, a window manager requires the scaled geometry to properly decorate the window. The X server can probably determine which client is acting as the window manager (e.g., because that client will select events that are used exclusively by the window manager). However, other Scaled Window Extension aware clients may also need to determine the scaled geometry. Therefore, at least two additional extension calls should be implemented: XGetScaledWindowAttributes and XGetScaledGeometry.

3.3.4 Popup and Child window positions

Some applications may position popup and child windows based on an unscaled notion of the main window geometry. In this case, additional modifications to the client would be required.

3.3.5 Events

Most events (e.g., for mouse motion) return information about the coordinates at which the even occurred. These coordinates would have to be modified so that unscaled values were presented to the client.

3.3.6 Implementation

There are many implementation issues, some of which are similar to the issues involved in implementing the X Video Extension for DMX. The window contents must be scaled, either by performing all operations to a frame buffer and then writing the image to the display (perhaps using hardware scaling support), or by modifying all of the various drawing operations to perform scaling. Because of the complexity involved, the frame buffer option is recommended.

4. Conclusion and Recommendations

We recommend a three phase implementation strategy, based on how an application could be written to take advantage of scaling:

1.

   The XCopyAreaScaled extension should be implemented, since this is the ideal solution for applications like VNC, and since making use of this extension will require minimal changes to applications that already use XPutImage or XCopyArea.

   The initial implementation work would include the design of the X protocol extension, writing this up in the usual format for extension documentation, implementation of the protocol transport pieces in XFree86, implementation of a software fall-back in XFree86 and DMX, one example hardware implementation for XFree86, and implementation of support for this extension in DMX.

   We suggest implementing the extension first on the ATI Radeon cards. However, since these cards do not provide a 2D scaled blit primitive, the implementation would have to make use of the 3D texture engine to emulate a scaled blit. This is recommended, since other modern graphics cards also do not provide a simple 2D scaled blit operation and an example of the more difficult implementation pathway would be helpful to others.

2.

   Until XCopyAreaScaled is widely supported, applications that require scaling will have to fall back to another scaling method. We suggest OpenGL as the first fall-back method because it is widely available and supported by DMX.

A project centered around OpenGL-based scaling would implement this scaling in VNC as an example. This work would include re-writing the `vncviewer` rendering engine to cache a master copy of the desktop image for all operations.

3.

Since OpenGL is not implemented everywhere, and may not provide hardware-assisted performance in every implementation, an application that requires scaling should also fall back to using the X Video Extension.

This project would add support for the X Video Extension to DMX and would add support to VNC to take advantage of this extension without introducing artifacts. This would require modifying the `vncviewer` rendering engine to cache a master copy of the desktop image for all operations. This project should also add support for the RGB format to at least one XFree86 driver (e.g., ATI Radeon).

The X Video Extension is one of the few popular extensions that DMX does not support. We recommend implementing the X Video Extension even if scaling is the specific goal of that work.

We do **not** recommend implementation of the XCreateScaledWindow extension because of the complexity involved. We do **not** recommend implementation of the XPutImageScaled extension because it requires the same amount of work as the XCopyAreaScaled extension, but provides less functionality. Further, server-side scaling with a large frame buffer is **not** recommended because of the performance implications.

The back-end scaling, especially with disconnect/reconnect support should be explored in the future after disconnect/reconnect is implemented, but not at the present time.

CONTENTS

$XFree86: xc/programs/Xserver/hw/dmx/doc/scaled.sgml,v 1.2 2005/02/15 01:05:26 dawes Exp $