Sun™ Elite3D Frame Lock and Buffer Swap Synchronization Installation Guide

THE NETWORK IS THE COMPUTER™

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900 USA
650 960-1300 Fax 650 969-9131

Part No. 806-0273-11
August 1999, Revision A

Send comments about this document to: docfeedback@sun.com

Please
Recycle

Regulatory Compliance Statements

Your Sun product is marked to indicate its compliance class:

- Federal Communications Commission (FCC) — USA
- Department of Communications (DOC) — Canada
- Voluntary Control Council for Interference (VCCI) — Japan

Please read the appropriate section that corresponds to the marking on your Sun product before attempting to install the product.

FCC Class A Notice

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions:

1. This device may not cause harmful interference.
2. This device must accept any interference received, including interference that may cause undesired operation.

Note: This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

Shielded Cables: Connections between the workstation and peripherals must be made using shielded cables in order to maintain compliance with FCC radio frequency emission limits. Networking connections can be made using unshielded twisted-pair (UTP) cables.

Modifications: Any modifications made to this device that are not approved by Sun Microsystems, Inc. may void the authority granted to the user by the FCC to operate this equipment.

FCC Class B Notice

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions:

1. This device may not cause harmful interference.
2. This device must accept any interference received, including interference that may cause undesired operation.

Note: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/television technician for help.

Shielded Cables: Connections between the workstation and peripherals must be made using shielded cables in order to maintain compliance with FCC radio frequency emission limits. Networking connections can be made using unshielded twisted pair (UTP) cables.

Modifications: Any modifications made to this device that are not approved by Sun Microsystems, Inc. may void the authority granted to the user by the FCC to operate this equipment.

DOC Class A Notice - Avis DOC, Classe A

This Class A digital apparatus meets all requirements of the Canadian Interference-Causing Equipment Regulations.

Cet appareil numérique de la classe A respecte toutes les exigences du Règlement sur le matériel brouilleur du Canada.

DOC Class B Notice - Avis DOC, Classe B

This Class B digital apparatus meets all requirements of the Canadian Interference-Causing Equipment Regulations.

Cet appareil numérique de la classe B respecte toutes les exigences du Règlement sur le matériel brouilleur du Canada.

VCCI 基準について

クラス A VCCI 基準について

クラス A VCCI の表示があるワークステーションおよびオプション製品は、クラス A 情報技術装置です。これらの製品には、下記の項目が該当します。

この装置は、情報処理装置等電波障害自主規制協議会 (VCCI) の基準に基づくクラス A 情報技術装置です。この装置を家庭環境で使用すると電波妨害を引き起こすことがあります。この場合には使用者が適切な対策を講ずるよう要求されることがあります。

クラス B VCCI 基準について

クラス B VCCI の表示  があるワークステーションおよびオプション製品は、クラス B 情報技術装置です。これらの製品には、下記の項目が該当します。

この装置は、情報処理装置等電波障害自主規制協議会 (VCCI) の基準に基づくクラス B 情報技術装置です。この装置は、家庭環境で使用することを目的としていますが、この装置がラジオやテレビジョン受信機に近接して使用されると、受信障害を引き起こすことがあります。取扱説明書に従って正しい取り扱いをしてください。

Contents

Figures

Tables

Preface

This document describes how to install the Frame Lock and Buffer Swap Synchronization system with your Sun Workstation™. It is intended for system administrators and programmers who want to display multiscreen applications over two or more Elite3D frame buffer monitors.

How This Book Is Organized

Chapter 1 describes the Frame Lock and Buffer Swap Synchronization product and an example on creating a multi-screen application.

Chapter 2 provides procedures for installing the software and hardware, and for configuring Elite3D graphics cards.

Appendix A provides a code example for a Buffer Swap Synchronization application.

Appendix B offers questions and answers pertaining to this product.

Appendix C provides the `afbconfig` manpage.

Sun Documentation on the Web

The docs.sun.com(sm) web site enables you to access Sun technical documentation on the Web. You can browse the docs.sun.com archive or search for a specific book title or subject at:

http://docs.sun.com

Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. You can email your comments to us at:

docfeedback@sun.com

Please include the part number of your document in the subject line of your email.

CHAPTER 1

Introduction

This chapter describes the Elite3D Frame Lock and Buffer Swap Synchronization product and provides an example of creating a multiscreen application.

Description

The Elite3D Frame Lock and Buffer Swap Synchronization system enables two or more Elite3D graphics device monitors to display multiscreen applications. This product includes the Frame Lock cable assembly and software. There are two main feature components: Frame Lock and Buffer Swap Synchronization. See Chapter 2 for installation and Elite3D device configuration information.

To order the Frame Lock cable assembly, contact:

On-Que Computers
2 Martel Way
Georgetown, Massachusetts 01833
Phone: 877-667-8383
Fax: 978-352-2828

Order number: F180-1910-01

Frame Lock

The Frame Lock synchronization feature enables vertical retracing to occur simultaneously on each Elite3D graphics subsystem in Frame Lock. The Frame Lock cable assembly is used to daisy-chain two or more Elite3D graphics subsystems. Vertical retracing eliminates flicker between multiscreen displays. You can Frame

Lock two or more Elite3D graphics subsystems across two or more computer systems. In multihead stereo mode, all displays are synchronized left and right so that a set of LCD stereo glasses will correctly show images on all displays.

Buffer Swap Synchronization

Buffer Swap synchronization enables a simultaneous swap of buffer memory contents on all Elite3D graphics subsystems to maintain image quality and to enable continuity between scenes on all applicable displays. This feature, however, is not applicable across two or more systems. The display can be a monitor or a large wall screen image displayed by a projector. See ““Creating a Multiscreen Application” on page 3” in this chapter for an example of creating a multiscreen application and Appendix A for the code used in this application example.

Note – If you Frame Lock multiple computer systems, only the Frame Lock feature will operate (that is, the Buffer Swap Synchronization feature is not available across multiple computer systems).

Supported Systems

TABLE 1-1 lists the supported systems. It also lists the supported number of Elite3D graphics cards for each, and the computer systems that support Buffer Swap synchronization.

TABLE 1-1 Supported Systems

	Maximum Number of Elite3D Devices Supported	Buffer Swap Synchronization
Sun Ultra 10 system	(Frame Lock only)	No
Sun Ultra 30 system	Two	Yes
Sun Ultra 60 system	Two	Yes
Sun Enterprise 3000 and 3500 system	Four	Yes
Sun Enterprise 4500 and 5500 system	Four	Yes
Sun Enterprise 6500 system	Eight	Yes

Creating a Multiscreen Application

The following is a programming example of how to create a Buffer Swap Synchronization (multiscreen) application. Refer to Appendix A for the code for this application example.

Main program:

- Create *n* full screen windows, one per screen.
- Create *n* rendering threads and associate one thread per screen.
- Create a master thread to synchronize rendering threads.
- Execute main window system event loop.

Master Thread run method:

- Do the following in a loop:
  - Notify all rendering threads to render a frame (possibly in response to an event)
  - Wait for all rendering threads to finish rendering
  - Notify all rendering threads to swap buffers
  - Wait for all rendering threads to finish swapping

Render Thread(s) run method:

- Create OpenGL context for this thread's window
- Make context current to this thread
- Initialize OpenGL context state
- Do the following in a loop:
  - Wait for master thread notification
  - Render image to back buffer for this screen
  - Notify master thread that this thread is done rendering
  - Wait for master thread notification
  - Swap buffers
  - Notify master thread that this thread is done swapping

CHAPTER 2

Elite3D Frame Lock and Buffer Swap Synchronization Installation

This chapter provides procedures for installing the Elite3D Frame Lock and Buffer Swap Synchronization software and hardware. It also describes how to configure an Elite3D graphics card as either master or slave devices within the Frame Lock system.

Software Requirements

- The latest OpenGL 1.1.2 patch
- The latest Elite3D patch for your release of Solaris

Installing the Software

1. **Boot each Elite3D system that will share Frame Lock.**
2. **Install the latest Elite3D patch for your release of Solaris.**
3. **Set the environment variable for** `AFB_SWAP_BUFFER_SYNC`**.**
4. **Reboot the system or systems.**

Configuring Elite3D Graphics Cards

- When rebooting your system after installing the Elite3D software, the Solaris operating system places *all* Elite3D cards in the system in master mode. Use `afbconfig` to set the slave Elite3D cards.
- There can only be *one* master Elite3D card in a system or systems, with a maximum number of seven (7) slave Elite3D cards supported.
- You must choose a resolution supported by Elite3D and assure that each Elite3D monitor in Frame Lock is set to that resolution.
- If you Frame Lock multiple systems, you must first determine which system will contain the master Elite3D card. From the system with the master Elite3D, you must remote log in to the other systems to reconfigure the Elite3D card to be slave.
- Refer to the `boot -r` man page for device location and device numbering information for how devices are numbered based on their physical location. For Frame Lock, you may select any device (that is, `afb0`, `afb1`, `afb2`,...`afb7`) to be the master Elite3D device.

1. **Select an Elite 3D card to designate as master for the Elite3D graphics cards to be in Frame Lock.**

   One solution for designating an Elite3D to be the master is to use the Elite3D that serves as the boot/console head for that system.

2. **Make sure that each Elite3D installed has the same monitor resolution as the Elite3D in master mode.**

   If the resolution on each Elite3D graphics card is not the same, you must change it to match the Elite3D card in master mode.

   To check the resolution of an Elite3D card, use the `afbconfig` command. For example:

```
% afbconfig -dev /dev/fbs/afb0 -prconf
```

   You must repeat this command for each Elite3D card in the system or systems.

   The `afbconfig` command displays the current monitor resolution setting. It also displays additional information such as whether the Elite3D is a master or slave.

   If you need to change the resolution of an Elite3D card, use the `afbconfig` command. For example:

```
% afbconfig -dev /dev/fbs/afb1 -res 1280x1024x76 now nocheck
```

3. **Connect the Frame Lock cable to each Elite3D graphics card. Make sure to first connect the top of the Frame Lock cable assembly to the master Elite3D card.**

   See the section ““Frame Lock Cable Installation” on page 7.”

4. **From the master Elite3D graphics card display window, configure the other Elite3D graphics cards in slave mode.**

   To do this, use the `afbconfig` command. For example:

```
% afbconfig -dev /dev/fbs/afb1 -slave on
```

   You must configure each card separately (that is, for `afb1`, `afb2`, `afb3`, and so on).

   In order to reconfigure your Elite3D graphics card from slave mode back to master mode, do one of the following:

   - Power cycle the system or systems which have Elite3D graphics cards installed.
   - Use the `afbconfig` command. For example:

```
% afbconfig -dev /dev/fbs/afb1 -slave off
```

   You must configure each card in slave mode separately (that is, for `afb1`, `afb2`, `afb3`, and so on).

   Your system is now ready for Frame Lock and Buffer Swap Synchronization software applications.

Frame Lock Cable Installation

The Frame Lock cable, FIGURE 2-1, is a Y-shaped cable assembly with three connectors for daisy-chaining multiple Elite3D graphics cards within a computer system.

Frame Lock Cable Assembly

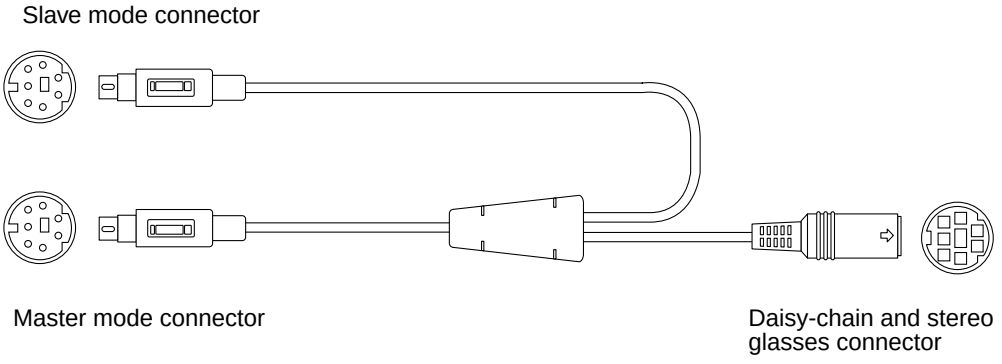

FIGURE 2-1 Frame Lock Cable Assembly

TABLE 2-1 Frame Lock Cable Connections

Frame Lock Connector	Description
Master connector (shortest cable-male)	Connects into the stereo connector located on the Elite3D graphics card that is designated as the master device. It also plugs into the daisy-chain connector for slave Elite3D cards.
Slave connector (longest cable male)	Connects into the stereo connector located on the Elite3D graphics card that is designated as a slave device.
Daisy chain / stereo glasses connector (female connector)	Can connect a pair of stereo glasses directly into this connector or the connector can be used to daisy chain to other Frame Lock cables for slave Elite3D devices.

Note – There can only be one master Elite3D graphics device. You must configure all other Elite3D devices as slaves. See the section “Configuring Elite3D Graphics Cards” on page 6.”

Elite3D Connector Pinout for Frame Lock and Buffer Swap Synchronization

FIGURE 2-2 and TABLE 2-2 shows the Elite3D stereo connector and pinout signals.

Elite3D 7-pin DIN female stereo connector

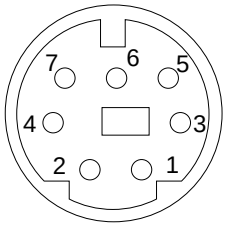

FIGURE 2-2 Elite3D Backplate Stereo Connector

TABLE 2-2 Elite3D Stereo Connector Pinout

Pin	Signal
1	DIN7_RETURN (signal ground)
2	No connect
3	3D_GLASSES_PWR +11V
4	FIELD
5	Slave FIELD_IN
6	DRAWING_L
7	No connect

Typical Cable Wiring Application

TABLE 2-3 shows a typical wiring schematic for a master Elite3D device and one or more slave Elite3D devices. You should wire the second to *n*th slave devices exactly as the “slave” column in this table.

TABLE 2-3 Wiring Schematic for Frame Lock Cable Assembly

Master Male DIN7	Slave Male DIN7	Glasses Female DIN7
FIELD, pin 4	FIELD_IN, pin 5	FIELD, pin 4
DIN7_RETURN, pin 1	DIN7_RETURN, pin 1	DIN7_RETURN, pin 1
DRAWING_L, pin 6	DRAWING_L, pin 6	DRAWING_L, pin 6 (see note)
3D_GLASSES_PWR, pin 3		3D_GLASSES_PWR, pin 3

Note – This wire should be connected so that the “Glasses” connector is used to daisy-chain multiple cables to additional slave Elite3D devices.

Connecting the Frame Lock Cable Assembly

1. **Locate the master Elite3D back panel on the rear of your system and connect the top of the Frame Lock cable assembly to the master Elite3D stereo connector.**
2. **Connect the slave cable connector to a slave Elite3D stereo connector.**
3. **Connect a second slave Elite3D, if applicable, or stereo glasses to the daisy-chain/ stereo connector.**

FIGURE 2-3 shows horizontal-type Elite3D cards, FIGURE 2-4 shows vertical-type.

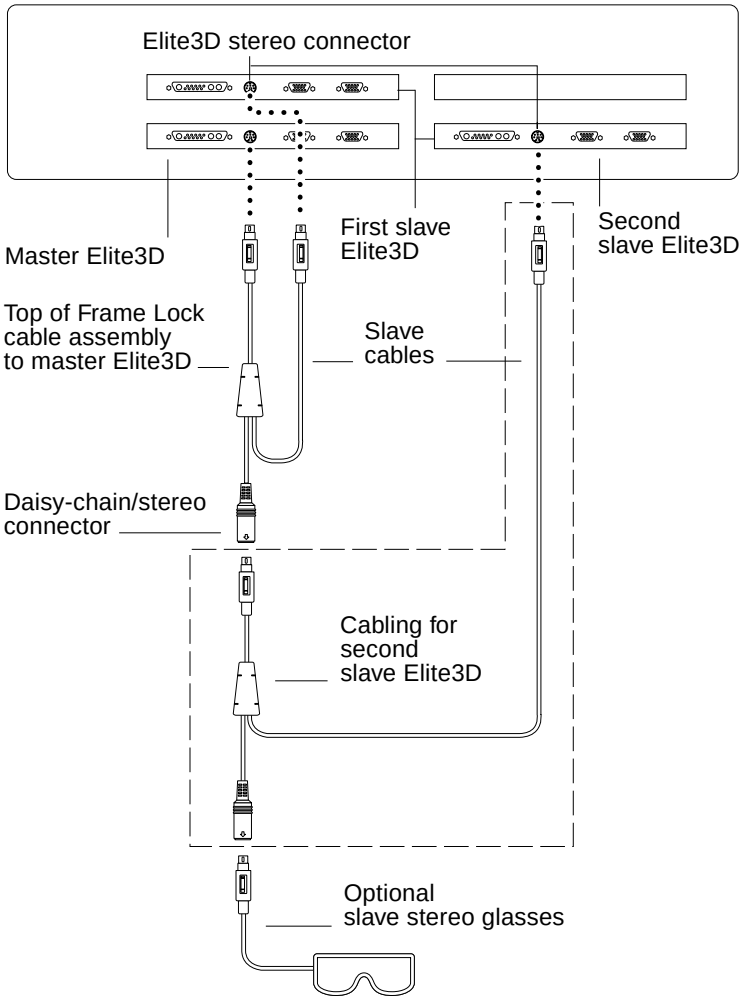

FIGURE 2-3 Horizontal Elite3D and Frame Lock Cable Assembly

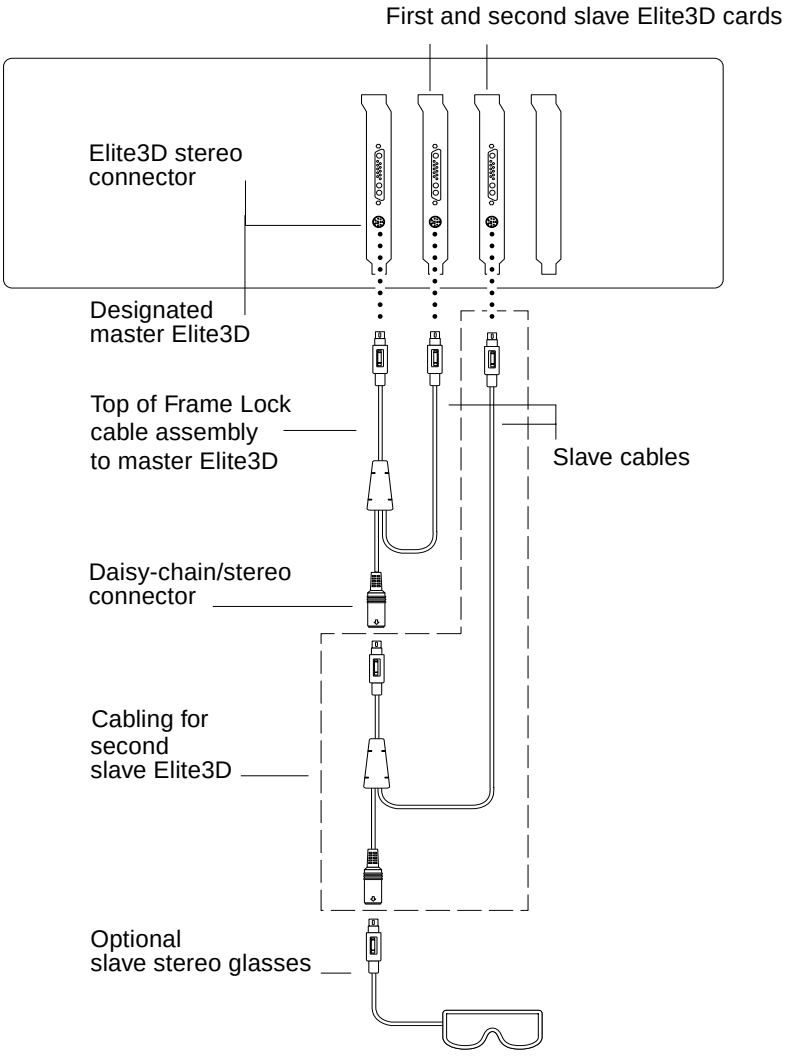

FIGURE 2-4 Vertical Elite3D and Frame Lock Cable Assembly

APPENDIX **A**

Multiscreen Application Code Example

This appendix provides the code for the multiscreen application example in “Creating a Multiscreen Application” on page 3 in Chapter 1.

```
/*************************************************************************
*
* @(#)multi_screen_MT.c 1.3 99/07/13 16:35:53
*
* Copyright (c) 1998-1999 Sun Microsystems, Inc. All Rights Reserved.
*
* Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
* modify and redistribute this software in source and binary code form,
* provided that i) this copyright notice and license appear on all copies of
* the software; and ii) Licensee does not utilize the software in a manner
* which is disparaging to Sun.
*
*
* This software is provided "AS IS," without a warranty of any kind. ALL
* EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING
* ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
* OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT
* BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING,
* MODIFYING OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT
* WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA,
* OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE
* DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING
* OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN
* ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
*
* This software is not designed or intended for use in on-line control of
* aircraft, air traffic, aircraft navigation or aircraft communications; or
```

```
 * in the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 *
 ************************************************************************
 *
 */
 /*
  * Multi-threaded, OpenGL example program that that creates a window on each
  * of multiple screens aligned in a rectangular grid. The projection matrix
  * for each screen is set up such that each screen represents one piece of a
  * larger, tessellated screen.
  *
  * This example program draws a single triangle that spans multiple screens.
  * Each screen has its own context and its own rendering thread. There is also
  * a master thread that synchronizes the rendering and buffer swapping of each
  * screen
  *
  * Usage: multi_frames_MT [-cells <numRows> <numCols>]
  *
  * Where the <numRows> and <numCols> specify the number of screens per row and
  * column in the grid of multiple screens.
  */

#include <thread.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glx.h>

#include <X11/Xlib.h>
#include <X11/Xutil.h>

/* Maximum number of screens */
#define MAXSCR 16

/*
 * The RenderThreadState structure defines all rendering state used by each
rendering thread. It
 * includes the X11 display and window state, OpenGL state, and thread
communication state.
 */
typedef struct {
    Display            *dpy;    /* X11 Display connection */
    XVisualInfo        *visInfo;/* X11 visual info */
    Window              win;    /* XID of Canvas window */
```

```
    int                 whichCtx;       /* Which context (1 thru n) */
    GLXContext          ctx;    /* GLX context ID */
    double              angle;  /* rotation angle */

    mutex_t            *masterMutex;    /* Used to notify master */
    cond_t             *masterCond;     /* Ditto */
    volatile GLboolean *masterFlag;     /* Ditto */
    mutex_t            *renderMutex;    /* Used to notify master */
    cond_t             *renderCond;     /* Ditto */
    volatile GLboolean *renderFlag;     /* Ditto */
}                   RenderThreadState;

/*
 * The MasterThreadState structure defines the thread communication state used
by the master thread
 * to control all of the rendering threads.
 */
typedef struct {
    mutex_t            *masterMutex[MAXSCR];    /* Used to notify master */
    cond_t             *masterCond[MAXSCR];     /* Ditto */
    volatile GLboolean *masterFlag[MAXSCR];     /* Ditto */
    mutex_t            *renderMutex[MAXSCR];    /* Used to notify master */
    cond_t             *renderCond[MAXSCR];     /* Ditto */
    volatile GLboolean *renderFlag[MAXSCR];     /* Ditto */
}                   MasterThreadState;

/* Global flags set by command line option */
/* static GLboolean mutexCreate = GL_TRUE;   Shouldn't need this */
static GLboolean    mutexCreate = GL_FALSE;
static int          numRows = 1;
static int          numCols = 1;

static int          numScreens = 1;
static int          screenWidth;
static int          screenHeight;

/* Mutex for serializing OpenGL context creation */
static mutex_t      glCreateMutex;

/* Main run methods for renderer threads and master thread */
void               *renderThreadRun(void *);
void               *masterThreadRun(void *);

/* Method to create a new render thread state structure */
static RenderThreadState *
newRenderThreadState(int, Display *, XVisualInfo *, Window,
    mutex_t *, cond_t *, GLboolean *,
    mutex_t *, cond_t *, GLboolean *);
```

```
/* Method to create a new master thread state structure */
static MasterThreadState *
newMasterThreadState(mutex_t *, cond_t *, GLboolean *,
    mutex_t *, cond_t *, GLboolean *);

/* Thread communication methods */
static void         post(mutex_t *, cond_t *, volatile GLboolean *);
static void         waitForPost(mutex_t *, cond_t *, volatile GLboolean *);

/* Window and context creation methods */
static Window       createWindow(Display *, int, Window, XVisualInfo *, int,
int);
static void         initContext(RenderThreadState *);

/* Render and buffer swap methods */
static void         render(RenderThreadState *);
static void         swap(RenderThreadState *);

/* X11 event loop for main thread */
static void         event_loop(Display *, Window[]);

/*
 * main
 *
 * Creates a window on each screen, creates and initializes all threads,
 * calls main event loop.
 */
main(int argc, char *argv[])
{
    GLboolean           glIsMTSafe = GL_FALSE;/* Flag indicating MT-safe OGL */
    Display            *dpy;    /* X11 display connection */
    int                 i;      /* Loop counter for screens */
    XWindowAttributes   xwa;    /* Default root window attrs */

    MasterThreadState  *masterThreadState;      /* Master state */
    thread_t            master_tid;     /* Thread ID for master thread */

    /* Per-screen information */
 RenderThreadState  *renderThreadState[MAXSCR];      /* Rendering state */
    XVisualInfo        *visInfo[MAXSCR];        /* X visual info */
    Window              root[MAXSCR];   /* Root window */
    Window              win[MAXSCR];    /* OpenGL window */
    thread_t            render_tid[MAXSCR];     /* Thread ID */
    mutex_t             masterMutex[MAXSCR];    /* Used to notify master */
    cond_t              masterCond[MAXSCR];     /* Ditto */
    GLboolean           masterFlag[MAXSCR];     /* Ditto */
    mutex_t             renderMutex[MAXSCR];    /* Used to notify master */
    cond_t              renderCond[MAXSCR];     /* Ditto */
```

```
    GLboolean           renderFlag[MAXSCR];     /* Ditto */

    static int          gl_visual_attrs[] = {
        GLX_USE_GL,
        GLX_RGBA,
        GLX_DOUBLEBUFFER,
        GLX_RED_SIZE, 2,
        GLX_GREEN_SIZE, 2,
        GLX_BLUE_SIZE, 2,

None

    };

    /* Parse command line options */
    while (--argc > 0) {
        static char  *usage = "multi_frames_MT [-cells <numRows> <numCols>]";

        ++argv;
        if (argv[0][0] == '-') {
            if (strcmp(argv[0], "-cells") == 0) {
                if (argc >= 3) {
                    numRows = atoi(argv[1]);
                    numCols = atoi(argv[2]);
                    argc -= 2;
                    argv += 2;
                } else {
                    fprintf(stderr, "%s\n", usage);
                    exit(1);
                }
            } else {
                fprintf(stderr, "%s\n", usage);
                exit(1);
 }
        } else {
            fprintf(stderr, "%s\n", usage);
            exit(1);
        }
    }
    numScreens = numRows * numCols;

    /*
     * Create display connection and initialize Xlib and OpenGL for MT rendering
     */
    XInitThreads();

    dpy = XOpenDisplay("");

#ifdef GLX_SUN_init_threads
```

```
    if (strstr(glXGetClientString(dpy, GLX_EXTENSIONS),
            "GLX_SUN_init_threads")) {

        if (glXInitThreadsSUN()) {
            glIsMTSafe = GL_TRUE;
        } else {
            fprintf(stderr, "Cannot initialize OpenGL for MT rendering\n");
        }
    }
#endif

    if (!glIsMTSafe) {
        fprintf(stderr, "OpenGL is not MT safe\n");
        exit(1);
    }

    XGetWindowAttributes(dpy, DefaultRootWindow(dpy), &xwa);
    screenWidth = xwa.width;
    screenHeight = xwa.height;
    fprintf(stderr, "screen size = %dx%d\n", screenWidth, screenHeight);

    /* Get screen and root window for each screen. Create N windows. */
    for (i = 0; i < numScreens; i++) {
        root[i] = RootWindow(dpy, i);
        visInfo[i] = glXChooseVisual(dpy, i, gl_visual_attrs);
        if (visInfo[i] && visInfo[i]->class == TrueColor) {
 fprintf(stderr,
             "Screen %d: found a %d-bit TrueColor visual (visual ID = 0x%x)\n",
              i, visInfo[i]->depth, visInfo[i]->visualid);
        } else {
            fprintf(stderr, "Screen %d: cannot find conforming visual\n", i);
            exit(1);
        }
        win[i] = createWindow(dpy, i, root[i], visInfo[i], 1, 1);
    }

    /* Initialize threads */
    mutex_init(&glCreateMutex, USYNC_THREAD, NULL);

    for (i = 0; i < numScreens; i++) {
        mutex_init(&masterMutex[i], USYNC_THREAD, NULL);
        cond_init(&masterCond[i], USYNC_THREAD, NULL);
        masterFlag[i] = GL_FALSE;
        mutex_init(&renderMutex[i], USYNC_THREAD, NULL);
        cond_init(&renderCond[i], USYNC_THREAD, NULL);
        renderFlag[i] = GL_FALSE;

        renderThreadState[i] =
            newRenderThreadState(i, dpy, visInfo[i], win[i],
```

```
            &masterMutex[i], &masterCond[i], &masterFlag[i],
            &renderMutex[i], &renderCond[i], &renderFlag[i]);

        if (thr_create(NULL, 0, renderThreadRun, renderThreadState[i],
                THR_DAEMON | THR_NEW_LWP, &render_tid[i]) != 0) {
            perror("thr_create");
        }
    }

    masterThreadState =
        newMasterThreadState(masterMutex, masterCond, masterFlag,
        renderMutex, renderCond, renderFlag);
    if (thr_create(NULL, 0, masterThreadRun, masterThreadState,
            THR_DAEMON | THR_NEW_LWP, &master_tid) != 0) {
        perror("thr_create");
    }

    /* Process window events from the main thread */
    event_loop(dpy, win);
}

/*
 * event_loop
 *
 * X11 Event processing loop. Upon the first expse event, each window is
 * resized to be full screen and positioned such that the window title bar
 * and borders are off screen. The values are hard-coded for Sun's olwm
 * window manager.
 */
static void
event_loop(Display * dpy,
    Window win[])
{
    static int          hasBeenResized[MAXSCR];
    XEvent              event;
    int                 scr;

    for (scr = 0; scr < numScreens; scr++)
        hasBeenResized[scr] = False;
    while (1) {
        XNextEvent(dpy, &event);
        switch (event.type) {
          case Expose:
            {
                XWindowChanges      xwc;
                Window              w;

                w = event.xexpose.window;
                for (scr = 0; scr < numScreens; scr++) {
```

```
                    if (w == win[scr])
                        break;
                }
                if (scr >= numScreens || hasBeenResized[scr])
                    break;

                /*
                 * Position the window such that the title bar and border are
                 * just off the screen.
                 * Resize the window so that the canvas will fill the entire
                 * screen.
 */
                xwc.x = -5;
                xwc.y = -25;
                xwc.width = screenWidth;
                xwc.height = screenHeight;
                XReconfigureWMWindow(dpy,
                    w,
                    scr,
                    CWX | CWY | CWWidth | CWHeight,
                    &xwc);
                hasBeenResized[scr] = True;
                break;
            }
          case ConfigureNotify:
            break;
          case KeyPress:
            {
                int                 i, j;
                char                text[3];

                i = XLookupString(&event, &text, sizeof(text), NULL, NULL);
                if (i == 1 && (text[0] == '\033' || text[0] == 'Q')) {
                    exit(0);
                }
                break;
            }
          case MappingNotify:
            XRefreshKeyboardMapping(&event);
            break;
        }
    }
}

/*
 * newRenderThreadState
 *
 * This method allocate and initializes a new render thread state object.
 */
```

```
static RenderThreadState *
newRenderThreadState(int whichCtx,      /* context number [0,N-1] */
    Display * dpy,              /* X11 display */
    XVisualInfo * visInfo,      /* X11 visual */
 Window win,                 /* X11 window */
    mutex_t * masterMutex,      /* Master thread mutex */
    cond_t * masterCond,        /* Master thread cond var */
    GLboolean * masterFlag,     /* Master thread "done" flag */
    mutex_t * renderMutex,      /* Render thread mutex */
    cond_t * renderCond,        /* Render thread cond var */
    GLboolean * renderFlag)     /* Render thread "done" flag */
{
    RenderThreadState  *ts =
    (RenderThreadState *) malloc(sizeof(RenderThreadState));
    char                devName[100];

    ts->dpy = dpy;
    ts->visInfo = visInfo;
    ts->win = win;

    ts->whichCtx = whichCtx;
    ts->ctx = NULL;
    ts->angle = 0.0;

    ts->masterMutex = masterMutex;
    ts->masterCond = masterCond;
    ts->masterFlag = masterFlag;
    ts->renderMutex = renderMutex;
    ts->renderCond = renderCond;
    ts->renderFlag = renderFlag;

    return ts;

}

/*
 * newMasterThreadState
 *
 * This method allocate and initializes a new render thread state object.
 */
static MasterThreadState *
newMasterThreadState(
    mutex_t * masterMutex,      /* Master thread mutex */
    cond_t * masterCond,        /* Master thread cond var */
    GLboolean * masterFlag,     /* Master "done" flag array */
    mutex_t * renderMutex,      /* Render thread mutex */
 cond_t * renderCond,        /* Render thread cond var */
    GLboolean * renderFlag)     /* Render "done" flag array */
{
```

```
    MasterThreadState  *ts =
    (MasterThreadState *) malloc(sizeof(MasterThreadState));
    int                 i;      /* Loop counter for screens */

    for (i = 0; i < numScreens; i++) {
        ts->masterMutex[i] = &masterMutex[i];
        ts->masterCond[i] = &masterCond[i];
        ts->masterFlag[i] = &masterFlag[i];
        ts->renderMutex[i] = &renderMutex[i];
        ts->renderCond[i] = &renderCond[i];
        ts->renderFlag[i] = &renderFlag[i];
    }

    return ts;
}

/*
 * post
 *
 * Notify other thread that this thread is done
 */
static void
post(mutex_t * m, cond_t * c, volatile GLboolean * flag)
{
    mutex_lock(m);
    *flag = GL_TRUE;
    cond_signal(c);
    mutex_unlock(m);
}

/*
 * waitForPost
 *
 * Wait for notification from other thread
 */
static void
waitForPost(mutex_t * m, cond_t * c, volatile GLboolean * flag)
{
    mutex_lock(m);
 while (!*flag) {
        cond_wait(c, m);
    }
    *flag = GL_FALSE;
    mutex_unlock(m);
}

/*
 * renderThreadRun
 *
```

```
 * Main run method for render threads.
 */
void               *
renderThreadRun(void *threadState)
{
    RenderThreadState  *ts = (RenderThreadState *) threadState;
    fprintf(stderr, "renderThreadRun(%d)\n", ts->whichCtx);

    initContext(ts);
    while (1) {
        /*
         * Wait for notification from master thread, perform one rendering
         * pass, and then notify the master thread that we are done.
         */
        waitForPost(ts->renderMutex, ts->renderCond, ts->renderFlag);
        render(ts);
        post(ts->masterMutex, ts->masterCond, ts->masterFlag);

        /*
         * Wait for notification from master thread, perform buffer swap,
         * and then notify the master thread that we are done.
         */
        waitForPost(ts->renderMutex, ts->renderCond, ts->renderFlag);
        swap(ts);
        post(ts->masterMutex, ts->masterCond, ts->masterFlag);
    }
}

/*
 * masterThreadRun
 *
 * Main run method for master thread
 */
void               *
masterThreadRun(void *threadState)
{
    MasterThreadState  *ts = (MasterThreadState *) threadState;
    int                 flag;
    int                 i;

    fprintf(stderr, "masterThreadRun\n");

    while (1) {
        /* Notify all renderers that rendering may begin */
        for (i = 0; i < numScreens; i++) {
            post(ts->renderMutex[i], ts->renderCond[i], ts->renderFlag[i]);
        }

        /* Wait for all renderers to finish rendering */
```

```
        for (i = 0; i < numScreens; i++) {
            waitForPost(ts->masterMutex[i],
                ts->masterCond[i],
                ts->masterFlag[i]);
        }

        /* Notify all renderers that the swap can occur */
        for (i = 0; i < numScreens; i++) {
            post(ts->renderMutex[i], ts->renderCond[i], ts->renderFlag[i]);
        }

        /* Wait for all renderers to finish buffer swap */
        for (i = 0; i < numScreens; i++) {
            waitForPost(ts->masterMutex[i],
                ts->masterCond[i],
                ts->masterFlag[i]);
        }
    }
}

/*
 * initContext
 *
 * Initialize OpenGL context and make it current to this thread
 */
static void
initContext(RenderThreadState * ts)
{
    double              aspect, xmin, xmax, ymin, ymax;
    int                 xCellPos, yCellPos;

    if (mutexCreate) {
        mutex_lock(&glCreateMutex);
    }

    ts->ctx = glXCreateContext(ts->dpy, ts->visInfo, 0, True);
    glXMakeCurrent(ts->dpy, ts->win, ts->ctx);

    glDrawBuffer(GL_BACK);

    /* Set the viewport */
    glViewport(0, 0, screenWidth, screenHeight);

    /* Compute cell position for this screen and use this to compute proj */
    aspect = (((double) screenHeight * (double) numRows) /
        ((double) screenWidth * (double) numCols));
    xCellPos = ts->whichCtx % numCols;
    yCellPos = ts->whichCtx / numCols;
    xmin = 2.0 * xCellPos / numCols - 1.0;
```

```
    xmax = 2.0 * (xCellPos + 1) / numCols - 1.0;
    ymin = 2.0 * (numRows - 1 - yCellPos) / numRows - 1.0;
    ymax = 2.0 * (numRows - yCellPos) / numRows - 1.0;
    ymin *= aspect;
    ymax *= aspect;

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(xmin, xmax, ymin, ymax, -1.0, 1.0);

    /* Initialize the FG and BG colors */
    glClearColor(0.0, 0.0, 0.4, 1.0);
    glColor3f(0.0, 1.0, 0.0);

    /*
     * fprintf(stderr,
     * "initContext(%d): xCellPos = %d, yCellPos = %d; ortho(%g, %g, %g, %g)\n",
     * ts->whichCtx, xCellPos, yCellPos, xmin, xmax, ymin, ymax);
 */

    if (mutexCreate) {
        mutex_unlock(&glCreateMutex);
    }
}

/*
 * render
 *
 * Render one frame for the current render thread.
 */
static void
render(RenderThreadState * ts)
{
    /*
     * NOTE: glXMakeCurrent is not necessary because each thread only uses
     * one context, and that context is always current to this thread.
     */
    /* glXMakeCurrent(ts->dpy, ts->win, ts->ctx); */

    /* Initialize transform */
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotated(ts->angle, 0.0, 0.0, 1.0);
    glRotated(ts->angle, 0.0, 0.0, 1.0);

    /* clear the display */
    glClear(GL_COLOR_BUFFER_BIT);

    /* Draw the triangle */
```

```
    glBegin(GL_TRIANGLES);
    glVertex3f(0.0, 0.7, 0.0);
    glVertex3f(-0.7, -0.5, 0.0);
    glVertex3f(0.7, -0.5, 0.0);
    glEnd();

    /* Update the rotation angle */
    ts->angle += 5.0;

    /* Flush and wait for the rendering to complete */
    glFinish();
}

/*
 * swap
 *
 * Swap the front & back buffers for the window associated with this thread.
 */
static void
swap(RenderThreadState * ts)
{
    glXSwapBuffers(ts->dpy, ts->win);
}

/*
 * createWindow
 *
 * Simple window creation routine. This creates a window with no border.
 */
Window
createWindow(
    Display * dpy,
    int screen,
    Window parent,
    XVisualInfo * visInfo,
    int width,
    int height)
{
    Window              window;
    XSetWindowAttributes wattrs;
    XEvent              event;
    Colormap            cmap;

    cmap = XCreateColormap(dpy, parent, visInfo->visual, AllocNone);

    /* Create the output window */
    wattrs.background_pixel = 0;
    wattrs.border_pixel = 0;
    wattrs.colormap = cmap;
```

```
    wattrs.event_mask = ExposureMask | StructureNotifyMask | KeyPressMask;
    window = XCreateWindow(dpy, parent, 0, 0, width, height, 1,
        visInfo->depth, InputOutput, visInfo->visual,
        (CWBackPixel | CWBorderPixel | CWEventMask |
 CWColormap),
        &wattrs);

    XMapWindow(dpy, window);

    return window;
}
```

APPENDIX **B**

Questions and Answers

This appendix provides answers to commonly asked questions about Frame Lock and Buffer Swap Synchronization.

Q - What is the maximum number of Elite3D graphics subsystems supported by Frame Lock and Buffer Swap Synchronization?

A - Eight (8).

Q - Which systems support Frame Lock?

A - Sun Ultra 10, Sun Ultra 30, Sun Ultra 60, and Sun Enterprise 3*x*00 through 6*x*00 systems.

Q - Which systems support buffer swap synchronization and multi-head stereo, and how many Elite3D graphics subsystems are supported in these systems?

A - (1) Sun Ultra 30 and Sun Ultra 60 systems support two (2) Elite3D graphics subsystems (2) Sun Enterprise 3000 and 3500 systems support four (4) Elite3D graphics subsystems (3) Sun Enterprise 4500 and 5500 systems support four (4) Elite3D graphics subsystems (4) Sun Enterprise 6500 system supports eight (8) Elite3D graphics subsystems

Q - In what mode does Elite3D graphics subsystem come up in after initial power on?

A - All Elite3D graphics subsystems come up in master mode at power on.

Q - Can Elite3D graphics subsystems numbering, that is, `afb0`, `afb1`,... `afb7`, be changed via software command?

A - Refer to the `boot -r` command manual page on device location and device numbering info for how devices are numbered based on their physical location.

Q - Does the `afb0` device need to be the master Elite3D?

A - No. The master/slave cable connection dictates which Elite3D graphics subsystem is master.

Q - What is the maximum number of master Elite3D cards allowed per Frame Lock system?

A - One (1).

Q - What is the maximum number of slave Elite3D cards allowed per Frame Lock system?

A - Maximum of seven (7) slave devices.

Q - Can a Creator3D graphics card be a master card?

A - This feature is not supported on Creator3D. However, the Creator3D stereo glasses output signal is identical to that used by Elite3D as a Frame Lock master.

Q - Can Elite3D graphics subsystem be a slave to an external source?

A - This feature is not supported. But it could work if the external source is a TTL or LVTTL square wave at 1/2 the vertical frequency.

Q - Do all Elite3D monitor display devices have to be at the same resolution for Frame Lock?

A - Yes. If not, than vertical retrace will not occur simultaneously on all displays.

Q - Can a slave Elite3D be changed to a master through software commands?

A - No. The Frame Lock cable connections shown in this manual determines the master Elite3D.

Q - Since all Elite3D graphics subsystems come up at power on in master mode, how do you configure an Elite3D graphics subsystem to a slave subsystem?

A - See “Configuring Elite3D Graphics Cards” on page 6 to make Elite3D graphics subsystem a slave device.

Q - What would happen if by mistake all Elite3D graphics subsystems are configured as slave devices?

A - Each of the screen displays in Frame Lock will go black.

Q - Is there another method of refreshing all Elite3D cards in Frame Lock if improper use of software commands has left Elite3D graphics subsystems in unknown states or has left all Elite3D graphics subsystems in slave mode?

A - Power cycle the applicable system or systems and restart the configuration procedure.

Q - Would a system reboot have the same effect as a power cycle on the system with regard to the Elite3D graphics systems?

A - It is difficult to say what state the system is in when either reboot or power cycle is required. Power cycle, however, may fix the problem or allow you to change master/slave configuration.

Q - When an Elite3D graphic system is configured via software to a slave, will its output video on the display monitor change?

A - No. The output video will remain the same but one will see the video display flicker a bit at the time the software command is invoked.

Q - When changing the Elite3D graphic subsystems to slave mode, does it matter if the subsystem is running any type of window system?

A - No. It will not matter what type of window system the subsystem is running at the time the subsystem is converted to a slave device.

APPENDIX **C**

`afbconfig` Manpage

This appendix provides the manpage for the `afbconfig(1M)` maintenance commands. The `afbconfig` command is used when configuring your Elite3D graphics cards as either master or slave. See “Configuring Elite3D Graphics Cards” on page 6. Refer to these pages for syntax and command option descriptions.

NAME

`afbconfig`- configure the Elite3D (AFB) Graphics Accelerator

SYNOPSIS

```
    /usr/sbin/afbconfig [ -dev device-filename ]
[ -res video-mode [ now | try ] [ noconfirm  |  nocheck
] ]
    [ -file machine | system ]
    [ -deflinear   true | false ]
    [ -defoverlay   true | false ]
    [ -linearorder   first | last ]
    [ -overlayorder   first | last ]
    [ -expvis   enable | disable ]
    [ -sov   enable | disable ]
    [ -maxwids n ]
    [ -extovl   enable | disable ]
    [ -g gamma-correction-value ]
    [ -gfile gamma-correction-file ]
    [ -slave   on | off ]
    [ -propt ] [ -prconf ] [ -defaults ]
/usr/sbin/afbconfig [ -propt ] [ -prconf ]
/usr/sbin/afbconfig [ -help ] [ -res ?  ]
```

AVAILABILITY

SUNWafbcf

DESCRIPTION

afbconfig configures the AFB Graphics Accelerator and some of the X11 window system defaults for AFB.

The first form of afbconfig shown in the synopsis above stores the specified options in the OWconfig file. These options will be used to initialize the AFB device the next time the window system is run on that device. Updating options in the OWconfig file provides persistence of these options across window system sessions and system reboots.

The second and third forms which invoke only the -prconf, -propt, -help, and -res ? options do not update the OWconfig file. Additionally, for the third form all other options are ignored.

Options may be specified for only one AFB device at a time. Specifying options for multiple AFB devices requires multiple invocations of afbconfig.

Only AFB-specific options can be specified through afbconfig. The normal window system options for specifying default depth, default visual class and so forth are still specified as device modifiers on the openwin command line (see the Xsun(1) manual page in the Openwindows Reference Manual).

The user can also specify the OWconfig file that is to be updated. By default, the machine-specific file in the /etc/openwin directory tree is updated. The -file option can be used to specify an alternate file to use. For example, the system-global OWconfig file in the /usr/openwin directory tree can be updated instead.

Both of these standard OWconfig files can only be written by root. Consequently, the afbconfig program, which is owned by the root user, always runs with setuid root permission.

OPTIONS

-dev device-filename

Specifies the AFB special file. The default is /dev/fbs/afb0.

-file machine | system-file machine | system

Specifies which OWconfig file to update. If machine, the machine-specific OWconfig file in the /etc/openwin directory tree is used. If system, the global OWconfig file in the /usr/openwin directory tree is used. If the file does not exist, it is created.

-res video-mode [now | try [noconfirm | nocheck]]

Specifies the video mode used to drive the monitor connected to the specified AFB device.

`widthxheightxrate`

where width is the screen width in pixels, height is the screen height in pixels, and rate is the vertical frequency of the screen refresh. The s suffix of 960x680x112s and 960x680x108s means that these are stereo video modes. The i suffix of 640x480x60i and 768x575x50i designates interlaced video timing. If absent, non-interlaced timing will be used. As a convenience, -res also accepts formats with '@' (at sign) in front of the refresh rate instead of x. For example: 1280x1024@76. Note, some video-modes, supported by AFB, may not be supported by the monitor. The list of video-modes supported by the AFB device and the mon itor can be obtained by running afbconfig with the -res ? option (the third form shown in the command synopsis above). A list of all possible video-modes supported on AFB is shown below.

```
1024x768x60
1024x768x70
1024x768x75
1024x768x77
1024x800x84
1152x900x66
1152x900x76
1280x800x76
1280x1024x60
1280x1024x67
1280x1024x76
960x680x112s (Stereo)
960x680x108s (Stereo)
640x480x60
640x480x60i (Interlaced)
768x575x50i (Interlaced)
```

`Symbolic names`

For convenience, some of the above video modes have symbolic names defined for them. Instead of the form width x height x rate, one of these names may be supplied as the argument to -res. The meaning of the symbolic name none is that when the window system is run the screen resolution will be the video mode that is currently programmed in the device.

Name	Corresponding Video Mode
`svga`	1024x768x60
`1152`	1152x900x76
`1280`	1280x1024x76
`stereo`	960x680x112s

Name	Corresponding Video Mode
ntsc	640x480x60i
pal	768x575x50i
none	(see text above)

The -res option also accepts additional, optional arguments immediately following the video mode specification. Any or all of these may be present.

now

If present, not only will the video mode be updated in the OWconfig file, but the AFB device will be immediately programmed to display this video mode. (This is useful for changing the video mode before starting the window system).

It is inadvisable to use this suboption with afbconfig while the configured device is being used (for example, while running the window system); unpredictable results may occur. To run afbconfig with the now suboption, first bring the window system down. If the now suboption is used within a window system session, the video mode will be changed immediately, but the width and height of the affected screen won't change until the window system is exited and reentered again. In addition, the system may not recognize changes in stereo mode. Consequently, this usage is strongly discouraged.

noconfirm

Using the -res option, the user could potentially put the system into an usable state, a state where there is no video output. This can happen if there is ambiguity in the monitor sense codes for the particular code read. To reduce the chance of this, the default behavior of afbconfig is to print a warning message to this effect and to prompt the user to find out if it is okay to continue. The noconfirm option instructs afbconfig to bypass this confirmation and to program the requested video mode anyway. This option is useful when afbconfig is being run from a shell script.

nocheck

If present, the normal error checking based on the monitor sense code (described above) will be suspended. The video mode specified by the user will be accepted regardless of whether it is appropriate for the currently attached monitor. (This option is useful if a different monitor is to be connected to the AFB device). Use of this option implies noconfirm well.

try

If present, the specified video mode will be programmed on a trial basis. The user will be asked to confirm the video mode by typing 'y' within 10 seconds. Or the user may terminate the trial before 10 seconds are up by typing any character. Any character other than 'y' or carriage return is considered a no and the previous video mode will be restored and afbconfig will not change the video mode in the OWconfig file (other options specified will still take effect). If a carriage return is

typed, the user is prompted for a yes or no answer on whether to keep the new video mode. This option implies the now suboption (see the warning note on the now suboption).

AFB possesses two types of visuals: linear and nonlinear. Linear visuals are gamma corrected and nonlinear visuals are not. There are two visuals that have both linear and nonlinear versions: 24-bit Truecolor and 8- bit StaticGray. If true, the default visual is set to the linear visual that satisfies other specified default visual selection options (specifically, the Xsun(1) defdepth and defclass options described in the *OpenWindows Reference Manual*). If false, or if there is no linear visual that satisfies the other default visual selection options, the non-linear visual specified by these other options will be chosen to be the default. This option cannot be used when the -defoverlay option is present, because AFB doesn't possess a linear overlay visual.

```
-defoverlay true | false
```

The AFB provides an 8-bit PseudoColor visual whose pixels are disjoint from the rest of the AFB visuals. This is called the overlay visual. Windows created in this visual will not damage windows created in other visuals. The converse, however, is not true. Windows created in other visuals will damage overlay windows. The number of colors available to the windows created using this visual depends on the settings for the extovl option. If the extovl is enabled, extended overlay with 256 opaque color values is available. (refer to the -extovl option). If extovl is disabled, extended overlay is not available and this visual has (256 - maxwids) number of opaque color values (refer to the -maxwids option). If the value of this option is true, the overlay visual will be made the default visual. If false, the nonoverlay visual that satisfies the other default visual selection options, such as defdepth and defclass, will be chosen as the default visual. See the Xsun(1) manual page in the *OpenWindows Reference Manual*.

Whenever -defoverlay true is used, the default depth and class chosen on the openwin command line must be 8-bit PseudoColor. If not, a warning message will be printed and the -defoverlay option will be treated as false. This option cannot be used when the -deflinear option is present, because AFB doesn't possess a linear overlay visual.

```
-linearorder first | last
```

If true, linear visuals will come before their nonlinear counterparts on the X11 screen visual list for the AFB screen. If false, the nonlinear visuals will come before the linear ones.

```
-overlayorder first | last
```

If true, the depth 8 PseudoColor Overlay visual will come before the non-overlay visual on the X11 screen visual list for the AFB screen. If false, the nonoverlay visual will come before the overlay one.

```
-expvis enable | disable
```

If enabled, Openly Visual Expansion will be activated. Multiple instances of selected visual groups (8-bit PseudoColor, 24-bit TrueColor... etc) can be found in the screen visual list.

```
-sov enable | disable
```

If enabled, the root window's SERVER_OVERLAY_VISUALS property will be advertised. SOV visuals will be exported and their transparent types, values and layers can be retrieved through this property. If disabled, the SERVER_OVERLAY_VISUALS property will not be defined. SOV visuals will not be exported.

```
-maxwids n
```

This option is available only if extovl is disabled. It specifies the maximum number of AFB X channel pixel values that are reserved for use as window IDs (WIDs). The remainder of the pixel values in overlay colormaps are used for normal X11 opaque color pixels. The reserved WIDs are allocated on a first-come first-serve basis by 3D graphics windows (such as XGL), MBX windows, and windows that have a non-default visual. The X channel codes 0 to (255 - n) will be opaque color pixels. The X channel codes (255 - n + 1) to 255 will be reserved for use as WIDs. Legal values: 1, 2, 4, 8, 16, 32, 64.

```
-extovl enable | disable
```

If enabled, extended overlay is available. The overlay visuals will have 256 opaque colors. The SOV visuals will have 255 opaque colors and 1 transparent color. Also, this option enables hardware supported transparency, thus provides better performance for windows using the SOV visuals.

```
-g gamma-correction value
```

This option allows changing the gamma correction value. All linear visuals provide gamma correction. By default the gamma correction value is 2.22. Any value less than zero is illegal. This option can be used while the window system is running. Changing the gamma correction value will affect all the windows being displayed using the linear visuals.

```
-gfile gamma-correction file
```

This option loads gamma correction table from the specified file. This file should be formatted to provide the gamma correction values for R, G and B channels on each line. Each of these values should be in hexadecimal format and separated from each other by at least 1 space. Also this file should provide 256 such triplets. An example of this file is as follows.

```
0x00 0x00 0x00
0x01 0x01 0x01
0x02 0x02 0x02
...
...
0xff 0xff 0xff
```

Using this option, the gamma correction table can be loaded while the window system is running. The new gamma correction will affect all the windows being displayed using the linear visuals. Note, when gamma correction is being done using user specified table, the gamma correction value is undefined. By default, the window system assumes a gamma correction value of 2.22 and loads the gamma table it creates corresponding to this value.

`-slave on | off`

This option is for the Frame Lock Configuration. Frame Lock Configuration is supported only on AFB part numbers X3665 and above. If set to on, the AFB hardware behaves as a slave in the Frame Lock Configuration. If set to off, the AFB hardware is the master in a Frame Lock Configuration. By default, AFB hardware is set as the master. The Frame Lock Configuration is not stored across system reboots.

`-defaults`

Resets all option values to their default values.

`-propt`

Prints the current values of all AFB options in the OWconfig file specified by the -file option for the device specified by the -dev option. Prints the values of options as they will be in the OWconfig file after the call to afbconfig completes. This is a typical display:

```
--- OpenWindows Configuration for /dev/fbs/afb0 ---
OWconfig: machine
Video Mode: 1280x1024x76
Default Visual: Non-Linear Normal Visual
Visual Ordering: Linear Visuals are last
Overlay Visuals are last
OpenGL Visual Expansion: enabled
Server Overlay Visuals: enabled
Extended Overlay: enabled
Underlay WIDs: 64 (not configurable)
Overlay WIDs: 4 (not configurable)
Gamma Correction Value: 2.220000
Gamma Correction Table: Available
```

-prconf

Prints the AFB hardware configuration. This is a typical display:

```
--- Hardware Configuration for /dev/fbs/afb0 ---
Type: double-buffered AFB with Z-buffer
Board: rev 0 (Horizontal)
Number of Floats: 6
PROM Information: @(#)afb.fth x.xx xx/xx/xx
AFB ID: 0x101df06d
DAC: Brooktree 9070, version 1 (Pac2)
3DRAM: Mitsubishi 130a, version x
Framelock Configuration: Master
EDID Data: Available - EDID version 1 revision x
Monitor Sense ID: 4 (Sun 37x29cm RGB color monitor)
Monitor possible resolutions: 1024x768x77, 1024x800x84,
1152x900x66,
1152x900x76, 1280x1024x67, 1280x1024x76, 960x680x112s, 960x680
x108s
Current resolution setting: 1280x1024x76
```

-help

Prints a list of the afbconfig command line options, along with a brief explanation of each.

DEFAULTS

For a given invocation of afbconfig command line if an option does not appear on the command line, the corresponding OWconfig option is not updated; it retains its previous value.

When the window system is run, if an AFB option has never been specified via afbconfig, a default value is used. The option defaults are as follows:

Option	Default
-dev	/dev/fbs/afb0
-file	machine
-res	none
-deflinear	false
-defoverlay	false
-linearorder	last
-overlayorder	last
-expvis	enabled
-sov	enabled
- maxwids	32
-extovl	enabled
-g	2.22
-slave	off

The default for the -res option of none means that when the window system is run the screen resolution will be the video mode that is currently programmed in the device.

Note – This provides compatibility for users who are used to specifying the device resolution through the PROM. On some devices (e.g. GX) this is the only way of specifying the video mode. This means that the PROM ultimately determines the default AFB video mode.

EXAMPLES

The following example switches the monitor type to the resolution of 1280 x 1024 at 76 Hz:

```
example% /usr/sbin/afbconfig -res 1280x1024x76
```

FILES

```
/dev/fbs/afb0            device special file
```

SEE ALSO

`mmap(2), fbio(7I), afb(7D)`