Indigo^2 IMPACT Video™ Programmer's Guide

Indigo² IMPACT Video™ Programmer's Guide
Document Number 007-3182-001

Contents

List of Figures

List of Tables

List of Examples

About This Guide

Indigo^2 IMPACT Video™ is a video option that provides a Silicon Graphics$^®$ Indigo^2 IMPACT™ workstation with the ability to input and output graphic and video images and record them to disk or videotape.

Indigo^2 IMPACT fully utilizes all calls and controls in the Silicon Graphics Digital Media library, such as the Video Library, as well as controls that are native to Indigo^2 IMPACT Video only.

This guide explains features of the Video Library™ (VL) and gives step-by-step instructions for creating VL programs that make use of the Indigo^2 IMPACT Video capabilities.

Audience

This guide is written for the sophisticated video user with a background in C programming who wishes to develop video programs for Indigo^2 IMPACT Video capabilities.

Structure of This Document

This guide contains the following chapters and appendixes:

*   Chapter 1, "Features of Indigo2 IMPACT Video," introduces features and capabilities of the Indigo^2 Video board for Indigo^2 IMPACT Video. It explains VL features and architecture, and presents the VL programming model.

*   Chapter 2, "Creating Video Programs With the Video Library," explains how to open a connection to the video daemon and set up a data path, how to set data transfer parameters, how to display video data onscreen, how to transfer video data, and how to end data transfer by

presenting an annotated sample program that displays live video input in a graphics window.

- Chapter 3, "Using VL Controls," explains VL control type and values, VL control fraction ranges, VL control classes, and VL control groupings.

- Chapter 4, "Event Handling," presents the VL events for Indigo^2 IMPACT Video and details querying VL events, creating a VL event loop, and creating a main loop with callbacks.

- Chapter 5, "Managing Connections," explains how to set up more complex paths in Indigo^2 IMPACT Video programs by specifying connectivity and avoiding dynamic switching problems. It explains connectivity for Indigo^2 IMPACT Video by presenting details of board and software architecture.

- Chapter 6, "Video Real-Time Capture and Playback," gives guidelines for optimizing capture or playback to system memory or disk.

- Chapter 7, "Blending, Keying, and Transitions," explains how to use VL to perform chroma keying, luma keying, alpha keying, and transitions. It explains the blend node, keying, the keyer, and blending controls for Indigo^2 IMPACT Video.

- Appendix A, "Video Basics," introduces basic video concepts.

- Appendix B, "Return Codes," lists and explains VL return messages for Indigo^2 IMPACT Video.

- Appendix C, "Indigo2 IMPACT Video Nodes and Their Controls," gives information on the Indigo^2 IMPACT Video nodes and their controls.

A glossary and an index complete this guide.

Conventions

These type conventions and symbols are used in this guide:

Helvetica Bold   Hardware labels

Italics   Executable names, filenames, IRIX commands, manual or book titles, new terms, program variables, tools, utilities, variable command line arguments, variable coordinates, and variables to be supplied by the user in examples, code, and syntax statements

Bold   Function name

`Fixed-width type`
    Error messages, prompts, and onscreen text

`Bold fixed-width type`
    User input, including keyboard keys (printing and nonprinting); literals supplied by the user in examples, code, and syntax statements (*see also* <>)

""   (Double quotation marks) Onscreen menu items and references in text to document section titles

[]   (Brackets) Surrounding optional syntax statement arguments

<>   (Angle brackets) Surrounding nonprinting keyboard keys, for example, **`<Esc>`**, **`<Ctrl-D>`**

Features of Indigo^2 IMPACT Video

The Indigo^2 IMPACT Video option board and the Video Library provide video input and output for Indigo^2 workstations equipped with IMPACT graphics.

This chapter introduces

- Indigo^2 IMPACT Video board capabilities
- Video Library capabilities
- the VL system software architecture
- the VL architectural model of video devices
- Indigo^2 IMPACT video formats

For an introduction to video, see Appendix A, "Video Basics."

Indigo^2 IMPACT Video Board Capabilities

Building on its broadcast-quality, 10-bit digital video architecture, Indigo^2 IMPACT Video provides a solid foundation for unlimited applications. You can use Indigo^2 IMPACT graphics with real-time video and keyed or alpha output of the 32-bit, double-buffered graphics for broadcast applications. Post-production professionals can use Indigo^2 IMPACT Video to capture and play back uncompressed 10-bit video to and from main system memory.

You can send and receive live component video from any serial CCIR-601/SMPTE-259M-compliant device. Compatible with 525-line (NTSC) and 625-line (PAL) standards, Indigo^2 IMPACT Video accommodates all major formats of serial digital video I/O:

- two channels of YUV 4:2:2 (8- or 10-bit) (single-link)
- one channel of YUVA 4:2:2:4 (8- or 10-bit) (dual-link)

- one channel of YUVA 4:4:4:4 (8- or 10-bit) (dual-link)

- one channel RP175 (RGBA 8- or 10-bit) (dual-link)

- two channels of arbitrary 8-bit data (single-link)

Additional connections provide genlock input, genlock loopthrough, and GPI trigger signals. You can use the genlock to lock output to analog house sync or to either digital input. For conversion to or from component or composite analog video, use third-party digital-to-analog and analog-to-digital solutions.

Indigo^2 IMPACT Video's serial digital I/O produces valid output only when it operates in CCIR non-square pixel mode. The I/O ports are not usable when the device operates in square pixel mode, although the remainder of the device is usable. The non-square pixel modes are used with the Indigo^2 IMPACT Compression™ option card to capture digital video signals to disk in real time. Serial digital video I/O coupled with low compression ratios dramatically reduces storage and network bandwidth requirements, thus facilitating demanding applications such as spot playback and nonlinear video editing.

For input, you can use, in any combination

- graphics screen

- Indigo^2 IMPACT Compression

- digital video

- main memory

The real-time 8-bit alpha blender and key generator enable live creation of many fundamental video effects, including overlays, dissolves, fades, wipes, chroma and luma keying, and shadow.

Video Library Capabilities

The Video Library provides a software interface to the Indigo^2 IMPACT Video board, enabling applications to

- display live video in a window

- capture live video to system memory

- encode graphics to video in real time

- produce high-quality full-rate video output

The Video Library (VL) is a collection of device-independent and device-dependent C language calls for Silicon Graphics workstations equipped with video options. The VL provides generic video tools, including simple tools for importing and exporting digital data to and from Silicon Graphics systems, as well as to and from third-party video devices that adhere to the Silicon Graphics architectural model for video devices. Video tools are described in the *Media Control Panels User's Guide*, which you can view using the IRIS InSight™ viewer; similar applications are supplied in source-code form as examples in the *4Dgifts* directory (*/usr/people/4Dgifts/impact/examples/dmedia/video/vl* and */usr/people/4Dgifts/OpenGL*).

The VL works with other Silicon Graphics libraries, such as the OpenGL® and IRIS Graphics Library™ (GL). The VL does not depend on the X Window System™, but you can use X Window System libraries or toolkits to create a windowing interface.

The VL allows programs to get events 60 times per second on a quiescent system; it also enables programs to share resources or to gain exclusive use of resources. It supports input and output of video data to or from locked-down memory at the nominal frame rate. The VL provides an API that enables applications to

- capture or play back video from system memory

- blend computer graphics with frames from videotape or any video source

- present video in a window on the workstation screen

The software for the Indigo^2 IMPACT video board includes a graphical user interface, */usr/sbin/vcp*, that makes it convenient to access VL capabilities.

VL System Software Architecture

This section describes features of these VL system components and tools:

- video daemon
- generic video tools
- library and header files

Figure 1-1 diagrams the interaction between the VL, the video daemon, the kernel, the hardware, and the X Window System server.
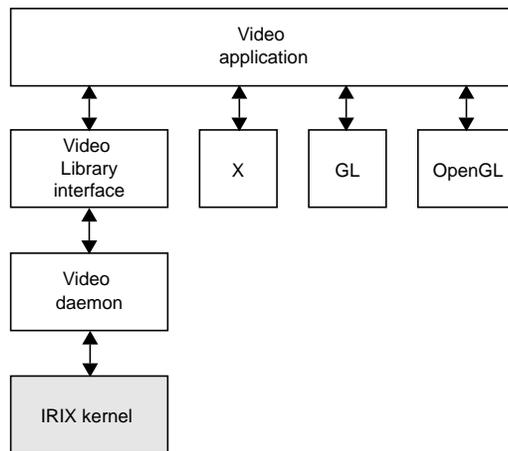
Figure 1-1   VL System Components

The VL communicates with the IRIX kernel for device initialization, vertical retrace, setup, and maintenance of any device-supported direct memory access (DMA).

Besides these components, the VL includes a collection of applications that support device configuration and control setting and retrieval, generic tools that display video on a workstation, and video control panels.

Video Daemon

The video daemon */usr/etc/videod,* which has device-dependent and device-independent portions, handles video device management and status information.

Device Management

Management that the video daemon performs includes

- multiple client access to multiple devices

  The library supports connections from multiple client applications and manages their access to a limited number of video devices.

- dispatching events

  As events are handled and noted by devices, the daemon notifies applications that have expressed interest in those events.

- handling events

  As events are generated by the various devices, the daemon initiates any action required by an event before it hands the event off to interested applications.

- maintaining exclusive use

  Types of data or control usage for video clients in a Video Library application are *Done Using, Read-only, Lock,* and *Shared.* These usage levels apply only to write access on controls, not read access. Any application can open and read the control's values at any time.

- client cleanup on exit

  When a client exits or is terminated abnormally, its connection to the daemon is broken; the daemon performs any cleanup required of the system. Any exclusive-use modes that have been set are cleared; interested clients are notified that the device is no longer in exclusive use. Controls set by the client might persist, but are not guaranteed to remain after the client closes the connection.

Status Information

Status information for which the video daemon is responsible includes

- system status of video devices

  The video devices installed in a system can be queried as to availability and control status.

- video positioning (offset) information

- control setting and retrieval

  Device-independent and device-dependent controls are set and retrieved through the video daemon.

Generic Video Tools

The generic video tools include:

videopanel (vcp)   Use this graphical user interface to set controls, such as hue or contrast, on devices. The panel resizes itself dynamically to reflect available video devices.

vlcmd   Use the Video Library command-line interface to enter Video Library shell-level and other commands.

videoin   Use the video input window tool to view input video in a window.

videoout   Use the video output tool to output video from a rectangular area of the screen on hardware that supports the screen-to-video path.

vlinfo   Use the video info tool to display information about video devices available through the VL, such as the name of the X server, number of devices on the server, and the types and ID numbers of nodes, sources, and drains on each device.

vintovout   Use this tool to display video input on the device attached to video output.

vidtomem   Use this tool to capture a single frame (the current video input) or a specified number of frames, depending on the hardware limits for burst capture, and write the data to disk. Capture size can also be specified. The data, which can be translated or left as raw data, can be used by the *memtovid* tool.

memtovid   Use this tool to output frames (images) to video out on hardware that supports the memory-to-video path.

The *vlinfo*, *vidtomem*, and *memtovid* tools are command-line tools. In addition to their reference pages, these tools have explanations in the *Media Control Panels User's Guide*, which you can view using the IRIS InSight viewer. Similar applications are supplied in source-code form as examples in the *4Dgifts* directory (*/usr/people/4Dgifts/impact/examples/dmedia/video/vl* and */usr/people/4Dgifts/examples/OpenGL*).

Library and Header Files

The client library is */usr/lib/libvl.so*. The header files for the VL are in */usr/include/dmedia*. The header file for the VL, *vl.h*, contains the main definition of the VL API and controls. The header files for Indigo^2 IMPACT Video are */usr/include/dmedia/dev_mgv.h* (linked to */usr/include/vl/vl_mgv.h*) and */usr/include/dmedia/dev_impact.h* (linked to */usr/include/vl/vl_impact.h*).

The header file */usr/include/dmedia/vl_impact.h* (linked to */usr/include/vl/dev_impact.h*) contains definitions common to the Indigo^2 IMPACT Video and Indigo^2 IMPACT Compression devices.

VL Architectural Model of Video Devices

The VL recognizes these classes of objects:

- *devices*, each including sets of nodes

- *nodes*: sources, drains, and internal nodes

- *paths*, connecting sources and drains

- *ports*, the entities on nodes that produce or consume video data

- *controls*, or parameters, that modify how data flows through nodes; for example:
    - video device parameters, such as blanking width, gamma value, horizontal phase, sync source
    - video data capture parameters
    - blending parameters
- *buffers*, for sending frame data to and receiving frame data from host memory; the VL buffers contain a number of blocks; each with a pointer, a size, and pointers to the head (oldest) and tail (newest) valid data

Central concepts for VL are *path*, *node*, and *port*.

Path

The path is an abstraction for a way of moving data around. A path is a set of nodes with video routes (connections) between the ports on the nodes.

Node

The node is an endpoint or internal processing element of the path, such as a video *source* like a VTR, video *drain* (such as to the Indigo^2 IMPACT screen), a *device* (video), or the *blender* in which video sources are combined for output to a drain.

A path defines the useful connections between video sources and video drains. Figure 1-2 shows a simple path in which a frame from a videotape is displayed in a workstation window.
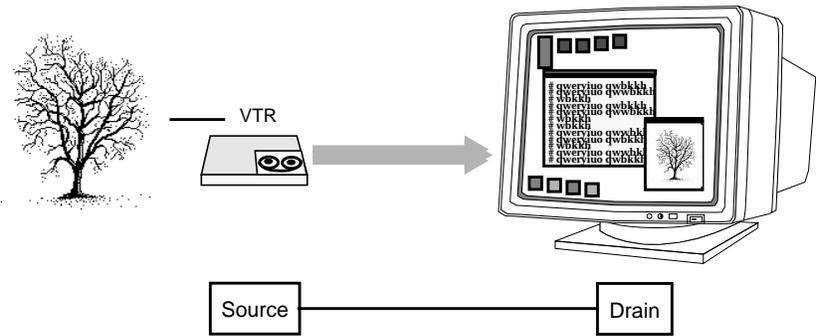
Figure 1-2   Simple VL Path

Figure 1-3 shows a more complex path with two video sources: a frame from a videotape and a computer-generated image are blended and output to a workstation window. This path is set up in stages.
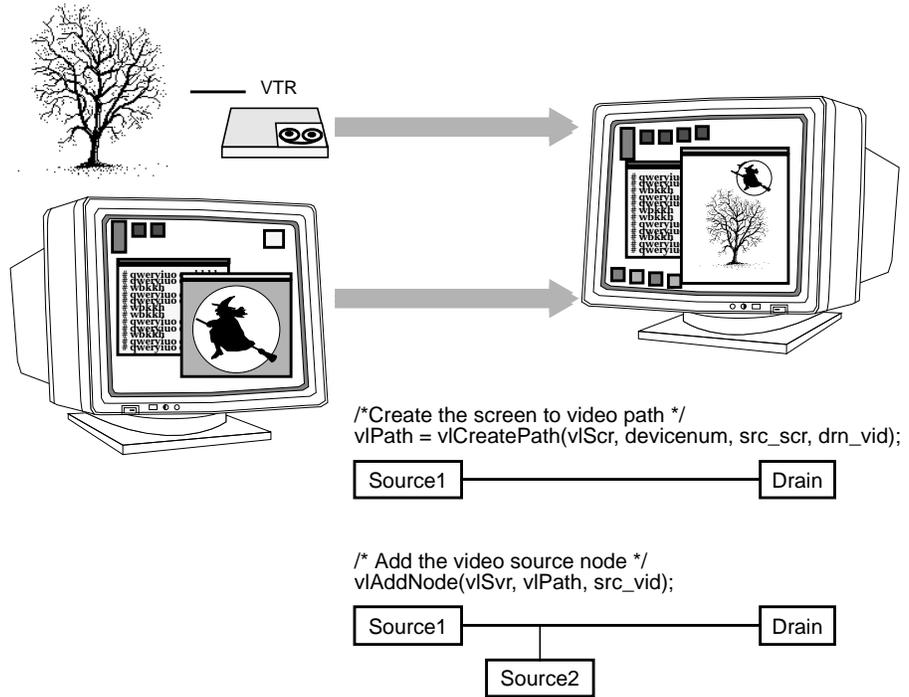
Figure 1-3   Simple VL Blending

Port

The port is an entity on a node that produces or consumes video data.

Most nodes have only one port, such as the video in or video out nodes. Each internal node has at least two ports, input (drain) and output (source). The blend node has several ports (A alpha in, A pixel in, B alpha in, B pixel in, pixel out, alpha out).

Ports have several attributes:

- *link type*: single-link or dual-link

- *data type*: alpha, pixel, or pixel-alpha (dual-link)

A device can use this attribute internally to handle data conversions or routing. For example, the Indigo^2 IMPACT Video board includes an alpha LUT to convert CCIR-range pixel data to full-range alpha values.

- *direction*: source or drain

- *enumerator*: A, B, C, and so on, used if a path has several ports with the same link type, data type, and direction

Ports produce or consume various types of data: pixel, alpha, or dual-link data. The identification of the port as pixel or alpha may cause the video stream to be treated differently. For example, alpha data, which can be supplied to Indigo^2 IMPACT video in the CCIR range only, is internally expanded to full range before it is used. No range expansion is performed for pixel data. Dual-link channels carry both alpha and pixel data, although one data type may be ignored depending on the format.

Ports have generic names; for example:

- VL_IMPACT_PORT_PIXEL_SRC_A: source of a pixel stream (first, or only, port instance)

- VL_IMPACT_PORT_ALPHA_DRN_B: drain of an alpha stream (second port instance)

For the symbolic names for ports, see */usr/include/dmedia/dev_impact.h*. Appendix C, "Indigo2 IMPACT Video Nodes and Their Controls," in this guide gives the ports associated with each node.

Connections

The connections between ports on nodes determine the topology of a path.

Single-link ports can be connected to single-link ports only; dual-link ports can be connected to double-link ports only.

Data flows from a source port to a drain port. It is not permissible to connect a source port to another source port, or a drain port to another drain port.

Connections obey stream-usage levels set with **vlSetupPaths()**. Usage is drain-centric: the usage levels of the path(s) using the drain node serve as the usage level of the connection.

The functions **vlSetConnection()** and **vlGetConnection()** manipulate connections:

- **vlSetConnection()** sets a connection between a source pair (node, port) pair and a drain pair (node, port).

- **vlGetConnection()** returns the set of connections entering or leaving a node or port.

Chapter 5, "Managing Connections," provides more information about specifying connections.

Indigo² IMPACT Video Formats

The Indigo² IMPACT Video board translates video signals into a form usable by the Indigo² workstation. It also does the reverse, translating graphics from the Indigo or Indigo² display into video signals. This section describes the Indigo² IMPACT video board's I/O interface.

Table 1-1 summarizes the formats that the Indigo² IMPACT Video board supports.

Table 1-1   Video Formats for Indigo² IMPACT Video

Format	Signal	Nodes
Digital component YCrCbA serial (VL_FORMAT_DIGITAL_COMPONENT_SERIAL)	YCrCb 4:2:2 serial digital signal with 8- or 10-bit words. Component ranges are 16 to 235 (8-bit) or 64 to 940 (10-bit).	All memory nodes
	This format is also used to specify YCrCbA 4:4:4:4. Two streams are required to carry this format. The first is 4:2:2 YCrCb (u0, y0, v0, y1, u2, y2...). The second is 4:2:2 ACrCb (u1, a0, v1, a1, u3, a2...).	
	Conforms to the CCIR-601/656 specification.	
SMPTE YUV (VL_FORMAT_SMPTE_YUV)	Contains YUV components in the range 1-254; superblack and superwhite values can be present.	All memory nodes

Table 1-1 (continued)   Video Formats for Indigo^2 IMPACT Video

Format	Signal	Nodes
Digital component RGB serial (VL_FORMAT_DIGITAL_COMPONENT_RGB_SERIAL)	Dual-link RGBA signal with GBR 4:2:2 (b0, g0, r0, g1, b2, g2, r2...) on the first link and ABR 4:2:2 (b1, a0, r1, a1, b3, a2, r3...) on the second link. Component ranges are 16 to 235 (8-bit) or 64-940 (10-bit). Conforms to the RP175 specification.	All VGI1 memory nodes
RGB (VL_FORMAT_RGB)	Full-range 8-bit or 10-bit per component RGBA. Component range is 0 to 255 (8-bit) and 0-1023 (10-bit).	All VGI1 memory nodes
Raw data (VL_FORMAT_RAW_DATA)	Used for encoding arbitrary 8-bit data values (0 to 255 range) in a 10-bit video signal. Within the coded 10-bit word, bit 9 is 0, bit 8 is 1, and bits 7 through 0 carry the 8-bit data value. When this format is used, the packing is irrelevant.	All VGI1 single-link nodes

For information on videotape formats, see "Videotape Formats" in Appendix A in this guide. For information on the requirements for recording to and from video, see the *Indigo^2 IMPACT Video Owner's Guide* (007-3184-001).

Creating Video Programs With the Video Library

Video Library (VL) calls let you perform video teleconferencing, blend computer-generated graphics with frames from videotape or any video source, and output the input video source to the graphics monitor, to a video device such as a VCR, or both.

This chapter explains the basics of creating video programs for Indigo^2 IMPACT Video:

- the VL programming model
- performing preliminary steps
- opening a connection to the video daemon
- specifying nodes
- creating and setting up a data path
- setting parameters for data transfer to or from memory
- displaying video data onscreen
- transferring video data to and from devices
- ending data transfer
- descriptions of some example programs

The VL Programming Model

Syntax elements are as follows:

- VL types and constants begin with uppercase VL; for example, VLServer
- VL functions begin with lowercase vl; for example, **vlOpenVideo()**

Data transfers fall into two categories:

- transfers involving memory (video to memory, memory to video), which require setting up a VL buffer

- transfers not involving memory (such as video to screen and graphics to video), which do not require a VL buffer

For the two categories of data transfer, based on the VL programming model, the process of creating a VL application consists of these steps:

1. Open a connection to the video daemon (**vlOpenVideo()**); if necessary, determine which device the application will use (**vlGetDevice()**, **vlGetDeviceList()**).

2. Specify nodes on the data path (**vlGetNode()**).

3. Create the path (**vlCreatePath()**).

4. (Optional step) Add more connections to a path **(vlAddNode())**.

5. Set up the hardware for the path (**vlSetupPaths()**).

6. Specify path-related events to be captured (**vlSelectEvents()**).

7. Set input and output parameters (controls) for the nodes on the path (**vlSetControl()**).

8. For transfers involving memory, create a VL buffer to hold data for memory transfers (**vlGetTransferSize()**, **vlCreateBuffer()**).

9. For transfers involving memory, register the buffer (**vlRegisterBuffer()**).

10. Set the path topology (**vlSetConnection()**).

11. Start the data transfer (**vlBeginTransfer()**).

12. For transfers involving memory, get the data (**vlGetNextValid()** or **vlGetLatestValid()**, **vlGetActiveRegion()**, **vlPutFree()**) to manipulate frame data.

13. Clean up (**vlEndTransfer()**, **vlDeregisterBuffer()**, **vlDestroyPath()**, **vlDestroyBuffer()**, **vlCloseVideo()**).

Table 2-1 lists calls explained in this chapter.

Table 2-1   Video Library Calls for Data Transfer

All Transfers	Transfers Involving Memory	Setting Controls
vlOpenVideo()	vlGetTransferSize()	vlSetControl()
vlGetDevice()	vlCreateBuffer()	vlGetControl()
vlGetDeviceList()	vlRegisterBuffer()	vlControlList()
vlGetNode()	vlGetNextValid()	vlGetControlInfo()
vlCreatePath()	vlGetLatestValid()	
vlSetConnection()	vlPutValid()	
vlGetConnection()	vlGetNextFree()	
vlAddNode()	vlGetActiveRegion()	
vlRemoveNode()	vlPutFree()	
vlSetupPaths()	vlGetDMediaInfo()	
vlSelectEvents()	vlGetImageInfo()	
vlBeginTransfer()	vlDeregisterBuffer()	
vlEndTransfer()	vlDestroyBuffer()	
vlDestroyPath()		
vlCloseVideo()		

Performing Preliminary Steps

To run VL, you must

- install the *dmedia_dev* option

- link with *libvl.so*

- include *vl.h* and *dev_mgv.h*

The client library is */usr/lib/libvl.so*. The header files for the VL are in */usr/include/dmedia*. The header file for the VL, *vl.h*, contains the main definition of the VL API and controls. The header files for Indigo[2] IMPACT Video are */usr/include/dmedia/dev_mgv.h* (linked to */usr/include/vl/vl_mgv.h*) and */usr/include/dmedia/dev_impact.h* (linked to */usr/include/vl/vl_impact.h*).

Note: When building a VL-based program, you must add -*lvl* to the linking command.

Opening a Connection to the Video Daemon

The first thing a VL application must do is open the device with
vlOpenVideo(). Its function prototype is

```
VLServer vlOpenVideo(const char *sName)
```

where *sName* is the name of the server to which to connect; set it to a NULL
string for the local server. For example:

```
vlSvr = vlOpenVideo("")
```

Specifying Nodes on the Data Path

Use **vlGetNode()** to specify nodes; this call returns the node's handle. Its
function prototype is

```
VLNode vlGetNode(VLServer vlSvr, int type, int kind, int
number)
```

where

VLNode   is a handle for the node, used when setting controls or
                setting up paths

vlSvr   names the server (as returned by **vlOpenVideo()**)

type   specifies the type of node:

- VL_SRC: source

- VL_DRN: drain

- VL_INTERNAL: internal node, such as the blend node

- VL_DEVICE: device for device-global controls

    Note: If you are using VL_DEVICE, the kind should be
    set to 0.

kind   specifies the kind of node:

- VL_BLENDER

    Note: The use of VL_BLENDER is explained in
    Chapter 7, "Blending, Keying, and Transitions," later in
    this guide.

- VL_FB: internal frame buffer node for freezing video

- VL_MEM: region of workstation memory

- VL_SCREEN: workstation screen

- VL_VIDEO: connection to a video device; for example, a video tape deck or camera

Note: Appendix C, "Indigo² IMPACT Video Nodes and Their Controls," gives full details of all Indigo² IMPACT Video nodes except VL_BLENDER and VL_CSC.

number   is the number of the node in cases of two or more identical nodes, such as two video source nodes

To discover which node the default is, use the control VL_DEFAULT_SOURCE after getting the node handle the normal way. The default video source is maintained by the VL. For example:

```
vlGetControl(vlSvr, path, VL_ANY, VL_DEFAULT_SOURCE, &ctrlval);
nodehandle = vlGetNode(vlSvr, VL_SRC, VL_VIDEO, ctrlval.intVal);
```

In the first line above, the last argument is a struct that retrieves the value. Corresponding to VL_DEFAULT_SOURCE, the control VL_DEFAULT_DRAIN gets the default VL_SRC node.

Creating and Setting Up the Data Path

Once nodes are specified, use VL calls to

- create the path

- get the device ID

- add nodes (optional step)

- set up the data path

- specify the path-related events to be captured

Creating the Path

Use **vlCreatePath()** to create the data path. Its function prototype is

```
VLPath vlCreatePath(VLServer vlSvr, VLDev vlDev
    VLNode src, VLNode drn)
```

This code fragment creates a path if the device is unknown:

```
if ((path = vlCreatePath(vlSvr, VL_ANY, src, drn)) < 0) {
    vlPerror(_progName);
    exit(1);
}
```

This code fragment creates a path that uses a device specified by parsing a *devlist*:

```
if ((path = vlCreatePath(vlSvr, devlist[devicenum].dev, src,
    drn)) < 0) {
    vlPerror(_progName);
    exit(1);
}
```

Note: If the path contains one or more invalid nodes, **vlCreatePath()** returns VLBadNode.

Getting the Device ID

If you specify VL_ANY as the device when you create the path, use **vlGetDevice()** to discover the device ID selected. Its function prototype is

```
VLDev vlGetDevice(VLServer vlSvr, VLPath path)
```

For example:

```
devicenum = vlGetDevice(vlSvr, path);
deviceName = devlist.devices[devicenum].name;
printf("Device is: %s/n", deviceName);
```

Adding a Node

For this optional step, use **vlAddNode()**. Its function prototype is

```
int vlAddNode(VLServer vlSvr, VLPath vlPath, VLNodeId node)
```

where

vlSvr   names the server to which the path is connected

vlPath   is the path as defined with **vlCreatePath()**

node   is the node ID

This example fragment adds a source node and a blend node:

```
vlAddNode(vlSvr, vlPath, src_vid);
vlAddNode(vlSvr, vlPath, blend_node);
```

Setting Up the Data Path

Use **vlSetupPaths()** to set up the data path. Its function prototype is

```
int vlSetupPaths(VLServer vlSvr, VLPathList paths,
      u_int count, VLUsageType ctrlusage,
      VLUsageType streamusage)
```

where

vlSvr   names the server to which the path is connected

paths   specifies a list of paths you are setting up

count   specifies the number of paths in the path list

ctrlusage   specifies usage for path controls:

- VL_SHARE: other paths can set controls on this node; this control is the desired setting for other paths, including *vcp*, to work

     Note: When using VL_SHARE, pay attention to events. If another user has changed a control, a VLControlChanged event occurs.

- VL_READ_ONLY: controls cannot be set, only read; for example, this control can be used to monitor controls

- VL_LOCK: prevents other paths from setting controls on this path; controls cannot be used by another path

- VL_DONE_USING: the resources are no longer required; the application releases this set of paths for other applications to acquire

streamusage   specifies usage for the data:

- VL_SHARE: transfers can be preempted by other users; paths contend for ownership

  Note: When using VL_SHARE, pay attention to events. If another user has taken over the node, a VLStreamPreempted event occurs.

- VL_READ_ONLY: the path cannot perform transfers, but other resources are not locked; set this value to use the path for controls

- VL_LOCK: prevents other paths that share data transfer resources with this path from transferring; existing paths that share resources with this path will be preempted

- VL_DONE_USING: the resources are no longer required; the application releases this set of paths for other applications to acquire

This example fragment sets up a path with shared controls and a locked stream:

```
if (vlSetupPaths(vlSvr, (VLPathList)&path, 1, VL_SHARE,
    VL_LOCK) < 0)
{
    vlPerror(_progName);
    exit(1);
}
```

Note: The Video Library infers the connections on a path if **vlBeginTransfer()** is called and no drain nodes have been connected using **vlSetConnection()** (implicit routing). To specify a path that does not use the default connections, use **vlSetConnection()** (explicit routing). Chapter 5, "Managing Connections," explains the use of this function and related requirements.The following rules are used in determining the connections:

- For each internal node on the path, all unconnected input ports are connected to the first source node added to the path. Pixel ports are connected to pixel ports and alpha ports are connected to alpha ports.

- For each drain node on the path, all unconnected input ports are connected to the first internal node placed on the path, if there is an internal node, or to the first source node placed on the path. Pixel ports are connected to pixel ports and alpha ports are connected to alpha ports.

Note: Do not combine implicit and explicit routing.

Specifying the Path-Related Events to Be Captured

Use **vlSelectEvents()** to specify the events you want to receive. Its function prototype is

```
int vlSelectEvents(VLServer vlSvr, VLPath path, VLEventMask eventmask)
```

where

vlSvr   names the server to which the path is connected

path   specifies the data path.

eventmask   specifies the event mask; Table 2-2 lists the possibilities

Table 2-2 lists and describes the VL event masks.

Table 2-2   VL Event Masks

Symbol	Meaning
VLStreamBusyMask	Stream is locked
VLStreamPreemptedMask	Stream was grabbed by another path
vlStreamChangedMask	Video routing on this path has been changed by another path
VLAdvanceMissedMask	Time was already reached
VLSyncLostMask	Irregular or interrupted signal

Table 2-2 (continued)   VL Event Masks

Symbol	Meaning
VLSequenceLostMask	Field or frame dropped
VLControlChangedMask	A control has changed
VLControlRangeChangedMask	A control range has changed
VLControlPreemptedMask	Control of a node has been preempted, typically by another user setting VL_LOCK on a path that was previously set with VL_SHARE
VLControlAvailableMask	Access is now available
VLTransferCompleteMask	Transfer of field or frame complete
VLTransferFailedMask	Error; transfer terminated; perform cleanup at this point, including **vlEndTransfer()**
VLEvenVerticalRetraceMask	Vertical retrace event, even field
VLOddVerticalRetraceMask	Vertical retrace event, odd field
VLFrameVerticalRetraceMask	Frame vertical retrace event
VLDeviceEventMask	Device-specific event, such as a trigger
VLDefaultSourceMask	Default source changed

For example:

```
vlSelectEvents(vlSvr, path, VLTransferCompleteMask);
```

Event masks can be Or'ed; for example:

```
vlSelectEvents(vlSvr, path, VLTransferCompleteMask |
     VLTransferFailedMask);
```

Setting Parameters for Data Transfer to or From Memory

Transferring data to or from memory requires creating a VL buffer; its size is determined by the size of the frame data you are transferring.

To set frame data size and to convert from one video format to another, apply controls to the nodes. The use of source node controls and drain node controls is explained separately in this section.

Setting Node Controls for Data Transfer

Important data transfer controls for source and drain nodes are summarized in Table 2-3. They should be set in the order in which they appear in the table.

These controls are highly interdependent, so the order in which they are set is important. In most cases, the value being set takes precedence over other values that were previously set.

Note: For drain nodes, VL_PACKING must be set first. Note that changes in one parameter may change the values of other parameters set earlier; for example, clipped size may change if VL_PACKING is set after VL_SIZE.

Table 2-3   Data Transfer Controls

Control	Basic Use	Video Nodes	Memory Nodes	Screen Nodes
VL_FORMAT	Video format on the physical connector	See "Using VL_FORMAT" in this chapter	N/A	N/A
VL_TIMING	Video timing	See Table 2-4 for values	Not applicable	Not applicable
VL_CAP_TYPE	Setting type of field(s) or frame(s) to capture; see "Interlacing" in Appendix A	Not applicable	VL_CAPTURE_NONINTERLEAVED VL_CAPTURE_INTERLEAVED VL_CAPTURE_EVEN_FIELDS VL_CAPTURE_ODD_FIELDS VL-CAPTURE_FIELDS	Not applicable
VL_PACKING	Pixel packing (conversion) format	Not applicable	Changes pixel format of captured data; see Table 2-6 for values	Not applicable

Table 2-3 (continued)   Data Transfer Controls

Control	Basic Use	Video Nodes	Memory Nodes	Screen Nodes
VL_SIZE	Clipping size	Full size of video; read only	Clipped size	Clipped size
VL_OFFSET	Position within larger area	Position of active region	Offset relative to video offset	Pan within the video
VL_ORIGIN	Position within video	Not applicable	Not applicable	Screen position of first pixel displayed
VL_WINDOW	Setting window ID for video in a window	Not applicable	Not applicable	Window ID
VL_RATE	Field or frame transfer speed	Depends on capture type as specified by VL_CAP_TYPE	Not applicable	Not applicable

To determine default values, use **vlGetControl()** to query the values on the video source or drain node before setting controls. The initial offset of the video node is the first active line of video.

Similarly, the initial size value on the video source or drain node is the full size of active video being captured by the hardware, beginning at the default offset. Because some hardware can capture more than the size given by the video node, this value should be treated as a default size.

For all these controls, it pays to track return codes. If the value returned is VLValueOutOfRange, the value set is not what you requested.

To specify the controls, use **vlSetControl()**, for which the function prototype is

```
int vlSetControl(VLServer vlSvr, VLPath vlPath, VLNode node,
        VLControlType type, VLControlValue * value)
```

The use of VL_TIMING, VL_FORMAT, VL_PACKING, VL_SIZE, VL_OFFSET, VL_CAP_TYPE, and VL_RATE is explained in more detail in the following sections.

Using VL_TIMING

Timing type expresses the timing of video presented to a source or drain. Table 2-4 summarizes dimensions for VL_TIMING.

Table 2-4   Dimensions for Timing Choices

Timing	Maximum Width	Maximum Height
VL_TIMING_525_SQ_PIX (12.27 MHz)	640	486
VL_TIMING_625_SQ_PIX (14.75 MHz)	768	576
VL_TIMING_525_CCIR601 (13.50 MHz)	720	486
VL_TIMING_625_CCIR601 (13.50 MHz)	720	576

Using VL_FORMAT

To specify video input and output formats of the video signal on the physical connector, use VL_FORMAT.

Table 2-5   VL_FORMAT

Format	Explanation	Supported by Node
VL_FORMAT_DIGITAL _COMPONENT_SERIAL	8- or 10-bit YCrCb	Single-link and dual-link
VL_FORMAT_SMPTE_YUV	Backwards compatibility: 8- or 10-bit YCrCb	Single-link and dual-link
VL_FORMAT_RAW_DATA	Arbitrary 8-bit data (non-video format)	Single-link only
VL_FORMAT_RGB	Full-range 8-bit (0-255) or 10-bit (0-1023) RGBA	Dual-link only
VL_FORMAT_DIGITAL _COMPONENT_RGB_SERIAL	RP175 standard RGBA	Dual-link only

Using **VL_PACKING**

A video *packing* describes how a video signal is stored in memory, in contrast with a video format, which describes the characteristics of the video signal. For example, the memory source nodes—CC1 and both VGI1 nodes—accept packed video from a VL buffer and output video in a given format.

Packings are specified through the VL_PACKING control on the memory nodes. This control also converts one video output format to another in memory, within the limits of the nodes.

Note: On dual-linked VGI1 memory nodes, only native packings are available; no conversions can be performed.

Packing types for eight bits per component are summarized in Table 2-6. In this table, the Native To column lists the nodes to which the packing is native; no software conversion is required, so these packings are fastest.

Table 2-6   Packing Types for Eight Bits per Component

Type	Native To	63-56	55-48	47-40	39-32	31-24	23-16	15-8	7-0
VL_PACKING_YVYU_422_8 YUV 4:2:2, single-link	All memory nodes	U0	Y0	V0	Y1	U2	Y2	V2	Y3
VL_PACKING_YUVA_4444_8 YUVA 4:4:4:4, dual-link	VGI1 memory nodes	A0	U0	Y0	V0	A1	U1	Y1	V1
VL_PACKING_AUYV_4444_8 AUYV 4:4:4:4, dual-link	VGI1 memory nodes	V0	Y0	U0	A0	V1	Y1	U1	A1
VL_PACKING_RGBA_8 RGBA, dual-link	VGI1 memory nodes	A0	B0	G0	R0	A1	B1	G1	R1
VL_PACKING_ABGR_8 ABGR, dual-link	VGI1 memory nodes	R0	G0	B0	A0	R1	G1	B1	A1
VL_PACKING_RGB_332_P RGB, single-link Each 8-bit pixel, P*n*, is shown as BBGGGRRR	None	P7	P6	P5	P4	P3	P2	P1	P0

Table 2-6 (continued)   Packing Types for Eight Bits per Component

Type	Native To	63-56	55-48	47-40	39-32	31-24	23-16	15-8	7-0
VL_PACKING_Y_8_P Grayscale (luminance only), single-link	None	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
VL_PACKING_RGB_8 RGB, single-link 24-bit word, X*n* are ignored	None	X0	B0	G0	R0	X1	B1	G1	R1

Packing types for ten bits per component are summarized in Table 2-7. The ten data bits are left-aligned within a 16-bit word. The hardware sets the lower six bits to zero before it writes them to memory. When reading from memory, the lower six bits are ignored.

Table 2-7   Packing Types for Ten Bits per Component

Type	Native To	63-48	47-32	31-16	15-0
VL_PACKING_YVYU_422_10 YUV 4:2:2, single-link	VGI1 memory nodes	[U0]000000	[Y0]000000	[V0]000000	[Y1]000000
VL_PACKING_YUVA_4444_10 YUVA 4:4:4:4, dual-link	VGI1 memory nodes	[A0]000000	[U0]000000	[Y0]000000	[V0]000000
VL_PACKING_AUYV_4444_10 AUYV 4:4:4:4, dual-link	VGI1 memory nodes	[V0]000000	[Y0]000000	[U0]000000	[A0]000000
VL_PACKING_RGBA_10 RGBA, dual-link	VGI1 memory nodes	[A0]000000	[B0]000000	[G0]000000	[R0]000000
VL_PACKING_ABGR_10 ABGR, dual-link	VGI1 memory nodes	[R0]000000	[G0]000000	[B0]000000	[A0]000000

In addition, Indigo[2] IMPACT Video also supports dual-link AYUAYV, a packed format with three 10-bit components per 32-bit word, with the lowest two bits set to 0. It is native to VGI1 memory nodes. Bits are

- 63-32: [U0][Y0][A0]00

- 31-0: [V0][Y1][A1]00

Note: Other libraries may use different packing names.

For example:

```
VLControlValue val;

val.intVal = VL_PACKING_RGBA_10;
vlSetControl(vlSvr, path, memdrn, VL_PACKING, &val);
```

If the single-link packings VL_PACKING_RGB_332_P, VL_PACKING_RGB_8, and VL_PACKING_Y_8_P are requested of a memory drain node, the Video Library performs a software conversion to translate the data from a native packing and format. The application receives data in the requested packing and format, although the capture rate may be degraded.

Using VL_ZOOM

VL_ZOOM controls the expansion or decimation of the video image. Values greater than one expand the video; values less than one perform decimation. Figure 2-1 illustrates zooming and decimation.

Note: Indigo[2] IMPACT Video screen drain nodes support the full range of VL_ZOOM (7/1, 6/1, 5/1, 4/1, 3/1, 2/1, 1/1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8). Screen source nodes support 1/1 and 1/2. Other nodes support zoom and decimation ratios of 1:1 only, that is, no zoom or decimation.

Original image

Zoom factor: 2/1

Decimation factor: 1/2
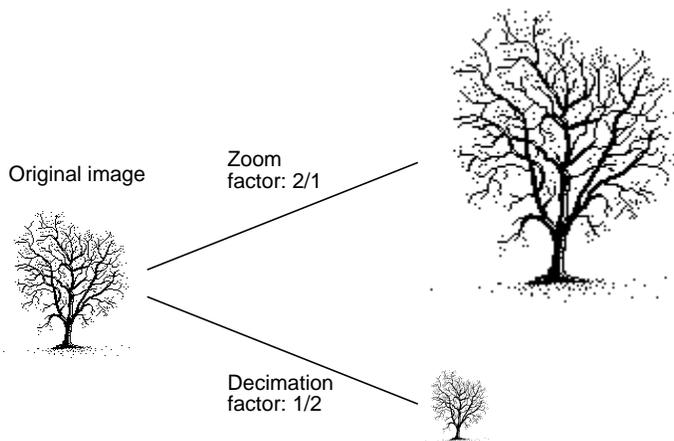
Figure 2-1   Zoom and Decimation

VL_ZOOM takes a nonzero fraction as its argument; do not use negative values. For example, this fragment captures half-size decimation video to the screen:

```
val.fractVal.numerator = 1;
val.fractVal.denominator = 2;
if (vlSetControl(server, screen_path, screen_drain_node,
VL_ZOOM, &val)){
   vlPerror("Unable to set zoom");
   exit(1);
}
```

Note: For a source, zooming takes place before blending; for a drain, blending takes place before zooming.

This fragment captures half-size decimation video to the screen, with clipping to 320×243 (NTSC size minus overscan):

```
val.fractVal.numerator = 1;
val.fractVal.denominator = 2;
if (vlSetControl(server,screen_path, screen_drain_node,
    VL_ZOOM, &val))
{
    vlPerror("Unable to set zoom");
    exit(1);
}
val.xyVal.x = 320;
val.xyVal.y = 243;
if (vlSetControl(server, screen_path, screen_drain_node,
    VL_SIZE, &val))
{
    vlPerror("Unable to set size");
    exit(1);
}
```

This fragment captures *xsize* × *ysize* video with as much decimation as possible, assuming the size is smaller than the video stream:

```
if (vlGetControl(server, screen_path, screen_source,
VL_SIZE, &val))
{
   vlPerror("Unable to get size");
   exit(1);
}
if (val.xyVal.x/xsize < val.xyVal.y/ysize)
   zoom_denom = (val.xyVal.x + xsize - 1)/xsize;
else
   zoom_denom = (val.xyVal.y + ysize - 1)/ysize;
val.fractVal.numerator = 1;
val.fractVal.denominator = zoom_denom;
   if (vlSetControl(server, screen_path, screen_drain_node,
   VL_ZOOM, &val))
{
   /* allow this error to fall through */
   vlPerror("Unable to set zoom");
}
val.xyVal.x = xsize;
val.xyVal.y = ysize;
if (vlSetControl(server, screen_path, screen_drain_node,
   VL_SIZE, &val))
{
   vlPerror("Unable to set size");
   exit(1);
}
```

Using VL_SIZE

VL_SIZE controls how much of the image sent to the drain is used, that is, how much clipping takes place. This control operates on the zoomed image; for example, when the image is zoomed to half size, the limits on the size control change by a factor of 2. Figure 2-2 illustrates clipping.

Clipping an unzoomed image

Original image   Image to fit into this space

Clipping a zoomed image

Placement of clipping area
depends on the value of VL_OFFSET
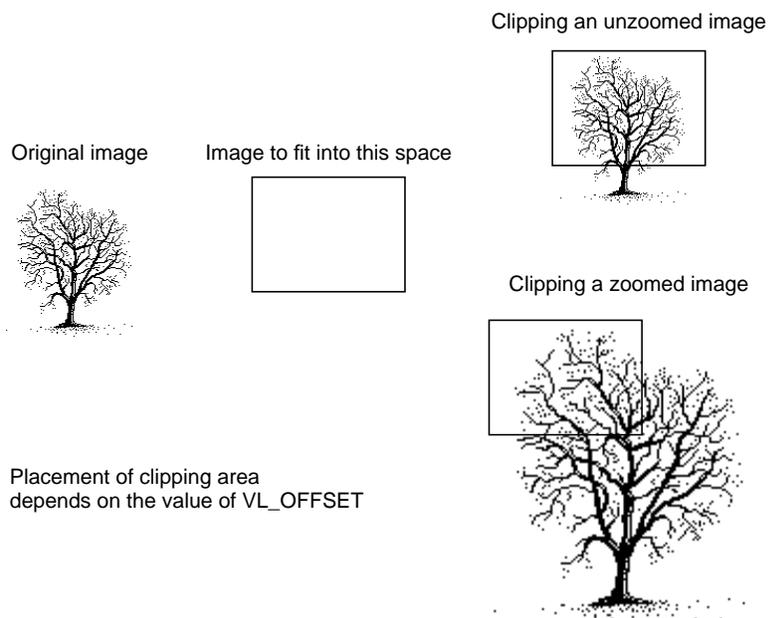
Figure 2-2   Clipping an Image

For example, to display PAL video in a 320×243 space, clip the image to that size, as shown in the following fragment:

```
VLControlValue value;
value.xyval.x=320;
value.xyval.y=243;
vlSetControl(vlSvr, path, drn, VL_SIZE, &value);
```

Note: Because this control is device-dependent and interacts with other controls, always check the error returns. For example, if offset is set before size and an error is returned, set size before offset.

Using VL_OFFSET

VL_OFFSET puts the upper left corner of the video data at a specific position; it sets the beginning position for the clipping performed by VL_SIZE. The values you enter are relative to the origin.

This example places the data ten pixels down and ten pixels in from the left:

```
VLControlValue value;
value.xyval.x=10;
value.xyval.y=10;
vlSetControl(vlSvr, path, drn, VL_OFFSET, &value);
```

To capture the blanking region, set offset to a negative value.

Figure 2-3 shows the relationships between the source and drain size, offset, and origin.

Note: For memory nodes, VL_OFFSET and VL_SIZE in combination define the active region of video that is transferred to or from memory.
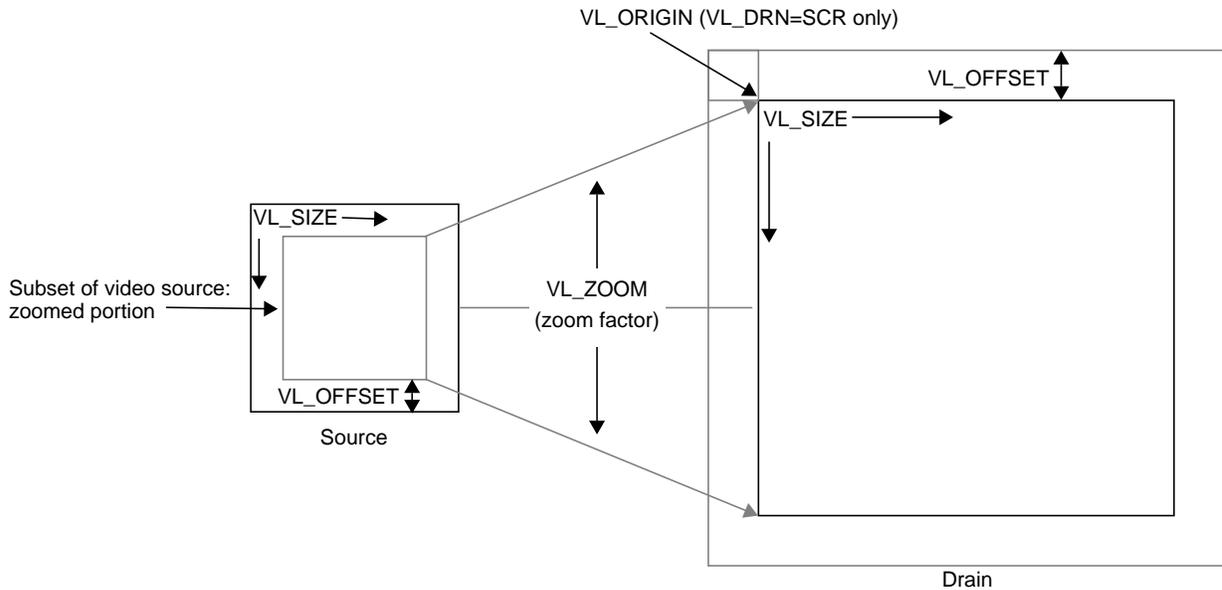
Figure 2-3   Zoom, Size, Offset, and Origin

Using VL_CAP_TYPE and VL_RATE

An application can request that Indigo^2 IMPACT Video capture or play back a video stream in a number of ways. For example, the application can request that each field be placed in its own buffer, that each buffer contain an interleaved frame, or that only odd or even fields be captured. This section enumerates the capture types that Indigo^2 IMPACT Video supports.

A *field mask* is useful for identifying which fields will be captured and played back and which fields will be dropped. A field mask is a bit mask of 60 bits for NTSC or 50 bits for PAL (two fields per frame). A numeral 1 in the mask indicates that a field is captured or played back, while a zero indicates that no action occurs.

For example, the following field mask indicates that every other field will be captured or played back:

```
101010101010101010...
```

Capture types are as follows:

- VL_CAPTURE_NONINTERLEAVED
- VL_CAPTURE_INTERLEAVED
- VL_CAPTURE_EVEN_FIELDS
- VL_CAPTURE_ODD_FIELDS
- VL_CAPTURE_FIELDS

VL_RATE determines the data transfer rate by field or frame, depending on the capture type as specified by VL_CAP_TYPE, as shown in Table 2-8.

Table 2-8   VL_RATE Values (Items per Second)

VL_CAP_TYPE Value	VL_RATE Value
VL_CAPTURE_NONINTERLEAVED, VL_CAPTURE_INTERLEAVED	NTSC: 1-30 frames/second PAL: 1-25 frames/second
VL_CAPTURE_EVEN_FIELDS, VL_CAPTURE_ODD_FIELDS	NTSC: 1-30 fields/second PAL: 1-25 fields/second
VL_CAPTURE_FIELDS	NTSC: 1-60 fields/second PAL: 1-50 fields/second

Note: Not all rates are supported on all memory nodes; see Appendix C, "Indigo2 IMPACT Video Nodes and Their Controls," for details. The buffer size must be set in accordance with the capture type, as listed in Table 2-9 later in this chapter.

VL_CAPTURE_NONINTERLEAVED

The VL_CAPTURE_NONINTERLEAVED capture type specifies that frame-size units are captured noninterleaved. Each field is placed in its own buffer, with the dominant field in the first buffer. If one of the fields of a frame is dropped, all fields are dropped. Consequently, an application is guaranteed that the field order is maintained; no special synchronization is necessary to ensure that fields from different frames are mixed.

The rate (VL_RATE) for noninterleaved capture is in terms of fields and must be even. For NTSC, the capture rate may be from 2-60 fields per second, and for PAL 2-50 fields per second. Because a frame is always captured as a whole, a rate of 30 fields per second results in the following field mask:

```
1100110011001100...
```

The first bit in the field mask corresponds to the dominant field of a frame. Indigo^2 IMPACT Video waits for a dominant field before it starts the transfer.

If VL_CAPTURE_NONINTERLEAVED is specified for playback, similar guarantees apply as for capture. If one field is lost during playback, it is not possible to "take back" the field. Indigo^2 IMPACT Video resynchronizes on the next frame boundary, although black or "garbage" video might be present between the erring field and the frame boundary.

The rate during playback also follows the rules for capture. For each 1 in the mask above, a field from the VL buffer is output. During the 0 fields, the previous frame is repeated. Note that the previous *frame* is output, not just the last field. If there are a pair of buffers, the dominant field is placed in the first buffer.

VL_CAPTURE_INTERLEAVED

Interleaved capture interleaves the two fields of a frame and places them in a single buffer; the order of the frames depends on the value set for VL_MGV_DOMINANCE_FIELD (see Table C-3 or Table C-4 in Appendix C for details). Indigo^2 IMPACT Video guarantees that the interleaved fields are from the same frame: if one field of a frame is dropped, then both are dropped.

The rate for interleaved frames is in frames per second: 1-30 frames per second for NTSC and 1-25 frames per second for PAL. A rate of 15 frames per second results in every other frame being captured. Expressed as a field mask, the following sequence is captured:

```
1100110011001100....
```

As with VL_CAPTURE_NONINTERLEAVED, Indigo^2 IMPACT Video begins processing the field mask when a dominant field is encountered.

During playback, a frame is deinterleaved and output as two consecutive fields, with the dominant field output first. If one of the fields is lost, Indigo^2 IMPACT Video resynchronizes to a frame boundary before playing the next frame. During the resynchronization period, black or "garbage" data may be displayed.

Rate control follows similar rules as for capture. For each 1 in the mask above, a field from the interleaved frame is output. During 0 periods, the previous frame is repeated.

VL_CAPTURE_EVEN_FIELDS

In the VL_CAPTURE_EVEN_FIELDS capture type, only even (F2) fields are captured, with each field placed in its own buffer. Expressed as a field mask, the captured fields are

```
1010101010101010...
```

Indigo^2 IMPACT Video begins processing this field mask when an even field is encountered.

The rate for this capture type is expressed in even fields. For NTSC, the range is 1-30 fields per second, and for PAL 1-25 fields per second. A rate of 15 fields per second (NTSC) indicates that every other even field is captured, yielding a field mask of

```
1000100010001000...
```

During playback, the even field is repeated as both the F1 and F2 fields, until it is time to output the next buffer. If a field is lost during playback, black or "garbage" data might be displayed until the next buffer is scheduled to be displayed.

VL_CAPTURE_ODD_FIELDS

The VL_CAPTURE_ODD_FIELDS capture type works the same way as VL_CAPTURE_EVEN_FIELDS, except that only odd (F1) fields are captured, with each field placed in its own buffer. The rate for this capture type is expressed in odd fields. A rate of 15 fields per second (NTSC) indicates that every other odd field is captured. Field masks are the same as for VL_CAPTURE_EVEN_FIELDS.

VL_CAPTURE_FIELDS

The VL_CAPTURE_FIELDS capture type captures both even and odd fields and places each in its own buffer. Unlike VL_CAPTURE_NONINTERLEAVED, there is no guarantee that fields are dropped in frame units. Field synchronization can be performed by examining the UST, the MSC, or the dmedia info sequence number associated with each field.

The rate for this capture type is expressed in fields. For NTSC, the range is 1-60 fields per second, and for PAL 1-50 fields per second. A rate of 30 fields per second (NTSC) indicates that every other field is captured, resulting in the following field mask:

```
101010101010101010...
```

Contrast this with the rate of 30 for VL_CAPTURE_NONINTERLEAVED, which captures every other frame.

Field mask processing begins on the first field after the transfer is started; field dominance, evenness, oddness plays no role in this capture type.

Displaying Video Data Onscreen

To set up a window for live video, follow these steps, as outlined in the example program *simplev2s.c.*

1. Open an X display window; for example:

    ```
    if (!(dpy = XOpenDisplay("")))
        exit(1);
    ```

2. Connect to the video daemon; for example:

    ```
    if (!(vlSvr = vlOpenVideo("")))
        exit(1);
    ```

3. Create a window to show the video; for example:

    ```
    vwin = XCreateSimpleWindow(dpy, RootWindow(dpy, 0), 10,
        10, 640, 486, 0,
        BlackPixel(dpy,DefaultScreen(dpy)),
        BlackPixel(dpy, DefaultScreen(dpy));
    XMapWindow(dpy, vwin);
    XFlush(dpy);
    ```

4. Create a source node on a video device and a drain node on the screen; for example:

    ```
    src = vlGetNode(vlSvr, VL_SRC, VL_VIDEO, VL_ANY);
    drn = vlGetNode(vlSvr, VL_DRN, VL_SCREEN, VL_ANY);
    ```

5. Create a path on the first device that supports it; for example:

    ```
    if ((path = vlCreatePath(vlSvr, VL_ANY, src, drn)) < 0)
        exit(1);
    ```

6. Set up the hardware for the path and define the path use; for example:

    ```
    vlSetupPaths(vlSvr, (VLPathList)&path, 1, VL_SHARE,
        VL_SHARE);
    ```

7. Set the X window to be the drain; for example:

    ```
    val.intVal = vwin;
    vlSetControl(vlSvr, path, drn, VL_WINDOW, &val);
    ```

8. Get X and VL into the same coordinate system; for example:

    ```
    XTranslateCoordinates(dpy, vwin, RootWindow(dpy,
        DefaultScreen(dpy)), 0, 0,&x, &y, &dummyWin);
    ```

9. Set the live video to the same location and size as the window; for example:

```
val.xyVal.x = x;
val.xyVal.y = y;
vlSetControl(vlSvr, path, drn, VL_ORIGIN, &val);
XGetGeometry(dpy, vwin, &dummyWin, &x, &y, &w, &h, &bw,
    &d);
val.xyVal.x = w;
val.xyVal.y = h;
vlSetControl(vlSvr, path, drn, VL_SIZE, &val);
```

10. Begin the data transfer:

```
vlBeginTransfer(vlSvr, path, 0, NULL);
```

11. Wait until the user finishes; for example:

```
printf("Press return to exit.\n");
c = getc(stdin);
```

12. End the data transfer, clean up, and exit:

```
vlEndTransfer(vlSvr, path);
vlDestroyPath(vlSvr, path);
vlCloseVideo(vlSvr);
```

Transferring Video Data to and From Devices

The processes for data transfer are as follows:

- creating a buffer for video data (for transfers involving memory)

- registering the VL buffer with the path (for transfers involving memory)

- starting data transfer

- reading data from the buffer (for transfers involving memory)

Each process is explained separately.

Creating a Buffer for Video Data

Once you have specified frame parameters in a transfer involving memory (or have determined to use the defaults), create a buffer for the video data. In this case, video data is frames or fields, depending on the capture type:

- frames if the capture type is VL_CAPTURE_NONINTERLEAVED

- fields if the capture type is anything else

Like other libraries in the IRIX digital media development environment, the VL uses VL buffers. Vl buffers provide a way to read and write varying sizes of video data. A frame of data consists of the actual frame data and an information structure describing the underlying data, including device-specific information.

When a VL buffer is created, constraints are specified that control the total size of the data segment and the number of frame or field buffers (sectors) to allocate.

A head and a tail flag are automatically set in a VL buffer so that the latest frame can be accessed. A sector is locked down if it is not called; that is, it remains locked until it is read. When the VL buffer is written to and all sectors are occupied, data transfer stops. The sector last written to remains locked down until it is released.

All sectors in a VL buffer must be of the same size, which is the value returned by **vlGetTransferSize()**. Its function prototype is

```
long vlGetTransferSize(VLServer vlSvr, VLPath path)
```

For example:

```
transfersize = vlGetTransferSize(vlSvr, path);
```

where *transfersize* is the size of the data in bytes.

41

To create a VL buffer for the frame data, use **vlCreateBuffer()**. Its function prototype is

```
VLBuffer vlCreateBuffer(VLServer vlSvr, VLPath path,
     VLNode node, int numFrames)
```

where

VLBuffer   is the handle of the buffer to be created

vlSvr   names the server to which the path is connected

path   specifies the data path

node   specifies the memory node containing data to transfer to or from the VL buffer

numFrames   specifies the number of sectors in the buffer (fields or frames, depending on the capture type)

For example:

```
buf = vlCreateBuffer(vlSvr, path, src, 1);
```

Table 2-9 shows the relationship between capture type and minimum VL buffer size.

Table 2-9   Buffer Size Requirements

Capture Type	Minimum Sectors for Capture	Minimum Sectors for Playback
VL_CAPTURE_NONINTERLEAVED	2	4
VL_CAPTURE_INTERLEAVED	1	2
VL_CAPTURE_EVEN_FIELDS	1	2
VL_CAPTURE_ODD_FIELDS	1	2
VL_CAPTURE_FIELDS	1	2

Note: For VGI1 memory nodes, real-time memory or video transfer can be performed only as long as buffer sectors are available to the Indigo^2 IMPACT Video device.

Registering the VL Buffer

Use **vlRegisterBuffer()** to register the VL buffer with the data path. Its function prototype is

```
int vlRegisterBuffer(VLServer vlSvr, VLPath path,
    VLNode memnodeid, VLBuffer buffer)
```

where

vlSvr   names the server to which the path is connected

path   specifies the data path

memnodeid   specifies the memory node ID

buffer   specifies the VL buffer handle

For example:

```
vlRegisterBuffer(vlSvr, path, drn, Buffer);
```

Starting Data Transfer

To begin data transfer, use **vlBeginTransfer()**. Its function prototype is

```
int vlBeginTransfer(VLServer vlSvr, VLPath path, int count,
    VLTransferDescriptor* xferDesc)
```

where

vlSvr   names the server to which the path is connected

path   specifies the data path

count   specifies the number of transfer descriptors

xferDesc   specifies an array of transfer descriptors

Tailor the data transfer by means of *transfer descriptors*. Multiple transfer descriptors are supplied; they are executed in order. The transfer descriptors are

xferDesc.mode   Transfer method:

- VL_TRANSFER_MODE_DISCRETE: a specified number of frames are transferred (burst mode)

43

- VL_TRANSFER_MODE_CONTINUOUS (default): frames are transferred continuously, beginning immediately or after a trigger event occurs (such as a frame coincidence pulse), and continues until transfer is terminated with **vlEndTransfer()**

- VL_TRANSFER_MODE_AUTOTRIGGER: frame transfer takes place each time a trigger event occurs; this mode is a repeating version of VL_TRANSFER_MODE_DISCRETE

xferDesc.count   Number of frames to transfer; if *mode* is VL_TRANSFER_MODE_CONTINUOUS, this value is ignored.

xferDesc.delay   Number of frames from the trigger at which data transfer begins.

xferDesc.trigger Set of events to trigger on; an event mask. This transfer descriptor is always required. VLTriggerImmediate specifies that transfer begins immediately, with no pause for a trigger event. VLDeviceEvent specifies an external trigger.

If *xferDesc* is NULL, then VL_TRIGGER_IMMEDIATE and VL_TRANSFER_CONTINOUS_MODE are assumed and one transfer is performed.

This example fragment transfers the entire contents of the buffer immediately.

```
xferDesc.mode = VL_TRANSFER_MODE_DISCRETE;
xferDesc.count = imageCount;
xferDesc.delay = 0;
xferDesc.trigger = VLTriggerImmediate;
```

This fragment shows the default descriptor, which is the same as passing in a null for the descriptor pointer. Transfer begins immediately; *count* is ignored.

```
xferDesc.mode = VL_TRANSFER_MODE_CONTINUOUS;
xferDesc.count = 0;
xferDesc.delay = 0;
xferDesc.trigger = VLTriggerImmediate;
```

Reading Data From the Buffer

If your application uses a buffer, use various VL calls for reading frames, getting pointers to active buffers, freeing buffers, and other operations. Table 2-10 lists the buffer-related calls.

Table 2-10   Buffer-Related Calls

Call	Purpose
vlGetNextValid()	Returns a handle on the next valid frame or field of data
vlGetLatestValid()	Reads only the most current frame or field in the buffer, discarding the rest
vlPutValid()	Puts a frame or field into the valid list (memory to video)
vlPutFree()	Puts a valid frame or field back into the free list (video to memory)
vlGetNextFree()	Gets a free buffer into which to write data (memory to video)
vlBufferDone()	Informs you if the buffer has been vacated
vlBufferReset()	Resets the buffer so that it can be used again

Figure 2-4 illustrates the difference between **vlGetNextValid()** and **vlGetLatestValid()**, and their interaction with **vlPutFree()**.
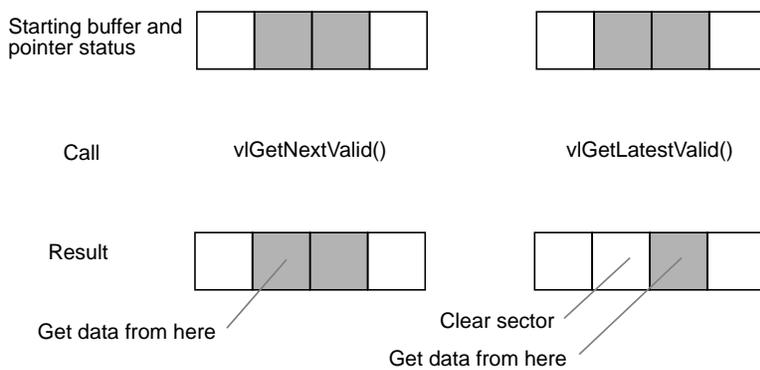
Figure 2-4   vlGetNextValid(), vlGetLatestValid(), and vlPutFree()

Table 2-11 lists the calls that extract information from a buffer.

Table 2-11   Calls for Extracting Data From a Buffer

Call	Purpose
vlGetActiveRegion()	Gets a pointer to the data region of the buffer (video to memory); called after **vlGetNextValid()** and **vlGetLatestValid()**
vlGetDMediaInfo()	Gets a pointer to the DMediaInfo structure associated with a frame; this structure contains timestamp and field count information
vlGetImageInfo()	Gets a pointer to the DMImageInfo structure associated with a frame; this structure contains image size information

Caution: None of these calls has count or block arguments; appropriate calls in the application must deal with a NULL return in cases of no data being returned.

In summary, for video-to-memory transfer, use

```
buffer = vlCreateBuffer(vlSvr, path, memnode1);
vlRegisterBuffer(vlSvr, path, memnode1, buffer);
vlBeginTransfer(vlSvr, path, 0, NULL);
info = vlGetNextValid(vlSvr, buffer);
/* OR vlGetLatestValid(vlSvr, buffer); */
dataptr = vlGetActiveRegion(vlSvr, buffer, info);

/* use data for application */
…
vlPutFree(vlSvr, buffer);
```

For memory-to-video transfer, use

```
buffer = vlCreateBuffer(vlSvr, path, memnode1);
vlRegisterBuffer(vlSvr, path, memnode1, buffer);
vlBeginTransfer(vlSvr, path, 0, NULL);
buffer = vlGetNextFree(vlSvr, buffer, bufsize);
/* fill buffer with data */
…
vlPutValid(vlSvr, buffer);
```

These calls are explained in separate sections.

Reading the Frames to Memory From the Buffer

Use **vlGetNextValid()** to read all the frames in the buffer or get a valid frame of data. Its function prototype is

```
VLInfoPtr vlGetNextValid(VLServer vlSvr, VLBuffer vlBuffer)
```

Use **vlGetLatestValid()** to read only the most current frame in the buffer, discarding the rest. Its function prototype is

```
VLInfoPtr vlGetLatestValid(VLServer vlSvr, VLBuffer vlBuffer)
```

After removing interesting data, return the buffer for use with **vlPutFree()** (video to memory). Its function prototype is

```
int vlPutFree(VLServer vlSvr, VLBuffer vlBuffer)
```

Sending Frames From Memory to Video

Use **vlGetNextFree()** to get a free buffer to which to write data. Its function prototype is

```
VLInfoPtr vlGetNextFree(VLServer vlSvr, VLBuffer vlBuffer,
      int size)
```

After filling the buffer with the data you want to send to video output, use **vlPutValid()** to put a frame into the valid list for output to video (memory to video). Its function prototype is

```
int vlPutValid(VLServer vlSvr, VLBuffer vlBuffer)
```

Caution: These calls do not have count or block arguments; appropriate calls in the application must deal with a NULL return in cases of no data being returned.

Getting DMediaInfo and Image Data From the Buffer

Use **vlGetActiveRegion()** to get a pointer to the active buffer. Its function prototype is

```
void * vlGetActiveRegion(VLServer vlSvr, VLBuffer vlBuffer,
     VLInfoPtr ptr)
```

Use **vlGetDMediaInfo()** to get a pointer to the DMediaInfo structure associated with a frame. This structure contains timestamp and field count information. The function prototype for this call is

```
DMediaInfo * vlGetDMediaInfo(VLServer vlSvr,
   VLBuffer vlBuffer, VLInfoPtr ptr)
```

Use **vlGetImageInfo()** to get a pointer to the DMImageInfo structure associated with a frame. This structure contains image size information. The function prototype for this call is

```
DMImageInfo * vlGetImageInfo(VLServer vlSvr,
   VLBuffer vlBuffer, VLInfoPtr ptr)
```

Ending Data Transfer

To end data transfer, use **vlEndTransfer()**. Its function prototype is

```
int vlEndTransfer(VLServer vlSvr, VLPath path)
```

A discrete transfer is finished when the last frame of the sequence is output. The two types of memory nodes behave differently at the last frame:

- The CC1 memory source stops transferring data from main memory to the Indigo^2 IMPACT Video device, but continues to output to video the last frame transferred, which is held in a frame buffer associated with the CC1 memory node.

- The VGI1 memory nodes have no associated frame buffer and consequently emit black video output after a transfer (discrete or continuous) has been completed.

To accomplish the necessary cleanup to exit gracefully, use the following functions:

- for transfers involving memory: **vlDeregisterBuffer()**, **vlDestroyPath()**, **vlDestroyBuffer()**

- for all transfers: **vlCloseVideo()**

The function prototype for **vlDeregisterBuffer()** is

```
int vlDeregisterBuffer(VLServer vlSvr, VLPath path,
   VLNode memnodeid, VLBuffer ringbufhandle)
```

where

vlSvr   is the server handle

path   is the path handle

memnodeid   is the memory node ID

ringbufhandle   is the VL buffer handle

The function prototypes for **vlDestroyPath()**, **vlDestroyBuffer()** and **vlCloseVideo()** are, respectively,

```
int vlDestroyPath(VLServer vlSvr, VLPath path)

int vlDestroyBuffer(VLServer vlSvr, VLBuffer vlBuffer)

int vlCloseVideo(VLServer vlSvr)
```

This example ends a data transfer that used a buffer:

```
vlEndTransfer(vlSvr, path);
vlDeregisterBuffer(vlSvr, path, memnodeid, buffer);
vlDestroyPath(vlSvr, path);
vlDestroyBuffer(vlSvr, buffer);
vlCloseVideo(vlSvr);
```

Example Programs

The directory */usr/people/4Dgifts/impact/examples/dmedia/video/vl* includes a number of example programs. These programs illustrate how to create simple video applications; for example:

*   a simple screen application: *simplev2s.c*

    This program shows how to send live video to the screen.

*   a video-to-memory frame grab: *simplegrab.c*

    This program demonstrates video frame grabbing.

*   a memory-to-video frame output *simplem2v.c*

    This program sends a frame to the video output.

*   a continuous frame capture: *simpleccapt.c*

    This program demonstrates continuous frame capture.

49

Note: To simplify the code, these examples do not check returns. However, you should always check returns.

See Chapter 4 for a description of *eventex.c* and Chapter 7 for descriptions of *simpleblend.c* and *simplewipe.c*.

The directory */usr/people/4Dgifts/examples/dmedia/video/vl/OpenGL* contains three example OpenGL programs:

- *contcapt.c*: performs continuous capture using buffering and *sproc*

- *mtov.c:* uses the Silicon Graphics Movie Library to play a movie on the selected video port

- *vidtomem.c*: captures an incoming video stream to memory

Note that these programs differ from the programs with the same names in */usr/people/4Dgifts/impact/examples/dmedia/video/vl*.

Using VL Controls

Video Library (VL) controls enable you to

- specify data transfer parameters, such as the frame rate or count
- specify the capture region and decimation, or output window
- specify video format and timing
- adjust signal parameters, such as hue, brightness, vertical sync, and horizontal sync
- specify sync source

This chapter explains

- VL control type and values
- VL control fraction ranges
- VL control classes
- VL control groupings

Device-independent controls are documented in */usr/include/dmedia/vl.h.* Device-dependent controls for the Indigo[2] IMPACT video option are documented in the header files */usr/include/dmedia/dev_mgv.h* (linked to */usr/include/vl/vl_mgv.h*) and */usr/include/dmedia/dev_impact.h* (linked to */usr/include/vl/vl_impact.h*).

Note: For information on the controls used for specific nodes, see Appendix C, "Indigo2 IMPACT Video Nodes and Their Controls." For information on controls for blending and keying, see Chapter 7, "Blending, Keying, and Transitions."

Table 3-1 lists device-independent VL controls alphabetically, along with their values or ranges.

Table 3-1   Device-Independent Controls for Indigo^2 IMPACT Video

Control	Purpose	Comments
VL_BLEND_A	Input source for foreground (channel A) image	VLNode type derived from **vlGetNode()**; must be one of the source nodes
VL_BLEND_B	Input source for background (channel B) image	VLNode type derived from **vlGetNode()**; must be one of the source nodes
VL_BLEND_A_ALPHA	Input source for foreground (channel A) alpha	
VL_BLEND_B_ALPHA	Input source for background (channel B) alpha	
VL_BLEND_A_FCN	Blend function that controls mixing of foreground (channel A) signals	VL_BLDFCN_ZERO VL_BLDFCN_ONE VL_BLDFCN_A_ALPHA VL_BLDFCN_MINUS_A_ALPHA
VL_BLEND_B_FCN	Blend function that controls mixing of background (channel B) signals	VL_BLDFCN_ZERO VL_BLDFCN_ONE VL_BLDFCN_B_ALPHA VL_BLDFCN_MINUS_B_ALPHA
VL_BLEND_A_NORMALIZE	Follows Porter-Duff model (background [channel B]' pixels premultiplied by their corresponding alphas before blending); premultiplies foreground (channel A) by alpha	1 = off
VL_BLEND_B_NORMALIZE	Follows Porter-Duff model (background [channel A]' pixels premultiplied by their corresponding alphas before blending); premultiplies foreground (channel B) by alpha	0 = off 1 = on
VL_CAP_TYPE	Type of frame(s) or field(s) to capture; see "Interlacing" in Appendix A	
VL_DEFAULT_SOURCE	Default source for the video path	
VL_DEFAULT_DRAIN	Default drain for the video path	
VL_FORMAT	Video format	
VL_FREEZE	Data transfer freeze; suspends transfer at the drain node, used only for video out (FB node)	0 = off 1 = on

Control	Purpose	Comments
VL_OFFSET	On VL_VIDEO nodes, the offset to the active region of the video; on all other nodes, the offset within the video Because the default is 0,0, use negative values to get blanking data	
VL_ORIGIN	Upper left corner of image in drain (usually a window); the offset within the node;	Coordinates; default is 0,0
VL_PACKING	Packing of video data at source or drain	
VL_RATE	Transfer rate in fields or frames	
VL_SIZE	On VL_VIDEO nodes, the size of the video; on all other nodes, the clipped size of the video	
VL_SYNC	Sync mode	VL_SYNC_INTERNAL VL_SYNC_GENLOCK
VL_SYNC_SOURCE	Sets sync source for analog breakout box	Reference input: GEN_PORT Input 1: GEN_DIN1 Input 2: GEN_DIN2
VL_TIMING	Video timing	
VL_WINDOW	Window ID for video in a window (screen node only)	Integer
VL_ZOOM	Zoom and decimation	Screen drain nodes support 7/1, 6/1, 5/1, 4/1, 3/1, 2/1, 1/1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8. Screen source nodes support 1/1 and 1/2. Other nodes support zoom and decimation ratios of 1:1 only, that is, no zoom or decimation.

Note: For information on controls for keying, blending, or wipes, see Chapter 7, "Blending, Keying, and Transitions." For detailed information on using VL_CAP_TYPE, VL_FORMAT, VL_OFFSET, VL_PACKING, VL_RATE, VL_SIZE, and VL_TIMING, see "Setting Parameters for Data Transfer to or From Memory" in Chapter 2.

VL Control Type and Values

The type of VL controls is

```
typedef long VLControlType;
```

Common types used by the VL to express the values returned by the controls are

```
typedef struct __vlControlInfo {
    char name[VL_NAME_SIZE]; /* name of control */
    VLControlType type;      /* e.g. WINDOW, HUE, BRIGHTNESS */
    VLControlClass ctlClass; /* SLIDER, DETENT, KNOB, BUTTON */
    VLControlGroup group;    /* BLEND, VISUAL QUALITY, SIGNAL, SYNC */
    VLNode node;             /* associated node */
    VLControlValueType valueType;   /* what kind of data do we have */
    int valueCount;          /* how many data items do we have */
    int numFractRanges;      /* number of ranges to describe control */
    VLFractionRange *ranges; /* range of values of control */

    int numItems;            /* number of enumerated items */
    VLControlItem *itemList; /* the actual enumerations */
} VLControlInfo;
```

To store the value of different controls, *libvl.a* uses this struct:

```
typedef union {
    VLFraction  fractVal;
    VLBoolean   boolVal;
    int         intVal;
    VLXY        xyVal;
    char        stringVal[96];  /* beware of trailing NULLs! */
    float       matrixVal[3][3];
    uint        pad[24];        /* reserved */
} VLControlValue;

typedef struct {
    int numControls;
    VLControlInfo *controls;
} VLControlList;
```

The control info structure is returned by a **vlGetControlInfo()** call, and it contains many of the items discussed above.

VLControlInfo.number is the number of the *VLControlInfo.node* that the information pertains to. There may be several controls of the same type on a particular node, but usually there is just one.

VLControlInfo.numFractRanges is the number of fraction ranges for a particular control. The names correspond 1-to-1 with the *rangeNames*, up to the number of range names, *numRangeNames*. That is, there may be fewer names than ranges, but never more.

VL Control Fraction Ranges

The VL uses fraction ranges to represent the values possible for a control. A VLFractionRange generated by the VL is guaranteed never to generate a fraction with a zero denominator, or a fractional numerator or denominator.

For a range type of VL_LINEAR, *numerator.increment* and *denominator.increment* are guaranteed to be greater than zero, and the limit is always guaranteed to be *{numerator,denominator}.base*, plus some integral multiple of *{numerator,denominator}.increment*.

The type definition for fraction types in the header file is

```
typedef struct {
    VLRange numerator;
    VLRange denominator;
} VLFractionRange;
```

VL Control Classes

The VL defines control classes for user-interface developers. The classes are hints only; they are the VL developer's idea of how the control is commonly represented in the real world.

```
#define VL_CLASS_NO_UI          0
#define VL_CLASS_SLIDER         1
#define VL_CLASS_KNOB           2
#define VL_CLASS_BUTTON         3
#define VL_CLASS_TOGGLE         4
#define VL_CLASS_DETENT_KNOB    5
#define VL_CLASS_LIST           6
```

In the list above, VL_CLASS_NO_UI is often used for controls that have no user-interface metaphor and are not displayed in the video control panel or saved in the defaults file.

The VL controls can be read-only, write-only, or both. The VL includes these macros:

```
#define VL_CLASS_RDONLY          0x8000  /* control is read-only */
#define VL_CLASS_WRONLY          0x4000  /* control is write-only */
#define VL_CLASS_NO_DEFAULT      0x2000  /* don't save in default files */

#define VL_IS_CTL_RDONLY(x)      ((x)->ctlClass & VL_CLASS_RDONLY)
#define VL_IS_CTL_WRONLY(x)      ((x)->ctlClass & VL_CLASS_WRONLY)
#define VL_IS_CTL_RW(x)          (!(VL_IS_CTL_RDONLY(x) || VL_IS_CTL_WRONLY(x)))
```

The macros test these conditions:

```
#define VL_CLASS_MASK        0xfff

typedef unsigned long VLControlClass; /* from list above */
```

VL Control Groupings

Like control class, control grouping is an aid for the user-interface developer. The groupings are the VL developer's idea of how the controls would be grouped in the real world. These groupings are implemented in the video control panel *vcp*.

The type definition for groupings is

```
typedef char NameString[80];
#define VL_CTL_GROUP_PATH    9    /* Path Controls */
```

The maximum length of a control or range name is VL_NAME_SIZE.

Table 3-2 summarizes the VL control groupings.

Table 3-2   VL Control Groupings

Grouping	Includes controls for...
VL_CTL_GROUP_BLENDING	Blending; for example, VL_BLEND_B_FCN
VL_CTL_GROUP_VISUALQUALITY	Visual quality of sources or drains; for example, VL_H_PHASE or VL_V_PHASE
VL_CTL_GROUP_SIGNAL	Signal of sources or drains; for example, VL_HUE
VL_CTL_GROUP_CODING	Encoding or decoding sources or drains; for example, VL_TIMING or VL_FORMAT
VL_CTL_GROUP_SYNC	Synchronizing video sources or drains; for example, VL_SYNC
VL_CTL_GROUP_ORIENTATION	Orientation or placement of video signals; for example, VL_ORIGIN
VL_CTL_GROUP_SIZING	Setting the size of the video signal; for example, VL_SIZE
VL_CTL_GROUP_RATES	Setting the rate of the video signal; for example, VL_RATE
VL_CTL_GROUP_WS	Specifying the windowing system of the workstation; for example, VL_WINDOW
VL_CTL_GROUP_PATH	Specifying the data path through the system; these controls, often marked with the VL_CLASS_NO_UI, are often internal to the VL, with no direct access for the user
VL_CTL_GROUP_SIGNAL_ALL	Specifying properties of all signals
VL_CTL_GROUP_SIGNAL_COMPOSITE	Specifying properties of composite signals
VL_CTL_GROUP_SIGNAL_CLUT_COMPOSITE	Specifying properties of composite color lookup table (CLUT) controls
VL_CTL_GROUP_KEYING	Specifying properties of chroma or luma keying controls, such as VL_KEYER_FG_OPACITY
VL_CTL_GROUP_PRO	Specifying values not commonly found on the front panel of a real-world video device; for example, a wipe control
VL_CTL_GROUP_MASK	Masking optional bits to extract only the control group

Event Handling

The Video Library (VL) provides several ways of handling data stream events, such as completion or failure of data transfer, vertical retrace event, loss of the path to another client, lack of detectable sync, or dropped fields or frames. The method you use depends on the kind of application you're writing:

- For a strictly VL application, use

    – **vlSelectEvents()** to choose the events to which you want the application to respond

    – **vlAddCallback()** to specify the function called when the event occurs

    – your own event loop or a main loop (**vlMainLoop()**) to dispatch the events

- For an application that also accesses another program or device driver, or if you're adding video capability to an existing X or OpenGL application, set up an event loop in the main part of the application and use the IRIX file descriptor (FD) of the event(s) you want to add.

This chapter explains

- Indigo^2 IMPACT Video VL events

- querying VL events

- creating a VL event loop

- creating a main loop with callbacks

It concludes with an example illustrating a main loop and event loops.

Indigo² IMPACT Video VL Events

This section describes the events that the Indigo^2 IMPACT Video device generates. Each event has a standard header, which can be followed by additional data. The additional data can be accessed through the appropriate structure member of the VLEvent union, specified for each of the events listed below.

The VLEvent union and its structures are found in */usr/include/dmediavl.h.*

The standard header for a VL event contains

- int *reason*: the event ID, such as VLControlChanged

- VLServer *server*: the server from which the event originated

- VLDev *device*: the device from which the event originated

- VLPath *path*: the path on which the event originated

- uint *serial*: the serial number of the last request read from the server connection

- uint *time*: the time at which the event was generated

Note: Hardware-generated events, such as vertical retrace, are not available on pure video source-to-video drain paths. To receive these events, a path must make use of the screen, blender, frame buffer, or memory nodes. A path receives a VLBadPath error from **vlSelectEvents()** if it attempts to register for events it cannot receive.

Table 4-1 summarizes the VL events for Indigo^2 IMPACT Video.

Table 4-1   VL Events for Indigo^2 IMPACT Video

Event	Structure	Description
VLStreamPreempted	vlstreampreempted	Generated when a path is preempted by another path that requires some resource that the first path also requires. The paths may be contending over a node (such as a video drain), a part of a node (such as a dual-link input node or one of the single-link nodes that comprise it), or other resource (such as a connector required to route a path).
		The preempted path is indicated by the path member of the vlstreampreempted structure. Once preempted, the path has a stream usage of VL_READ_ONLY. When the stream becomes available again, the path is downgraded to a control usage of VL_SHARE, unless control usage was at VL_READ_ONLY before the stream was preempted. In this case, the level remains at VL_READ_ONLY.
		A VLStreamAvailable event is delivered when the path can be set up again to a stream usage of VL_SHARE or VL_LOCK.
VLStreamAvailable	vlstreamavailable	Generated when all nodes required by a path become available for setup with a stream usage of VL_SHARE or VL_LOCK. Typically, such a path becomes available when another path that was using the nodes is set up with stream usage VL_READ_ONLY or VL_DONE_USING, or is destroyed. The path in question is indicated by the path member of the vlstreamavailable structure.
		VLStreamAvailable is delivered to all registered paths with a stream usage of VL_READ_ONLY. Consequently, a rare condition can occur in which several paths are set up when they receive this event, so that the last path that was set up "wins."
VLSyncLost	vlsynclost	Generated when a node on a path detects invalid timing. The path on which the timing error occurred is specified by the path member of the vlsynclost structure. Some memory nodes, such as the VGI1 memory nodes, have controls to abort a transfer when they detect invalid timing. In that case, a VLTransferFailed event is generated in its place.

Table 4-1 (continued)   VL Events for Indigo[2] IMPACT Video

Event	Structure	Description
VLSequenceLost	vlsequencelost	Generated when a video unit (field or frame, depending on the capture type) is dropped. The path on which the unit was dropped is specified by the path member of the vlsequencelost structure. If a group of contiguous units is dropped, only one VLSequenceLost event is generated. The client can register for VLTransferComplete events to determine when capture or playback resumes.
		Note that VLSequenceLost represents a "soft" error and video transfer continues on the path. This event is in contrast to VLTransferFailed, which signals a "hard" error that causes the transfer to abort.
		The event is delivered as soon as the missed unit is detected. Note that for VGI1 memory nodes; this event may not be generated until a valid unit is transferred.
VLControlChanged	vlcontrolchanged	Generated when a control's value changes. In order for a path to receive this event, it must contain the node on which the control resides. The node is specified in the node member of the vlcontrolchanged structure, and the control's ID is specified by the type member. Use vlGetControl to retrieve the new value of the control.
		This event is never delivered to the path causing the event, that is, the path on which vlSetControl was called.
		Note that the vlcontrolchanged structure contains a value member. This member is not currently used and does not contain the new value of the control.
VLTransferComplete	vltransfercomplete	Generated each time a video unit is captured or played back on a path. The video unit is a field or a frame, depending on the capture type. The path on which the event occurred is specified in the path member of the vltransfercomplete structure.
		This event is generated by paths containing memory nodes only. VLTransferComplete is not sent on "jack-to-jack" paths, for example, a video input to video output path.

Table 4-1 (continued)   VL Events for Indigo^2 IMPACT Video

Event	Structure	Description
VLTransferFailed	vltransferfailed	Generated when a catastrophic error occurs while a path is capturing or playing back a video unit. The memory transfer is halted. The path on which the failure occurred is specified by the path member of the vltransferfailed structure. Note that this event is in contrast to the VLSyncLost or VLSequenceLost events, which are generated when noncatastrophic errors are detected.
		This event is generated by paths containing memory nodes only. VLTransferFailed is not sent on "jack-to-jack" paths, for example, a video input to video output path.
VLEvenVerticalRetrace	vlevenverticalretrace	Generated at the vertical retrace for each even field in the video stream. The path on which the event occurred is specified by the path member of the vlevenverticalretrace structure.
		A path must contain a memory, screen, or blender node to receive VLEvenVerticalRetrace events.
VLOddVerticalRetrace	vloddverticalretrace	Generated at the vertical retrace for each odd field in the video stream. The path on which the event occurred is specified by the path member of the vloddverticalretrace structure.
		A path must contain a memory, screen, or blender node to receive VLOddVerticalRetrace events.
VLFrameVerticalRetrace	vlframeverticalretrace	Generated at the vertical retrace for each frame. The path to which the event is delivered is specified by the path member of the vlframeverticalretrace structure.
		A path must contain a memory, screen, or blender node to receive VLFrameVerticalRetrace events.
VLDeviceEvent	vldeviceevent	Generated when the external trigger fires. The event is delivered to all paths registered for it. The path to which an event record is delivered is specified by the path member of the vldeviceevent structure.
		Trigger polarity, trigger line, and other parameters controlling the trigger are specified by controls on the device node.
VLDefaultSource	vldefaultsource	Generated when a **vlSetControl()** on the VL_DEFAULT_SOURCE control changes the default video source. The new source is specified by the node member of the vldefaultsource structure.
		In order for a path to receive this event, it must contain the new default source node.

Table 4-1 (continued)   VL Events for Indigo² IMPACT Video

Event	Structure	Description
VLControlRangeChanged	vlcontrolrangechanged	Generated when the range for a control changes. In order for a path to receive this event, it must contain the node on which the control resides. The node is specified in the node member of the vlcontrolrangechanged structure, and the control's ID is specified by the type member.
VLControlPreempted	vlcontrolpreempted	Delivered to a path that has acquired a node with VL_SHARE control usage (the preempted path) when a path with VL_LOCK control usage (the preempting path) is set up. The preempted path retains VL_SHARE control usage, but is prevented from changing any controls while the preempting path is set up with control usage VL_LOCK. A VLControlAvailable event is sent when the controls are unlocked.
		The node whose controls have been locked is specified by the node member of the vlcontrolpreempted structure. The path containing the node is identified by the path member.
VLControlAvailable	vlcontrolavailable	Delivered to a path whose controls were previously preempted (see VLControlPreempted), when controls are unlocked, that is, when the control usage of the locking path is dropped to VL_SHARE, VL_READ_ONLY, or VL_DONE_USING.
		The node whose controls have been unlocked is specified by the node member of the vlcontrolavailable structure. The path containing the node is identified by the path member.
VLDefaultDrain	vldefaultdrain	Generated when a **vlSetControl()** changes the default video drain to VL_DEFAULT_DRAIN control. The new drain is specified by the node member of the vldefaultdrain structure.
		In order to receive this event, the path must contain the new default drain node.
VLStreamChanged	vlstreamchanged	Generated when the connectivity of the device is changed by **vlSetConnection()**, or by connections generated by the Indigo² IMPACT Video device on a path's behalf. This event is sent to all paths containing the drain node whose input has changed. Paths containing only the source node do not receive this event, nor does the path causing the connectivity change. The affected path is specified by the path member of the vlstreamchanged structure. The affected drain (node, port) pair are specified by the drnnode and drnport members. The new source (node, port) pair is specified by the srcnode and srcport members.
		If the source or drain (node, port) pair cannot be represented on the path because it does not contain the node in question, then the (node, port) pair has the value (VLUnknownNode, VLUnknownPort).

Querying VL Events

General VL event handling routines are summarized in Table 4-2.

Table 4-2   VL Event Handling Routines

Routine	Use
vlGetFD()	Retrieves a file descriptor for a VL server
vlNextEvent()	Obtains the next event; blocks until the next event from the queue is obtained
vlCheckEvent()	Like a nonblocking **vlNextEvent()**, checks to see if you have an event waiting of the type you specify and reads it off the queue without blocking
vlPeekEvent()	Copies the next event from the queue but, unlike **vlNextEvent()**, does not update the queue, so that you can see the event without processing it
vlSelectEvents()	Selects video events of interest
vlPending()	Queries whether there is an event waiting for the application
vlEventToName()	Retrieves the character string with the name of the event; for example, to use in messages
vlAddCallback()	Adds a callback; use for VL events
vlRemoveCallback()	Removes a callback for the events specified if the client data matches that supplied when adding the callback
vlRemoveAllCallbacks()	Removes all callbacks for the specified path and events
vlCallCallbacks()	Creates a handler; used when creating a main loop or using a supplied, non-VL main loop
vlRegisterHandler()	Registers an event handler; use for non-VL events
vlRemoveHandler()	Removes an event handler

The event type is an integer. **vlEventToName()** allows you to get the character string with the name of the event, so that you can use the event name, for example, in messages.

Table 4-3 summarizes VL event masks.

Table 4-3   VL Event Masks

Symbol	Meaning
VLStreamBusyMask	Stream is locked
VLStreamPreemptedMask	Stream was grabbed by another application
VLStreamChangedMask	Stream topology has changed
VLAdvanceMissedMask	Time was already reached
VLSyncLostMask	Irregular or interrupted signal
VLSequenceLostMask	Field or frame dropped
VLControlChangedMask	A control has changed
VLControlRangeChangedMask	A control range has changed
VLControlPreemptedMask	Control of a node has been preempted, typically by another user setting VL_LOCK on a path that was previously set with VL_SHARE
VLControlAvailableMask	Access is now available
VLTransferCompleteMask	Transfer of field or frame complete
VLTransferFailedMask	Error; transfer terminated; perform cleanup at this point, including **vlEndTransfer()**
VLEvenVerticalRetraceMask	Vertical retrace event, even field
VLOddVerticalRetraceMask	Vertical retrace event, odd field
VLFrameVerticalRetraceMask	Frame vertical retrace event
VLDeviceEventMask	Device-specific event, such as a timing change on a node
VLDefaultSourceMask	Default source changed

Call **vlGetFD()** to get a file descriptor usable from *select*(2) or *poll*(2).

Call **vlSelectEvents()** to express interest in one or more event. For example:

```
vlSelectEvents(svr, path, VLTransferCompleteMask);
```

The VLEvent structure returned by vlNextEvent or vlCheckEvent identifies the type of event that occurred and provides additional information on the event; for example, the VLControlChanged event, accompanied by the node on which the control resides and by the new value of the control. These additional pieces of information can be obtained through the members of the VLEvent union corresponding to each event.

Event masks can be Or'ed together. For example:

```
vlSelectEvents(svr, path, VLTransferCompleteMask |
                VLTransferFailedMask);
```

Depending on whether you want to block processing or not, use **vlNextEvent()** (blocking) or **vlCheckEvent()** (nonblocking) to get the next event.

Use **vlPeekEvent()** to see what the next event in the queue is without removing it from the queue. For example, the part of the code that actually gets the event from the event loop uses **vlNextEvent()**, whereas another part of the code that just wants to know about it, for example, for priority purposes, uses **vlPeekEvent()**.

Creating a VL Event Loop

You can set an event loop to run until a specific condition is fulfilled. The routine **vlSelectEvents()** allows you to specify which event the application will receive.

Using an event loop requires creating an *event mask* to specify the events you want. The VL event mask symbols are combined with the bitwise OR operator. For example, to set an event mask to express interest in either transfer complete or control changed events, use

```
VLTransferCompleteMask | VLControlChangedMask
```

To create an event loop, follow these steps:

1. Define the event; for example:

    ```
    VLEvent ev;
    ```

2.   Set the event mask; for example:

```
vlSelectEvents(vlServer, path, VLTransferCompleteMask |
VLControlChangedMask)
```

3.   Block on the transfer process until at least one event is waiting:

```
for(;;){
vlNextEvent(vlServer, &ev);
```

4.   Create the loop and define the choices; for example:

```
switch(ev.reason){
        case VLTransferComplete:
            …
        break;
    case VLControlChanged:
            …
        break;
    }
}
```

Creating a Main Loop With Callbacks

vlMainLoop() is provided as a convenience routine and constitutes the main
loop of VL applications. This routine first reads the next incoming video
event; it then dispatches the event to the appropriate registered procedure.
Note that the application does not return from this call.

Applications are expected to exit in response to some user action. There is
nothing special about **vlMainLoop()**; it is simply an infinite loop that calls
the next event and then dispatches it. An application can provide its own
version of this loop, for example, to test a global termination flag or to test
that the number of top-level widgets is larger than zero before circling back
to the call to the next event.

To specify callbacks, that is, routines that are called when a particular VL
event arrives, use **vlAddCallback()**. Its function prototype is

```
int vlAddCallback(VLServer vlServer, VLEvent * event,
    void * clientdata, VLEventMask events,
    VLCallbackProc callback, void *clientData)
```

Example 4-1 illustrates the use of **vlAddCallback()**.

Example 4-1   Using VL Callbacks

```
main()
{
  …
      /* Set up the mask for control changed events and Stream preempted events */
   if (vlSelectEvents(vlSvr, vlPath, VLTransferComplete | VLStreamPreemptedMask))
         doErrorExit("select events");

   /* Set ProcessEvent() as the callback for VL events */
   vlAddCallback(vlSvr, vlPath, VLTransferCompleteMask | VLStreamPreemptedMask,
                 ProcessEvent, NULL);

   /* Start the data transfer immediately (i.e. don't wait for trigger) */
   if (vlBeginTransfer(vlSvr, vlPath, 0, NULL))
        doErrorExit("begin transfer");

   /* Get and dispatch events */
   vlMainLoop();
}

/* Handle VL events */
void
ProcessEvent(VLServer svr, VLEvent *ev, void *data)
{
   switch (ev->reason)
   {
      case VLTransferComplete:
       /* Get the valid video data from that frame */
          dataPtr = vlGetActiveRegion(vlSvr, transferBuf, info);
       /* Done with that frame, free the memory used by it */
             vlPutFree(vlSvr, transferBuf);
             frameCount++;
    break;

    case VLStreamPreempted:
        fprintf(stderr, "%s: Stream was preempted by another Program\n",
        _progname);
        docleanup(1);
    break;

    default:
    break;
    }
}
```

Delete a callback with **vlRemoveCallback()** or **vlRemoveAllCallbacks()**. Their function prototypes are

```
int vlRemoveCallback(VLServer vlServer, VLPath * path,
      VLEventMask events, VLCallbackProc callback, void
      *clientData)
```

```
int vlRemoveAllCallbacks(VLServer vlServer, VLPath * path, VLEventMask events)
```

The functions **vlAddHandler()** and **vlRemoveHandler()** are analogous to **vlAddCallback()** and **vlRemoveCallback()**, respectively. Use them for non-VL events.

In */usr/people/4Dgifts/examples/dmedia/video/vl*, the example program *eventex.c* illustrates how to create a main loop and event loops.

Caution: To simplify the code, this example does not check returns. You should, however, always check returns.

Managing Connections

You can use the Video Library to set up complex paths in Indigo^2 IMPACT Video programs. This chapter explains

- specifying connectivity
- avoiding dynamic switching problems

Specifying Connectivity

The Video Library infers the connections on a path if **vlBeginTransfer()** is called and no drain nodes have been connected using **vlSetConnection()**. This situation simplifies application development for simple paths and supports the existing set of applications that do not use **vlSetConnection()**

Thus, the use of **vlSetConnection()** to specify the path connectivity is optional. The following rules are used in determining the connections:

- For each internal node on the path, all unconnected input ports are connected to the first source node added to the path. Pixel ports are connected to pixel ports and alpha ports are connected to alpha ports.

- For each drain node on the path, all unconnected input ports are connected to the first internal node placed on the path, if there is an internal node, or to the first source node placed on the path. Pixel ports are connected to pixel ports and alpha ports are connected to alpha ports.

Because existing connections are preserved, **vlSetConnection()** can be used to override part of the default routes, as long as all drain nodes remain unconnected.

The function prototype of **vlSetConnection()** is

```
int vlSetConnection(VLServer vlSvr, VLPath path, VLNode source_node, VLPort
source_port, VLNode drain_node, VLPort drain_port, VLBoolean preempt)
```

where

vlSvr	names the server to which the path is connected.
path	specifies the data path whose connectivity is being changed.
source_node	names the node that data will flow from. This is either a source or internal node.
source_port	specifies the port on the source node to use.
drain_node	specifies the node that data will flow to. This is either a drain or internal node.
drain_port	specifies the port on the drain node to use.
preempt	specifies whether other paths should be preempted (TRUE) or not (FALSE) in order to route the connection.

Connections must be specified only if

- the path contains multiple internal nodes; in this case, the ordering of the internal nodes is ambiguous and may not be inferred properly by the Video Library

- the default connections, described below, are not the ones that the application desires

- the application wants to change a path's topology after the path has started transferring (note that the change in hardware route may cause a timing glitch in the video stream, depending on the device)

Connections are set up one at a time using **vlSetConnection()** and take effect either immediately or at the next vertical interval, depending on the device. In other words, if **vlSetConnection()** completes successfully, the hardware connection has been established.

Paths may be preempted in order to set a connection since scarce connector resources may be required to route a connection from the source (node, port) to the drain (node, port). The ability to preempt a path follows the rules for stream usage defined with **vlSetupPaths()**. The ability to set a connection, as opposed to only getting (retrieving) it, follows the rules for control usage.

This example fragment sets the blender node's foreground pixel input, that is, input A, to come from the frame buffer node output:

```
if (vlSetConnection(vlSvr, path, fb_node, VL_IMPACT_PORT_PIXEL_SRC,
blender_node, VL_IMPACT_PORT_PIXEL_DRN_A, FALSE) < 0)
{
    vlPerror(_progName);
    exit(1);
}
```

If **vlSetConnection()** returns with -1, an error has occurred. In addition to the standard error codes, the following have special meaning for **vlSetConnection()**:

VLNoRoute   No physical route could be found from the source to the drain.

VLPathInUse   A physical route exists between the source and drain, but the required connector resources are in use. The application has requested that no paths be preempted by specifying FALSE as the *preempt* parameter, or another path has the resources locked.

Getting Connections

Use **vlGetConnection()** to retrieve the connections originating or terminating at a given (node, port). Its function prototype is

```
int vlGetConnection(VLServer vlSvr, VLPath path, VLNode
node, VLPort port, VLNode *nodelist, VLPort *portlist, int
*n)
```

where

vlSvr   specifies the server the application is connected to

path   specifies the path whose connectivity is being checked

node   specifies the node on the path

port   specifies the port on the node

nodelist   is an array of VLNode where the connected nodes will be returned

73

portlist is an array of VLPort where the connected ports are
returned

*n specifies the size of the nodelist and portlist arrays; on exit,
*n is updated to reflect the actual number of elements filled
in

On successful exit, each (nodelist[i], portlist[i]) pair specifies one connection
to (node, port). If port is a source port, then (nodelist[i], portlist[i]) specifies
the drain ports it broadcasts to. If the port is a drain port, then (nodelist[0],
portlist[0]) specifies the input. Because a drain port can have only one input,
only the first entry is used.

If **vlGetConnection()** returns with -1, an error has occurred. In addition to
the generic error codes, the error code VLNotEnoughSpace has special
meaning to **vlGetConnection()**. It indicates that the array size, **n*, is too
small to contain the list of connections. **n* is updated to reflect the required
array length.

The fragment in Example 5-1 illustrates **vlGetConnection()**.

Example 5-1 **vlGetConnection()** Example

```
/*
 * Connect to the video daemon
 */
svr = vlOpenVideo("");

/*
  * Acquire video source and drain nodes.
  */
srcnode = vlGetNode(svr, VL_SRC, VL_VIDEO, VL_ANY);
drnnode = vlGetNode(svr, VL_DRN, VL_VIDEO, VL_ANY);

/*
 * Create a path with these nodes.
 */
path = vlCreatePath(svr, VL_ANY, srcnode, drnnode);

/*
 * Connect the two nodes. Since this is a simple path with obvious
 * video routing, this step is optional. If the path contained
 * multiple internal nodes or the video route was not obvious, then
 * we would need to explicitly state the connections.
```

```
 */
vlSetConnection(svr, path, srcnode, VL_IMPACT_PORT_PIXEL_SRC, drnnode,
VL_IMPACT_PORT_PIXEL_DRN, FALSE);

/*
 * Begin the transfer
 */
vlBeginTransfer(svr, path, 0, NULL);
```

Figure 5-1 diagrams the Indigo^2 IMPACT Video board architecture and shows which data can flow between blocks.
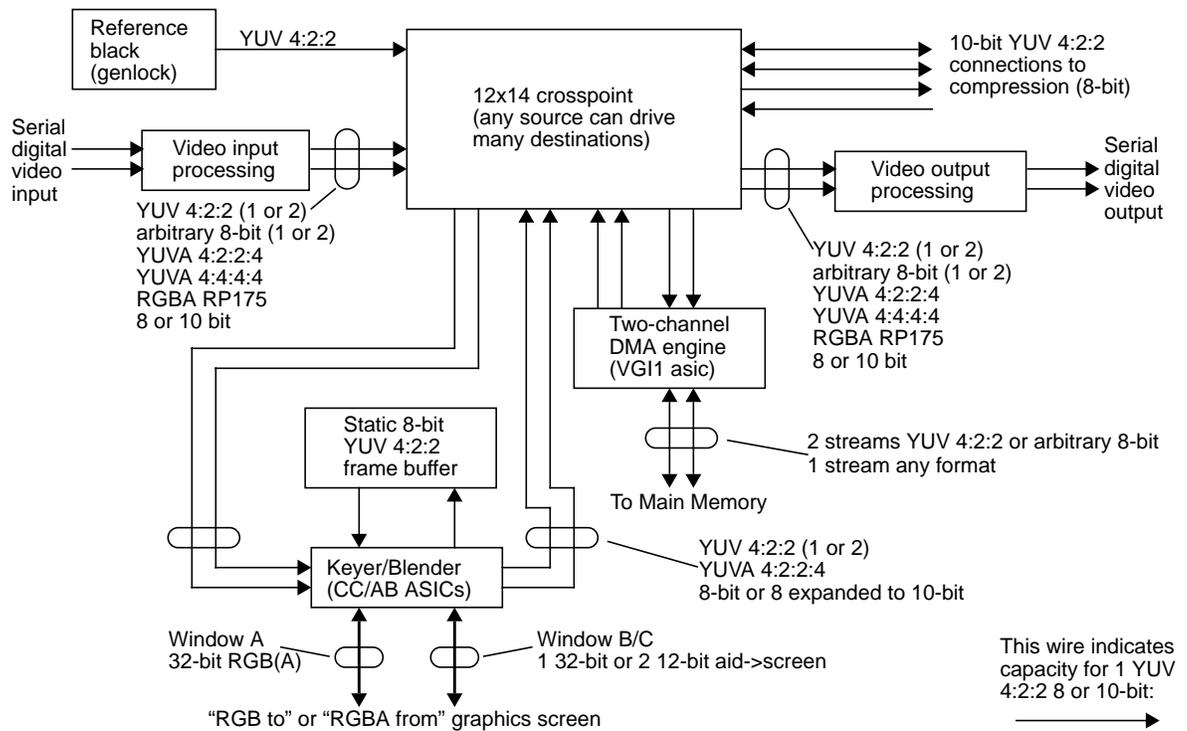
Figure 5-1   Hardware Representation

Figure 5-2 is a software model of Indigo² IMPACT Video node connectivity. Each multiplexer (mux) is a full crossbar: any source can simultaneously broadcast to all destinations. An ellipsis joining two links indicates that the two links can be used either as separate nodes or as a single dual-link mode.
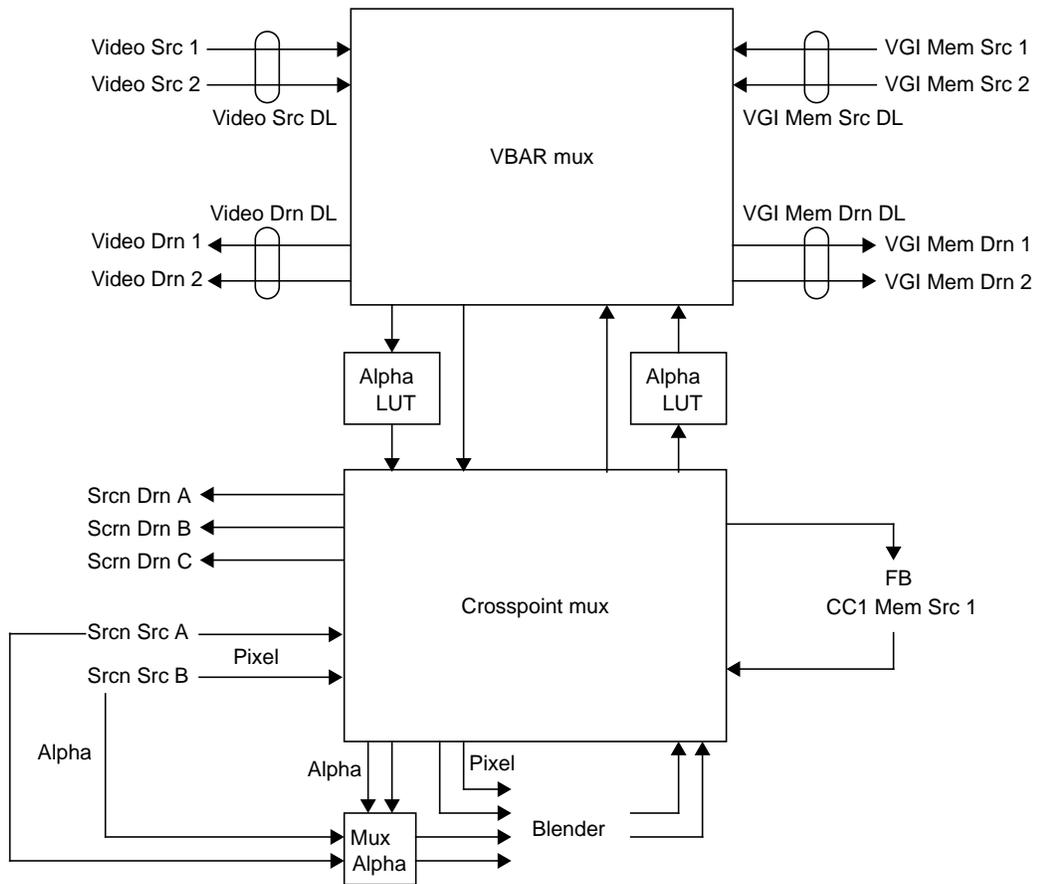
Figure 5-2   Software Representation

The link with the alpha is called the pixel-alpha link, since it can carry pixel or alpha streams. See the VL_MGV_ALPHA_LUT control in "VL_DEVICE" in Appendix C for an explanation of this LUT.

- At most two video streams can flow from the VBAR mux to the crosspoint mux, and at most two video streams can flow from the crosspoint mux to the VBAR mux. Attempts to set up paths that require more cross-mux streams result in VLPathInUse return codes, or in the path(s) using the links to be preempted.

- Because only one link in each direction has a specific alpha LUT, at most one alpha stream can be routed into and at most on one alpha stream routed out of the crosspoint mux.

- If a path uses a pixel-alpha link for a pixel stream and locks it (that is, sets stream usage to VL_LOCK), no alpha stream can be routed in that direction.

The allocation of the links is as follows:

- If the target is a blender alpha input, the pixel-alpha link is allocated.

- If the target is anything other than a blender alpha input, then the pixel link is used if it is available. Otherwise, the pixel-alpha link is used.

The dual-link video and memory nodes are composed of their single-link counterparts. Consequently, when a dual-link node is used, its single-link counterparts become unavailable. For example, when the dual-link video source is in use, video source 1 and video source 2 become unavailable.

Contention for the component single-link resources by the dual-link and single-link nodes follow normal preemption rules. A path using a dual-link node with stream usage VL_SHARE can be preempted by another path using one of its single-link constituents with stream usage VL_SHARE or VL_LOCK.

The CC1 memory source node and the frame buffer node share the same processing element. Consequently, only one can be in use at a time.

Unlike most sources, which feed directly into one of the muxes, the alpha from a screen source node can be fed only to blender alpha input. This restriction is enforced both by the VL_BLEND_A_ALPHA and VL_BLEND_B_ALPHA controls as well as by **vlSetConnection()**.

Note: When a screen source is designated as source A alpha, the blender can extract alpha from it, because a screen source "contains" alpha. Otherwise, any other pixel stream is 4:2:2. If a source other than screen is designated as source A alpha, the blender uses the Y value as alpha and discards the U and V values.

Avoiding Dynamic Switching Problems

vlSetConnection() allows path topology to be changed while the path is transferring, with certain restrictions. This section explains what connections should and should not be switched during a transfer (dynamic switching). Possible failures include the following:

- Picture glitch: Dynamic switching causes the output for one field to consist partly of the field before the switch (usually the top part), the field after the switch (usually the bottom part), and potentially a few garbage pixels at the transition. This glitch is due to the lack of vertical boundary resampling of the switching. However, there is no timing or synchronization glitch.

- Timing glitch: Dynamic switching causes a temporary loss of synchronization. Picture monitors can tear, VTRs can slew, and downstream equipment hiccups in one way or another but eventually recovers. This glitch is unacceptable for any "live" situation.

In general, a glitch ripples downstream to other nodes. Changing video standards (PAL/NTSC or square pixel/CCIR601) always causes a timing glitch. Table 5-1 shows the effects of changing various controls. Glitches caused by changes in the video routing are explained more fully in the following section.

Table 5-1   Dynamic Effects of Various Video Data Path Controls

At Video Node	Timing Glitch Possible Causes	Picture Glitch Possible Causes	Causes No Glitch
Serial digital video output	Synchronizer bypass, horizontal phase, genlock reference sources	Crosspoint selection to video out; various blanking	Freeze
Serial digital video input	External interruption, autophase mode	Truncation or rounding	N/A

Table 5-1	(continued)	Dynamic Effects of Various Video Data Path Controls	
At Video Node	**Timing Glitch Possible Causes**	**Picture Glitch Possible Causes**	**Causes No Glitch**
VGI1 memory drain	VBAR select to VGI1	Truncation or rounding	N/A
VGI1 memory source	VGI1 timing source	N/A	N/A
Blender and screen source	VBAR to crosspoint mux	Alpha LUT select, various blanking and rounding, internal crosspoint	Keyer/blender controls

As indicated in Figure 5-1, Indigo^2 IMPACT Video has timing restrictions for the crosspoint mux and for the VGI1 memory source.

Crosspoint Mux Timing Restrictions

The two outputs of the VBAR mux going into the crosspoint mux must be locked to within six pixels of each other if they are to be blended or displayed in windows B and C. To do this, make sure that the two video input channels are locked by enabling the input autophaser: set the VL_MGV_AUTO_PHASE to a value other than VL_MGV_AUTOPHASE_OFF. (Autophasing locks the video inputs to each other if, on input, their phase differences are not too great.)

Note: "VL_DEVICE" in Appendix C describes the use of these controls.

When one of the VBAR-to-crosspoint links is not in use, the following is done:

- If the pixel-alpha link is not in use, it is set to the same input as the pixel link.

- If the pixel link is not in use, it is set to the same input as the pixel-alpha link.

- If neither links is in use, the output of the black generator is sent to both links.

This arrangement guarantees that if only one link is in use, the two links are locked. If two links are in use, then the application should ensure that the inputs are locked by enabling the input autophaser.

Note: The autophaser can affect only the two serial digital inputs.

Sending a video signal with bad timing into the crosspoint mux from the VBAR mux may cause the nodes attached to the crosspoint mux to operate inconsistently. These nodes derive master timing from the signal on the pixel-alpha link.

VGI1 Memory Source Timing Restrictions

The VGI1 memory sources derive their timing from the genlock source, or from an internal black generator if genlock is disabled. Consequently, changing controls associated with the genlock source may result in a transfer error on the memory node. Changing the autophase mode, for example, causes a timing glitch in the video stream.

Disconnecting the genlock source while a transfer is in progress causes the transfer to fail.

Video Real-Time Capture and Playback

The Indigo^2 IMPACT Video VGI1 memory nodes are capable of full video-rate capture and playback to the Video Library buffers. This chapter explains how to optimize capture or playback to system memory or disk.

- Video Library buffers
- caching
- buffer alignment
- direct I/O to disk
- **syssgi**
- asynchronous I/O

This chapter concludes with descriptions of examples.

Video Library Buffers

Data transfer between the VL and an application takes place through a VL buffer. This buffer is created through the **vlCreateBuffer()** routine and is associated with a memory node by the **vlRegisterBuffer()** routine. A VL buffer is composed of a number of elements of field or frame size, depending on the capture type. The number of elements comprising a buffer is specified to **vlCreateBuffer()**.

When Indigo^2 IMPACT Video transfers data from the Video Library to an application, it places data in a buffer element and marks the element as *valid*. The application can retrieve the element through the **vlGetNextValid()** and **vlGetLatestValid()** routines. When the application is done, it uses the **vlPutFree()** routine to alert the video device that the buffer element can be reused.

When Indigo[2] IMPACT Video transfers data from the application to the Video Library, the application retrieves an empty buffer using **vlGetNextFree()**. After placing data in it, the application marks the buffer valid using the **vlPutValid()** routine. When the video device is done reading from the buffer, it marks the buffer as free.

All operations on a VL buffer operate in FIFO order. That is, the first element retrieved by **vlGetNextValid()** is the first returned by **vlPutFree()**. This function does not take an element as a parameter and always puts back the oldest outstanding element.

Caching

Use the **vlBufferAdvise()** routine to mark a VL buffer as cacheable or not. It has the following prototype:

```
int vlBufferAdvise(VLBuffer buffer, int advice)
```

where

buffer   specifies the ring buffer to be advised

advice   specifies the type of advisory being made:

- VL_BUFFER_ADVISE_NOACCESS marks the buffer as non-cacheable

- VL_BUFFER_ADVISE_ACCESS marks the buffer as cacheable

Marking the buffer non-cacheable indicates that the CPU cache does not have to be flushed or invalidated when data is read or written to system memory via DMA. However, any access to the buffer through the CPU must then bypass the cache and must always go to system memory. This arrangement can severely degrade the performance of an application that directly manipulates the video data.

Consequently, marking a buffer cacheable or noncacheable is application-dependent. In general:

- If the application manipulates the data, even if it is only to copy the data into or out of another region of system memory, the buffer should be set cacheable. This setting is the default for a buffer.

- If the application does not manipulate data, and all transfer is done strictly through DMA, then performance is optimized by setting the buffer to noncacheable. This is the case, for example, when video is read into the ring buffer and then written directly to disk with raw or direct I/O.

  Note: If raw or direct I/O is not used, the data is first copied into the filesystem cache. In that case, the buffer should be kept cacheable.

Buffer Alignment

The performance of the **memcpy()** and **bcopy()** routines is greatly affected by the alignment of the source and destination buffers. For copy operations between buffers with the same alignment, throughput is approximately 400% greater than between buffers with mismatched alignments. On a 200 MHz R4400 IMPACT system, aligned copies typically yield 80 MB per second. In contrast, unaligned copies yield approximately 21 MB per second.

For **memcpy()** and **bcopy()**, the source and target buffers can be considered aligned if the following condition is met:

```
(src % 4) == (dest % 4)
```

In other words, the source and destination buffer addresses are equally distant from a word boundary.

Because the VL buffers used with the Indigo^2 IMPACT Video device are page-aligned, performance is maximized if the application's buffers are word-aligned. Note that the memory allocation routines such as **malloc()** return double-word (64-bit aligned) buffers.

Direct I/O to Disk

Capture or playback from a disk subsystem can be greatly improved by using direct I/O. Direct I/O bypasses the filesystem's buffer cache, eliminating a data copy and other overhead. The buffer can also be marked noncacheable, yielding further performance gains.

Because the filesystem cache is bypassed, device buffer alignment and block size restrictions fall onto the application. These restrictions can be obtained using

```
fcntl(int fd, F_DIOINFO, struct dioattr *dioattr)
```

The device can, for example, require that the buffer be page-aligned. Disk devices usually require that the buffer's size be a multiple of 512 bytes (the disk sector size), or a multiple of the stripe size.

In addition, device performance can be improved with certain alignments or sizes. For example, a device operating on a non-page-aligned buffer can internally break the request into a nonaligned part and an aligned part, yielding the overhead of two requests instead of one. In striped disk subsystems, performance is usually improved by reading or writing entire stripes at a time.

VL buffer elements used with the Indigo^2 IMPACT Video device are always page-aligned, which satisfies the alignment constraints of most devices.

The VL_MGV_BUFER_QUANTUM control is provided so that an application can specify the block size that should be applied to a video unit. (The video unit is a field or frame, depending on the capture type.) For example, setting this control to 512 rounds the frame or field size, as reported by **vlGetTransferSize()**, up to a multiple of 512. This control should be set to a multiple of the block size returned by *fcntl(fd, F_DIOINFO, ...)*, or to the optimal block size for the device.

When VL_MGV_BUFFER_QUANTUM is set to a value other than 1, the video data is padded at the end with random values. Consequently, it is important to use the same value for VL_MGV_BUFFER_QUANTUM on capture and on playback. Making the value the same can be a problem if a file is copied from one device to another with a different allowable block size. It is recommended that the control be set to a common multiple of the

allowable sizes. For example, 4096 satisfies most devices. Otherwise, the file may need to be reformatted.

syssgi

Some of the standard I/O routines support files sizes only up to 2 GB because file position is expressed as a signed integer. **lseek**, for example, only operates up to a 2 GB range. (Note that it is possible to use the **read** or **write** system calls to read or write past the 2 GB mark, up to the filesystem size).

The **syssgi** system call can be used to read or write raw disk partitions greater than 2 GB when used with the following parameters:

```
int syssgi(int request, int fd, char *data, int blockoffset,
int numblocks)
```

where

request	is SGI_READB for a read operation or SGI_WRITEB for a write operation
fd	is a file descriptor of a character special device, as obtained by the **open** system call
data	points to the buffer to be written from or read to
blockoffset	is the block position where reading or writing should commence
numblocks	is the number of blocks to read or write starting at blockoffset

Note that **syssgi** operates in units of device blocks as opposed to bytes. For disk subsystems, a block is usually 512 bytes, allowing 2^{40} bytes of disk space to be addressed.

As with direct I/O, the application is responsible for ensuring that the data buffer is properly aligned and that block size constraints are followed.

Asynchronous I/O

Asynchronous I/O allows an application to process multiple read or write requests simultaneously. On Silicon Graphics platforms, asynchronous I/O is available through the *aio* facility. The *aio64* facility additionally supports 64-bit file sizes and offsets.

Because multiple I/O requests might be outstanding when asynchronous I/O is used, the round-trip delay between making a request, having it serviced, and issuing another request is removed. Asynchronous I/O also eliminates any process-scheduling delay between these steps. In addition, the device being read from or written to might be able to optimize performance by carrying out the requests simultaneously.

The following points should be kept in mind when using asynchronous I/O with VL buffers:

- The VL buffer is a first-in first-out mechanism. When putting a buffer element back into the buffer using **vlPutValid()**, the "oldest" element retrieved by **vlGetNextFree()** is used. There is no way to specify that a different element should be used.

- Because asynchronous I/O operations can complete out of order, the application may need to keep a list of filled elements. When the oldest element is filled, the application can then call **vlPutValid()** to place it back into the buffer, and check to see if any other elements are also ready.

- The same restriction applies to **vlPutFree()** for elements obtained with **vlGetNextValid()** or **vlGetLatestValid()**.

Caution: Software conversion can severely degrade capture or playback performance.

Capture and Playback Examples

The following examples of real-time capture and playback are available in *~4Dgifts/examples/dmedia/video/impact*:

- *vidtodsk*: video to disk using direct I/O (up to the disk subsystem rate)

- *dsktovid*: disk to video using direct I/O (up to the disk subsystem rate)

- *vidtodsk_aio*: video to disk using asynchronous and direct I/O (up to the disk subsystem rate)

- *dsktovid_aio*: disk to video using asynchronous and direct I/O (up to the disk subsystem rate)

Blending, Keying, and Transitions

This chapter explains how to combine video frame information and computer-generated graphics on the Indigo^2 workstation. Use the VL and the Indigo^2 IMPACT Video board to perform three types of blending:

- Chroma keying: overlaying one image on another by choosing a key color. For example, if chroma keying is set to blue, image A might show through image B everywhere the color blue appears in image B. A common example is the TV weather reporter standing in front of the satellite weather map. The weather reporter, wearing any color but blue, stands in front of a blue background; keying on blue shows the satellite picture everywhere blue appears. Because there is no blue on the weatherperson, he or she appears to be standing in front of the weather map.

- Luma keying: overlaying one image on another by choosing a level of luminance. For example, to overlay bright text (such as a caption) on video, a graphics source is created with the text on a dark background. The video source is made to show through the dark areas of the graphics; the bright text remains on top of the video.

- Transitions: fades, tiles, and wipes, such as single, double, or corner wipes, for which you can set the angle or center.

The choice "Blend/Wipe Node" in the Pro menu of the panel *vcp*, a graphical user interface for VL and the Indigo^2 IMPACT Video board, provides convenient access to blending, keying, and transition controls.

This chapter explains

- the VL blender node
- VL keying
- the keyer

The chapter concludes with descriptions of example application programs that are included in the software.
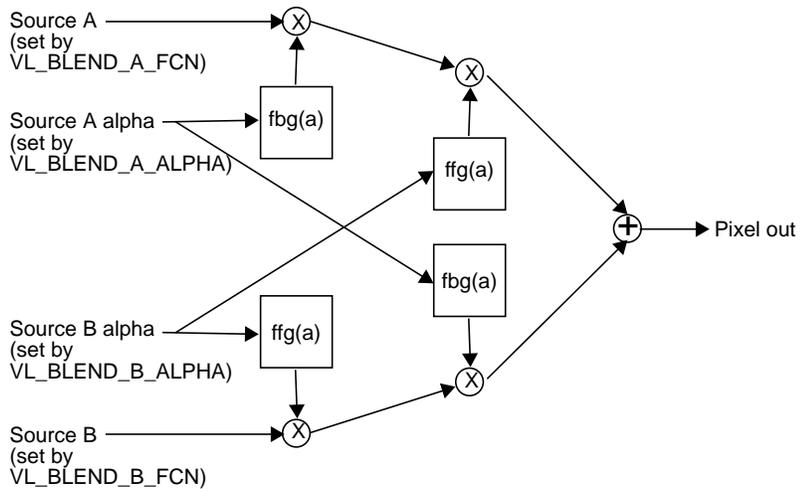
The Blender Node

Blending takes place in the VL's internal *blender node*, which mixes the foreground and background video signals by applying a blend function to the foreground and background pixels.

The blender node is supplied by four independent inputs:

- pixel from a foreground source (A)

- the alpha value for source A

- pixel from a background source (B)

- the alpha value for source B

Figure 7-1 diagrams the blender node.

Figure 7-1   Blender Node

The blender node has four multiplier stages, indicated by $\otimes$ in Figure 7-1, and one adder stage, indicated by $\oplus$. The values in the four multiplier stages are based on the blending functions selected and on the input normalization controls.

Of the four inputs shown in Figure 7-1, two have alternate sources. The Indigo^2 IMPACT Video keyer is hard-wired to alpha source A and the flat-background generator is hard-wired to pixel source B, as diagrammed in Figure 7-2.
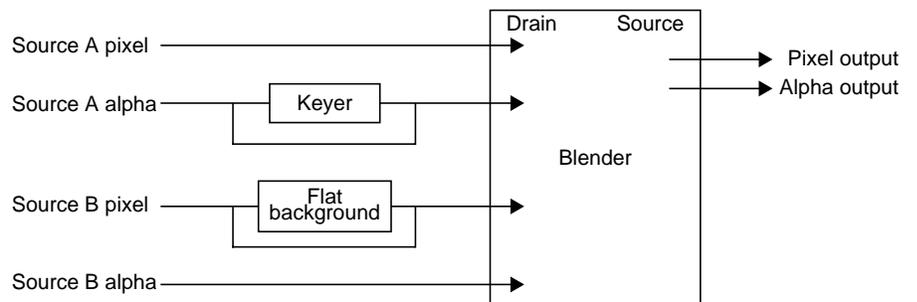
Figure 7-2   Keyer and Flat-Background Generator Locations on Source Nodes

The keyer produces an alpha stream from a pixel stream, generating a key for each pixel in each source node. It is described in detail in "The Keyer" later in this chapter. The flat-background generator sets the background pixel stream (source B pixel) to a default background color or to another color.

Note: When a screen source is designated as source A alpha, the blender can extract alpha from it, because a screen source "contains" alpha. Otherwise, any other pixel stream is 4:2:2. If a source other than screen is designated as source A alpha, the blender uses the Y value as alpha and discards the U and V values.

The rest of this section explains

- setting up the blender node
- setting normalization
- setting and turning off flat background
- adding shadows

Setting Up the Blender Node

Figure 7-3 diagrams setting up the blender node.

```
drn_scr = vlGetNode(vlSvr, VL_DRN, VL_SCREEN, VL_ANY);
drn_vid = vlGetNode(vlSvr, VL_DRN, VL_VIDEO, VL_ANY);
src_scr = vlGetNode(vlSvr, VL_SRC, VL_SCREEN, VL_ANY);
src_vid = vlGetNode(vlSvr, VL_SRC, VL_VIDEO, VL_ANY);
```

```
blend_node = vlGetNode(vlSvr, VL_INTERNAL, VL_BLENDER, VL_ANY);
```
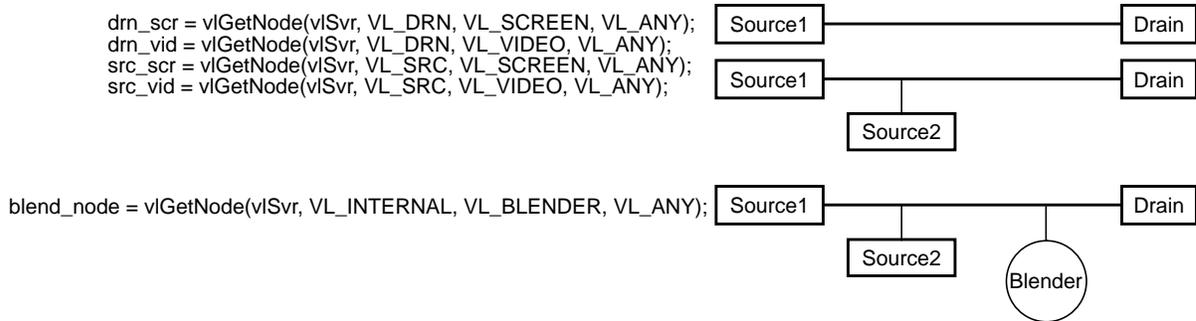

Figure 7-3   Setting Up the Blender Node

The blender node is created with the **vlGetNode()** function. The code fragment in Example 7-1 sets up source, drain, and blender nodes. Notice that the drain nodes are set up before the source nodes.

Example 7-1   Setting Up Source, Drain, and Blender Nodes

```
/* variable definitions */
{
 VLServer vlSvr;
 VLPath path;
 VLNode drn_scr, drn_vid, src_scr, src_vid, blend_node;
}

/* Open a video device */
vlSvr = vlOpenVideo("");

/* Set up drain nodes on the screen and video */
drn_scr = vlGetNode(vlSvr, VL_DRN, VL_SCREEN, VL_ANY);
drn_vid = vlGetNode(vlSvr, VL_DRN, VL_VIDEO, VL_ANY);

/* Set up source nodes on the screen and video */
src_scr = vlGetNode(vlSvr, VL_SRC, VL_SCREEN, VL_ANY);
src_vid = vlGetNode(vlSvr, VL_SRC, VL_VIDEO, VL_ANY);
```

```
/* Set up internal blending node */
blend_node = vlGetNode(vlSvr, VL_INTERNAL, VL_BLENDER,
                       VL_ANY);
```

Table 7-1 summarizes the generic VL blending controls. For all these controls, access is GST:

- G: The value can be retrieved through **vlGetControl().**

- S: The value can be set through **vlSetControl()** while the path is not transferring.

- T: The value can be set through **vlSetControl()** while the path is transferring.

Table 7-1   General Blender Controls

Control	Values	Selects
VL_BLEND_A_FCN type *intVal*	VL_BLDFCN_ZERO VL_BLDFCN_ONE (default) VL_BLDFCN_B_ALPHA VL_BLDFCN_MINUS_B_ALPHA	Blend function that controls mixing of foreground signals, and, with VL_BLEND_B_FCN, provides 16 possible blends of Pixel A and Pixel B, although not all are useful. This control is superseded by **vlSetConnection()**.
VL_BLEND_B_FCN type *intVal*	VL_BLDFCN_ZERO VL_BLDFCN_ONE VL_BLDFCN_A_ALPHA VL_BLDFCN_MINUS_A_ALPHA (default)	Blend function that controls mixing of background signals; see VL_BLEND_B_FCN for more information. This control is superseded by **vlSetConnection()**.
VL_BLEND_A type *intVal*	VLNode type, derived from **vlGetNode()**; must be one of the two pixel source nodes	Sets input source A pixel (foreground).
VL_BLEND_B type *intVal*	Same as for VL_BLEND_A	Sets input source B pixel (background).
VL_BLEND_A_ALPHA type *intVal*	VLNode type, derived from **vlGetNode()**; must be one of the two alpha source nodes	Sets input source A alpha (foreground). If this control is set to a screen node, screen alpha is used.
VL_BLEND_B_ALPHA type *intVal*	Same as for VL_BLEND_A_ALPHA	Sets input source B alpha (background). If this control is set to a screen node, screen alpha is used.

Table 7-1 (continued)   General Blender Controls

Control	Values	Selects
VL_BLEND_A_NORMALIZE type *boolVal*	(0,1) 0 = off (not supported), 1 = on (default)	Selects normalization for A pixel and alpha streams. The pixel and alpha of each stream are applied before the blend operation. Follows Porter-Duff model (background pixels premultiplied by their corresponding alphas before blending)
VL_BLEND_B_NORMALIZE type *boolVal*	(0,1) 0 = off, 1 = on (default)	If set to TRUE, the pixel and alpha of each stream are applied before the blend operation. For screen inputs, set this control to FALSE if you are in VL_MGV_KEYERMODE_NONE mode; alpha has been applied once by the graphics in most cases. Otherwise, leave set to TRUE. Many special effects can be done by altering this control. Follows Porter-Duff model.

Some of the operations are not very useful, for example VL_BLDFCN_ONE, VL_BLDFCN_ONE adds the two images; the hardware clips pixels that are too bright.

Note: When sending the blender output to video, it is best to blank the chroma.

Setting Normalization

You can compose the 12 standard Porter-Duff operations by combining the values of VL_BLEND_A_FCN and VL_BLEND_B_FCN. Figure 7-4 gives some examples of compositing, assuming normalized inputs. Normalized background pixels for a frame are premultiplied by their corresponding alphas before they are blended.
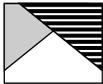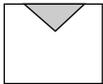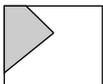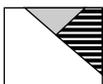
Operation	Diagram	f(A)=	f(B)=
Clear		0	0
A		1	
B			1
A over B		1	1−f(A)
B over A		1−f(B)	1
A in B		f(B)	
B in A			f(A)
A held out by B		1−f(B)	
B held out by A			1−f(A)
A atop B		f(B)	1−f(A)
B atop A		1−f(B)	f(A)
A xor B (union of A out B and B out A)		1−f(B)	1−f(A)
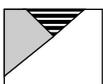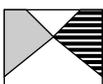
Table derived from Thomas Porter and Tom Duff, "Compositing Digital Images," published by the Association of Computing Machinery, 1984.

Figure 7-4   Binary Compositing

Table 7-2 shows the choices for the two blend functions A and B, which correspond exactly.

Table 7-2   Choices for Blend Functions A and B

Blend Function A	Blend Function B
f(A) = 0.0	f(B) = 0.0
f(A) = 1.0	f(B) = 1.0
f(A) = f(A)	f(B) = f(B)
f(A) = 1 - f(A)	f(B) = 1 - f(B)

The value 0.0 sets the display to black (cut the foreground or background value); the value 1.0 sets the display to white (pass the foreground or background value):

- If both foreground and background are set to 0.0, the result is black (both foreground and background are cut).

- If foreground is set to 0.0 and background is set to 1.0, foreground is cut (ignored) and background is passed (displayed).

- If foreground is set to 1.0 and background is set to 1-f(A), background obscures and overlaps foreground, resulting in compositing.

Normally, chroma is multiplied (scaled) by the selected alpha. For example, the value on source A can be multiplied by its own alpha value or that from source B. In a normal blend, f(A), the incoming alpha of source A is applied to the value for A. In the inverse of this blend, f(A)=1-f(A), the region that was considered opaque (turned off), that is, outside the volume defined for keying, is applied to source A.

In another way of blending, the alpha from source B can be applied to the component represented by source A. In the inverse of this blend, f(A)=1-f(B), the region that was turned off for source B is applied to source A.

For screen inputs, set VL_BLEND_B_NORMALIZE to FALSE if the keyer mode is set to pass-through (VL_KEYERMODE_NONE), because the alpha has been applied once by the graphics in most cases. In other words, set VL_BLEND_B_NORMALIZE to FALSE if it is following another blender.

For foreground-to-background wipes, background alpha is set to a constant value of 1.0, so that the background shows through the foreground.

Setting and Turning Off Flat Background

For Indigo^2 IMPACT Video, pixel source B is normally a flat background, supplied by a flat-background generator on this wire (see Figure 7-2). However, you can use device-dependent controls to set the background as desired:

- VL_MGV_BLEND_B_FLAT (default: off)

  When this control is TRUE (on), the background pixel source is used for pixel timing only and live video from pixel source B goes to the blender.

  When this control is off, you can set three controls listed below to values of your choosing for the background. The default value for the background is gray.

  Note: Set this value before the background is turned on if you wish to avoid a flash.

- VL_MGV_BLEND_B_Y (default: 128)

  The legal range of Y is 16 to 235.

- VL_MGV_BLEND_B_U (default: 128 (50% gray))

  The legal range of U is 16 to 240.

- VL_MGV_BLEND_B_V (default: 128)

  The legal range of V is 16 to 240.

The software does not prevent you from using values outside of the range (1-254):

- The values of 1 and 254 are superblack and superwhite.
- The setting Y=235, U=128, V=128 is 100% white.

Adding Shadows

The shadow hardware adds back the Y information from the area that was cut by the keyer. If a shadow exists in the cut area, the effect is to dim the pixels in the area of that shadow in the replaced background.

Use these device-dependent controls to set the shadow values:

- VL_MGV_BLEND_SHADOW_ON

  Set this control to TRUE to activate the shadow hardware.

- VL_MGV_BLEND_SHADOW_GAIN

  The range for this control is 0.0 to 3.0, which shifts the value.

  Use this control and the next one to make the shadow darker or lighter than the "real" shadow you see in the input video.

  Note: Darkening a very light shadow can result in noise.

- VL_MGV_BLEND_SHADOW_OFFSET

  The range for this control is 0 to 255, which is added to the value.

Shadows that are very dark may be hard to key. These registers affect only the background pixels; foreground areas are passed through by the blender. Tune the values for the best effect.

Keying

For each kind of keying—luma keying, chroma keying, and transitions—
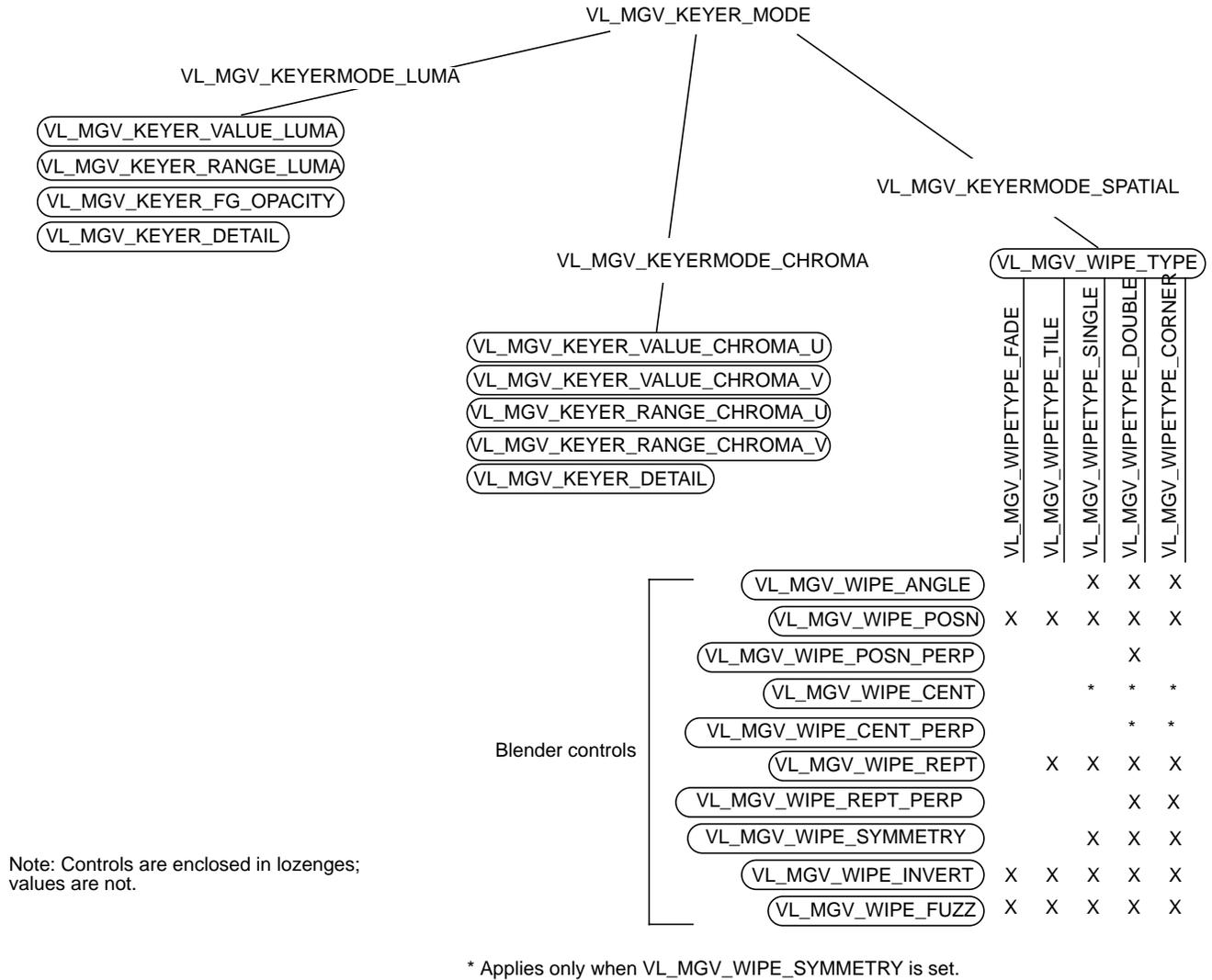further VL controls enable you to specify the properties of the blend.

The values for the Indigo^2 IMPACT Video "master" keyer control,
VL_MGV_KEYER_MODE, determine the type of keying performed:

- luma keying: VL_MGV_KEYERMODE_LUMA

- chroma keying: VL_MGV_KEYERMODE_CHROMA

- transitions, that is, fades, tiles, or wipes:
  VL_MGV_KEYERMODE_SPATIAL

For example, the following fragment specifies a fade:

```
VLControlType val;
val.intVal = VL_MGV_WIPETYPE_FADE;
vlSetControl(vlSvr, vlPath, blend_node, VL_MGV_WIPE_TYPE,
             &val);
```

Each type of keying is explained separately in this section. Figure 7-5 shows
the relationships between the Indigo^2 IMPACT Video board keying and
wipe controls.

VL_MGV_KEYER_MODE

VL_MGV_KEYERMODE_LUMA

(VL_MGV_KEYER_VALUE_LUMA)
(VL_MGV_KEYER_RANGE_LUMA)
(VL_MGV_KEYER_FG_OPACITY)
(VL_MGV_KEYER_DETAIL)

VL_MGV_KEYERMODE_SPATIAL

VL_MGV_KEYERMODE_CHROMA

(VL_MGV_WIPE_TYPE)

(VL_MGV_KEYER_VALUE_CHROMA_U)
(VL_MGV_KEYER_VALUE_CHROMA_V)
(VL_MGV_KEYER_RANGE_CHROMA_U)
(VL_MGV_KEYER_RANGE_CHROMA_V)
(VL_MGV_KEYER_DETAIL)

	VL_MGV_WIPETYPE_FADE	VL_MGV_WIPETYPE_TILE	VL_MGV_WIPETYPE_SINGLE	VL_MGV_WIPETYPE_DOUBLE	VL_MGV_WIPETYPE_CORNER
VL_MGV_WIPE_ANGLE			X	X	X
VL_MGV_WIPE_POSN	X	X	X	X	X
VL_MGV_WIPE_POSN_PERP				X	
VL_MGV_WIPE_CENT			*	*	*
VL_MGV_WIPE_CENT_PERP				*	*
VL_MGV_WIPE_REPT		X	X	X	X
VL_MGV_WIPE_REPT_PERP				X	X
VL_MGV_WIPE_SYMMETRY			X	X	X
VL_MGV_WIPE_INVERT	X	X	X	X	X
VL_MGV_WIPE_FUZZ	X	X	X	X	X

Blender controls

Note: Controls are enclosed in lozenges;
values are not.

* Applies only when VL_MGV_WIPE_SYMMETRY is set.

Figure 7-5   Indigo[2] IMPACT Video Keying, Wipe, and Blender Control Relationships

Luma Keying

Luma keying is typically used to overlay a fixed image on video, such as the name and title of an individual being interviewed, a cable channel's logo, or a symbol that denotes an ongoing news story during a newscast. Figure 7-6 illustrates the results of luma keying.
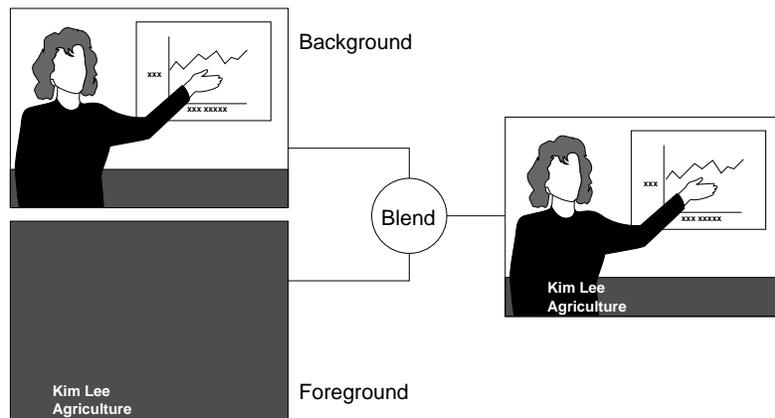
Figure 7-6   Luma Keying Application: Titling

The Indigo^2 IMPACT Video luma keying controls are summarized in Table 7-3. For each, the type is intVal and access is GST. The default value is persistent: it is initially obtained from the defaults file, but is never reset. Many controls available through the video control panel *vcp* (for example, the default video input) fall into this category. For this value, changes made by **vlSetControl()** are persistent across paths, even if the node goes into an unused state.
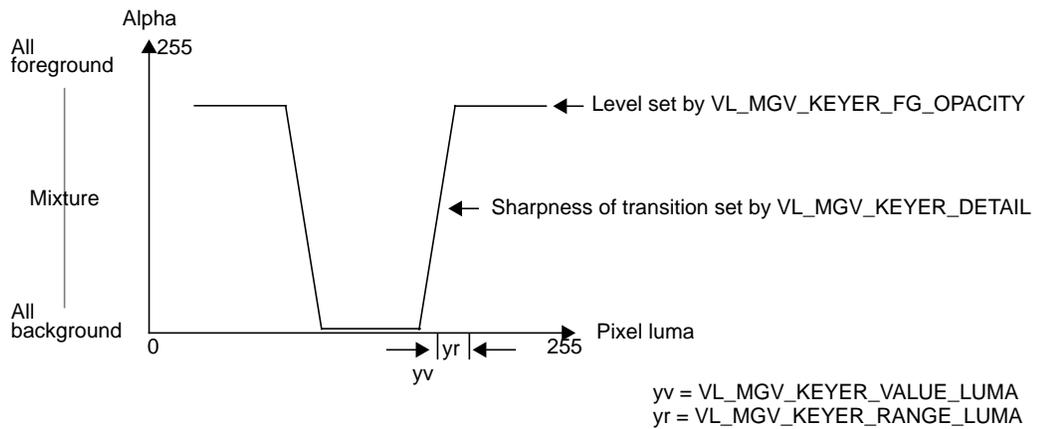
Table 7-3   Indigo^2 IMPACT Luma Keying Controls

Control	Range	Sets
VL_MGV_KEYER_VALUE_LUMA	(0,255)	Central luma value. This control sets the luma value at which the background shows through the foreground.
VL_MGV_KEYER_RANGE_LUMA	(0,255)	One-sided range of the center value. This control determines the range of luma values where the background shows through the foreground.

Table 7-3 (continued)   Indigo^2 IMPACT Luma Keying Controls

| Control	Range	Sets
VL_MGV_KEYER_FG_OPACITY	(0,255)	Opacity of the foreground, thus limiting the value of foreground alpha at any point.
VL_MGV_KEYER_DETAIL, VL_MGV_KEYER_SHARPNESS, VL_MGV_KEYER_FUZZ	(-8,7)	Sharpness of transition between foreground and background allowing blurring of edges. The value -8 yields the most gradual transition, +7 the sharpest.

Figure 7-7 diagrams the relationships between these controls.

Figure 7-7   Relationships Between Indigo^2 IMPACT Video Luma Keying Controls

Chroma Keying

Chroma keying overlays one image on another based on the color value. Figure 7-8 illustrates an example of chroma keying.
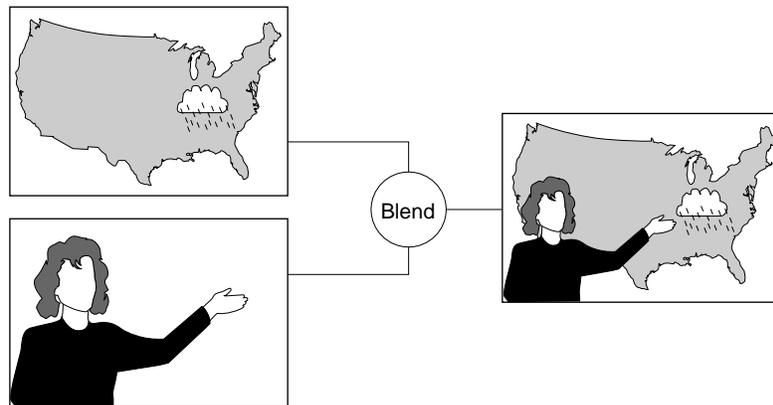
Figure 7-8   Chroma Keying Application: TV Weather Map

Table 7-4 summarizes the controls for Indigo^2 IMPACT video chroma keying and gives their ranges. For each, default is persistent, access is GST, and type is intVal.
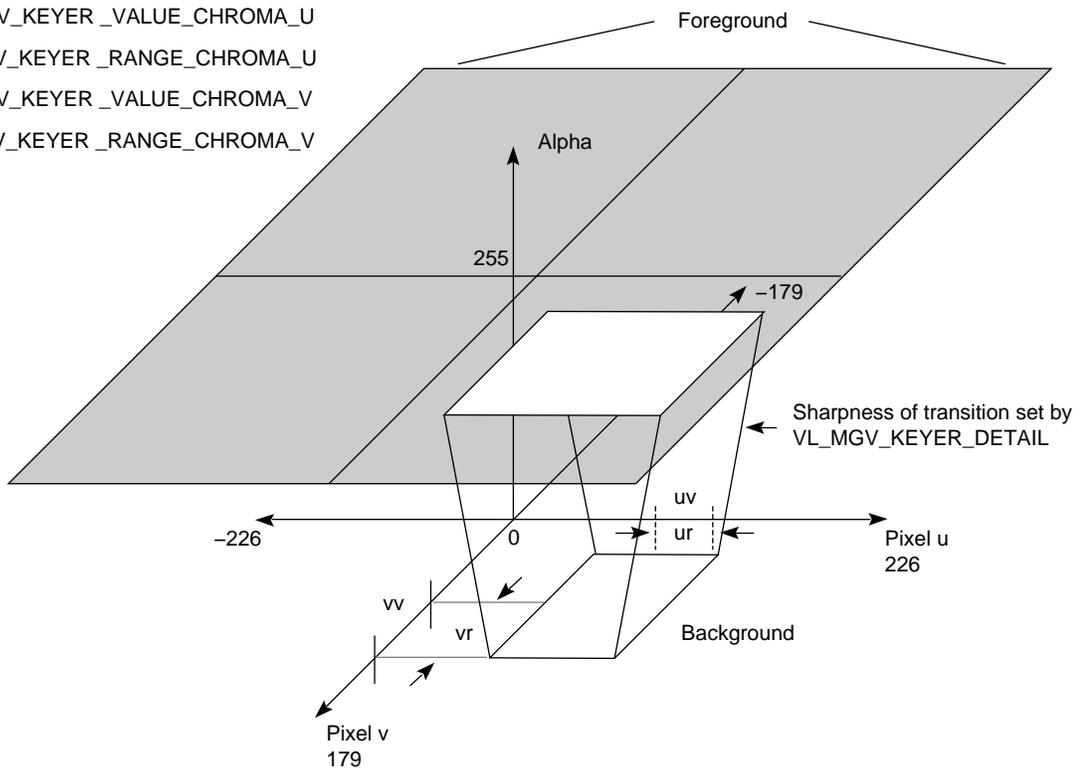
Table 7-4   Indigo^2 IMPACT Video Chroma Keying Controls

Control	Range	Sets
VL_MGV_KEYER_VALUE_CHROMA_U	(-226,226)	Central U value at which the background shows through the foreground.
VL_MGV_KEYER_RANGE_CHROMA_U	(0,452)	One-sided range of U where the background shows through the foreground.
VL_MGV_KEYER_VALUE_CHROMA_V	(-179,179)	Central V value at which the background shows through the foreground.
VL_MGV_KEYER_RANGE_CHROMA_V	(0,358)	One-sided range of V where the background shows through the foreground.
VL_MGV_KEYER_DETAIL, VL_MGV_KEYER_SHARPNESS, VL_MGV_KEYER_FUZZ	(-8,7)	Sharpness of transition between foreground and background

Note: VL_MGV_KEYER_FG_OPACITY has no effect on Indigo[2] IMPACT Video in chroma key mode.

Figure 7-9 diagrams the relationships between these controls.

uv = VL_MGV_KEYER _VALUE_CHROMA_U

ur = VL_MGV_KEYER _RANGE_CHROMA_U

vv = VL_MGV_KEYER _VALUE_CHROMA_V

vr = VL_MGV_KEYER _RANGE_CHROMA_V

Foreground

Alpha

255

−179

Sharpness of transition set by
VL_MGV_KEYER_DETAIL

uv

ur

−226

0

Pixel u
226

vv

vr

Background

Pixel v
179

Figure 7-9   Relationships Between Indigo[2] IMPACT Video Chroma Keying
Controls

Fades, Tiles, and Wipes

The values used with the control VL_MGV_WIPE_TYPE determine the type of blending performed:

- from all-foreground to all-background: VL_MGV_WIPETYPE_FADE

- from all-foreground to all-background by randomly tiling screen with rectangles of a specified size: VL_MGV_WIPETYPE_TILE

- wipe to cross the screen as a vertical, diagonal, or horizontal "front," with a specified angle: VL_MGV_WIPETYPE_SINGLE

- wipe in two orthogonal directions simultaneously (two single wipes at the same time): VL_MGV_WIPETYPE_DOUBLE

- wipe in two orthogonal directions, with the perpendicular position locked to the normal, or in-line position: VL_MGV_WIPETYPE_CORNER

For example, the following fragment specifies that a fade is to be performed:

```
VLControlType val;
val.intVal = VL_MGV_WIPETYPE_FADE;
vlSetControl(vlSvr, vlPath, blend_node, VL_MGV_WIPE_TYPE,
             &val);
```

Fades, tiles, and wipes go from all-foreground (VL_MGV_WIPE_POSN=0) to all-background (VL_MGV_WIPE_POSN=1000), unless VL_MGV_WIPE_INVERT control is set, in which case they go from all-background (VL_MGV_WIPE_POSN = 0) to all-foreground (VL_MGV_WIPE_POSN = 1000).

Table 7-5 summarizes controls common to all wipe types.

Table 7-5   Controls for Fades, Tiles, and Wipes

Control	Values	Sets
VL_MGV_WIPE_POSN type *fractVal*	Numerator (0,1000) Denominator (1000)	Amount of progress of wipe, from none (numerator = 0) to full (numerator = 1000).
VL_MGV_WIPE_REPT type *intVal*	(0,15)	Number of repetitions of pattern in direction of wipe, usually louvers on single, corner, or double wipe, and length of one side of rectangles for a tile wipe. Note that this control does not apply to fades.
VL_MGV_WIPE_INVERT type *intVal*	(0,1) 0 = off, 1 = on	Reversal of foreground and background regions of a wipe. When set to 0, wipes proceed from foreground (position = minimum) to background (position = maximum). When set to 1, wipes proceed from background (position = minimum) to foreground (position = maximum). This value is buffered (does not go into effect) until another blending control is set.

Table 7-6 summarizes the controls specific to wipes or that work differently for wipes. For each, access is GST and the default is persistent, except VL_MGV_WIPE_SYMMETRY and VL_MGV_WIPE_INVERT, for which it is FALSE. Some of these controls work in conjunction with each other.

Table 7-6   Indigo[2] IMPACT Video Controls Specific to Wipes

Control	Values	Sets
VL_MGV_WIPE_DIRECTION (VL_MGV_WIPE_ANGLE) type *intVal*	VL_MGV_WIPEANGLE_E VL_MGV_WIPEANGLE_NE VL_MGV_WIPEANGLE_N VL_MGV_WIPEANGLE_NW VL_MGV_WIPEANGLE_W VL_MGV_WIPEANGLE_SW VL_MGV_WIPEANGLE_S VL_MGV_WIPEANGLE_SE	Wipe vector direction, that is, the direction in which the wipe appears to be proceeding as its position increases. Note: VL_MGV_WIPEANGLE_N and VL_MGV_WIPEANGLE_S do not work for the wipe types VL_MGV_WIPETYPE_DOUBLE and VL_MGV_WIPETYPE_CORNER
VL_MGV_WIPE_FUZZ (VL_MGV_WIPE _SHARPNESS) type *intVal*	(-8,7)	Sharpness of wipe transition band. As for VL_MGV_KEYER_DETAIL, -8 is most gradual, +7 is sharpest.
VL_MGV_WIPE_SYMMETRY type *intVal*	(0,1) 0 = off, 1 = on	Wipe symmetry (on or off) so that wipe proceeds in both directions at once from the center line. Effect depends on type of wipe: no effect for fades or tiling; enables VL_MGV_WIPE_CENT for single, double, and corner wipes; enables VL_MGV_WIPE_CENT_PERP control for double and corner wipes.
VL_MGV_WIPE_POSN_PERP type *fractVal*	numerator (0,1000) denominator (1000)	Amount of progress of wipe, from none (numerator = 0) to full (numerator = 1000), along a direction perpendicular to normal wipe position VL_MGV_WIPE_POSN.
VL_MGV_WIPE_CENT type *fractVal*	numerator (0,1000) denominator (1000)	Offset that is center of a symmetrical wipe along wipe position. 0 means center is where VL_MGV_WIPE_POSN is 0, and 1000 means center is where VL_MGV_WIPE_POSN is 1000. For this control to work for single, double, and corner wipes, VL_MGV_WIPE_SYMMETRY must be on.

Table 7-6 (continued)		Indigo² IMPACT Video Controls Specific to Wipes

| Control	Values	Sets
VL_MGV_WIPE_CENT_PERP type *fractVal*	numerator (0,1000) denominator (1000)	Offset that is center of a symmetrical wipe along a perpendicular wipe position. 0 means center is where VL_WIPE_POSN_PERP is 0, and 1000 means center is where VL_WIPE_POSN_PERP is 1000. VL_WIPE_SYMMETRY must be on for this control to work for double and corner wipes.
VL_MGV_WIPE_REPT_PERP type *intVal*	(0,15)	Number of repetitions perpendicular to wipe direction for single, double, and corner wipes, and length of other side of rectangles for tile wipe.

The Keyer

The role of the keyer is to take a pixel stream and produce an alpha stream. It generates a key for each pixel in each source node:

- If luma keying is set, the keyer assesses the brightness of each pixel.

- If chroma keying is set, the keyer assesses the color of each pixel.

- If spatial, or transition, keying (fade, tile, wipe) is set, the keyer assesses the (x,y) coordinates for each pixel.

The keyer determines the alpha value (opacity) of a pixel and sets a value for it ranging from 0 (completely transparent) to 1 (completely opaque). This alpha value can be used downstream for further layering operations. The program *simpleblend.c* illustrates this procedure; it is included in the software and described at the end of this chapter.

The control VL_MGV_BLEND_H_FILT is a horizontal smoothing filter that, if set to TRUE, filters pixel information before the alpha extraction. It smooths the alpha output of the key generator, softening the edges of the image.

Figure 7-10 shows the relationships between value, range, and detail (transition) for a single channel (for example, A).
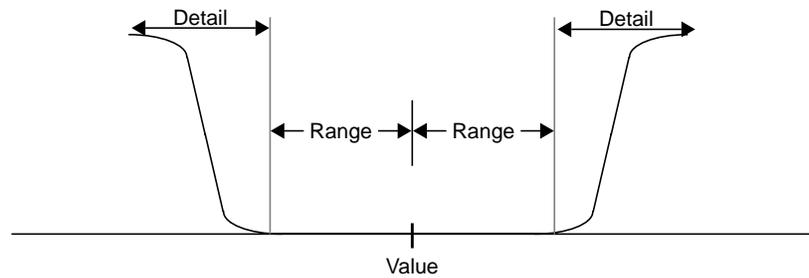
Figure 7-10   Value, Range, and Transition (Keyer Detail) for a Channel

VL Blending Examples

This section explains two example programs from */usr/people/4Dgifts/impact/ examples/dmedia/video/vl*:

- *simpleblend.c*

- *simplewipe.c*

Because the programs are lengthy, they are not duplicated here. Look at the source code in a separate window, or print them out to look at while you read their descriptions.

Caution: To simplify the code, these examples do not check returns. However, you should always check returns.

Blending Video and Graphics

simpleblend.c blends video with graphics and outputs it to both a graphics window and video out. The program

- constrains the window's aspect ratio

- checks that the device the user requested is in the device list

- sets up a path between the source (screen) and the drain (video)

- adds video source and a screen drain nodes to create the blend

- sets the keyer mode, keyer source, and blend controls

- displays the drain window and sets the video to appear in it

- specifies appropriate event handling

- starts data transfer

- specifies that video is updated if the user changes the size of the window

Creating a Simple Wipe Effect

Like *simpleblend.c*, *simplewipe.c* blends video with graphics and outputs it to a graphics window and video out. When the user presses the **w** key, it executes a wipe.

Specifically, in addition to doing everything that *simpleblend.c* does, *simplewipe.c*

- sets up blend parameters (VL_WIPE_TYPE, VL_WIPE_ANGLE or VL_WIPE_DIRECTION, VL_WIPE_CENT, VL_WIPE_REPT)

- calls a loop that sets the keyer mode to spatial and sets the position in the loop; **doswitchloop()** and **dowipe()** execute the loop

- checks for the **w** key and calls **dowipe()**, which in turn calls **doswitchloop()**

Video Basics

Computer graphics and video differ in a number of ways; understanding the differences can help you produce better results with the VL and your Silicon Graphics video option. This appendix introduces some of the important terms and concepts used in conjunction with video. For more detail about a particular term, see the Glossary included in this guide.

Video differs from computer graphics in these ways:
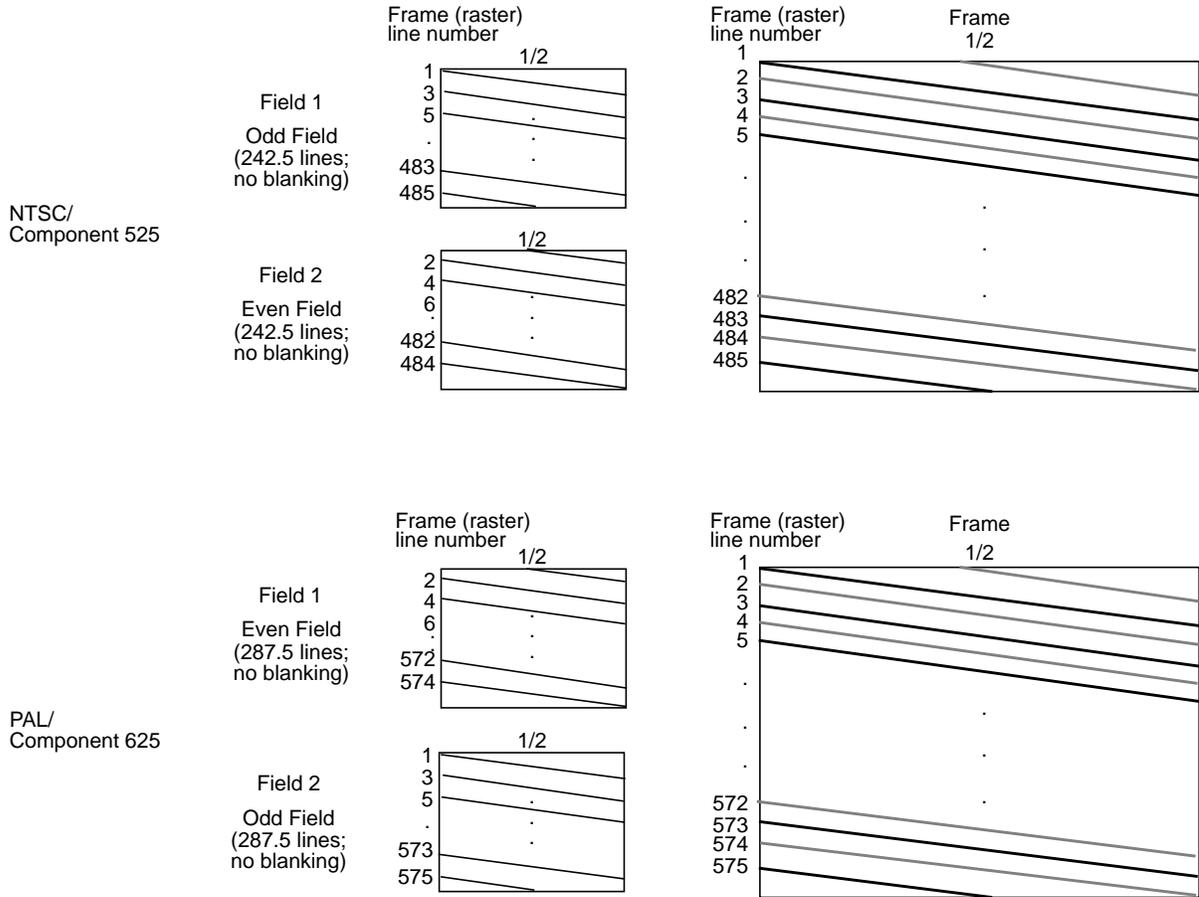
- interlacing
- broadcast standards
- color encoding
- video signals
- tape formats

Interlacing

Most video signals are *interlaced*: each time the video screen is refreshed, only every other one of the horizontal lines are drawn. On the next refresh, the alternate lines are drawn. That is, each *frame* is composed of two *fields*.

During one screen refresh, the video monitor draws the first field, which contains all the odd-numbered lines; during the next refresh, it draws the second field, which contains all the even-numbered lines. Therefore, two refresh cycles are required to draw one frame.

The display rate of interlaced video signals can be measured either in terms of field rate or refresh rate, or in terms of frame rate, which equals half of the field rate, because each frame contains two fields.

Figure A-1 shows a frame and its two fields for NTSC, the broadcast standard used in North America and some other parts of the world, and PAL, the broadcast standard used in much of Europe and elsewhere.

Figure A-1   Fields and Frame

In contrast, the Silicon Graphics workstation monitor is typically noninterlaced: it draws every line each time it refreshes the screen. Refresh rates vary, depending on the type of monitor your Silicon Graphics workstation has. The video output capability of the graphics subsystem for some Silicon Graphics workstation models supports interlaced monitor formats, including component RGB at 525 and 625 lines per frame.

Broadcast Standards

Broadcast standards, or video timing formats, are ways of encoding video information for broadcast to television receivers. These standards are also used to describe the display capabilities of video monitors and are thus also called video timing formats or video output formats (VOFs). The three broadcast standards are:

NTSC   Named after the National Television Systems Committee, which developed it, this standard is used in all of North and South America, except Brazil, and in much of East Asia.

PAL   (Phase Alternated by Line) This standard is used in western Europe, including the United Kingdom but excluding France, and in East Asia, including Australia.

SECAM   (Sequentiel Couleur avec Memoire) This standard is used in France, eastern Europe, the Near East and Mideast, and parts of Africa and the Caribbean.

Note: NTSC implementations can vary slightly by country; PAL and SECAM implementations can vary considerably.

NTSC employs a total of 525 horizontal lines per frame, with two fields per frame of 262.5 lines each. Each field refreshes at 60 Hz (actually 59.94 Hz). NTSC encodes brightness, color, and synchronizing information in one signal.

PAL employs a total of 625 horizontal lines per frame, with two fields per frame of 312.5 lines per frame. Each field refreshes at 50 Hz. PAL encodes brightness, color, and synchronizing information in one signal also, but in a different way from NTSC.

SECAM transmits the same number of lines at the same rate as PAL, but transmits each color difference signal on alternate lines, using the frequency modulation of the subcarrier.

These numbers of horizontal lines—525 and 625, respectively—are a shorthand description of what actually happens. For NTSC, the first (odd) field starts with a whole line and ends with a half line; the second (even) field starts with a half line and ends with a whole line. Each NTSC field contains 242.5 active lines and 20 lines of vertical blanking.

Similarly, for PAL, the first (even) field starts with a half line and ends with a whole line; the second (odd) field starts with a whole line and ends with a half line. Each PAL field contains 287.5 active lines and 25 lines of vertical blanking.

In each case, the numbers 525 and 625 refer to transmitted lines; the active video lines are fewer—typically, 486 for NTSC and 576 for PAL. The remaining lines are used for delimiting frame boundaries and for synchronization and other information.

To minimize frame flickering and reduce the bandwidth of the video signal, the active video lines are *interlaced*, as explained earlier in this chapter.

NTSC and PAL can be recorded digitally; these recording techniques are referred to as D2 525 (digital NTSC) and D2 625 (digital PAL).

Color Encoding

Color-encoding methods are:

- RGB (component)
- YUV (component)
- YIQ (component)
- YC (separate luminance (Y) and chrominance (C)), YC-358, YC-443, S-Video
- composite video

RGB

RGB is the color-encoding method used by most graphics computers, as well as some professional-quality video cameras. The three colors red, green, and blue are generated separately; each is carried on a separate wire.

YUV

YUV, a form of which is used by the PAL video standard and by Betacam[®] and D1 cameras and VCRs, is also a component color-encoding method, but in a different way from RGB. In this case, brightness, or *luminance*, is carried on a signal known as Y. Color is carried on the color difference signals, U and V, which are B-Y and R-Y respectively.

The YUV matrix multiplier derives colors from RGB via the following formula:

$Y = .299R + .587 G + .114 B$
$C_R = R\text{-}Y$
$C_B = B\text{-}Y$

in which Y represents luminance and R-Y and B-Y represent the color difference signals used by this format. In this system, which is sometimes referred to as Y/R-Y/B-Y, R-Y corresponds to C_R and V, and B-Y corresponds to C_B and U. R-Y and B-Y are obtained by subtracting luminance (Y) from the red (R) and blue (B) camera signals, respectively. C_R, C_B, V, and U are derived through different normalization methods, depending on the video format used. The U and V signals are sometimes subsampled by a factor of 2 and then carried on the same signal, which is known as 4:2:2.

YUV component color encoding can be recorded digitally, according to the CCIR 601 standard; this recording technique is referred to as D1.

YIQ

YIQ color encoding, which is typically used by the NTSC video format, encodes color onto two signals called I and Q (for intermodulation and quadrature, respectively). These two signals have different phase modulation in NTSC transmission. Unlike the U and V components of YUV, I and Q are carried on different bandwidths.

The YIQ formula is as follows:

Y = .299 R + .587 G + .114 B (the same as for YUV)
I = .596 R - .275 G - .321 B
Q = .212 R - .523 G + .311 B

YC, YC-358, YC-443, or S-Video

YC, a two-wire signal, results when I and Q are combined into one signal, called chrominance (C). Chrominance is a quadrature phase amplitude-modulated signal. In the NTSC broadcast standard, U is the 0-degree modulation and V is at 90 degrees. In the PAL broadcast standard, the V component is modulated at +/- 90 degrees line-to-line for the active picture and +/- 135 degrees for the reference burst.

 YC-358 is the most common NTSC version of this luminance/chrominance format; YC-443 is the most common PAL version. These formats are also known as S-Video; S-Video is one of the formats used for S-VHS™ videotape recorders.

Composite Video

The composite color-encoding schemes combine the brightness and color signals into one signal for broadcast. NTSC and PAL both combine brightness and color but use different methods.

Figure A-2 shows the relationships between color-encoding methods and video formats.
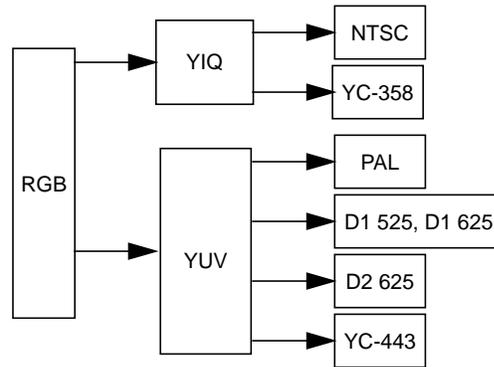
Figure A-2   Relationships Between Color-Encoding Methods and Video Formats

Video Signals

The video signal, whatever the broadcast standard being used, carries other information besides video (luminance and chrominance) and audio. For example, horizontal and vertical synchronization information is required, as well as a color phase reference, which is called color sync burst. Figure A-3 shows a composite video signal waveform.
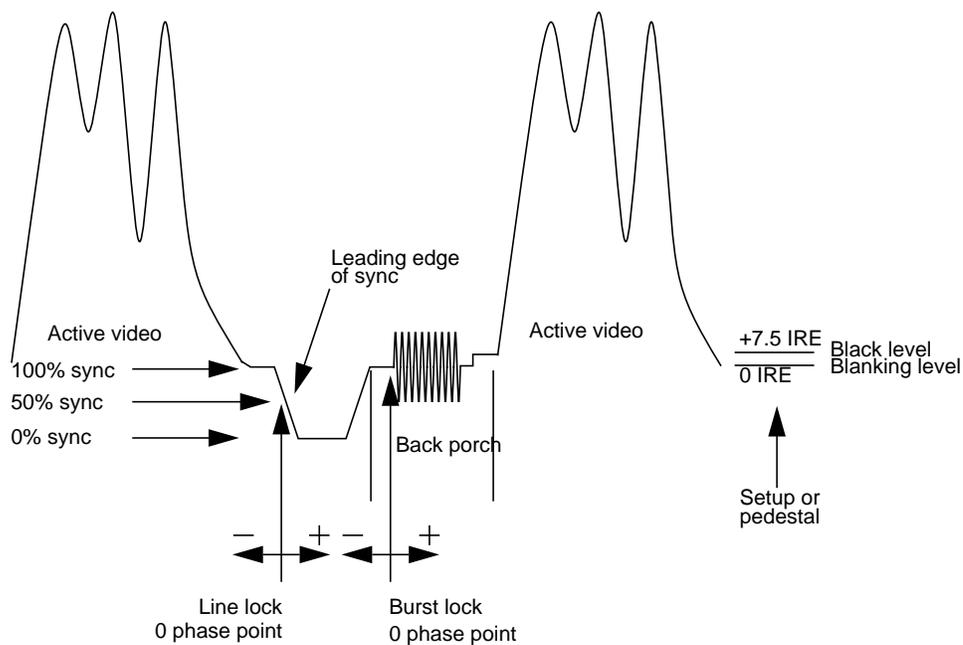
Figure A-3   Composite Video Waveform

Videotape Formats

Videotape recorders are available for analog and digital recording in various formats. They are further classified by performance level or market: consumer, professional, and broadcast. In addition, during postproduction (editing, including addition of graphics), the original footage can be transferred to digital media; digital videotape formats are available for composite and component video formats. There are no official standards for videotape classifications.

Table A-1 summarizes the formats.

Table A-1   Tape Formats and Video Formats

Electronics	Consumer	Professional	Broadcast	Postproduction
Analog	VHS cassette (composite)	U-Matic™ (SP) cassette, 3/4-inch (composite)	Type C reel-to-reel, 1-inch (composite)	
	S-VHS (YC, composite)		Type B (Europe) (composite)	
	S-Video (YC-358)	S-Video (YC-358)		
	Beta (composite)		Betacam (component)	
	8 mm (composite)		Betacam SP (YUV, YIQ, composite)	
	Hi-8mm™ (YC, composite)	Hi-8mm (YC)	MII™ (YUV, YIQ, composite)	
Digital				D1 525 (YUV)
				D1 625 (YUV)
				D2 525 (NTSC)
				D2 625 (PAL)

Although the VL and other software for Silicon Graphics video options do not distinguish between videotape formats, you need to know what kind of connector your video equipment uses. For example, the Galileo board has composite and S-Video connectors.

Most home VCRs use composite connectors. S-Video, on the other hand, carries the color and brightness components of the picture on separate wires; hence, S-Video connectors are also called Y/C connectors. Most S-VHS and Hi-8mm VCRs feature S-Video connectors.

Note: For definitions of video terms, consult the Glossary at the end of this guide.

Return Codes

This appendix explains the return codes that are used with the Video Library for Indigo^2 IMPACT Video. The return code is accessible through the **vlGetErrno()** routine; see also **vlPerror()** and **vlStrError()**.

VLSuccess

> The Video Library routine completed without error.

VLBadAccess

> The client attempted to perform an operation that is illegal given the state of the client, the node, or the path. This error is returned, for example, if the client attempts to add a node to a path that has been set up, or call **vlSetControl()** on a path with control usage est to VL_READ_ONLY.

VLBadAlloc

> The Video Library could not allocate the system resources required for the requested operation, for example, memory and semaphores. If the source of the error is not evident (that is, sufficient physical memory and paging space was present), report this error to technical support.

VLBadAtom

> The server does not recognize the value specified by the atom parameter in the request as a valid atom ID.

VLBadBuffer

> The value of the buffer parameter is not a ring buffer ID recognized by the Video Library.

VLBadControl

> The value specified by the control parameter is not recognized by the node the request was made to.
> The node to which the request was made does not recognize the value specified by the control parameter for **vlSetControl()** or **vlGetControl()**.

VLBadDevice

> The server does not recognize the value specified by the device parameter in the request as a valid device ID. See also `VLBadMatch`.

VLBadImplementation

> An internal processing error occurred. Report the error and the context in which it occurred to Customer Service.

VLBadIoctl

> An error occurred between the video daemon and the device driver associated with the video device. This error can result from an invalid parameter setting in **vlSetControl()**, although it can also represent an internal processing error. This error should be reported to technical support.

VLBadLength

> The video daemon received a request with an invalid length. Report this internal processing error to Customer Support.

VLBadMatch

> The arguments specified for the node, path, or device parameters are not consistent. The node may not reside on the path, or the path may not reside on the device.

VLBadNode

> The server does not recognize the value specified by the node parameter in the request as a valid node ID. See also `VLBadMatch`.

VLBadPath

The server does not recognize the value specified by the
path parameter in the request as a valid path. See also
`VLBadMatch`.

VLBadPort

The value specified by the port parameter is not a
recognized port on the associated node.

VLBadRequest

The daemon has received a bad request code. Report this
internal processing error to technical support.

VLBadServer

The value of the server parameter is not a server ID
recognized by the Video Library.

VLBadSize

The size of the ring buffer elements associated with a
memory node are not compatible with the size of a video
unit (field or frame), given the node's control settings.

VLBadValue

The value of a parameter is invalid. When generated by
vlGetControl(), VLBadValue can indicate that the incorrect
control value type was used, that the value is not within the
range for the control, or that the node cannot accept the
specified value due to a conflict with other node settings.

VLBadWinAlloc

vlSetupPaths() can return this code if there is insufficient
screen space to place a screen source or drain. See
Appendix C, "Indigo2 IMPACT Video Nodes and Their
Controls," for placement constraints.

VLBufferTooSmall

> The size of the ring buffer elements associated with a memory node are smaller than the size of a video unit (field or frame) given the node's control settings.

VLInputsNotLocked

> The processing element associated with a node cannot lock to the input signal. This code may indicate that no signal is present or that the supplied video signal uses a different timing standard than that expected by the node (see VL_TIMING on the input or device node).

VLNoRoute

> **vlSetConnection()** can return this error if no route could be found from the source (node, port) to the drain (node, port). The probable reason is that all connector resources are in use.

VLNotEnoughSpace

> The supplied data region did not contain enough space to hold the information returned by the server.

VLNotSupported

> **vlSetConnection()** or **vlGetConnection()** can return this code if the video device does not support explicit connections.

VLPathInUse

This error is generated if a required resource, for example a node (**vlSetupPaths()**) or a connector (**vlSetConnection()**) cannot be acquired.

- In the case of **vlSetupPaths()**, the node cannot be acquired because the path has requested VL_SHARE stream usage while another path has the required nodes with a stream of VL_LOCK, or the path has requested VL_SHARE control usage while another path has the required nodes with a control usage of VL_LOCK.

- For **vlSetConnection()**, the paths using the required connector could not be preempted because the application has requested that no preemption occur, or because a path using the connector has stream usage set to VL_LOCK.

VLSetupFailed

A general failure occurred during a **vlSetupPaths()** request. If multiple paths were specified for **vlSetupPaths()**, some or none of the paths may have been set up. In addition, some paths may have been preempted in order to set up those paths.

It is recommended that the application set up the paths again to stream usage VL_READ_ONLY and control usage VL_READ_ONLY or VL_SHARE in order to reset the state of all paths. This combination of control and stream usage is guaranteed to succeed.

VLValueOutOfRange

The control value specified for a **vlSetControl()** operation is not within the range accepted by the node. The value was adjusted before being set. (Compare with VLBadValue, where the control's value is not changed at all.) Use the **vlGetControlInfo()** routine to retrieve the valid ranges for the control.

Indigo² IMPACT Video Nodes and Their Controls

This appendix describes the nodes available to Indigo² IMPACT Video. It lists the ports and controls associated with each node, as well as special considerations involved in node usage.

In the tables that summarize the control set for a node, the columns are as follows:

Default   The default value for the control. If the value is Dynamic, the default value depends on the value of other controls. For example, frame size is dependent on device timing. The default value is described in the verbose description of the control.

                 If the value is Persistent, the default value is initially obtained from the defaults file, but is never reset. Many controls available through the video control panel *vcp* (for example, the default video input) fall into this category. For this value, changes made by **vlSetControl()** are persistent across paths, even if the node goes into an unused state.

                 If the default is a specific value or is Dynamic, the control is reinitialized to the default value when the node is no longer in use, that is, when all application paths have been destroyed and the only applications remaining are supervisory. At present, the *vcp* is the only supervisory application.

                 Some controls, such as VL_WINDOW, have a default value of None. This value means that the control *must* be set before a transfer can be started on a path containing the node.

Type   The member of the VLControlValue union used to set or get the value of the control.

Access   Access is one or more of the following:

- G: The value can be retrieved through **vlGetControl().**

- S: The value can be set through **vlSetControl()** while the path is not transferring.

- T: The value can be set through **vlSetControl()** while the path is transferring.

The nodes are

- VL_DEVICE

- VL_BLENDER

- VL_FB: internal frame buffer node for freezing video

- VL_MEM: region of workstation memory

- VL_SCREEN: workstation screen

- VL_VIDEO: connection to a video device; for example, a video tape deck or camera

VL_DEVICE

The device node (digital video source node) provides controls that affect the operation of the Indigo^2 IMPACT Video device as a whole. These controls include global parameters such as timing, as well as default information such as the default source or drain.

- type: VL_DEVICE

- kind: 0

- number: 0

- port: none

Table C-1 lists device node controls. For all these controls, access is GST, except VL_MGV_TRIGGER_WAIT, which is G only.

Table C-1   Device Node Controls

Control	Default	Type	Use
VL_DEFAULT_DRAIN	Persistent	intVal	The VL_DEFAULT_DRAIN control determines the drain node the Video Library selects when a node is acquired with **vlGetNode**(VL_DRAIN, VL_VIDEO, VL_ANY). The value of the control is a video drain node number, as reported by **vlGetDeviceList()**.
			Once a path is set up, the node number is fixed for the lifetime of the path. Consequently, changing this control does not change paths previously set up using a default drain node. Paths can register for the VLDefaultDrain event to be notified when this control's value is changed.
VL_DEFAULT_SOURCE	Persistent	intVal	The VL_DEFAULT_SOURCE control determines the source node the Video Library selects when a node is acquired with **vlGetNode**(VL_SRC, VL_VIDEO, VL_ANY). The value of the control is a video source node number, as reported by **vlGetDeviceList()**.
			Once a path is set up, the node number is fixed for the lifetime of the path. Consequently, changing this control does not change paths previously set up using a default source node. Paths can register for the VLDefaultSource event to be notified when this control's value is changed using **vlSelectEvents()**.

Table C-1 (continued)   Device Node Controls

Control	Default	Type	Use
VL_SYNC	Persistent	intVal	The Indigo[2] IMPACT Video device can derive timing from an external source or can use an internal free-running clock. If VL_SYNC is set to VL_SYNC_INTERNAL, the internal timing source is used. When VL_SYNC is set to VL_SYNC_GENLOCK, timing is derived from an external clock selected by the VL_SYNC_SOURCE control.
VL_SYNC_SOURCE	Persistent	intVal	When the VL_SYNC control is set to VL_SYNC_GENLOCK, this control selects the source of synchronization for the Indigo[2] IMPACT Video device. The device can accept external timing from GEN_PORT analog reference input GEN_DIN1 serial digital input 1 GEN_DIN2 serial digital input 2
VL_TIMING	Persistent	intVal	This control selects the input timing for the Indigo[2] IMPACT Video device and affects the timing for all nodes. The device supports the following modes: VL_TIMING_525_SQ_PIX: NTSC, 525-line square pixel timing VL_TIMING_525_CCIR601: CCIR 601, 525-line non-square pixel timing VL_TIMING_625_SQ_PIX: PAL, 625-line square pixel timing VL_TIMING_625_CCIR601: CCIR 601, 625-line non-square pixel timing Square pixel modes are used with the Indigo[2] IMPACT Compression option only.

Table C-1 (continued)   Device Node Controls

Control	Default	Type	Use
VL_MGV_AUTOPHASE	Persistent	intVal	Autophasing allows the video inputs to be locked to each other if, on input, their phase differences are not too great. This control selects whether input autophasing is enabled and, if enabled, the type of autophasing performed. Options for this control are:
			VL_MGV_AUTOPHASE_NORMAL: This mode synchronizes the video inputs to the reference input. It can accommodate a vertical interval switch with a maximum deviation of +/- 1/2 line from the reference.
			VL_MGV_AUTOPHASE_EXTENDED: This extended autophase mode allows frequency-locked inputs to be +/- 4 lines relative to the reference input.
			VL_MGV_AUTOPHASE_VARIABLE: The variable mode allows the two inputs to be nonsynchronous with the reference and can accommodate inputs that are offset by up to +/- 4 lines. Variable autophasing uses the clock from the "last" input as the clock for both video channels. The last input is determined by monitoring the field (F) bit of channel 1 and 2. If channel 2 is already in the odd field when channel 1 enters the odd field, channel 1 is assumed to be the latest input.
			VL_MGV_AUTOPHASE_OFF: Autophasing is disabled; signals are passed through with their original timing.

Table C-1 (continued)   Device Node Controls

Control	Default	Type	Use
VL_MGV_INPUT _ALPHA_LUT_SELECT	VL_MGV _ALPHA_LUT _CCIR601	intVal	This control selects the type of LUT to be used when video data is routed into the crosspoint mux (see "Getting Connections" in Chapter 5) to a blender alpha input. The LUT is generally used to expand limited-range data to full-range value for use as alpha. The following LUTs are available:
			VL_MGV_ALPHA_LUT_PASS: A pass-through LUT for use when the input is video (not alpha). Values 2-253 are passed unmodified. Input values 254 and 1 are mapped to 255 and 0, respectively.
			VL_MGV_ALPHA_LUT_CCIR601: Used to expand CCIR-range input to full-range input. Values >=235 are mapped to 255, values <=16 are mapped to 0, and values 17-234 are mapped to 1-254.
			VL_MGV_ALPHA_LUT_SUPERBLACK: Used for inputs that have been quantized with setup and the key extends into the blanking level. Values >=235 are mapped to 255, value 1 to 0, and values 2-234 to 1-254.
			VL_MGV_ALPHA_LUT_REDUCED_RANGE: Used for input keys that do not extend the full range. Values >=224 are mapped to 255, values <=32 to 0, and values 32-221 to 1-254. This control takes effect only when video is fed from the VBAR mux to the blender alpha input; when video is fed into a pixel input, then VL_MGV_ALPHA_LUT_PASS is always selected.
VL_MGV_OUTPUT _ALPHA_LUT_SELECT	VL_MGV _ALPHA_LUT _CCIR601	intVal	This control selects the LUT to use when alpha data is routed from the blender out of the crosspoint mux. Its function is to map the full-range alpha values to CCIR or other range values. The values for this control are the same as for VL_MGV_INPUT_ALPHA_LUT_SELECT, with the inverse mappings applied.

<div style="text-align: center;">Table C-1 (continued)   Device Node Controls</div>

Control	Default	Type	Use
VL_MGV_TRIGGER _WAIT Default: none	None	boolVal	Allows a client to "sniff" the GPI trigger. A **vlGetControl()** call with VL_MGV_TRIGGER_WAIT blocks the client application until the trigger goes off. While the application is blocked, events received from the video daemon are queued on the client's connection to the daemon.
			For an application to be notified when the trigger fires, it must be registered for VLDeviceEvent. Using the event mechanism has the advantage that the client application can continue to interact with the Video Library or perform local processing while waiting for the event. VL_MGV_TRIGGER_WAIT is provided for applications that require a faster response to the trigger than can be provided with VLDeviceEvent.
VL_MGV_TRIGGER _LINE	Persistent	intVal	This control reports the line on which the GPI trigger input is sampled. The Indigo^2 IMPACT Video device samples the trigger on a per-field basis at lines 4 and 266 for NTSC (reported as line 4 by **vlGetControl()**) and lines 1 and 313 for PAL (reported as 1 by **vlGetControl()**).
VL_MGV_TRIGGER _POLARITY	Persistent	intVal	This control selects the polarity, indicating that the GPI trigger has gone off. Valid values are POLAR_NEG: The GPI trigger input has become negative. POLAR_POS: The GPI trigger input has become positive.

VL_BLENDER

The blender node provides two-layer blending and keying. The foreground pixel input (PIXEL_DRN_A) can be used as an input to the keyer to provide for chroma and luma key generation. The blender can also be used with external pixel and alpha sources to perform user-defined blend operations:

- type: VL_INTERNAL

- kind: VL_BLENDER

- number: VL_MGV_NODE_NUMBER_BLENDER

- ports:

    - VL_IMPACT_PORT_PIXEL_DRN_A, foreground pixel input

    - VL_IMPACT_PORT_ALPHA_DRN_A, foreground pixel input

    - VL_IMPACT_PORT_PIXEL_DRN_B, background pixel input

    - VL_IMPACT_PORT_ALPHA_DRN_B, background pixel input

    - VL_IMPACT_PORT_PIXEL_SRC_A, pixel output

    - VL_IMPACT_PORT_ALPHA_SRC_B, alpha output

The blender is the only node that can use the screen node alpha information. To use the screen alpha you must route the screen pixel data to blender alpha inputs. The blender alpha output can then be sent to the video outputs or memory.

Note: The blender operates only on YUV:4:2:2 8 bit video. When sending the blender output to video, it is best to blank the chroma.

The Indigo^2 IMPACT Video blender is a Porter-Duff style blender; see "Setting Normalization" in Chapter 7, "Blending, Keying, and Transitions," for more information.

The blender does not stop you from doing special-effects blends. The output is clipped into the standard range. For a nice effect, try looping the output of the blender through a frame buffer back to the input of the blender in various modes.

Note: For more information on blending and keying, see Chapter 7, "Blending, Keying, and Transitions."

Table C-2 lists blender controls. Access for all controls is GST. For more information on blender node controls, see Chapter 7, "Blending, Keying, and Transitions."

Table C-2   Blender Node Controls

Control	Default	Type	Use
VL_BLEND_A_FCN	VL_BLDFCN_MINUS _A_ALPHA	intVal	Sets blend function that controls mixing of foreground signals.
VL_BLEND_B_FCN	VL_BLDFCN_ONE	intVal	Sets blend function that controls mixing of background signals.
VL_BLEND_A	Dynamic	intVal	Sets input source for foreground image.
VL_BLEND_B	Dynamic	intVal	Sets input source for background image.
VL_BLEND_A_ALPHA	Dynamic	intVal	Sets input source for foreground alpha.
VL_BLEND_B_ALPHA	Dynamic	intVal	Sets input source for background alpha.
VL_BLEND_A_NORMALIZE	TRUE	boolVal	Sets normalization; off is not supported by Indigo^2 IMPACT Video.
VL_BLEND_B_NORMALIZE	TRUE	boolVal	Sets normalization, following Porter-Duff model (background pixels premultiplied by their corresponding alphas before blending).
VL_MGV_KEYER_MODE	Persistent	intVal	Selects "master" keyer control that determines the type of keying performed (luma, chroma, or spatial).
VL_MGV_KEYER_DETAIL	Persistent	intVal	Sets sharpness of transition between foreground and background allowing blurring of edges. The value -8 yields the most gradual transition, +7 the sharpest.
VL_MGV_KEYER_FG_OPACITY	Persistent	intVal	Sets opacity of the foreground, thus limiting the value of foreground alpha at any point.
VL_MGV_KEYER_VALUE_LUMA	Persistent	intVal	Sets central luma value. This control sets the luma value at which the background shows through the foreground.

Table C-2   Blender Node Controls

Control	Default	Type	Use
VL_MGV_KEYER_RANGE _LUMA	Persistent	intVal	Sets one-sided range of the center value. This control determines the range of luma values where the background shows through the foreground.
VL_MGV_KEYER_VALUE _CHROMA_U	Persistent	intVal	Sets central U value at which the background shows through the foreground.
VL_MGV_KEYER_RANGE _CHROMA_U	Persistent	intVal	Sets one-sided range of U where the background shows through the foreground.
VL_MGV_KEYER_VALUE _CHROMA_V	Persistent	intVal	Sets central V value at which the background shows through the foreground.
VL_MGV_KEYER_RANGE _CHROMA_V	Persistent	intVal	Sets one-sided range of V where the background shows through the foreground.
VL_MGV_WIPE_SYMMETRY	FALSE	intVal	Sets wipe symmetry (on or off) so that wipe proceeds in both directions at once from the center line. Effect depends on type of wipe: no effect for fades or tiling; enables VL_MGV_WIPE_CENT for single, double, and corner wipes; enables VL_MGV_WIPE_CENT_PERP control for double and corner wipes.
VL_MGV_WIPE_INVERT	FALSE	intVal	Reverses foreground and background regions of a wipe. When set to 0, wipes proceed from foreground (position = minimum) to background (position = maximum). When set to 1, wipes proceed from background (position = minimum) to foreground (position = maximum). This value is buffered (does not go into effect) until another blending control is set.
VL_MGV_WIPE	Persistent	intVal	Sets autowiper on.
VL_MGV_WIPE_TYPE	Persistent	intVal	Selects type of blending (wipe) performed.

Table C-2   Blender Node Controls

Control	Default	Type	Use
VL_MGV_WIPE_ANGLE VL_MGV_WIPE_DIRECTION	Persistent	intVal	Sets wipe vector direction, that is, the direction in which the wipe appears to be proceeding as its position increases. Note: VL_MGV_WIPEANGLE_N and VL_MGV_WIPEANGLE_S do not work for the wipe types VL_MGV_WIPETYPE_DOUBLE and VL_MGV_WIPETYPE_CORNER.
VL_MGV_WIPE_SHARPNESS	Persistent	intVal	Sets sharpness of wipe transition band. As for VL_MGV_KEYER_DETAIL, -8 is most gradual, +7 is sharpest.
VL_MGV_WIPE_FUZZ	Persistent	intVal	Same as VL_MGV_WIPE_SHARPNESS.
VL_MGV_WIPE_SPEED	Persistent	intVal	Sets speed at which the autowiper sweeps the wipe. The value is the speed of the wipe in units of number of fields for each wipe position change.
VL_MGV_WIPE_POSN	Persistent	fractVal	Sets amount of progress of wipe, from none (numerator = 0) to full (numerator = 1000).
VL_MGV_WIPE_POSN_PERP	Persistent	fractVal	Sets amount of progress of wipe, from none (numerator = 0) to full (numerator = 1000), along a direction perpendicular to normal wipe position VL_MGV_WIPE_POSN.
VL_MGV_WIPE_CENT	Persistent	intVal	Sets offset that is center of a symmetrical wipe along wipe position. 0 means center is where VL_MGV_WIPE_POSN is 0, and 1000 means center is where VL_MGV_WIPE_POSN is 1000. For this control to work for single, double, and corner wipes, VL_MGV_WIPE_SYMMETRY must be on.
VL_MGV_WIPE_CENT_PERP	Persistent	intVal	Sets offset that is center of a symmetrical wipe along a perpendicular wipe position. 0 means center is where VL_WIPE_POSN_PERP is 0, and 1000 means center is where VL_WIPE_POSN_PERP is 1000. VL_WIPE_SYMMETRY must be on for this control to work for double and corner wipes.

Table C-2   Blender Node Controls

Control	Default	Type	Use
VL_MGV_WIPE_REPT	Persistent	intVal	Sets number of repetitions of pattern in direction of wipe, usually louvers on single, corner, or double wipe, and length of one side of rectangles for a tile wipe. This control does not apply to fades.
VL_MGV_WIPE_REPT_PERP	Persistent	intVal	Sets number of repetitions perpendicular to wipe direction for single, double, and corner wipes, and length of other side of rectangles for tile wipe.
VL_MGV_WIPE_EXT_TRIG	FALSE	boolVal	If set to TRUE, causes the trigger to initiate an automatic wipe (autowipe).
VL_MGV_WIPE_SPEED	10	intVal	Sets duration of an autowipe in tenths of a second.
VL_MGV_BLEND_B_FLAT	Persistent	intVal	Sets flat-background generator on, so that background pixel source is used for pixel timing only and live video from pixel source B goes to the blender.
VL_MGV_BLEND_B_Y	Persistent	intVal	Sets value of background Y.
VL_MGV_BLEND_B_U	Persistent	intVal	Sets value of background U.
VL_MGV_BLEND_B_V	Persistent	intVal	Sets value of background V.
VL_MGV_BLEND_SHADOW_ON	Persistent	boolVal	Activates shadow hardware. See "Adding Shadows" in Chapter 7 for information.
VL_MGV_BLEND_SHADOW _GAIN	Persistent	intVal	Sets value shift for shadow.
VL_MGV_BLEND_SHADOW _OFFSET	Persistent	intVal	Adds to shadow value. Note that darkening a very light shadow can result in noise.
VL_MGV_BLEND_H_FILT	Persistent	boolVal	Sets horizontal smoothing filter that filters pixel information before the alpha extraction and smooths the alpha output of the key generator.

VL_FB

The frame buffer node provides a mechanism for freezing a video stream. This node is most useful when it is used with the video source nodes, which lack freeze capability. It is also suitable when a snapshot of a video stream is required and the application cannot freeze the input because the live feed is used elsewhere.

Note that the memory and screen source nodes have inherent freeze capability. For the frame buffer node,

- type: VL_INTERNAL
- kind: VL_FB
- number: VL_MGV_NODE_NUMBER_FB
- ports:
    - VL_IMPACT_PORT_PIXEL_SRC_A, 8-bit single-link output
    - VL_IMPACT_PORT_PIXEL_DRN_A, 8-bit single-link input

The frame buffer node imposes a one-frame delay on the video stream.

The frame buffer element is shared between this node and the CC1 memory source node. Consequently, only one of the two can be in use at a time. Attempts to set up both on a path with stream usage VL_SHARE or VL_LOCK result in the first path being preempted. If the frame buffer node and the CC1 memory source node are set up on the same path, an error is returned.

The frame buffer node is internal to the crosspoint mux. Consequently, to avoid consuming the (scarce) VBAR-crosspoint connectors, ensure that its use is required in the path. For example, this node is usually not needed to freeze the video output, since the video drain nodes have freeze capability

The default control for this node is VL_FREEZE. If set to TRUE, this control freezes the video stream passing through the frame buffer. If set to FALSE, live video resumes. For this control, the default is FALSE, type is boolVal, access is GST.

VL_MEM

This discussion divides the VL_MEM nodes into their manifestations as source and drain.

VL_MEM Source

Indigo^2 IMPACT Video supports four memory source nodes: VGI1 1, VGI1 2, VGI1 DL, and CC1. The VGI1 memory sources provide real-time single and dual-link paths from main memory to Indigo^2 IMPACT Video. For the memory source node,

- type for all four memory source nodes: VL_SRC

- kind for all four memory source nodes: VL_MEM

- number: VL_MGV_NODE_NUMBER_VGI_1, VL_MGV_NODE_NUMBER__VGI_2, and VL_MGV_NODE_NUMBER__VGI_DL, and VL_MGV_NODE_NUMBER__CC, respectively

- ports:

  - memory source nodes VGI1 1 and VGI1 2: VL_PORT_PIXEL_SRC_A: single-link 8- or 10-bit video stream capable of real-time operation

  - dual-link video source node: VL_PORT_DUALLINK_SRC_A: dual-link 8- or 10-bit video stream capable of real-time operation

  - memory source node CC1: VL_PORT_PIXEL_SRC_A: single-link 8-bit video stream, no 10-bit support, does not guarantee real-time operation, but has attached frame buffer

    The CC1 node is used mostly to support alpha for the blender. If the blender uses an image from this frame buffer, the Y value is interpreted as alpha and the blender uses it accurately. Data in this buffer allows you to put a nonrectangular shape (for example, a heart or an irregularly shaped logo) as a matte around an image.

The CC1 memory source node is also useful for slide shows or other static-image situations in which video input changes only every 10 or 20 seconds and real-time performance is not critical. Because the CC1 node has its own frame buffer, there is no CPU overhead.

Table C-3 lists memory source node controls. For all these controls, access is GS.

Table C-3   Memory Source Node Controls

Control	Default	Type	Use
VL_CAP_TYPE	CAP_TYPE _INTERLEAVED	intVal	Specifies the type of video units—fields or frames—that the application obtains from the ring buffer. Valid capture types are VL_CAPTURE_NONINTERLEAVED, VL_CAPTURE_INTERLEAVED, VL_CAPTURE_EVEN_FIELDS, VL_CAPTURE_ODD_FIELDS, and VL_CAPTURE_FIELDS. See "Using VL_CAP_TYPE and VL_RATE" in Chapter 2 for information on capture types.
VL_FORMAT	VL_FORMAT_DIGITAL _COMPONENT_SERIAL	intVal	Specifies the type of video format to be produced. See "Using VL_FORMAT" in Chapter 2 for formats and explanations.
VL_PACKING	Single link: VL_PACKING_YVVU _422_8 Dual link: VL_PACKING_YUVA _4444_8	intVal	Specifies the bit order in which the video components are stored in memory. The native packings supported by VL_MGV_NODE_NUMBER_VGI_[1,2] are VL_PACKING_YVVU_422_8 and VL_PACKING_YVVU_422_10. Supported non-native single link packings (implemented automatically in software) are VL_PACKING_Y_8_P, VL_PACKING_RGB_332_P, and VL_PACKING_RGB_8. The native packings supported by VL_MGV_NODE_NUMBER_VGI_DL are VL_PACKING_YUVA_4444_8, VL_PACKING_YUVA_4444_10, VL_PACKING_AUYV_4444_8, VL_PACKING_AUYV_4444_10, VL_PACKING_RGBA_8, VL_PACKING_RGBA_10, VL_PACKING_ABGR_8, VL_PACKING_ABGR_10, and VL_PACKING_AYU_AYV_10. See "Using VL_PACKING" in Chapter 2 for the specifications of each packing.

Table C-3 (continued)   Memory Source Node Controls

Control	Default	Type	Use
VL_OFFSET	(0,0)	xyVal	Specifies the upper left corner of a video region to be output. The coordinates are offsets of the upper left corner of the active video and take precedence over the size. Therefore, in order to accommodate the given offset, the size may be changed. A VLControlChanged event is generated to inform interested parties of any change in size.
VL_RATE	Dynamic; depends on timing and capture type	fractVal	Specifies the rate at which the hardware extracts video units (fields or frames, depending on the capture type) are extracted from the ring buffer. The video unit is repeated, or black is output, to achieve the video output rate of 60 fields per second (NTSC) or 50 fields per second (PAL). The VGI1 memory source nodes can consume video units from system memory at any rate up to the video standard rate.
			For VL_CAPTURE_NONINTERLEAVED and VL_CAPTURE FIELDS, valid ranges are:
			NTSC: 1 through 60 units per second (must be multiple of fields per frame for noninterleaved)
			PAL: 1 through 50 units per second (must be multiple of fields per frame for noninterleaved)
			For VL_CAPTURE_INTERLEAVED, VL_CAPTURE_EVEN_FIELDS, or VL_CAPTURE_ODD_FIELDS, valid ranges are 1 through 30 units per second for NTSC and 1 through 25 units per second for PAL.
VL_SIZE	Dynamic; depends on timing and capture type	xyVal	Specifies the width (pixels) and height (lines) of the video data contained within each ring buffer entry. These values, along with VL_PACKING, determine the size in bytes of each ring buffer entry and thus the transfer size. The width must be a multiple of four pixels. The length must be a minimum of one line for field capture types, and two lines for frames.
			The specified size is constrained by the maximum allowable (as dictated by the device timing) and by the current offset position (VL_OFFSET). If the size is too large, it is reduced. The offset is not changed. It is recommended that VL_OFFSET be set before VL_SIZE.

Table C-3 (continued)   Memory Source Node Controls

Control	Default	Type	Use
VL_TIMING	Dynamic; from device node	intVal	Retrieves the current device-wide video timing value. See "VL_DEVICE" in this chapter for more details. Setting this control on any other node type has no effect.
VL_ZOOM	1.0	fractVal	Specifies the amount of scaling to be applied to the video before it is transferred to memory. The VGI1 memory source nodes have no scaling ability. The only legal value is 1.0.

Table C-3 (continued) Memory Source Node Controls

Control	Default	Type	Use
VL_MGV _DOMINANCE _FIELD	VL_MGV_DOMINANCE _F1	intVal	Sets the field dominance mode, which determines the order in which the fields are read from memory. This control applies only to the frame-oriented capture types (VL_CAPTURE_INTERLEAVED and VL_CAPTURE_NONINTERLEAVED).
			For VL_CAPTURE_INTERLEAVED, values are:
			VL_MGV_DOMINANCE_F1: For video timings VL_TIMING_525_CCIR601 and VL_TIMING_525_SQ_PIX, F1 (also known as odd) dominance dictates that data for the F1 field resides in memory *after* that for F2. For VL_TIMING_625_CCIR601 and VL_TIMING_625_SQ_PIX, the data for F1 resides in memory *before* that of F2.
			VL_MGV_DOMINANCE_F2: For VL_TIMING_525_CCIR601 and VL_TIMING_525_SQ_PIX, F2 (also known as even timings), dominance dictates that data for the F1 field resides in memory *before* that for F2. For VL_TIMING_625_CCIR601 and VL_TIMING_625_SQ_PIX, the data for F1 resides in memory *after* that of F2.
			The meaning of *before* and *after* depends on the capture type. For interleaved frames, *before* indicates that the data comprising the first line of the designated field begins at the first byte of the buffer. In this format, the lines of F1 and F2 are interleaved within the one ring buffer, thus the second line of the buffer belongs to the other field, and so forth.
			For noninterleaved frames, *before* indicates that the dominant field is in a buffer preceding the buffer(s) containing nondominant fields.
			For VL_CAPTURE_NONINTERLEAVED, values are:
			VL_MGV_DOMINANCE_F1: The F1 field is in the first buffer of the pair, and the F2 field in the second.
			VL_MGV_DOMINANCE_F2: The F2 field is in the first buffer of the pair, the F1 field in the second.

Table C-3 (continued)   Memory Source Node Controls

Control	Default	Type	Use
VL_MGV_BUFFER _QUANTUM	1	intVal	The granularity, or quantum, of data transfer required by the application. The video data is padded at the end so that the size of a field/frame is a multiple of VL_MGV_BUFFER_QUANTUM. This control is intended for applications that do I/O directly from the ring buffer, and may consequently require the frame or field size to be a multiple of the device block size. Direct I/O, for example, usually requires that 512 bytes of data be transferred at a time.
VL_MGV_DMA _ERROR_RESTART	VL_MGV_DMA_RESTART _ON	intVal	If enabled (VL_MGV_DMA_RESTART_ON), a video transfer continues when an error is encountered. Otherwise (VL_MGV_DMA_RESTART_OFF), the video transfer is aborted. This control covers three types of errors:
			The reference video timing is not clean, resulting in short/long lines, fields, or both. These errors are with respect to the programmed size and offset.
			The system GIO bus bandwidth was insufficient to transfer video from system memory at video rates.
			The video clock was interrupted.
VL_MGV_DMA _VOUT_EXPAND	VL_MGV_DMA_EXPD _OFF	intVal	Specifies whether or not 8-bit data read from memory is expanded to 10-bit data before being output by the DMA channel to the VBAR mux. If enabled (VL_MGV_DMA_EXPD_ON), then zeroes are inserted into the least significant two bits; otherwise (VL_MGV_DMA_EXPD_OFF), all 10 bits are output unmodified.

Table C-3 (continued)　　　Memory Source Node Controls

Control	Default	Type	Use
VL_MGV_DMA _VOUT _STARVATION	VL_MGV_DMA_VO _STARV_RPT	intVal	Sets the video output policy to use when the memory node underflows the ring buffer (that is, the application has not filled the ring buffer at the rate that the memory node consumes it). An application can choose between two starvation policies. In each case, video output from system memory resumes when the application places the next field/frame in the ring buffer via *vlPutValid()*.
			VL_MGV_DMA_VO_STARV_BLK: Outputs black fields or frames. This choice does not involve further access to memory until a new buffer becomes available.
			VL_MGV_DMA_VO_STARV_RPT: Repeats the last unit (field or frame) that was transferred from main memory. The repetition is performed by continuing to transfer the same field/frame from memory to video until a new buffer becomes available or the transfer is ended. This results in system bus bandwidth continuing to be consumed.
			Caution: In order to maintain compatibility with the behavior of the CC1 memory source node as well as the earlier Galileo Video™ products, where a frame buffer is incorporated, the default value for this control is VL_MGV_DMA_VO_STARV_RPT. Therefore the ring buffer used in the transfer must contain a minimum of two buffer entries (four for VL_CAPTURE_NONINTERLEAVED), so that one buffer can be repeated by the system while the application is filling the second. If only one buffer is used, then the first buffer output is repeated indefinitely and **vlGetNextFree()** never returns a free buffer.

VL_MEM Drain

Indigo[2] IMPACT Video supports three memory drain nodes: VGI1 1, VGI1 2, and VGI1 DL. The VGI1 memory drains provide real-time single- and dual-link paths from the Indigo[2] IMPACT Video device to ring buffers. For the memory drain,

- type for all three memory drain nodes: VL_DRN

- kind for all three memory drain nodes: VL_MEM

- number: VL_MGV_NODE_NUMBER_VGI1_1, VL_MGV_NODE_NUMBER__VGI1_2, and VL_MGV_NODE_NUMBER__VGI1_DL, respectively

- ports:

  - memory drain nodes VGI1 1 and VGI 2: VL_PORT_PIXEL_DRN_A: single-link source for 8- or 10-bit video stream capable of real-time operation

  - dual-link video drain node: VL_PORT_DUALLINK_DRN_A: dual-link source for 8- or 10-bit video stream capable of real-time operation

With the VL_MGV_DMA_VIN_ROUND control enabled, the components of the 10-bit video signal applied to the VGI1 memory drain are rounded to 8 bits; otherwise all 10 bits are passed through and written to memory.

With rounding disabled, setting an 8-bit packing while capturing 10-bit data truncates the data to 8 bits.

Figure C-1 shows the bit relationships for the CCIR 601 8- and 10-bit video format components
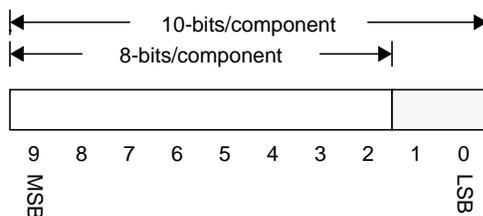
Figure C-1   Rounding for Memory Drain

When rounding is enabled, 10-bit data is converted to 8-bit data depending on the rounding type and the randomized rounding mode. In simple rounding, if bit 1 is set, then the value is rounded up (one is added to bit 2), otherwise it is rounded down.

Randomized rounding involves using a 22-bit shift register to generate two pseudo-random bits to be added to bits 1 and 0 of the 10-bit component, which may or may not result in a carry to bit 2.

The behavior of the shift register is dictated by the randomized rounding mode. With repeated randomized rounding, the shift register is initialized to the same value at the start of each odd (F1) field. Thus, the same pseudo-random sequence will be used for each frame. However, in free-wheel mode, the shift register is never reset and the sequence becomes totally random. The shift register is guaranteed never to become stuck at zero.

Rounding occurs only on active lines and during the digital active line between, and not including, SAV and EAV. The digital blanking data is not modified.

Table C-4 lists memory drain node controls. For all these controls, access is GS.

Table C-4   Memory Drain Node Controls

Control	Default	Type	Use
VL_CAP_TYPE	CAP_TYPE_ INTERLEAVED	intVal	Specifies the type of video units—fields or frames—that the application obtains from the ring buffer by the application. Valid capture types are VL_CAPTURE_NONINTERLEAVED, VL_CAPTURE_INTERLEAVED, VL_CAPTURE_EVEN_FIELDS, VL_CAPTURE_ODD_FIELDS, and VL_CAPTURE_FIELDS. See "Using VL_CAP_TYPE and VL_RATE" in Chapter 2 for information on capture types.
VL_FORMAT	Dynamic	intVal	Specifies the type of video format to be produced. See "Using VL_FORMAT" in Chapter 2 for formats and explanations.

Table C-4 (continued)   Memory Drain Node Controls

Control	Default	Type	Use
VL_PACKING	Single link: VL_PACKING_YVYU _422_8 Dual link: VL_PACKING_YUVA _4444_8	intVal	Specifies the bit order in which the video components are stored in memory. The native packings supported by VL_MGV_NODE_NUMBER_VGI_[1,2] are VL_PACKING_YVYU_422_8 and VL_PACKING_YVYU_422_10. Supported non-native single link packings (implemented automatically in software) are VL_PACKING_Y_8_P, VL_PACKING_RGB_332_P, and VL_PACKING_RGB_8. The native packings supported by VL_MGV_NODE_NUMBER_VGI_DL are VL_PACKING_YUVA_4444_8, VL_PACKING_YUVA_4444_10, VL_PACKING_AUYV_4444_8, VL_PACKING_AUYV_4444_10, VL_PACKING_RGBA_8, VL_PACKING_RGBA_10, VL_PACKING_ABGR_8, VL_PACKING_ABGR_10, and VL_PACKING_AYU_AYV_10. See "Using VL_PACKING" in Chapter 2 for the specifications of each packing.
VL_OFFSET	(0,0)	xyVal	Specifies the upper left corner of a video region to be output. The coordinates are offsets of the upper left corner of the active video and takes precedence over the size. Therefore, in order to accommodate the given offset, the size may be changed. A VLControlChanged event is generated to inform interested parties of any change in size.

Table C-4 (continued)   Memory Drain Node Controls

Control	Default	Type	Use
VL_RATE	Dynamic; depends on timing and capture type	fractVal	Specifies the rate at which video units (fields or frames depending on capture type) the hardware extracts from the ring buffer. The video unit is repeated, or black is output, to achieve the video output rate of 60 fields per second (NTSC) or 50 fields per second (PAL). The VGI1 memory source nodes can consume video units from system memory at any rate up to the video standard rate.
			For VL_CAPTURE_NONINTERLEAVED and VL_CAPTURE FIELDS, valid rates are
			NTSC: 1 through 60 units per second (must be multiple of fields per frame for noninterleaved)
			PAL: 1 through 50 units per second (must be multiple of fields per frame for noninterleaved)
			For VL_CAPTURE_INTERLEAVED, VL_CAPTURE_EVEN_FIELDS, or VL_CAPTURE_ODD_FIELDS, valid ranges are 1 through 30 units per second for NTSC and 1 through 25 units per second for PAL.
VL_SIZE	Dynamic; depends on timing and capture type	xyVal	Specifies the width (pixels) and height (lines) of the video data contained within each ring buffer entry which, along with VL_PACKING, determines the size in bytes of each ring buffer entry and thus the transfer size. The width must be a multiple of four pixels. The length must be a minimum of one line for field capture types, and two lines for frames.
			The specified size is constrained by both the maximum allowable (as dictated by the device timing and capture type) as well as the current offset position (VL_OFFSET). If the size is too large, it is reduced. The offset is not changed. It is recommended that VL_OFFSET be set before VL_SIZE.
VL_TIMING	Dynamic; from device node	intVal	Retrieves the current device-wide video timing value. See "VL_DEVICE" in this chapter for more details. Setting this control on any other node type has no effect.
VL_ZOOM	1.0	fractVal	Specifies the amount of scaling to be applied to the video before it is transferred to memory. The VGI1 memory drain nodes have no scaling ability. The only legal value is 1.0.

| | | Table C-4 (continued) | Memory Drain Node Controls |

Control	Default	Type	Use
VL_MGV_BUFFER _QUANTUM	1	intVal	The granularity, or quantum, of data transfer required by the application. The video data is padded at the end so that the size of a field/frame is a multiple of VL_MGV_BUFFER_QUANTUM. This control is intended for applications that do I/O directly from the ring buffer, and may consequently require the frame or field size to be a multiple of the device block size. Direct I/O, for example, usually requires that 512 bytes of data be transferred at a time.
VL_MGV _DOMINANCE _FIELD	VL_MGV_DOMINANCE _F1	intVal	Sets the field dominance mode, determining the order in which the fields are read from memory. This control applies only to the frame-oriented capture types (VL_CAPTURE_INTERLEAVED and VL_CAPTURE_NONINTERLEAVED. See the discussion of VL_MGV_DOMINANCE_FIELD in Table C-3 earlier in this appendix for more details.
VL_MGV_DMA _ERROR_RESTART	VL_MGV_DMA _ERROR_RESTART_OFF	intVal	If enabled (VL_MGV_DMA_RESTART_ON), a video transfer continues when an error is encountered. Otherwise (VL_MGV_DMA_RESTART_OFF), the video transfer is aborted. This control covers three types of errors: The reference video timing is not clean, resulting in short/long lines, fields, or both. These errors are with respect to the programmed size and offset. The system GIO bus bandwidth was insufficient to transfer video from system memory at video rates. The video clock was interrupted.
VL_MGV_DMA _ROUND_TYPE	VL_MGV_DMA_RND _SMPLE	intVal	Specifies type of rounding algorithm to be used. The simple rounding method (VL_MGV_DMA_RND_SMPLE) rounds up if bit 1 is one or rounds down if bit 1 is zero. The randomized rounding method (VL_MGV_DMA_RND_RAND) makes the decision whether or not to round up based on comparing the two least significant bits to a random sequence.

Table C-4 (continued)   Memory Drain Node Controls

Control	Default	Type	Use
VL_MGV_DMA _RAND_ROUND _MODE	VL_MGV_DMA _RND_RAND _RPT	intVal	Determines whether or not the random sequence used for randomized rounding is repeated. If the sequence is to be repeated (VL_MGV_DMA_RND_RAND_RPT), then a shift register is seeded to a fixed value at the start of each odd field. Otherwise, the shift register free-wheels.
VL_MGV_DMA _VIN_ROUND	VL_MGV_DMA_RND_OFF	intVal	Enables (VL_MGV_DMA_RND_ON) or disables (VL_MGV_DMA_RND_OFF) rounding of 10-bit video data to 8 bits per component. Only the active area data is rounded.

VL_SCREEN

This discussion divides the VL_SCREEN nodes into their manifestations as source and drain.

VL_SCREEN Source

Indigo^2 IMPACT Video supports two screen source nodes: A and B.

The screen source nodes provide a means of using the graphics screen as a source of video data. Both pixel and alpha information can be extracted from the screen source area (although note that the alpha data can be sent only to the blender node's alpha inputs).

- type for both screen drain nodes: VL_SRC

- kind for both screen drain nodes: VL_SCREEN

- number: VL_MGV_NODE_NUMBER_SCREEN_A andVL_MGV_NODE_NUMBER_SCREEN_B, respectively

- ports:

  – VL_IMPACT_PORT_ALPHA_SRC_[A,B], single-link 8-bit CCIR pixel stream derived from a graphics window's alpha contents

  – VL_IMPACT_PORT_PIXEL_SRC_[A,B], single-link 8-bit CCIR pixel stream derived from a graphics window's pixel contents

Certain constraints apply to window positioning. If an application attempts to place the window at an illegal position, the node attempts to place the window at a valid location. The size of the window can also be changed. If either the position or size is changed, the application is notified by a VLValueChanged event. If the window cannot be placed anywhere on the screen, **vlSetControl()** returns VLBadValue.

Make sure your application meets these constraints:

- Windows A and B must not overlap vertically.

- The vertical distance between windows A and B must be greater than 12 pixels.

Table C-5 lists screen source node controls. For all these controls, access is GST.

Table C-5   Screen Source Node Controls

Control	Default	Type	Use
VL_FREEZE	FALSE	boolVal	When set to TRUE, this control freezes the contents of the screen drain. Updates to the graphics frame buffer continue to be displayed on the graphics display but are not reflected on the video output. If set to FALSE, live output resumes.
VL_OFFSET	(0, 0)	xyVal	Specifies the upper left corner of a subregion of the graphics area used to produce the video output. The offset is relative to VL_ORIGIN. See also VL_SIZE, which defines the size of the subregion.
VL_ORIGIN	(0, 0)	xyVal	Specifies the upper left corner of a frame-size graphics area used to produce the video. The origin is specified in X Windows root-window coordinates. VL_OFFSET and VL_SIZE can be used to specify a subregion of this area.
VL_SIZE	CCIR 601 525: 720x486 CCIR 601 625: 768x576 NTSC: 640x486 PAL: 768x576	xyVal	Specifies the size of a subregion of the graphics area used to produce the video output. See also VL_OFFSET, which specifies the location of the subregion, and VL_ORIGIN, which maps a graphics window area to a frame. This control is applied before VL_ZOOM.

Table C-5 (continued)   Screen Source Node Controls

Control	Default	Type	Use
VL_ZOOM	1.0	fractVal	Sets the amount of zoom that is applied to the graphics area before it is converted to video. Valid values are 1 and 1/2.
			Note: The 1/2 zoom value selects full-screen video output and should be used only when the Indigo^2 IMPACT Video device is operating on square-pixel timing (used with the Indigo^2 IMPACT Compression option only).
VL_MGV_ DEINTERLACE	TRUE	boolVal	Specifies how the video fields are generated from the frame-size graphics area with origin VL_ORIGIN. If this control is set to TRUE, a video field line is produced by averaging the corresponding graphics frame lines. If set to FALSE, the corresponding graphics line is selected depending on the field dominance and is output verbatim.

VL_SCREEN Drain

Indigo^2 IMPACT Video supports three screen drain nodes: A, B, and C.

The screen drain nodes provide a means of displaying video data in a graphics window. The Indigo^2 IMPACT Video device displays the video over a specified window, obscuring any graphics contents that may have been there. Note that the Indigo^2 IMPACT Video device does not place the video data into the frame buffer, but instead injects data directly into the raster. Consequently, an **lrectwrite()** operation returns the contents that were drawn into the window, not the video data. For the screen drain,

- type for all three screen drain nodes: VL_DRN

- kind for all three screen drain nodes: VL_SCREEN

- number: VL_MGV_NODE_NUMBER_SCREEN_A, VL_MGV_NODE_NUMBER_SCREEN_B, and VL_MGV_NODE_NUMBER_SCREEN_C, respectively

- port: VL_IMPACT_PORT_PIXEL_DRN_[A,B,C], single-link 8-bit drain of CCIR-601 range video for display in a graphics window

To display live video using the graphics frame buffer, video frames should be captured using one of the VGI1 memory drain nodes, and then drawn using GL or OpenGL functions.

Screen drains B and C share the same physical frame buffer. When only drain B is used, the frame buffer is 24 bits deep. When drains B and C are used, or when only C is used, the frame buffer is split into two 12-bit logical frame buffers.

A 12- to 24-bit dithering is applied to produce the output of each window. While windows B and C can accept data from different sources, the following controls affect both nodes when applied to either:

- VL_SIZE

- VL_ZOOM

- VL_FREEZE

Window positioning has certain constraints. If an application attempts to place the window at an illegal position, the node attempts to place the window at a valid location. The size of the window can also be changed. If either the position or size is changed, the application is notified by a VLValueChanged event. If the window cannot be placed anywhere on the screen, **vlSetControl()** returns VLBadValue.

Make sure your application meets these constraints:

- Windows A, B, and C must not overlap.

- Window A must not overlap window B or C vertically.

- Window C must be to the right of window B by at least 45 pixels.

- The top of window C must be the same as or below window B.

- The bottom of window B must be the same as or higher than the bottom of window C.

- The vertical distance between window A and window B or C must be greater than 12 pixels.

Table C-6 lists screen drain node controls. For all these controls, access is GST, except VL_MGV_DEINTERLACE and VL_WINDOW, which are GS.

Table C-6   Screen Drain Node Controls

Control	Default	Type	Use
VL_FREEZE	FALSE	boolVal	If set to TRUE, this control freezes the contents of the screen drain. If set to FALSE, live video display resumes.
VL_OFFSET	(0, 0)	xyVal	Specifies the upper left corner of a subregion of the video frame to be displayed. See also VL_SIZE, which specifies the size of the subregion. When used with VL_ZOOM, VL_OFFSET is applied after VL_ZOOM.
VL_ORIGIN	(0, 0)	xyVal	Specifies the location on the screen where the video is displayed. The window coordinates are X-Server window coordinates.
VL_SIZE	CCIR 601 525: 720 x 486 CCIR 601 625: 768 x 576 NTSC: 640 x 486 PAL: 768 x 576	xyVal	Specifies the size of a subregion of the video frame to be displayed. See also VL_OFFSET, which specifies the location of the subregion. When used with VL_ZOOM, VL_SIZE is applied after VL_ZOOM.
VL_WINDOW	None	intVal	Specifies the window in which the video is displayed. The value set is the X Server window ID, as returned by **XtWindow()**, for example. The window ID cannot be changed while a screen drain is transferring.
VL_ZOOM	1.0	fractVal	Sets the amount of zoom applied to the video data before it is displayed. Valid values are 7/1, 6/1, 5/1, 4/1, 3/1, 2/1, 1/1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, and 1/8. Zoom is applied before offset (pan) and size.

			Table C-6 (continued) Screen Drain Node Controls
Control	**Default**	**Type**	**Use**
VL_MGV_ALPHA _NOT_PIXEL	TRUE	boolVal	Specifies whether the screen drain should take pixel or alpha data from the source node. The preferred mechanism for specifying this information is to use **vlSetConnection()**; this function overrides the value of this control.
			If VL_MGV_ALPHA_NOT_PIXEL is set to TRUE, the screen drain takes input from the source's alpha port. Otherwise, it takes input from the source's pixel port. If the source node has no alpha port, then the pixel port is used.
VL_MGV _DEINTERLACE	FALSE	boolVal	Sets deinterlacing method.
			Converting video, which is interlaced, to graphics, which is progressive scan, requires deinterlacing the image: that is, replacing the missing lines with something. Two types of deinterlace methods are available: replacing the missing lines with black lines (the simplest method) or interpolating missing lines by simple filtering of adjacent lines.
			If set to TRUE, then the average of the adjacent lines is used to produce a full-brightness deinterlaced screen. If set to FALSE, then the lines contain black, producing a half-brightness window but with the same interlacing as video.

VL_VIDEO

This discussion divides the VL_VIDEO node into its manifestations as source and drain.

VL_VIDEO Source

Indigo^2 IMPACT Video supports three digital video source nodes: 1, 2, and dual-link.

The video source nodes correspond to two video input connectors available on the Indigo^2 IMPACT Video device. These connectors can be used separately to feed CCIR-601 video to Indigo^2 IMPACT Video, or as a dual link to supply RP-175 RGB or YUV 4:4:4:4 or 4:2:2:4 video. When used in dual-link mode, the connector labelled 1 is used for pixel input while the connector labelled 2 is used for alpha input.

The uses of the connectors in single- or dual-link modes are mutually exclusive. When either of the single-linked nodes are in use, the dual-linked node is unavailable. Similarly, if the dual-link node is in use, both single-linked nodes are considered to be in use. Mutual exclusion takes place when paths are set up with stream usage VL_SHARE or VL_LOCK. Mutual exclusion conditions are not applied to paths with stream usage VL_READ_ONLY. For the video source,

- type for all three screen drain nodes: VL_SRC

- kind for all three screen drain nodes: VL_VIDEO

- number: VL_MGV_NODE_NUMBER_VIDEO_1
  VL_MGV_NODE_NUMBER_VIDEO_2, and
  VL_MGV_NODE_NUMBER_VIDEO_DL, respectively

- ports:

  - video source nodes 1 and 2: VL_IMPACT_PORT_PIXEL_SRC_A. single-link serial digital video input

  - dual-link video source node:
    VL_IMPACT_PORT_DUALLINK_SRC_A - RP-175 style RGB or YUV dual-link serial digital video

Table C-7 lists video source node controls. For all these controls, access is GST, except VL_FORMAT, which is GS.

Table C-7   Video Source Node Controls

Control	Default	Type	Use
VL_FREEZE	FALSE	boolVal	Freezes the input video stream. Because the Indigo^2 IMPACT Video device does not support frozen inputs, this control can be set only to FALSE.
VL_FORMAT	VL_FORMAT_DIGITAL_ COMPONENT_SERIAL (single-link) VL_FORMAT_DIGITAL_ COMPONENT_DUAL_ SERIAL (dual-link)	intVal	Specifies the format of the incoming video. Valid values for the single-link nodes are VL_FORMAT_DIGITAL_COMPONENT_SERIAL and VL_FORMAT_RAW_DATA. Valid values for the dual-link node are VL_FORMAT_DIGITAL_COMPONENT_DUAL_SERIAL and VL_FORMAT_DIGITAL_COMPONENT_RGB_SERIAL. See "Using VL_FORMAT" in Chapter 2 for format explanations.

Table C-7 (continued)   Video Source Node Controls

Control	Default	Type	Use
VL_OFFSET	(0, 0)	xyVal	Pans within the video. The Indigo[2] IMPACT Video source nodes support an offset of (0, 0) only.
VL_SIZE	Dynamic	xyVal	Reports the width and height of the active video region. The values are fixed for each timing mode: CCIR 525: 720 x 486 CCIR 625: 720 x 576 NTSC square pixel: 640 x 486 PAL square pixel: 768 x 576 Square pixel modes are used with the Indigo[2] IMPACT Compression option only. Timing is specified through the VL_TIMING control on the device node.
VL_MGV_INPUT_8BIT	FALSE	boolVal	Enables 10-bit to 8-bit truncating when 10-bit video input is supplied. Internally, Indigo[2] IMPACT Video treats all video streams as 10-bit. As a result, when this control is set to TRUE, the lower two bits are forced to zero. If set to FALSE, the input stream passes unmodified. If the input video contains 8-bit data, then it is left-shifted two bits to produce a 10-bit value. The lower two bits contain zeros.

VL_VIDEO Drain

Indigo^2 IMPACT Video supports three digital video drain nodes: 0, 1, and 2.

The video drain nodes correspond to two video output connectors available on the Indigo^2 IMPACT Video device. These connectors can be used separately to output CCIR-601 video from Indigo^2 IMPACT Video, or as a dual link to output RP-175 RGB or YUV 4:4:4:4 or 4:2:2:4 video. When used in dual-link mode, the connector labelled 1 is used for pixel output while the connector labelled 2 is used for alpha output.

The uses of the connectors in single or dual-link modes are mutually exclusive. When either of the single-linked nodes are in use, the dual-linked node is unavailable. Similarly, if the dual-link node is in use, both single-linked nodes are considered to be in use. Mutual exclusion takes place when paths are set up with stream usage VL_SHARE or VL_LOCK. Mutual exclusion conditions are not applied to paths with stream usage VL_READ_ONLY. For the video drain,

- type for all three video drain nodes: VL_DRN

- kind for all three video drain nodes: VL_VIDEO

- number: VL_MGV_NODE_NUMBER_VIDEO_1
  VL_MGV_NODE_NUMBER_VIDEO_2, and
  VL_MGV_NODE_NUMBER_VIDEO_DL, respectively

- ports:

  - video source nodes 1 and 2: VL_IMPACT_PORT_PIXEL_SRC_A: single-link serial digital video input

  - dual-link video source node:
    VL_IMPACT_PORT_DUALLINK_SRC_A: RP-175 style RGB or YUV dual-link serial digital video

Table C-8 lists video drain node controls. For all these controls, access is GST, except VL_FORMAT, which is GS.

Table C-8   Video Drain Node Controls

Control	Default	Type	Use
VL_FREEZE	FALSE	boolVal	If set to TRUE, the output of the drain node is frozen. For dual-link, both outputs are frozen simultaneously. Use the VL_MGV_OUTPUT_DL_SELECT_FREEZE control to set which links of the dual-link node are frozen. Note: VL_MGV_OUTPUT_FSYNC must be set to TRUE in order for the video to freeze.
VL_FORMAT	VL_FORMAT_DIGITAL_ COMPONENT_SERIAL (single-link) VL_FORMAT_DIGITAL_ COMPONENT_DUAL_ SERIAL (dual-link)	intVal	Specifies the format of the incoming video. Valid values for the single-link nodes are VL_FORMAT_DIGITAL_COMPONENT_SERIAL and VL_FORMAT_RAW_DATA: Arbitrary 8-bit data. Valid values for the dual-link node are VL_FORMAT_DIGITAL_COMPONENT_DUAL_SERIAL and VL_FORMAT_DIGITAL_COMPONENT_RGB_SERIAL.
VL_OFFSET	(0, 0)	xyVal	Sets offset; the Indigo^2 IMPACT Video drain nodes support an offset of (0, 0) only.
VL_SIZE	Dynamic	xyVal	Reports the width and height of the active video region. The values are fixed for each timing mode: CCIR 525: 720 x 486 CCIR 625: 72 0x 576 NTSC square pixel: 640 x 486 PAL square pixel: 768 x 576 Square pixel modes are used with the Indigo^2 IMPACT Compression option only. Timing is specified through the VL_TIMING control on the device node.
VL_MGV_ALPHA _NOT_PIXEL	FALSE	boolVal	If the node supplying the video drain node has both pixel and alpha outputs, this control selects whether the alpha (TRUE) or pixel (FALSE) channel is selected. This control is provided only for compatibility with Galileo Video applications. It is recommended that the application use **vlSetConnection()** to specify the output port.

Table C-8 (continued)   Video Drain Node Controls

Control	Default	Type	Use
VL_MGV_OUTPUT _BLANK	Persistent	boolVal	If this control is set to TRUE, the video output is blanked, that is, video black is output on the serial digital port. If set to FALSE, live video is displayed. VL_MGV_OUTPUT_FSYNC must be set to TRUE for this control to have any effect.
VL_MGV_OUTPUT _CHROMA	Persistent	boolVal	If set to TRUE, the chroma portion of a video stream is passed through. If set to FALSE, chroma is blanked. VL_MGV_OUTPUT_FSYNC must be set to TRUE for this control to have any effect.
VL_MGV_OUTPUT _FSYNC	Persistent	boolVal	If set to TRUE, this control enables the output synchronization hardware. The output synchronizer must be enabled for the VL_FREEZE, VL_MGV_OUTPUT_BLANK, VL_MGV_OUTPUT_HPHASE, or VL_MGV_OUTPUT_CHROMA controls to have any effect.
VL_MGV_OUTPUT _HPHASE	0xC00	intVal	Specifies the horizontal phase of the video output with respect to the video input. It is a 12-bit unsigned integer that increments in steps of the pixel clock (typically 74 nsec). The output occurs later in time as the value of this control increases. This control has a range of 1 to 0xFFF, which can advance the output by slightly more than three lines or delay the output by slightly more than one line. The default value 0xC00 makes the output match the timing of the video input. The value 0 is illegal. VL_MGV_OUTPUT_FSYNC must be set to TRUE for this control to have any effect.
VL_MGV_L _BLANK	VL_MGV_DL_SELECT _ALL	intVal	Selects the channels (pixel, alpha, both) on the dual-link node to blank. Valid values are VL_MGV_DL_SELECT ALPHAL (alpha channel only). VL_MGV_DL_SELECT_PIXEL (pixel channel only), and VL_MGV_DL_SELECT_AL (pixel and alpha channels).
VL_MGV_OUTPUT _DL_SELECT _CHROMA	VL_MGV_DL_SELECT _ALL	intVal	Selects the channels (pixel, alpha, both) on the dual-link node on which chroma should be blanked. Valid values are the same as for VL_MGV_OUTPUT_DL_SELECT_BLANK.

Table C-8 (continued)   Video Drain Node Controls

Control	Default	Type	Use
VL_MGV_OUTPUT _DL_SELECT _FREEZE	VL_MGV_DL_SELECT _ALL	intVal	Selects the channels (pixel, alpha, both) on the dual-link node to freeze. If both channels are selected, the freeze is performed atomically on both channels. Valid values are the same as for VL_MGV_OUTPUT_DL_SELECT_BLANK.
VL_MGV_OUTPUT _DL_SELECT _FSYNC	VL_MGV_DL_SELECT _ALL	intVal	Selects which output on the dual-link node synchronizers to enable or disable. Valid values are the same as for VL_MGV_OUTPUT_DL_SELECT_BLANK.

Glossary

active video

The portion of the video signal containing the chrominance or luminance information; all video lines not occurring in the vertical blanking signal containing the chrominance or luminance information. See also *chrominance*, *composite video*, *horizontal blanking*, *luminance*, and *video waveform*.

aliasing

One of several types of digital video artifact appearing as jagged edges. Aliasing results when an image is sampled that contains frequency components above the Nyquist limit for the sampling rate. See also *Nyquist limit*.

alpha

See *alpha value*.

alpha blending

Overlaying one image on another so that some of the underlying image may or may not be visible. See also *key*.

alpha plane

A bank of memory that stores alpha values; the values are 8 bits per pixel.

alpha register

Registers that stores an alpha value.

alpha value

The component of a pixel that specifies the pixel's opacity, translucency, or transparency. The alpha component is typically output as a separate component signal.

antialiasing

Filtering or blending lines of video to smooth the appearance of jagged edges in order to reduce the visibility of aliasing.

APL

Average Picture Level, with respect to blanking, during active picture time, expressed as a percentage of the difference between the blanking and reference white levels. See also *blanking level.*

artifact

In video systems, an unnatural or artificial effect that occurs when the system reproduces an image; examples are aliasing, pixellation, and contouring.

aspect ratio

The ratio of the width to the height of an electronic image. For example, the standard aspect ratio for television is 4:3.

back porch

The portion of the horizontal pedestal that follows the horizontal synchronizing pulse. In a composite signal, the color burst is located on the back porch, but is absent on a YUV or GBR signal. See also *blanking level, video waveform.*

Betacam

A component videotape format developed by Sony® that uses a Y/R-Y/B-Y video signal and 1/2-inch tape.

Betacam format

Advanced form (Superior Performance) of Betacam using special metal tape and offering longer recording time (90 minutes instead of 30 minutes) and superior performance.

bit map

A region of memory that contains the pixels representing an image. The pixels are arranged in the sequence in which they are normally scanned to display the image.

bitplane

One of a group of memory arrays for storing an image in bitmap format on a workstation. The workstation reads the bitplanes in parallel to re-create the image in real time.

black burst

Active video signal that has only black in it. The black portion of the video signal, containing color burst. See also *color burst*.

black level

In the active video portion of the video waveform, the voltage level that defines black. See also *horizontal blanking* and *video waveform*.

blanking level

The signal level at the beginning and end of the horizontal and vertical blanking intervals, typically representing zero output (0 IRE). See also *video waveform* and *IRE units*.

blend

To combine proportional amounts of a 3D graphic over a clip frame by frame, pixel by pixel, with the alpha determining how they are combined. See also *key*, *frame*, and *alpha*.

breezeway

In the horizontal blanking part of the video signal, the portion between the end of the horizontal sync pulse and the beginning of the color burst. See also *horizontal blanking* and *video waveform*.

broad pulses

Vertical synchronizing pulses in the center of the vertical interval. These pulses are long enough to be distinguished from other pulses in the signal; they are the part of the signal actually detected by vertical sync separators.

Bruch blanking

In PAL signals, a four-field burst blanking sequence used to ensure that burst phase is the same at the end of each vertical interval.

burst, burst flag

See *color burst*.

burst lock

The ability of the output subcarrier to be locked to input subcarrier, or of output to be genlocked to an input burst.

burst phase

In the RS-170A standard, burst phase is at field 1, line 10; in the European PAL standards, it is at field 1, line 1. Both define a continuous burst waveform to be in phase with the leading edge of sync at these points in the video timing. See also *vertical blanking interval* and *video waveform*.

B-Y (B minus Y) signal

One of the color difference signals used on the NTSC and PAL systems, obtained by subtracting luminance (Y) from the blue camera signal (B). This signal drives the horizontal axis of a vectorscope. Color mixture is close to blue; phase is 180 degrees opposite of color sync burst; bandwidth is 0.0 to 0.5 MHz. See also *luminance*, *R-Y signal*, *Y signal*, and *Y/R-Y/B-Y*.

C signal

Chrominance; the color portion of the signal. For example, the Y/C video format used for S-VHS has separate Y (luminance) and C (chrominance) signals. See also *chrominance*.

CAV

Component Analog Video; a generic term for all analog component video formats, which keep luminance and chrominance information separate. D1 is a digital version of this signal. See also *component video*.

C format

Type C, or one-inch reel-to-reel videotape machine; an analog composite recording format still used in some broadcast and postproduction applications.

CCIR 601

The digital interface standard developed by the CCIR (Comite' Consultatif International de Radiodiffusion, International Radio Consultative Committee) based on component color encoding, in which the luminance and chrominance (color difference) sampling frequencies are related in the ratio 4:2:2: four samples of luminance (spread across four pixels), two samples of C_R color difference, and two samples of C_B color difference. The

standard, which is also referred to as 4:2:2, sets parameters for both 525-line and 625-line systems.

chroma

See *chrominance*.

chroma keying

Overlaying one video source on another by choosing a key color. For example, if chroma keying is on blue, video source A might show through video source B everywhere the color blue appears in video source B. A common example is the TV weather reporter standing in front of the satellite weather map. The weather reporter, wearing any color but blue, stands in front of a blue background; keying on blue shows the satellite picture everywhere blue appears. Because there is no blue on the weatherperson, he or she appears to be standing in front of the weather map.

chroma signal

A 3.58 MHz (NTSC) or 4.43 MHz (PAL) subcarrier signal for color in television. SECAM uses two frequency-modulated color subcarriers transmitted on alternate horizontal lines; SC_R is 4.406 MHz and SC_B is 4.250 MHz.

chrominance

In an image reproduction system, a separate signal that contains the color information. Black, white, and all shades of gray have no chrominance and contain only the luminance (brightness) portion of the signal. However, all colors have both chrominance and luminance.

Chrominance is derived from the I and Q signals in the NTSC television system and the U and V signals in the PAL television system. See also *luminance*.

chrominance signal

Also called the chroma, or C, signal. The high-frequency portion of the video signal (3.58 MHz for NTSC, 4.43 MHz for PAL) color subcarrier with quadrature modulation by I (R-Y) and Q (B-Y) color video signals. The amplitude of the C signal is saturation; the phase angle is hue. See also *color subcarrier*, *hue*, and *saturation*.

client

In the context of the Video Library, an application that has connected to the video daemon to perform video requests.

clip

Segment of video, audio, or both. An image is a clip that is one frame long.

color bars

A test pattern used by video engineers to determine the quality of a video signal, developed by the Society of Television and Motion Picture Engineers (SMPTE). The test pattern consists of equal-width bars representing black, white, red, green, blue, and combinations of two of the three RGB values: yellow, cyan, and magenta. These colors are usually shown at 75% of their pure values. Figure Gl-1 diagrams the color bars.
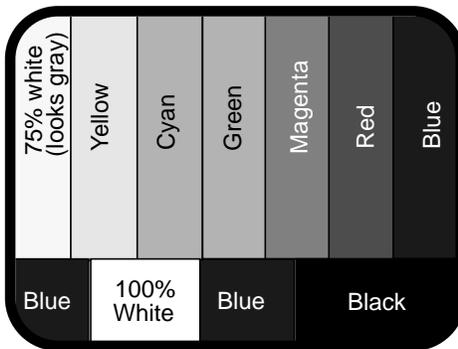
Figure Gl-1   SMPTE Color Bars (75%)

color burst

Also called burst and burst flag. The segment of the horizontal blanking portion of the video signal that is used as a reference for decoding color information in the active video part of the signal. The color burst is required for synchronizing the phase of 3.58 MHz oscillator in the television receiver for correct hues in the chrominance signal.

In composite video, the image color is determined by the phase relationship of the color subcarrier to the color burst. The color burst sync is 8 to 11 cycles of 3.58 MHz color subcarrier transmitted on the back porch of every horizontal pulse; the hue of the color sync phase is yellow-green.

Figure Gl-2 diagrams the relationship of the color burst and the chrominance signal. See also *color subcarrier* and *video waveform.*
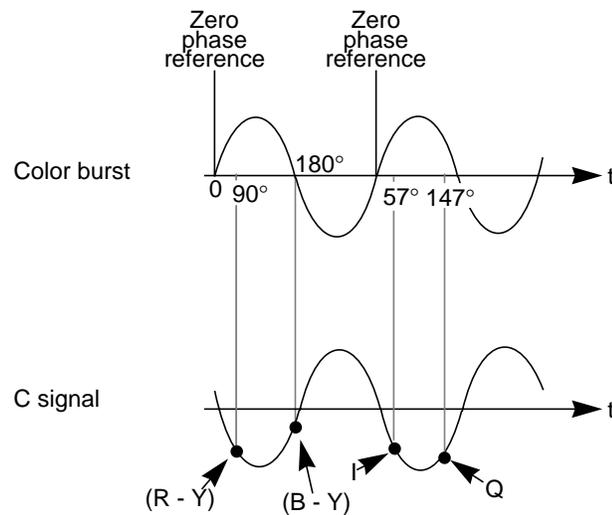
Figure Gl-2   Color Burst and Chrominance Signal

color difference signals

Signals used by color television systems to convey color information so that the signals go to zero when the picture contains no color; for example, unmodulated R-Y and B-Y, I and Q, U, and V.

color-frame sequence

In NTSC and S-Video, a two-frame sequence that must elapse before the same relationship between line pairs of video and frame sync repeats itself. In PAL, the color-frame sequence consists of four frames.

color space

A color component encoding format defined by three color components, such as R, G, and B or Y, U, and V.

color subcarrier

A portion of the active portion of a composite video signal that carries color information, referenced to the color burst. The color subcarrier's amplitude

determines saturation; its phase angle determines hue. Hue and saturation are derived with respect to the color burst. Its frequency is defined as 3.58 MHz in NTSC and 4.43 MHz in PAL. See also *color burst.*

complementary color

Opposite hue and phase angle from a primary color. Cyan, magenta, and yellow are complementary colors for red, green, and blue, respectively.

comb filtering

Process that improves the accuracy of extracting color and brightness portions of the signal from a composite video source.

component video

A color encoding method for the three color signals—R, G, and B; Y, I, and Q; or Y, U, and V—that make up a color image. See also *RGB, YIQ,* and *YUV.*

component video signals

A video signal in which luminance and chrominance are send as separate components, for example:

- RGB (basic signals generated from a camera)

- YIQ (used by the NTSC broadcasting standard)

- Y/R-Y/B-Y (used by Betacam and M-II recording formats and SECAM broadcasting standard)

- YUV (subset of Y/R-Y/B-Y used by the PAL broadcasting standard)

Separating these components yields a signal with a higher color bandwidth than that of composite video.

Figure Gl-3 depicts video signals for one horizontal scan of a color-bar test pattern. The RGB signals change in relation to the individual colors in the test pattern. When a secondary color is generated, a combination of the RGB signals occurs. Since only the primary and secondary colors are being displayed at 100% saturation, the R, G, and B waveforms are simply on or off. For more complex patterns of color, the individual R, G, and B signals would be varying amplitudes in the percentages needed to express that particular color.

See also *composite video, RGB, YUV, Y/R-Y/B-Y,* and *YIQ.*
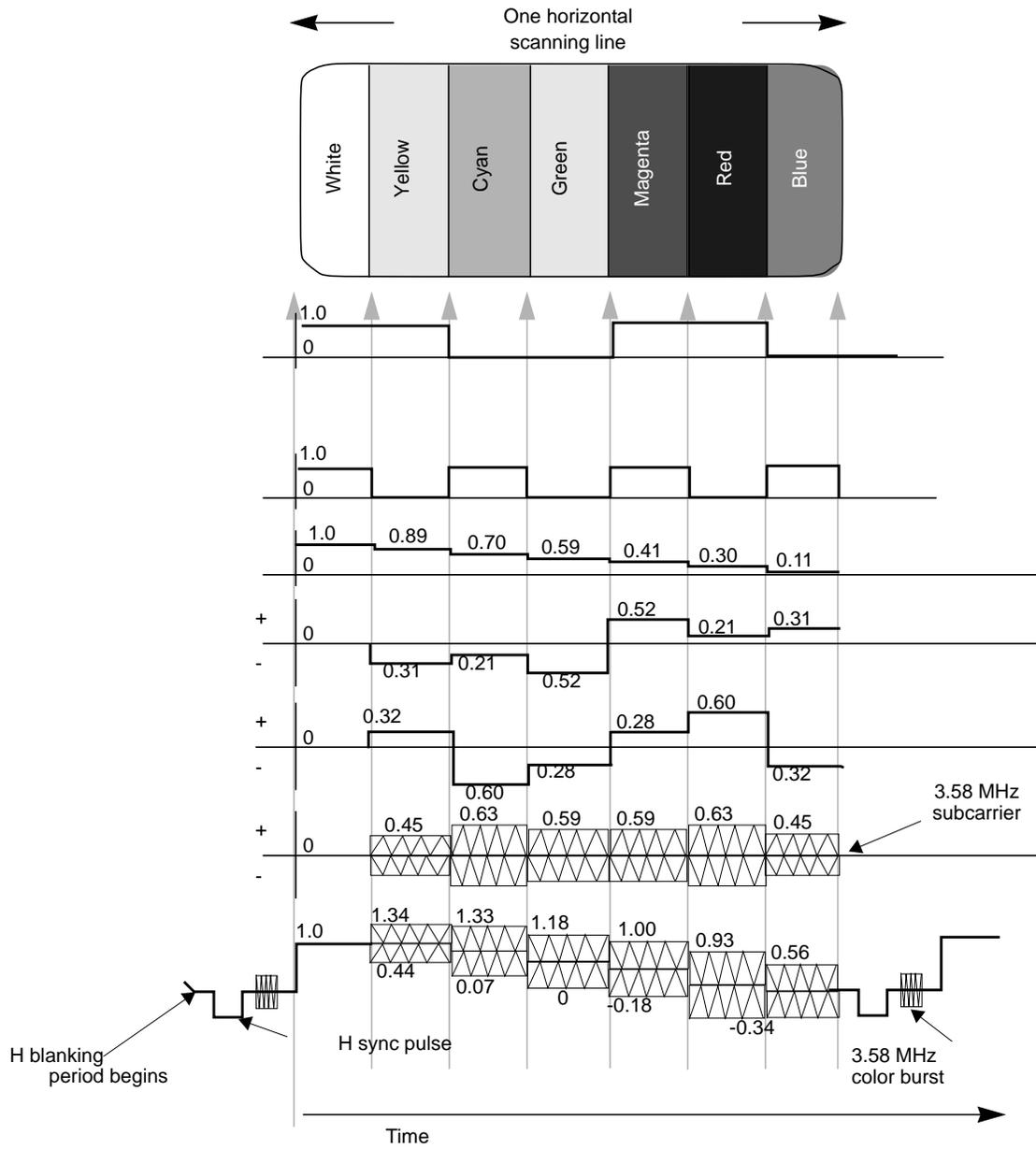
Figure GI-3   Component Video Signals

compositing

Combining graphics with another image.

composite video

A color encoding method or a video signal that contains all of the color, brightness, and synchronizing information in one signal. The chief composite television standard signals are NTSC, PAL, and SECAM. See also *NTSC*, *PAL*, and *SECAM*.

cross-chrominance, cross-luminance

Also known as cross-color, hanging dots, dot crawl; moving colors on stationary objects. This undesirable artifact is caused by high bandwidth luminance information being misinterpreted as color information. Hanging dots are a byproduct of the comb filters (used to help separate the color and brightness information) found in most modern television receivers. This artifact can be reduced or eliminated by using S-Video or a component video format.

cross-fade

A type of transition in which one video clip is faded down while another is faded up.

D1

Digital recording technique for component video; also known as CCIR 601, 4:2:2. D1 is the best choice for high-end production work where many generations of video are needed. D1 can be an 8-bit or 10-bit signal. See also *CCIR 601*.

D2

Digital recording technique for composite video. As with analog composite, the luminance and chrominance information is sent as one signal. A D2 VTR offers higher resolution and can make multiple generation copies without noticeable quality loss, because it samples an analog composite video signal at four times the subcarrier (using linear quantization), representing the samples as 8-bit digital words. D2 is not compatible with D1.

D3, DX

Developed by Panasonic, a 1/2-inch tape version of D2. More often called DX.

decoder

Hardware or software that converts, or decodes, a composite video signal into the various components of the signal. For example, to grab a frame of composite video from a VHS tapedeck and store it as an RGB file, it would have to be decoded first. Several Silicon Graphics video options have on-board decoders.

dithering

Approximating a signal value on a chroma-limited display device by producing a matrix of color values that fool human perception into believing that the signal value is being reproduced accurately. For example, dithering is used to display a true-color image on a display capable of rendering only 256 unique colors, such as IndigoVideo images on a Starter Graphics display.

drain

In the context of the Video Library, a target or consumer of video signals.

editing

The process in which data is examined, created, and modified. In video, the part of the postproduction process in which the finished videotape is derived from raw video footage. Animation is a subset of editing.

encoder

Device that combines the R, G, and B primary color video signals into hue and saturation for the C portion of a composite signal. Several Silicon Graphics video options have on-board encoders.

equalizing pulse

Pulse of one half the width of the horizontal sync pulse, transmitted at twice the rate of the horizontal sync pulse, during the portions of the vertical blanking interval immediately before and after the vertical sync pulse. The equalizing pulse makes the vertical deflection start at the same time in each interval, and also keeps the horizontal sweep circuits in step during the portions of the vertical blanking interval immediately before and after the vertical sync pulse.

event

Exceptional or noteworthy condition produced during video processing, such as loss of sync, dropping of frames or fields, and synchronization with other applications.

exclusive use

A term applied to usage of the video data stream and controls on a pathway. A pathway in exclusive-use mode is available for writing of controls only to the client that requested the exclusive use, yet any application may read the controls on that pathway.

fade

To modify the opacity and/or volume of a clip. A faded-up clip is unaffected, a clip faded down to 50% has 50% less opacity or volume, and a faded-down clip is completely transparent of turned off.

field

One of two (or more) equal parts of information in which a frame is divided in interlace scanning. A vertical scan of a frame carrying only its odd-numbered or its even-numbered lines. The odd field and even field make up the complete frame. See also *frame* and *interlace*.

field averaging

A filter that corrects flicker by averaging pixel values across successive fields. See also *flicker*.

field blanking

The blanking signals at the end of each field, used to make the vertical retrace invisible. Also called vertical blanking; see *vertical blanking* and *vertical blanking interval*.

filter

To process a clip with spatial or frequency domain methods. Each pixel is modified by using information from neighboring (or all) pixels of the image. Filter functions include blur (low-pass) and crisp (high-pass).

flicker

The effect caused by a one-pixel-deep line in a high-resolution graphics frame that is output to a low-resolution monitor, because the line is in only one of the alternating fields that make up the frame. This effect can be filtered out by field averaging. See also *field* and *frame*.

frame

The result of a complete scanning of one image. In television, the odd field (all the odd lines of the frame) and the even field (all the even lines of the frame) make up the frame. In motion video, the image is scanned repeatedly, making a series of frames.

freeze, freeze-frame

A condition on the digitized video signal where the digitizing is stopped and the contents of the signal appear frozen on the display or in the buffer. Sometimes used to capture the video data for processing or storage.

frequency

Signal cycles per second.

frequency interlace

Placing of harmonic frequencies of C signal midway between harmonics of horizontal scanning frequency Fh. Accomplished by making color subcarrier frequency exactly 3.579545 MHz. This frequency is an odd multiple of H/2.

front porch

The portion of the video signal between the end of active video and the falling edge of sync. See also *back porch*, *horizontal blanking*, and *video waveform*.

G-Y signal

Color mixture close to green, with a bandwidth 0.0 MHz to 0.5 MHz. Usually formed by combining B-Y and R-Y video signals.

gamma correction

Correction of gray-scale inconsistency. The brightness characteristic of a CRT is not linear with respect to voltage; the voltage-to-intensity characteristic is usually close to a power of 2.2. If left uncorrected, the resulting display has too much contrast and detail in black regions is not reproduced.

To correct this inconsistency, a correction factor using the 2.2 root of the input signal is included, so that equal steps of brightness or intensity at the input are reproduced with equal steps of intensity at the display.

genlocking

Synchronizing with another video signal serving as a master timing source. The master timing source can be a composite video signal, a video signal with no active video (only sync information), or, for video studio, a device called house sync. When there is no master sync available, VideoFramer, for example, can be set to "free run" (or "stand-alone") mode, so that it becomes the master timing device to which other devices sync. See also *line lock*.

gray-scale

Monochrome or black-and-white, as in a monitor that does not display color.

H rate

Number of complete horizontal lines, including trace and retrace, scanned per second.

HDTV

High-definition television. Though there is more than one proposal for a broadcast standard for HDTV, most currently available equipment is based on the 1125/60 standard, that is, 1125 lines of video, with a refresh rate of 60Hz, 2:1 interlacing (same as NTSC and PAL), and aspect ratio of 16:9 (1920 x 1035 viewable resolution), trilevel sync, and 30 MHz RGB and luminance bandwidth.

Hi-8mm

An 8mm recording format developed by Sony; accepts composite and S-Video signals.

horizontal blanking

The period when the electron beam is turned off, beginning when each scan line finishes its horizontal path (scan) across the screen (see Figure Gl-4).

FCC NTSC standards:
Front porch = 1.5 sec.
Hor. sync = 4.7 sec
Back porch = 4.7 sec.
Blanking period = 10.9 sec.

(NOT DRAWN TO SCALE)

Visible Video Picture

Setup Level 7.5 IRE

Setup Level 7.5 IRE

Active Video Area

Front Porch   Hor. Sync Pulse   Back Porch

Front Porch   Hor. Sync Pulse   Back Porch

Blanking Period
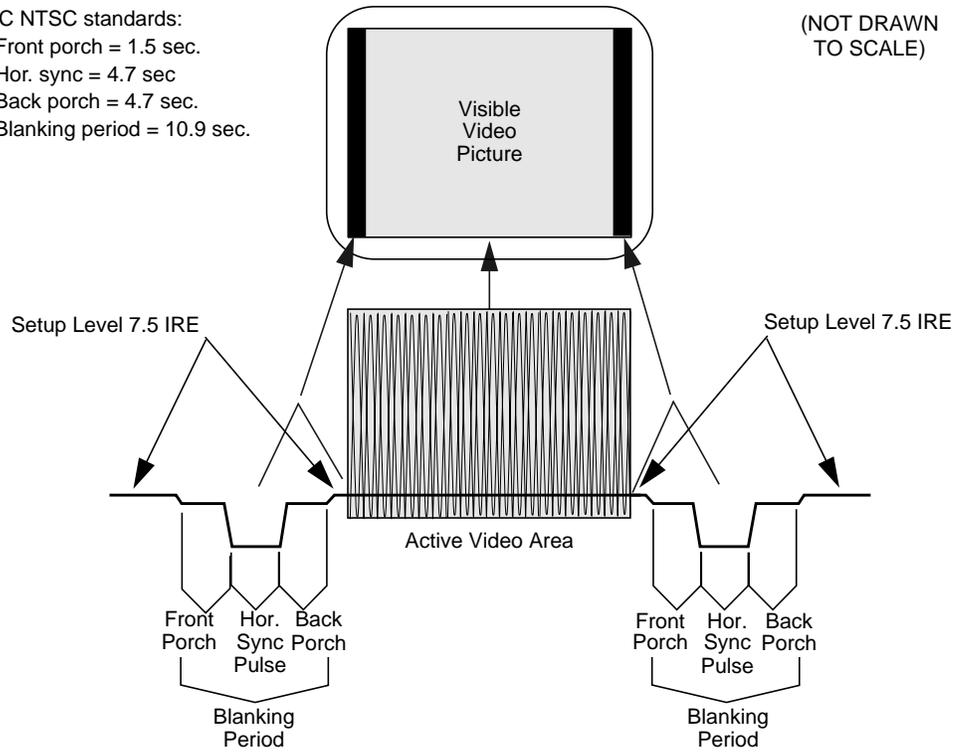
Blanking Period

Figure Gl-4   Horizontal Blanking

horizontal blanking interval

Also known as the horizontal retrace interval, the period when a scanning process is moving from the end of one horizontal line to the start of the next line. This portion of the signal is used to carry information other than video information. See also *video waveform.*
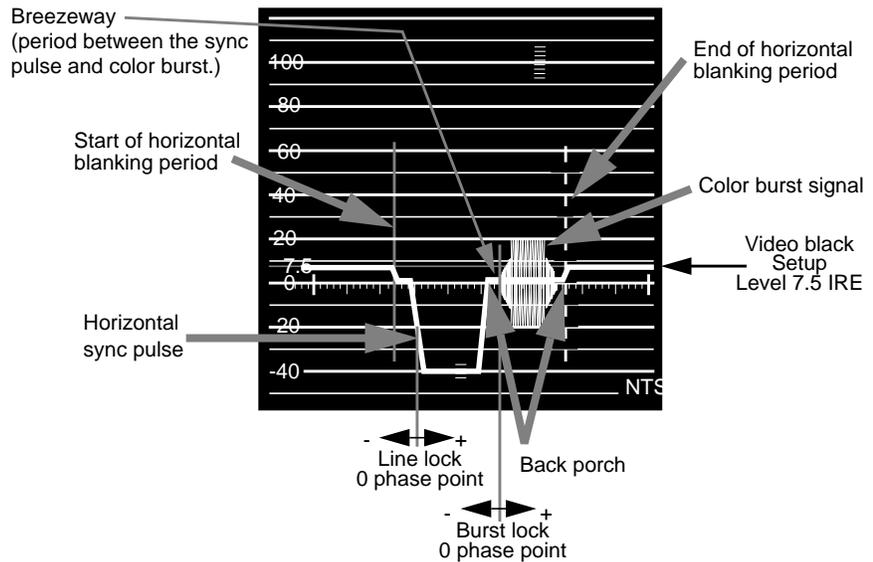
Figure GI-5   Horizontal Blanking Interval

horizontal drive

The portion of the horizontal blanking part of the video signal composed of the sync pulse together with the front porch and breezeway; that is, horizontal blanking minus the color burst. See also *video waveform.*

horizontal sync

The lowest portion of the horizontal blanking part of the video signal, it provides a pulse for synchronizing video input with output. Also known as h sync. See also *horizontal blanking* and *video waveform.*

HSI

See *hue-saturation-intensity*.

HSV

Hue-saturation-value; see *hue-saturation-intensity*.

hue

The designation of a color in the spectrum, such as cyan, blue, magenta. Sometimes called tint on NTSC television receivers. The varying phase angles in the 3.58 MHz (NTSC) or 4.43 MHz (PAL) C signal indicate the different hues in the picture information.

hue-saturation-intensity

A tristimulus color system based on the parameters of hue, saturation, and intensity (luminance). Also referred to as HSI or HSV.

I signal

Color video signal transmitted as amplitude modulation of the 3.58 MHz C signal (NTSC). The hue axis is orange and cyan. This signal is the only color video signal with a bandwidth of 0 to 1.3 MHz.

image plane

See *bitplane*.

image processing

Manipulating an image by changing its color, brightness, shape, or size.

interlace

A technique that uses more than one vertical scan to reproduce a complete image. In television, the 2:1 interlace used yields two vertical scans (fields) per frame: the first field consists of the odd lines of the frame, the other of the even lines. See also *field* and *frame*.

IRE units

A scale for measuring analog video signal levels, normally starting at the bottom of the horizontal sync pulse and extending to the top of peak white. Blanking level is 0 IRE units and peak white level is 100 IRE units (700mv). An IRE unit equals 7.14mv (+100 IRE to -40 IRE = 1v). IRE stands for Institute of Radio Engineers, a forerunner of the IEEE.

keying

Combining proportional amounts of two frames, pixel by pixel, with optional opacity. This process resembles taking two panes of glass with images on them and placing one pane on top of the other. The opacity of the top pane determines the parts of the bottom pane that show. Usually, keying is a real-time continuous process, as in the "over the shoulder" graphics in TV news programs. The alpha component of each pixel, which defines its opacity, determines how the images are combined. Combining images based on the alpha component is often called alpha keying or luma keying. See also *compositing* and *mixing*.

leading edge of sync

The portion of the video waveform after active video, between the sync threshold and the sync pulse. See also *video waveform*.

level

Signal amplitude.

line

The result of a single pass of the sensor from left to right across the image.

line blanking

The blanking signal at the end of each horizontal scanning line, used to make the horizontal retrace invisible. Also called horizontal blanking.

line frequency

The number of horizontal scans per second, normally 15,734.26 times per second for NTSC color systems. The line frequency for the PAL 625/50H. system is 15,625 times per second.

line lock

Input timing that is derived from the horizontal sync signal, also implying that the system clock (the clock being used to sample the incoming video) is an integer multiple of the horizontal frequency and that it is locked in phase to the horizontal sync signal. See also at *video waveform*.

linear matrix transformation

The process of combining a group of signals through addition or subtraction; for example, RGB signals into luminance and chrominance signals.

live video

Video being delivered at a nominal frame rate appropriate to the format.

luma

See *luminance.*

luminance

The video signal that describes the amount of light in each pixel. Luminance is a weighted sum of the R, G, and B signals. See also *chrominance* and *Y signal.*

map

Numerical lookup of pixel data that modifies each pixel without using neighboring pixels. This large category of video editing functions includes clip/gain, solarization, and histogram equalization.

MII (M2)

A second-generation recording format based on a version of the Y/R-Y/B-Y video signal. Developed by Panasonic, MII is also marketed by other video manufacturers. Though similar to Betacam, it is nonetheless incompatible.

matrix transformation

The process of converting analog color signals from one tristimulus format to another, for example, RGB to YUV. See also *tristimulus color system.*

mixing

In video editing, combining two clips frame by frame, pixel by pixel. Usually, a linear interpolation between the pixels in each clip is used, with which one can, for example, perform a cross-fade. Other operations include averaging, adding, differencing, maximum (non-additive mix), minimum, and equivalence (white where equal, else black). See also *compositing* and *keying.*

multiburst

A test pattern consisting of sets of vertical lines with closer and closer spacing; used for testing horizontal resolution of a video system.

NTSC

A color television standard or timing format encoding all of the color, brightness, and synchronizing information in one signal. Used in North America, most of South America, and most of the Far East, this standard is named after the National Television Systems Committee, the standardizing body that created this system in the U.S. in 1953. NTSC employs a total of 525 horizontal lines per frame, with two fields per frame of 262.5 lines each. Each field refreshes at 60Hz (actually 59.94Hz).

Nyquist limit

The highest frequency of input signal that can be correctly sampled without aliasing. The Nyquist limit is equal to half of the sampling frequency.

offset

In the context of a video signal, the relative coordinates from the upper left corner of the video image where signal sampling begins.

overscan

To scan a little beyond the display raster area of the monitor so that the edges of the raster are not visible. Television is overscanned; computer displays are underscanned.

PAL

A color television standard or timing format developed in West Germany and used by most other countries in Europe, including the United Kingdom but excluding France, as well as Australia and parts of the Far East. PAL employs a total of 625 horizontal lines per frame, with two fields per frame of 312.5 lines per frame. Each field refreshes at 50Hz. PAL encodes color differently from NTSC. PAL stands for Phase Alternation Line or Phase Alternated by Line, by which this system attempts to correct some of the color inaccuracies in NTSC. See also *NTSC* and *SECAM.*

pathway

In the Video Library, a connection of sources and drains that provide useful processing of video signals. Pathways have controls and video streams. Pathways can be locked for exclusive use, and are the target of events generated during video processing. See also *exclusive use* and *event.*

pedestal

See *setup*; see also *video waveform*.

pixel

Picture element; the smallest addressable spatial element of the computer graphics screen. A digital image address, or the smallest reproducible element in analog video. A pixel can be monochrome, gray-scale, or color, and can have an alpha component to determine opacity or transparency. Pixels are referred to as having a color component and an alpha component, even if the color component is gray-scale or monochrome.

pixel map

A two-dimensional piece of memory, any number of bits deep. See also *bitmap*.

postproduction

The processes that occur before release of the finished video product, including editing, painting (2D graphics), production, and 3D graphics production.

primary colors

Red, green, and blue. Opposite voltage polarities are the complementary colors cyan, magenta, and yellow.

Q signal

The color video signal that modulates 3.58 MHz C signal in quadrature with the I signal. Hues are green and magenta. Bandwidth is 0.0 MHz to 0.5 MHz. See also *C signal*, *I signal*, *YC*, and *YIQ*.

quantization error

The magnitude of the error introduced in a signal when the actual signal is between levels, resulting from subdividing a video signal into distinct increments, such as levels from 0 to 255.

raster

The scanning pattern for television display; a series of horizontal lines, usually left to right, top to bottom. In NTSC and PAL systems, the first and last lines are half lines.

raster operation, raster op

A logical or arithmetic operation on a pixel value.

registration

The process of causing two frames to coincide exactly. In component video cameras or displays, the process of causing the three color images to coincide exactly, so that no color fringes are visible.

resolution

Number of horizontal lines in a television display standard; the higher the number, the greater a system's ability to reproduce fine detail.

RGB

Red, green, blue; the basic component set used by graphics systems and some video cameras, in which a separate signal is used for each primary color.

RGB format

The technical specification for NTSC color television. Often (incorrectly) used to refer to an RGB signal that is being sent at NTSC composite timings, for example, a Silicon Graphics computer set to output 640 x 480. The timing would be correct to display on a television, but the signal would still be split into red, green and blue components. This component signal would have to go through an encoder to yield a composite signal (RS-170A format) suitable for display on a television receiver.

R-Y (R minus Y) signal

A color difference signal obtained by subtracting the luminance signal from the red camera signal. It is plotted on the 90 to 270 degree axis of a vector diagram. The R-Y signal drives the vertical axis of a vectorscope. The color mixture is close to red. Phase is in quadrature with B-Y; bandwidth is 0.0 MHz to 0.5 MHz. See also *luminance*, *B-Y (B minus Y) signal*, *Y/R-Y/B-Y*, and *vectorscope*.

sample

To read the value of a signal at evenly spaced points in time; to convert representational data to sampled data (that is, synthesizing and rendering).

sampling rate, sample rate

Number of samples per second.

saturation

Color intensity; zero saturation is white (no color) and maximum saturation is the deepest or most intense color possible for that hue. Different saturation values are varying peak-to-peak amplitudes in the 3.58 MHz modulated C signal. In signal terms, saturation is determined by the ratio between luminance level and chrominance amplitude. See also *hue.*

scaling

To change the size of an image.

scan

To convert an image to an electrical signal by moving a sensing point across the image, usually left to right, top to bottom.

SECAM

Sequentiel Couleur avec Memoire, the color television system developed in France and used there as well as in eastern Europe, the Near East and Mideast, and parts of Africa and the Caribbean.

setup

The difference between the blackest level displayed on the receiver and the blanking level (see Figure Gl-6). A black level that is elevated to 7.5 IRE instead of being left at 0.0 IRE, the same as the lowest level for active video. Because the video level is known, this part of the signal is used for black-level clamping circuit operation. Setup is typically used in the NTSC video format and is typically not used in the PAL video format; it was originally introduced to simplify the design of early television receivers, which had trouble distinguishing between video black levels and horizontal blanking. Also called pedestal.

Figure Gl-6 shows waveform displays of a signal with and without setup. See also *video waveform*.

With Setup level   Without Setup level
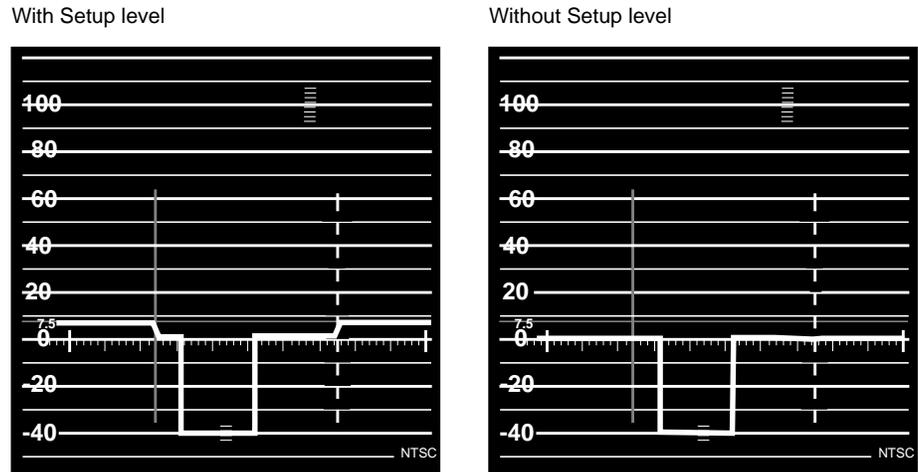
Figure Gl-6   Waveform Monitor Readings With and Without Setup

smear

An artifact usually caused by mid-frequency distortions in an analog system that results in the vertical edges of the picture spreading horizontally.

SMPTE time code

A signal specified by the Society of Motion Picture and Television Engineers for facilitating videotape editing; this signal uniquely identifies each frame of the video signal. Program originators use vertical blanking interval lines 12 through 14 to store a code identifying program material, time, frame number, and other production information (see Figure Gl-7).
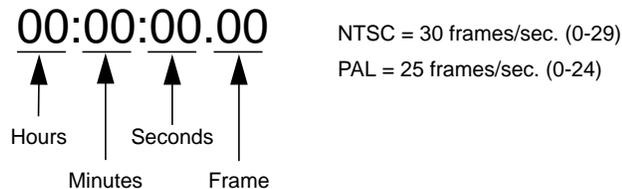
Figure Gl-7   SMPTE Time Code

source

In the context of the Video Library, a provider of video input signals.

subcarrier

A portion of a video signal that carries a specific signal, such as color. See *color subcarrier*.

subpixel

A unit derived from a pixel by using a filter for sizing and positioning.

S-VHS, S-Video

Video format in which the Y (luminance) and C (chrominance) portions of the signal are kept separate. Also known as YC.

sync information

The part of the television video signal that ensures that the display scanning is synchronized with the broadcast scanning. See also *video waveform*.

sync pulse

A vertical or horizontal pulse (or both) that determines the display timing of a video signal. Composite sync has both horizontal and vertical sync pulses, as well as equalization pulses. The equalization pulses are part of the interlacing process.

sync tip

The lowest part of the horizontal blanking interval, used for synchronization. See also *video waveform*.

synchronize

To perform time shifting so that things line up.

texturing

Applying images to three-dimensional objects to give additional realism to displayed renderings.

termination

To send a signal through a transmission line accurately, there must be an impedance at the end which matches the impedance of the source and of the line itself. Amplitude errors, frequency response, and pulse distortions and reflections (ghosting) occur on a line without proper termination. Video is a 75Ohm system; therefore a 75Ohm terminator of .5% to .25% accuracy must be installed at the end of the signal path.

threshold

In a digital circuit, the signal level that is specified as the division point between levels used to represent different digital values; for example, the sync threshold is the level at which the leading edge of sync begins. See also *video waveform.*

time-base errors

Analog artifacts caused by nonuniform motion of videotape or of the tape head drum. Time-base errors usually cause horizontal display problems, such as horizontal jitter.

time code

See *SMPTE time code.*

time-delay equalization

Frame-by-frame alignment of all video inputs to one sync pulse, so that all frames start at the same time. This alignment is necessary because cable length differences cause unequal delays. See *time-base errors.*

transcoder

A device that converts a component video signal to a different component video signal, for example, RGB to Y/R-Y/B-Y, or D1 to RGB.

transducer

A microphone, video camera, or other device that can convert sounds or images to electrical signals.

transform

The geometric perspective transformation of 3-D graphics models and planar images.

tristimulus color system

A system of transmitting and reproducing images that uses three color signals, for example, RGB, YIQ, and YUV.

U signal

One of the chrominance signals of the PAL color television system, along with V. Sometimes referred to as B-Y, but U becomes B-Y only after a weighting factor of 0.493 is applied. The weighting is required to reduce peak modulation in the composite signal.

U-Matic

Sony trademark of its 3/4-inch composite videotape format. SP U-Matic is an improved version using metal tape.

underscan

To scan a television screen so that the edges of the raster are visible. See also *overscan.*

V signal

One of the chrominance signals of the PAL color television system, along with U. Sometimes referred to as R-Y, but V becomes R-Y only after a weighting factor of 0.877 is applied. The weighting is required to reduce peak modulation in the composite signal.

vectorscope

A specialized oscilloscope that demodulates the video signal and presents a display of R-Y versus B-Y for NTSC (V and U for PAL). Video engineers use vectorscopes to measure the amplitude (gain) and phase angle (vector) of the primary (red, green, and blue) and the secondary (yellow, cyan, and magenta) color components of a television signal.

vertical blanking

The portion of the video signal that is blanked so that the vertical retrace of the beam is not visible.

vertical blanking interval

The blanking portion at the beginning of each field. It contains the equalizing pulses, the vertical sync pulses, and vertical interval test signals (VITS). Also the period when a scanning process is moving from the lowest horizontal line back to the top horizontal line.

video level

Video signal amplitude.

video output

See *drain.*

video signal

The electrical signal produced by a scanning image sensor.

videotape formats

Table Gl-1 lists major videotape formats.

Table Gl-1   Videotape Formats

Electronics	Consumer	Professional	Broadcast	Postproduction
Analog	VHS cassette	U-Matic (SP) cassette, 3/4-inch	Type C reel-to-reel, 1-inch composite	
	S-VHS		Type B (Europe), composite	
	S-Video (YC-358)	S-Video (YC-358)		
	Beta		Betacam (component)	
	8mm		Type MII (component)	
	Hi-8mm (YC)	Hi-8mm (YC)		
Digital				D1 525/625 (YUV)
				D2 525 (NTSC)
				D2 625 (PAL)

video waveform

The main components of the video waveform are the active video portion and the horizontal blanking portion. Certain video waveforms carry information during the horizontal blanking interval.

Figure Gl-8 and Figure Gl-9 diagram a typical red or blue signal, which carries no information during the horizontal blanking interval, and a typical Y or green-plus-sync signal, which carries a sync pulse.

Active Video

Horizontal Blanking

Active Video

Figure Gl-8   Red or Blue Signal

Active Video

Horizontal Blanking
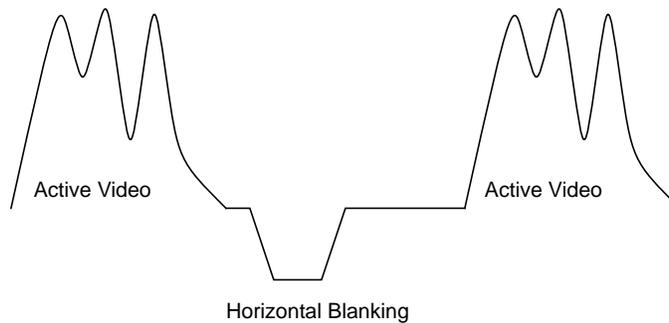
Active Video

Figure Gl-9   Y or Green Plus Sync Signal

193

Figure Gl-10 and Figure Gl-11 show the video waveform and its components for composite video in more detail. The figures show the composite video waveform with and without setup, respectively.

Figure Gl-10 shows a composite video signal with setup.
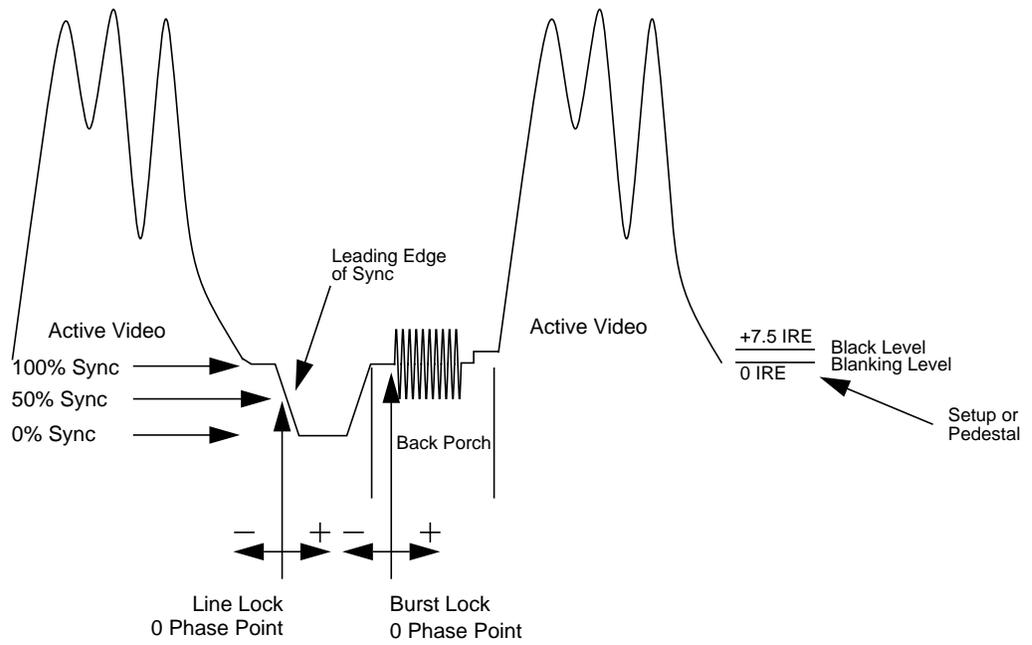
Figure Gl-10   Video Waveform: Composite Video Signal With Setup (Typical NTSC)

Figure Gl-11 shows a composite video signal without setup.

White Level — 100 IRE / 1.0 Volts

Horizontal Blanking

Active Video

Blanking and Black Level — 0 IRE

Sync Tip — -40 IRE / 0.0 Volts

Active Video

Burst

Breezeway

Front Porch

Sync or H Sync

Back Porch
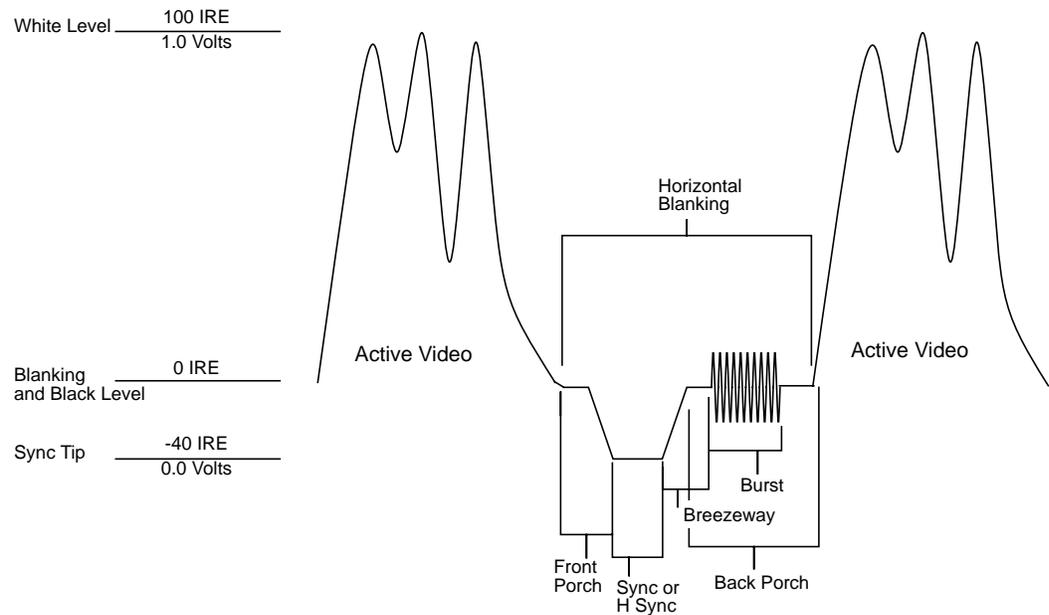
Figure Gl-11   Video Waveform: Composite Video Signal (Typical PAL)

white level

In the active video portion of the video waveform, the 1.0-volt (100 IRE) level. See also *video waveform*.

Y signal

Luminance, corresponding to the brightness of an image. See also *luminance* and *Y/R-Y/B-Y*.

YC

A color space (color component encoding format) based on YIQ or YUV. Y is luminance, but the two chroma signals (I and Q or U and V) are combined into a composite chroma called C, resulting in a two-wire signal. C is derived from I and Q as follows:

C - I cos(2 fsct) + Q sin(2 fsct)

where f_{sc} is the subcarrier frequency. YC-358 is the NTSC version of this luminance/chrominance format; YC-443 is the PAL version. Both are referred to as S-Video formats.

YIQ

A color space (color component encoding format) used in decoding, in which Y is the luminance signal and I and Q are the chrominance signals. The two chrominance signals I and Q (in-phase and quadrature, respectively) are two-phase amplitude-modulated; the I component modulates the subcarrier at an angle of 0 degrees and the Q component modulates it at 90 degrees. The color burst is at 33 degrees relative to the Q signal.

The amplitude of the color subcarrier represents the saturation values of the image; the phase of the color subcarrier represents the hue value of the image.

Y = 0.299R + 0.587G + 0.114B
I = 0.596R - 0.275G - 0.321B
Q = 0.212R - 0.523G + 0.311B

Y/R-Y/B-Y

A name for the YUV color component encoding format that summarizes how the chrominance components are derived. Y is the luminance signal and R-Y and B-Y are the chrominance signals. R-Y (red minus Y) and B-Y (blue minus Y) are the color differences or chrominance components. The color difference signals R-Y and B-Y are derived as follows:

Y = 0.299R + 0.587 + 0.114B

Y/R-Y/B-Y has many variations, just as NTSC and PAL do. All component and composite color encoding formats are derived from RGB without scan standards being changed. The matrix (amount of red, green, and blue) values and scale (amplitude) factors can differ from one component format to another (YUV, Y/R-Y, B-Y, SMPTE Y/R-Y, B-Y).

YUV

A color space (color component encoding format) used by the PAL video standard, in which Y is the luminance signal and U and V are the chrominance signals. The two chrominance signals U and V are two-phase amplitude-modulated. The U component modulates the subcarrier at an angle of 0 degree, but the V component modulates it at 90 degrees or 180 degrees on alternate lines. The color burst is also line-alternated at +135 and -135 degrees relative to the U signal. The YUV matrix multiplier derives colors from RGB via the following formula:

$Y = .299R + .587\,G + .114\,B$
$C_R = R - Y$
$C_B = B - Y$

In this formula, Y represents luminance; red and blue are derived from it: C_R denotes red and (V), C_B denotes blue. V corresponds to C_R; U corresponds to C_B c. The U and V signals are carried on the same bandwidth. This system is sometimes referred to as Y/R-Y/B-Y.

The name for this color encoding method is YUV, despite the fact that the order of the signals according to the formula is YVU.

Index

Tell Us About This Manual

As a user of Silicon Graphics documentation, your comments are important to us. They help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics to comment on:

- General impression of the document

- Omission of material that you expected to find

- Technical errors

- Relevance of the material to the job you had to do

- Quality of the printing and binding

Please include the title and part number of the document you are commenting on. The part number for this document is 007-3182-001.

Thank you!

Three Ways to Reach Us

The **postcard** opposite this page has space for your comments. Write your comments on the postage-paid card for your country, then detach and mail it. If your country is not listed, either use the international card and apply the necessary postage or use electronic mail or FAX for your reply.

If **electronic mail** is available to you, write your comments in an e-mail message and mail it to either of these addresses:

- If you are on the Internet, use this address: techpubs@sgi.com

- For UUCP mail, use this address through any backbone site:
  [your_site]!sgi!techpubs

You can forward your comments (or annotated copies of manual pages) to Technical Publications at this **FAX** number:

415 965-0964