# Indigo$^2$ Video™ for Indigo$^2$ IMPACT™ Programmer's Guide

Document Number 007-2917-001

CONTRIBUTORS

Written by Carolyn Curtis
Illustrated by Cheri Brown and Carolyn Curtis
Production by Heather Hermstad
Engineering contributions by I-Ching Wang, Ed Goldberg, Alan Chang, John Gibbon, Bob Abbott, Judy Ting, and Jeff Schmidt
Cover design and illustration by Rob Aguilar, Rikk Carey, Dean Hodgkinson, Erik Lindholm, and Kay Maitz



Figure 5-6, in Chapter 5, is derived from Thomas Porter and Tom Duff, “Compositing Digital Images,” published by the Association for Computing Machinery, 1984.

Indigo$^2$ Video™ for Indigo$^2$ IMPACT™ Programmer’s Guide
Document Number 007-2917-001

# Contents

# List of Figures

# List of Tables

# About This Guide

Indigo$^2$ Video™ for Indigo$^2$ IMPACT™ 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, as well as to conduct video conferencing. Video formats include composite or S-Video input, composite output with genlock, alpha blending in real time, and chroma and luma keying.

Indigo$^2$ Video for 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$^2$ Video for Indigo$^2$ IMPACT capabilities.

### Audience

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

### Structure of This Document

This guide contains the following chapters and appendixes:

- Chapter 1, “Features of the Indigo2 Video Option for Indigo2 IMPACT,” introduces features and capabilities of the Indigo$^2$ Video board for Indigo$^2$ IMPACT and presents an annotated sample program that displays live video input in a graphics window to help you get started with the Video Library. 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.
- Chapter 3, “Using VL Controls,” explains VL control type and values, VL control fraction ranges, VL control classes, and VL control groupings.
- Chapter 4, “VL Event Handling,” details querying VL events, creating a VL event loop, and creating a main loop with callbacks.
- Chapter 5, “Blending, Keying, and Transitions,” explains how to use VL to perform chroma keying, luma keying, and transitions. It explains the VL key generator, blend node, and blending controls.
- Appendix A, “Video Basics,” introduces basic video concepts.

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>** |

*Chapter 1*

# Features of the Indigo$^2$ Video Option for Indigo$^2$ IMPACT

Indigo$^2$ IMPACT workstation video capabilities are provided by the Video Library and Indigo$^2$ Video option board for Indigo$^2$ IMPACT. The board provides video input and output for Indigo$^2$ workstations equipped with IMPACT graphics.

This chapter introduces

- Video Library capabilities
- VL system software architecture
- VL architectural model of video devices
- VL programming model
- Indigo$^2$ Video formats

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

## Video Library Capabilities

The Video Library provides a software interface to the Indigo$^2$ Video board for Indigo$^2$ IMPACT, 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 single-frame 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/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

- perform video teleconferencing
- blend computer graphics with frames from videotape or any video source
- present video in a window on the workstation screen
- digitize video data

The software for the Indigo$^2$ Video board for Indigo$^2$ IMPACT includes a graphical user interface that makes it convenient to access VL capabilities. Features of this panel, */usr/sbin/vcp*, are referred to in the course of this guide.

## 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.

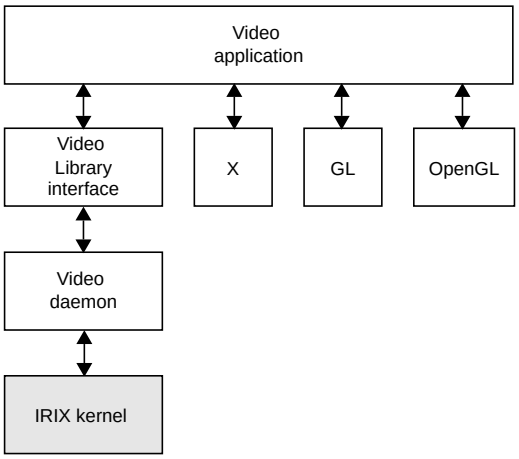



**Figure 1-1** VL System Components

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 video in a window. |
| *videoout* | Use the video output tool to output video from a rectangular or full-screen 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/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 file for Indigo$^2$ Video is */usr/include/vl/dev_ev1.h* (linked to */usr/include/dmedia/vl_ev1.h*).

## VL Architectural Model of Video Devices

The two central concepts for VL are:

- *path*: an abstraction for a way of moving data around
- *node*: an endpoint of the path, such as a video *source* (such as a VTR), video *drain* (such as the screen), a *device* (Indigo$^2$ video), or the *blender* in which video sources are combined for output to a drain

VL routines explained in this chapter enable you to build a fully connected topology of sources and drains.

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.

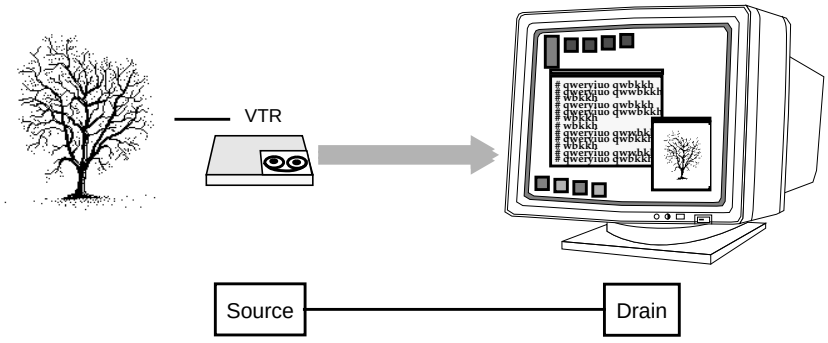



**Figure 1-2** Simple VL Path

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.

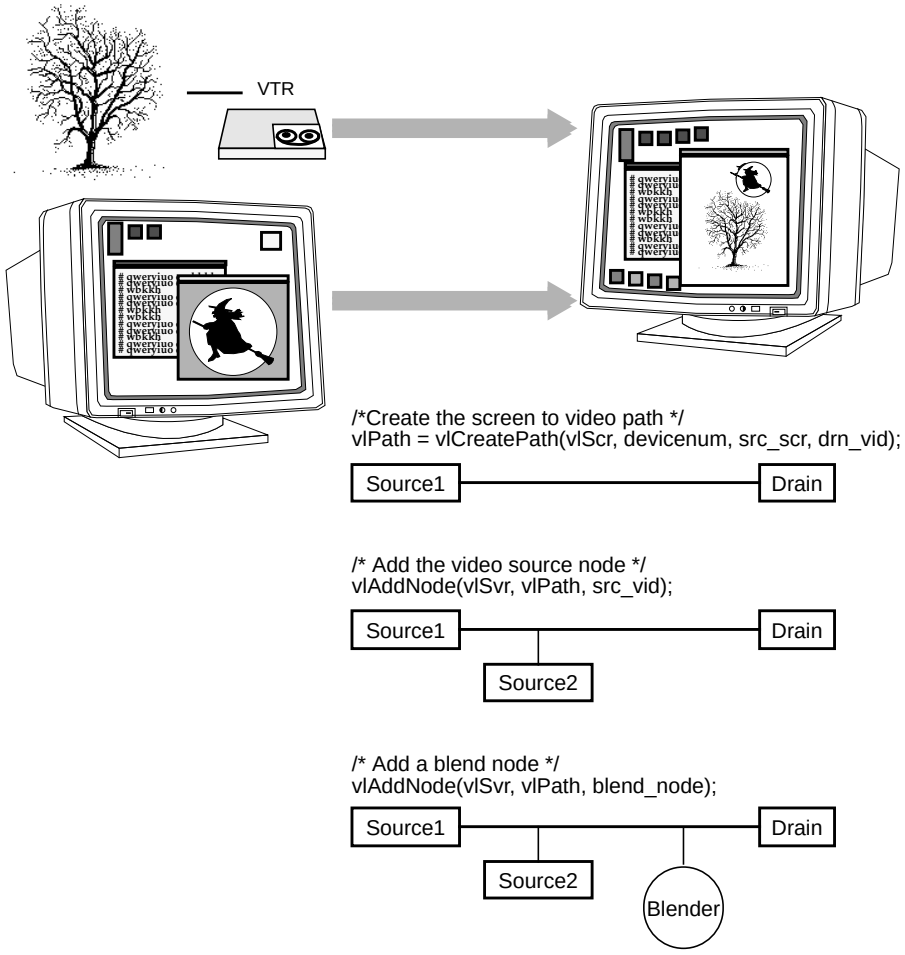



**Figure 1-3** Simple VL Blending

## VL Programming Model

The VL recognizes five classes of objects:

- *devices*, each including sets of nodes
- *nodes*: sources, drains, and internal nodes
- *paths*, connecting sources and drains
- *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 are implemented as ring buffers containing a number of blocks; each maintains a pointer, a size, and pointers to the head (oldest) and tail (newest) valid data

Data transfers fall into two categories:

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

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()**

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, determining 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 va ring buffer to hold data for memory transfers (**vlGetTransferSize()**, **vlCreateBuffer()**).
9. For transfers involving memory, register the buffer (**vlRegisterBuffer()**).
10. Start the data transfer (**vlBeginTransfer()**).
11. For transfers involving memory, get the data (**vlGetNextValid()** or **vlGetLatestValid()**, **vlGetActiveRegion()**, **vlPutFree()**) to manipulate frame data.
12. Clean up (**vlEndTransfer()**, **vlDeregisterBuffer**(), **vlDestroyPath()**, **vlDestroyBuffer()**, **vlCloseVideo()**).

Table 1-1 lists calls explained in this chapter.

**Table 1-1** 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() | |
| vlAddNode() | vlPutValid() | |
| vlRemoveNode() | vlGetNextFree() | |
| vlSetupPaths() | vlGetActiveRegion() | |
| vlSelectEvents() | vlPutFree() | |
| vlBeginTransfer() | vlGetDMediaInfo() | |
| vlEndTransfer() | vlGetImageInfo() | |
| vlDestroyPath() | vlDeregisterBuffer() | |
| vlCloseVideo() | vlDestroyBuffer() | |

## Indigo$^2$ Video for Indigo$^2$ IMPACT Formats

The Indigo$^2$ Video board for Indigo$^2$ IMPACT translates video signals into a form usable by the Indigo$^2$ workstation. It also does the reverse, translating graphics from the workstation display into video signals. This section describes the Indigo$^2$ Video board's I/O interface.

The Indigo$^2$ Video board's native video format is YUV 4:2:2. The board also supports 32-bit RGB video data. Data coming from or going to the Indigo$^2$ Video board for Indigo$^2$ IMPACT is always ordered top to bottom. The Indigo$^2$ workstation, on the other hand, commonly stores image data with lines ordered bottom to top. Both the Indigo$^2$ workstation and the Indigo$^2$ Video board for Indigo$^2$ IMPACT store pixels from left to right within lines.

The Indigo$^2$ Video board for Indigo$^2$ IMPACT uses 32-bit RGB format for single frame output, and it is produced by some of the video capture convenience routines. The format of these pixels is shown in Figure 1-4.

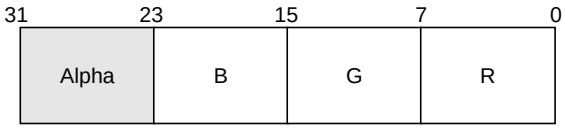



**Figure 1-4** Format of 32-Bit RGB Pixels

In Figure 1-4, the bits are numbered from right to left, with the least significant (rightmost) bit numbered zero.

**Note:** The alpha stored in the higher eight bits is not accessible except for memory transfers of RGB data to the Indigo$^2$ Video board for Indigo$^2$ IMPACT (such as in the program *memtovid.c*). When graphics data is input in real time to the video over a dedicated path, this alpha is not accessible.

You can use the capabilities of the Indigo$^2$ Video board to record video signals in composite or S-Video format. For information on the requirements for recording to and from video, see the *Indigo$^2$ Video Owner's Guide*. For information on videotape formats, see "Videotape Formats" in Appendix A.

*Chapter 2*

# Creating Video Programs With the Video Library

VL calls let you perform video teleconferencing on platforms that support it, 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

- opening a connection to the video daemon 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
- some example programs

The chapter concludes with example code illustrating a simple screen application and frame grabs (video to memory, memory to video, and continuous frame capture).

To run VL, you must

- install the *dmedia_dev* option
- link with *libvl.so*
- include *vl/vl.h* and *vl/dev_ev1.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 file for Indigo$^2$ Video for Indigo$^2$ IMPACT is *dev_ev1.h* (linked to */usr/include/dmedia/vl_ev1.h*).

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

## Opening a Connection to the Video Daemon and Setting Up a Data Path

Preliminary procedures required to create the data path are:

- opening the device
- specifying nodes on the data path
- creating and setting up the data path

Each procedure is explained separately.

### 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_DEVICE: device for device-global controls

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

- VL_INTERNAL: internal node, such as the blend node

*kind* specifies the kind of node:

- VL_VIDEO: connection to a video device; for example, a video tape deck or camera
- VL_MEM: region of workstation memory
- VL_SCREEN: workstation screen
- VL_BLENDER: a blender node

  **Note:** The use of VL_BLENDER is explained in Chapter 5, "Blending, Keying, and Transitions," later in this guide.

- VL_ANY: use any available node

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

To use the default node kind, use VL_ANY.

```
nodehandle = vlGetNode(vlSvr, VL_SRC, VL_VIDEO, VL_ANY);
```

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 second line above, the last argument is a struct that gets the value.

## 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, but can only be 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 device, 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);
}
```

### 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 2-1 lists the possibilities.

Table 2-1 lists and describes the VL event masks

**Table 2-1** VL Event Masks

| Symbol | Meaning |
|---|---|
| VLStreamBusyMask | Stream is locked |
| VLStreamPreemptedMask | Stream was grabbed by another application |
| 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 |

**Table 2-1** (continued) VL Event Masks

| Symbol | Meaning |
|---|---|
| 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 ring 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 Source Node Controls for Data Transfer

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

**Table 2-2** Data Transfer Controls for Source Nodes

| Control | Values | Basic Use |
|---|---|---|
| VL_MUXSWITCH | See Table 2-3 | Determines physical input for path |
| VL_TIMING | Default: timing produced by active signal | Set or get video timing |
| | VL_TIMING_525_SQ_PIX<br>VL_TIMING_625_SQ_PIX | For Betacam, MII, composite tape formats:<br>Analog: 12.27 MHz, 646 x 486<br>Analog: 14.75 MHz, 768 x 576 |
| VL_SIZE | Coordinates | Set or get active unmodified video area |
| VL_SYNC_SOURCE | Composite 1: set 0<br>Composite 2: set 2 | |

The use of VL_MUXSWITCH and VL_TIMING is explained in further detail in the following sections.

### Using VL_MUXSWITCH

Use VL_MUXSWITCH to switch between physical inputs on a single path. Table 2-3 summarizes values to set.

**Table 2-3** VL_MUXSWITCH Values

| Connector | Value |
|---|---|
| Y/C (RCA jacks) | 0 |
| Y/C (S-Video connector) | 1 |
| Composite input 1 | 3 |
| Composite input 2 | 4 |

For the VL, the default source is the one that is plugged in and selected.

#### 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 |

## Setting Drain Node Controls for Data Transfer

Important data transfer controls for drain nodes are summarized in Table 2-5. They should be set in the order in which they appear in the table.

**Table 2-5** Data Transfer Controls for Drain Nodes

| Control | Basic Use | Video Nodes | Memory Nodes | Screen Nodes |
|---|---|---|---|---|
| VL_FORMAT | Video format on the physical connector | See "Using VL_FORMAT" in this chapter | | |
| VL_TIMING | Video timing | See Table 2-2 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<br>VL_CAPTURE_INTERLEAVED<br>VL_CAPTURE_EVEN_FIELDS<br>VL_CAPTURE_ODD_FIELDS<br>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 2-5** (continued) Data Transfer Controls for Drain Nodes

| Control | Basic Use | Video Nodes | Memory Nodes | Screen Nodes |
|---|---|---|---|---|
| VL_ZOOM | Decimation or zoom factor (fraction) for screen: 1/1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 2/1, 4/1 | Not applicable | Decimation or zoom: 1/1, 1/4, 1/8 | Decimation or zoom: resizes data to remain within limits |
| 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 |

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:** 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.

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_FORMAT, VL_PACKING, VL_ZOOM, VL_SIZE, VL_OFFSET, VL_RATE, and VL_CAP_TYPE is explained in more detail in the following sections.

### Using VL_FORMAT

To specify video input and output formats of the video signal on the physical connector, use VL_FORMAT. For Indigo$^2$ Video for Indigo$^2$ IMPACT, the native format is YUV 4:2:2; this format is always fastest. The Indigo$^2$ Video option also supports 32-bit RGB.

**Note:** To convert formats, use VL_PACKING, which is explained in the next section.

### Using VL_PACKING

To convert a video output format to another in memory, use the control VL_PACKING. Packing type expresses the packing in memory of the video data at the source or drain.

Packing types are summarized in Table 2-6, which shows the most significant byte (MSB) on the left. An `X` means don't care; this bit is not used.

**Table 2-6** Packing Types and Their Sizes and Formats

| Type | Size | Format<br>MSB ----------LSB |
|---|---|---|
| VL_PACKING_RGB_332_P | 8-bit word | BBGGGRRR (four pixels packed into a 32-bit word) |
| VL_PACKING_RGBA_8 | 32-bit word | AAAAAAAA BBBBBBBB GGGGGGGG RRRRRRRR |
| VL_PACKING_RGB_8 | 24-bit word | XXXXXXXX BBBBBBBB GGGGGGGG RRRRRRRR |

**Table 2-6** (continued) Packing Types and Their Sizes and Formats

| Type | Size | Format<br>MSB ----------LSB |
|---|---|---|
| VL_PACKING_Y_8_P | 8-bit word | YYYYYYYY (four pixels packed into a 32-bit word) |
| VL_PACKING_YVYU_422_8 (native format) | 32-bit word | UUUUUUUU YYYYYYYY VVVVVVVV YYYYYYYY |

**Note:** The packing names follow the naming conventions used by the IRIS Graphics Library; other libraries such as the OpenGL may use different names.

For example:

```
VLControlValue val;

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

### 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.

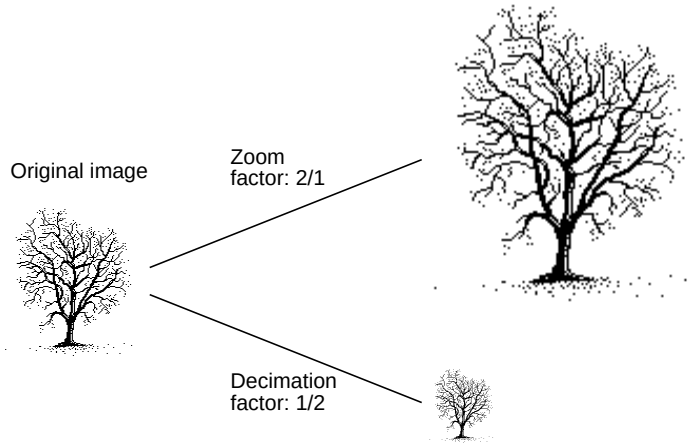



**Figure 2-1** Zoom and Decimation

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

```
val.fractVal.numerator = 1;
val.fractVal.denominator = 2;
if (vlSetControl(server, memory_path, memory_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 memory, with clipping to 320 × 243 (NTSC size minus overscan).

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

```
}
val.xyVal.x = 320;
val.xyVal.y = 243;
if (vlSetControl(server, memory_path, memory_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, memory_path, video_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, memory_path, memory_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, memory_path, memory_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.

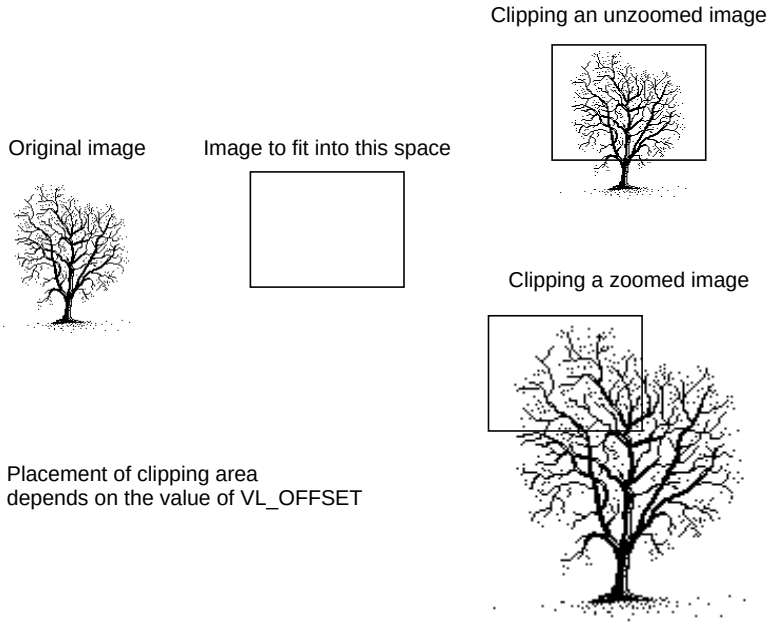



**Figure 2-2** Clipping an Image

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.

VL_OFFSET operates on the unzoomed image; it does not change if the zoom factor is changed.

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.

### Using VL_RATE and VL_CAP_TYPE

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 2-7.

**Table 2-7** VL_RATE Values (Items per Second)

| VL_CAP_TYPE Value | VL_RATE Value |
|---|---|
| VL_CAPTURE_NONINTERLEAVED only | NTSC: 10, 12, 20, 24, 30, 36, 40, 48, 50, 60<br>PAL: 5, 10, 15, 20, 25 |
| VL_CAPTURE_INTERLEAVED, VL_CAPTURE_EVEN_FIELDS, VL_CAPTURE_ODD_FIELDS, and VL_CAPTURE_FIELDS | NTSC: 5, 6, 10, 12, 15, 18, 20, 24, 25, 30<br>PAL: 10, 20, 30, 40, 50 |

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

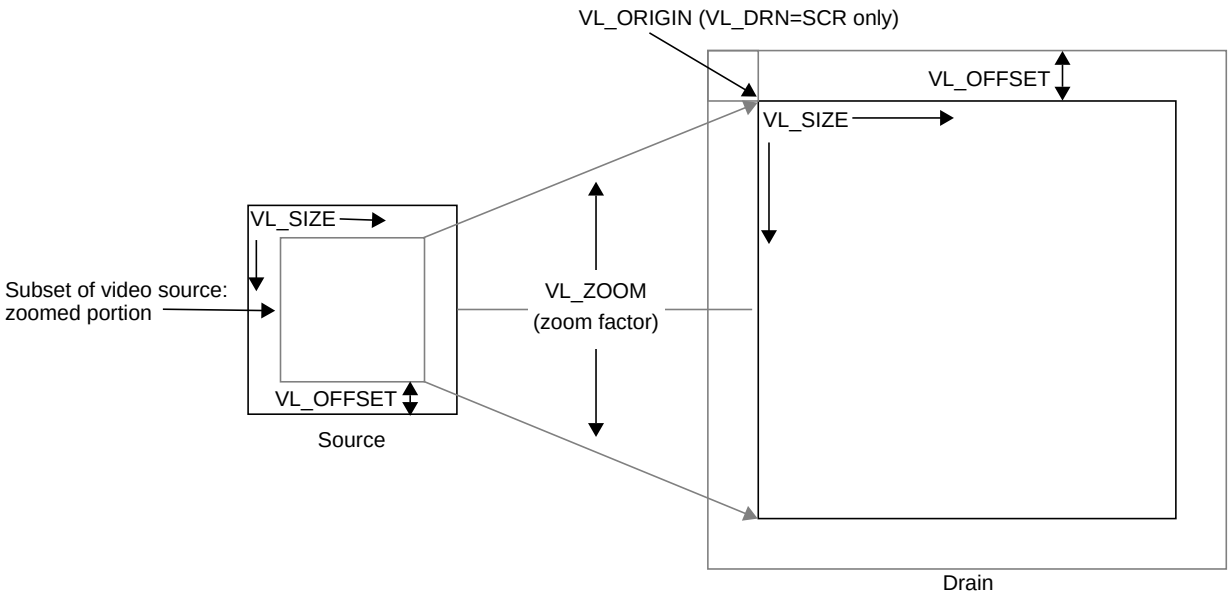



**Figure 2-3** Zoom, Size, Offset, and Origin

## 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:

- creating a buffer for the frames (for transfers involving memory)
- registering the ring 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 the Frames

Once you have specified frame parameters in a transfer involving memory (or have determined to use the defaults), create a buffer for the frames.

Like other libraries in the IRIX digital media development environment, the VL uses ring buffers. Ring buffers provide a way to read and write varying sizes of frames of 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 ring buffer is created, constraints are specified that control the total size of the data segment and the number of information buffers to allocate.

A head and a tail flag are automatically set in a ring 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 ring buffer is written to and all sectors are occupied, data transfer stops. The sector last written to remains locked down until it is released.

The ring buffer can accommodate data of varying sizes. You can specify a ring buffer at a fixed size, or you can determine the size of the data in the buffer. To determine frame data size, use **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.

To create a ring 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 ring buffer |
| *numFrames* | specifies the number of frames in the buffer |

For example:

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

## Registering the Ring Buffer

Use **vlRegisterBuffer()** to register the ring 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 ring 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 a transfer descriptor |

Tailor the data transfer by means of *transfer descriptors*. The transfer descriptors are:

*xferDesc.mode* Transfer method:

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

- 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.

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 2-8 lists the buffer-related calls.

**Table 2-8** Buffer-Related Calls

| Call | Purpose |
|---|---|
| **vlGetNextValid()** | Returns a handle on the next valid frame of data |
| **vlGetLatestValid()** | Reads only the most current frame in the buffer, discarding the rest |
| **vlPutValid()** | Puts a frame into the valid list (memory to video) |
| **vlPutFree()** | Puts a valid frame 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()**.

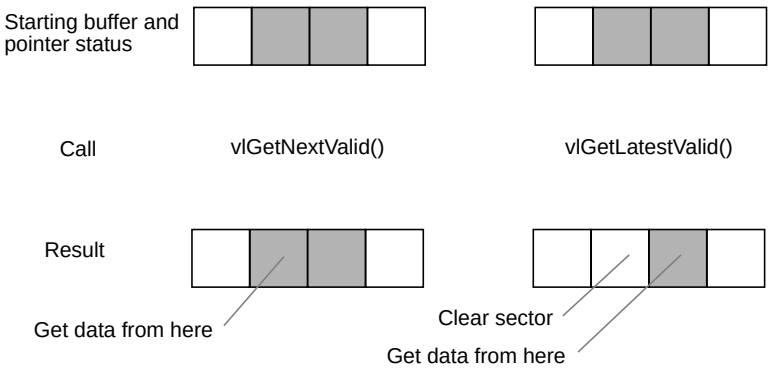



**Figure 2-4** **vlGetNextValid()**, **vlGetLatestValid()**, and **vlPutFree()**

Table 2-9 lists the calls that extract information from a buffer.

**Table 2-9** 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)
```

To accomplish the necessary cleanup to exit gracefully, use:

- for transfer 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 ring 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/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.

**Note:** To simplify the code, these examples do not check returns. The programmer should, however, always check returns.

See Chapter 4 for a description of *eventex.c* and Chapter 5 for descriptions of *simpleblend.c* and *simplewipe.c*.

The directory */usr/people/4Dgifts/examples.OpenGL* contains three example OpenGL programs:

- *contcapt.c*: 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/examples/dmedia/video/vl*.

*Chapter 3*

# Using VL Controls

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$^2$ Video board for Indigo$^2$ IMPACT are documented in the header file */usr/include/media/vl_ev1.h*).

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

**Table 3-1** Device-Independent VL Controls

| 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<br>VL_BLDFCN_ONE<br>VL_BLDFCN_A_ALPHA: foreground_alpha/255<br>VL_BLDFCN_MINUS_A_ALPHA: 1 - (foreground_alpha/255) |
| VL_BLEND_B_FCN | Blend function that controls mixing of background (channel B) signals | VL_BLDFCN_ZERO<br>VL_BLDFCN_ONE<br>VL_BLDFCN_B_ALPHA: background_alpha/255<br>VL_BLDFCN_MINUS_B_ALPHA: 1 - (background_alpha/255) |
| VL_BLEND_A_NORMALIZE | | 1 = on; off is not supported |
| 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<br>1 = on |
| VL_BLEND_OUT_NORMALIZE | | Not supported |
| VL_BRIGHTNESS | Brightness | |
| VL_CAP_TYPE | Type of frame(s) or field(s) to capture; see "Interlacing" in Appendix A | |
| VL_CONTRAST | | |
| VL_DEFAULT_SOURCE | Default source for the video path | |
| VL_FORMAT | Video format | |

**Table 3-1** (continued) Device-Independent VL Controls

| Control | Purpose | Comments |
|---|---|---|
| VL_FREEZE | Data transfer freeze; suspends transfer at the drain node, with no picture regeneration | 0 = off<br>1 = on |
| VL_H_PHASE | Horizontal phase | |
| VL_HUE | Hue; the control panel *vcp* calculates numerator and denominator | |
| VL_MUXSWITCH | Switch between inputs on a single path, corresponding to the physical connector to the option | Y/C (RCA jacks)<br>set 0<br>Y/C (S-Video connector):<br>set 1<br>Composite input 1: set 3; input 2: set 4 |
| VL_OFFSET | On VL_VIDEO nodes, the offset to the active region of the video; on all other nodes, the offset within the video<br>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_SIGNAL | | N/A |
| 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<br>VL_SYNC_GENLOCK<br>VL_SYNB_SLAVE |
| VL_SYNC_SOURCE | Sets sync source for analog breakout box | Composite 1: set 0<br>Composite 2: set 2 |
| VL_TIMING | Video timing | |
| VL_V_PHASE | Vertical phase | Integer |
| VL_WINDOW | Window ID for video in a window (screen node only) | Integer |
| VL_ZOOM | Zoom factor for video stream; fractions greater than 1 expand the picture, fractions less than one reduce the picture | 4/1, 2/1, 1/1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8 |

**Note:** For information on controls for keying, blending, or wipes, see Chapter 5, "Blending, Keying, and Transitions."

For detailed information on using VL_CAP_TYPE, VL_FORMAT, VL_MUXSWITCH, VL_OFFSET, VL_PACKING, VL_RATE, VL_SIZE, VL_TIMING, and VL_ZOOM, 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 the 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 info 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 VLProgressionType 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}.bas*e, 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)))
```

to 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 |
| 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_MUXSWITCH or 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 |

**Table 3-2** (continued) VL Control Groupings

| Grouping | Includes controls for... |
|---|---|
| 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 |

*Chapter 4*

# VL Event Handling

The 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

- 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.

## Querying VL Events

General VL event handling routines are summarized in Table 4-1.

**Table 4-1** VL Event Handling Routines

| Routine | Use |
|---|---|
| **vlGetFD()** | Gets a file descriptor for a VL server |
| **vlNextEvent()** | Gets the next event; blocks until you get the next event from the queue |
| **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()** | Gets 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 4-2 summarizes VL event masks.

**Table 4-2** VL Event Masks

| Symbol | Meaning |
|---|---|
| VLStreamBusyMask | Stream is locked |
| VLStreamPreemptedMask | Stream was grabbed by another application |
| 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);
```

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 which 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.

*Chapter 5*

# 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$^2$ Video board for Indigo$^2$ IMPACT 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$^2$ Video board for Indigo$^2$ IMPACT, provides convenient access to blending, keying, and transition controls.

This chapter explains

- the blend node
- keying

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

## The Blend Node

Blending takes place in the VL's internal *blend node*, which is created with the **vlGetNode()** function. The code fragment in Example 5-1 sets up source, drain, and blend nodes.

**Example 5-1** Setting Up Source, Drain, and Blend 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);
```

The blend node is supplied by up to 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 5-1 diagrams the blend node.

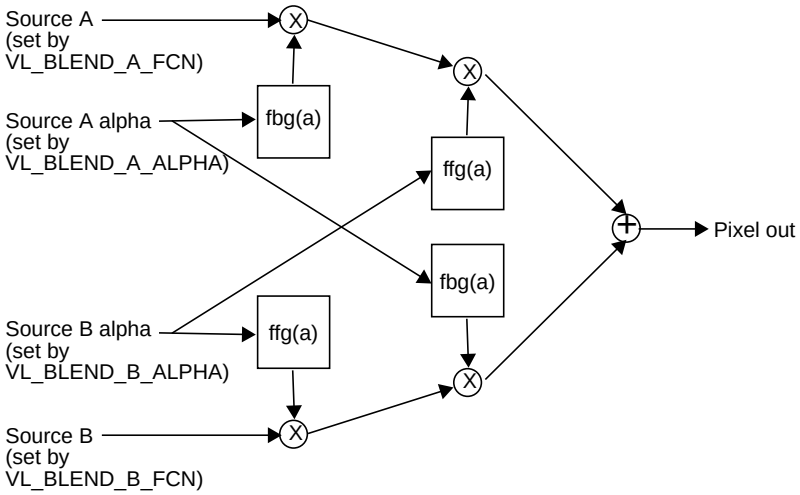



**Figure 5-1** Blend Node

The keyer can serve as the alpha value for source A.

The blend node has four multiplier stages, indicated by ⊗ in Figure 5-1, and one adder stage, indicated by ⊕. The values in the four multiplier stages are based on the blending functions selected and on the input normalization controls.

The rest of this section explains

- using VL to set up the blend node
- using VL blending controls to set blend function values
- using *vcp* to set blend function values

## Using VL to Set Up the Blend Node

Figure 5-2 diagrams setting up the blend node.

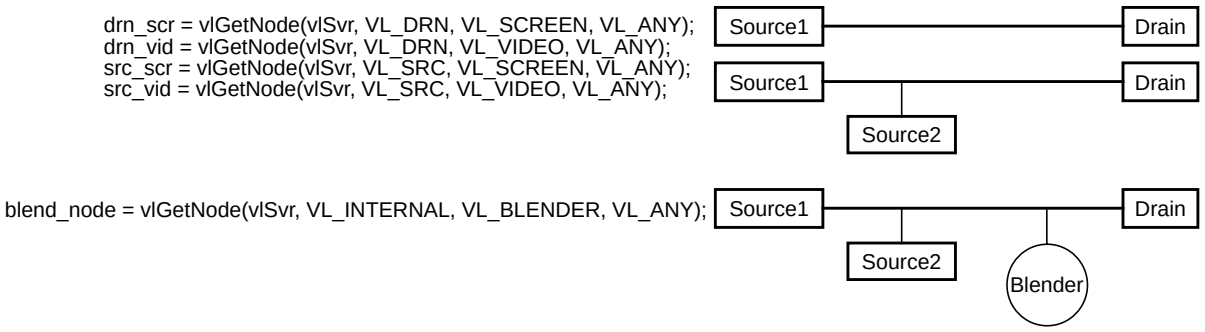



**Figure 5-2** Setting Up the Blend Node

The blend node mixes the foreground and background video signals by applying a blend function to the foreground and background pixels.

The blend node then sends the data to the drain node. For example, blending analog video with part of a graphics screen and sending it to video out can be diagrammed as shown in Figure 5-3.

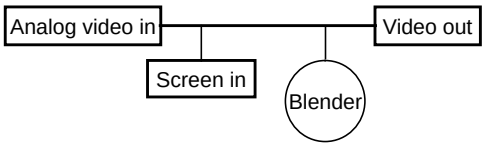



**Figure 5-3** Blending Analog Video With Part of a Graphics Screen

Blending analog video with static frame data and sending it to video out can be diagrammed as shown in Figure 5-4.

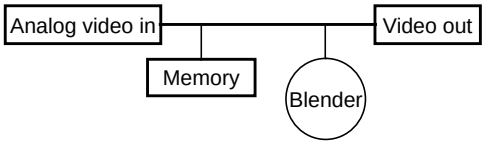



**Figure 5-4** Blending Analog Video With Static Frame Data

## Using VL Blending Controls to Set Blend Function Values

The VL uses blending controls to set blending options.

All blending controls—that is, all the controls discussed in this chapter—apply only to blend nodes, except for VL_EV1_ALPHA_NOT_PIXEL, which applies to drain nodes. The order of blending and zooming depends on the node type: for a source, zooming takes place before blending; for a drain, blending takes place before zooming.

Table 5-1 summarizes the VL controls for blending.

**Table 5-1** VL Blend Controls

| Control | Values | Selects |
|---|---|---|
| VL_BLEND_A<br>type *intVal* | VLNode type, derived from **vlGetNode()**; must be one of the two source nodes | Input source for foreground image |
| VL_BLEND_B<br>type *intVal* | VLNode type, derived from **vlGetNode()**; must be one of the two source nodes | Input source for background image |
| VL_BLEND_A_ALPHA<br>type *intVal* | VLNode type, derived from **vlGetNode()**; must be one of the two source nodes | Input source for foreground alpha |
| VL_BLEND_B_ALPHA<br>type *intVal* | VLNode type, derived from **vlGetNode()**; must be one of the two source nodes | Input source for background alpha |
| VL_BLEND_A_FCN<br>type *intVal* | VL_BLDFCN_ZERO<br>VL_BLDFCN_ONE<br>VL_BLDFCN_B_ALPHA<br>background_alpha/255<br>VL_BLDFCN_MINUS_B_ALPHA:<br>1 - (background_alpha/255) | Blend function that controls mixing of foreground signals |

**Table 5-1** (continued) VL Blend Controls

| Control | Values | Selects |
|---|---|---|
| VL_BLEND_B_FCN<br>type *intVal* | VL_BLDFCN_ZERO<br>VL_BLDFCN_ONE<br>VL_BLDFCN_A_ALPHA<br>foreground_alpha/255<br>VL_BLDFCN_MINUS_A_ALPHA<br>1 – (foreground_alpha/255) | Blend function that controls mixing of background signals |
| VL_BLEND_A_NORMALIZE<br>type *boolVal* | (0,1)<br>0 = off, 1 = on | Off is not supported by Indigo2 Video for Indigo2 IMPACT |
| VL_BLEND_B_NORMALIZE<br>type *boolVal* | (0,1)<br>0 = off, 1 = on | Follows Porter-Duff model (background pixels premultiplied by their corresponding alphas before blending) |
| VL_BLEND_OUT_NORMALIZE<br>type *boolVal* | 1 = on | Not supported by IMPACT Indigo$^2$ Video |

## Using *vcp* to Set Blend Function Values

The blender has two inputs, path A and path B. Each has an associated alpha that is separately controllable. IMPACT Indigo$^2$ Video-specific VL controls set these inputs; they are discussed in "Using VL to Set Up the Blend Node," later in this chapter. Settings in the panel control blending of these paths.

To use the panel to set the blend functions that control mixing of frames from paths A and B, select "Blend/Wipe Node" in the Pro menu. The following choices appear:

- All
- Keying Controls
- Path Controls
- Blending Controls
- Orientation Controls
- Rates Controls
- Visual Quality Controls

Select “Path Controls”; the Blend/Wipe Node-Path Controls window appears, as shown in Figure 5-5.

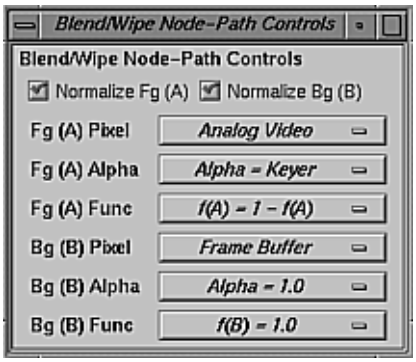



**Figure 5-5** *vcp* Path Controls Window

Notice that Blend Function A serves as foreground and Blend Function B serves as background.

The choices for foreground and background pixel and alpha values differ, however, as shown in Table 5-2.

**Table 5-2** Choices for Foreground and Background Pixel and Alpha Values

| Fg (A) Pixel | Bg (B) Pixel | Fg (A) Alpha | Bg (B) Alpha |
|---|---|---|---|
| Analog Video | Analog Video | Analog Video | Analog Video |
| IndyCam™ | IndyCam | IndyCam | IndyCam |
| Frame Buffer | Frame Buffer | Frame Buffer | Frame Buffer |
| Graphics A | Graphics A | Graphics A | Graphics A |
| Graphics B | Graphics B | Graphics B | Graphics B |
| | Blender Alpha | Alpha = Keyer | Alpha = 0.0 |
| | Blender Pixel | Alpha = 1.0 | Alpha = 1.0 |

Table 5-3 shows the choices for the two blend functions A and B, which correspond exactly.

**Table 5-3** 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.

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

The checkboxes at the top of the Path Controls window set normalization on for each blending source. Normalized background pixels for a frame are premultiplied by their corresponding alphas before they are blended (Porter-Duff model). Figure 5-6 gives some examples of compositing, assuming normalized inputs.

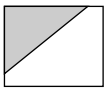
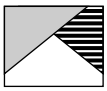
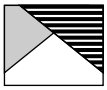
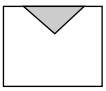
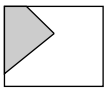
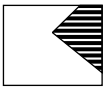
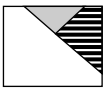
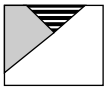
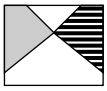
| Operation | Diagram | f(A)= | f(B)= |
|---|---|---|---|
| Clear | | 0 | 0 |
| A |  | 1 | |
| B |  | | 1 |
| A over B |  | 1 | 1−f(A) |
| B over A |  | 1−f(B) | 1 |
| A in B |  | f(B) | |
| B in A |  | | f(A) |
| A held out by B |  | 1−f(B) | |
| B held out by A |  | | 1−f(A) |
| A atop B |  | f(B) | 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 fo Computing Machinery, 1984.

**Figure 5-6** Binary 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.

## 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$^2$ Video board for Indigo$^2$ IMPACT "master" keyer control, VL_EV1_KEYER_MODE, determine the type of keying performed:

- luma keying: VL_EV1_KEYERMODE_LUMA
- chroma keying: VL_EV1_KEYERMODE_CHROMA
- transitions, that is, fades, tiles, or wipes: VL_EV1_KEYERMODE_SPATIAL

For example, the following fragment specifies that chroma keying is to be performed:

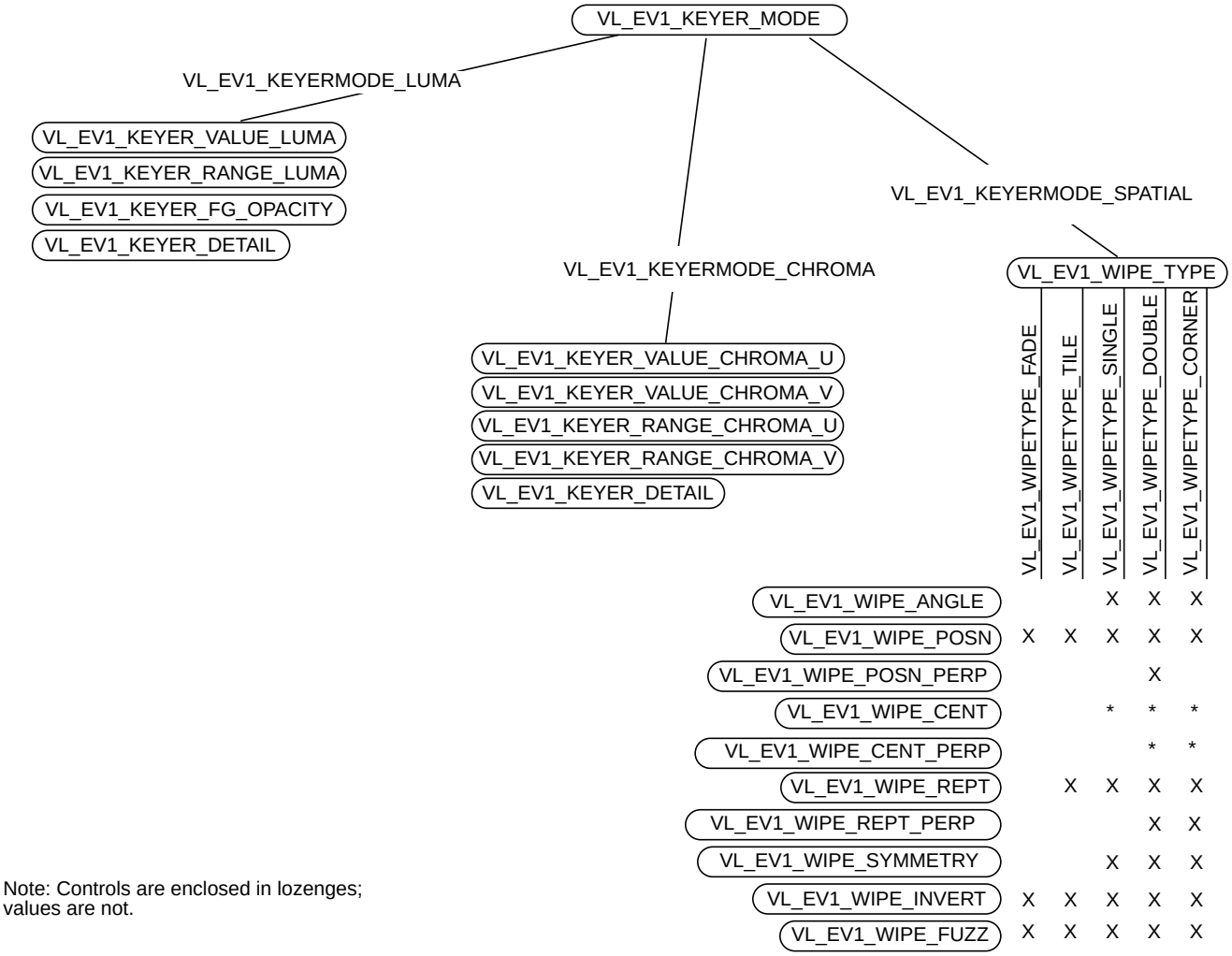
```
VLControlType val;
val.intVal = VL_EV1_WIPETYPE_FADE;
vlSetControl(vlSvr, vlPath, blend_node, VL_EV1_WIPE_TYPE,
             &val);
```

Keying controls fall into three groups:

- luma keying
- chroma keying
- fades, tiles, and wipes

Each type of keying is explained separately in this section. The section concludes with a discussion of the Indigo$^2$ Video for Indigo$^2$ IMPACT keyer.

Figure 5-7 shows relationships between the Indigo$^2$ Video board keying controls.




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

**Figure 5-7** Indigo$^2$ Video for Indigo$^2$ IMPACT Keying Controls

## 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 5-8 diagrams an application.

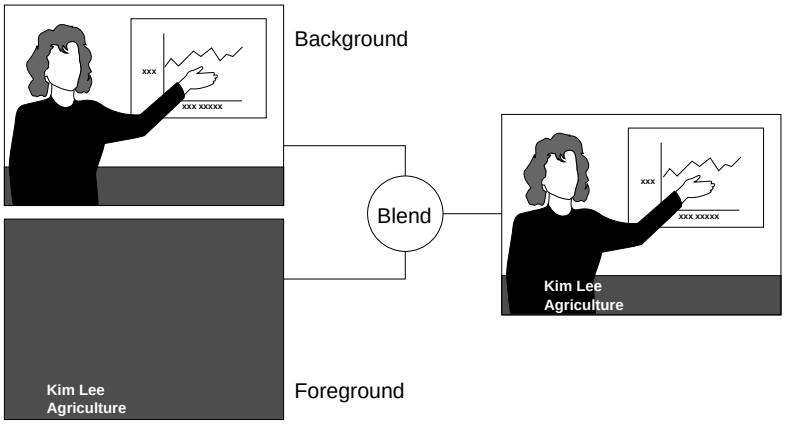



**Figure 5-8** Luma Keying Application: Titling

The Indigo$^2$ Video for Indigo$^2$ IMPACT luma keying controls are summarized in Table 5-4; each is of type **intVal**.

**Table 5-4** Indigo$^2$ Video for Indigo$^2$ IMPACT Luma Keying Controls

| Control | Range | Sets |
|---|---|---|
| VL_EV1_KEYER_VALUE_LUMA | (0,255) | Central luma value. This control sets the luma value at which the background shows through the foreground. |
| VL_EV1_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. |
| VL_EV1_KEYER_FG_OPACITY | (0,255) | Opacity of the foreground, thus limiting the value of foreground alpha at any point. |
| VL_EV1_KEYER_DETAIL | (-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 5-9 diagrams the relationships between these controls.

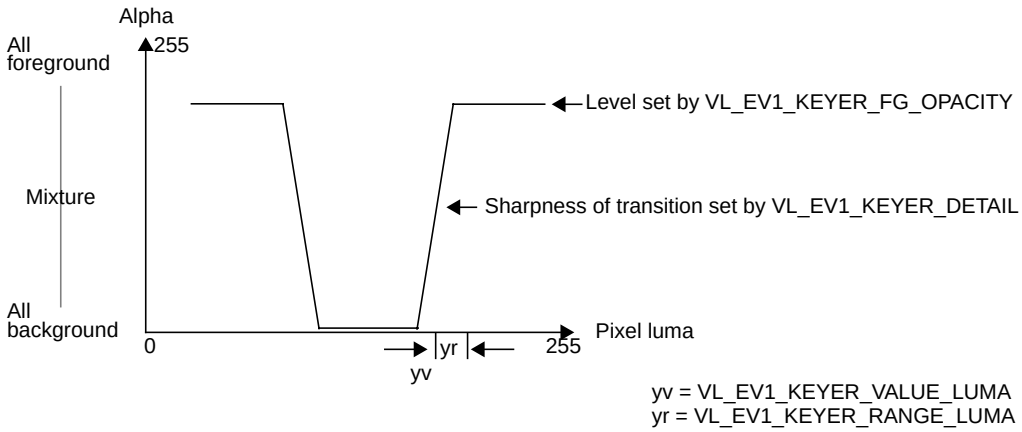



**Figure 5-9** Relationships Between Indigo$^2$ Video for Indigo$^2$ IMPACT Luma Keying Controls

## Chroma Keying

Chroma keying overlays one image on another based on the color value. Figure 5-10 diagrams a common application.

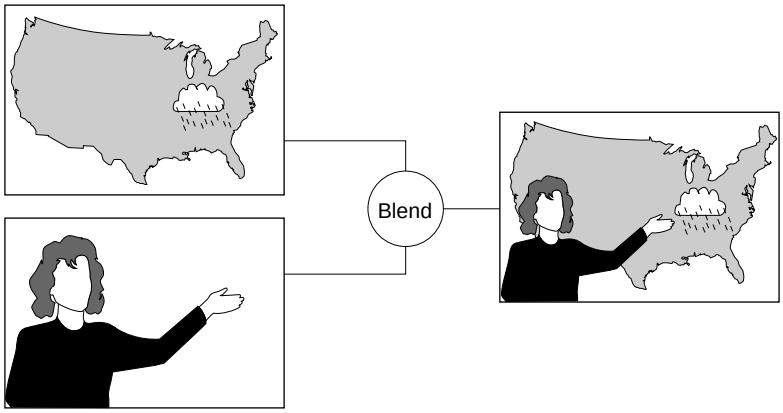



**Figure 5-10** Chroma Keying Application: TV Weather Map

Table 5-5 summarizes the controls for Indigo$^2$ Video for Indigo$^2$ IMPACT chroma keying and gives their ranges. These controls are all of type **intVal**.

**Table 5-5** Indigo$^2$ Video for Indigo$^2$ IMPACT Chroma Keying Controls

| Control | Range | Sets |
|---|---|---|
| VL_EV1_KEYER_VALUE_CHROMA_U | (-226,226) | Central U value at which the background shows through the foreground. |
| VL_EV1_KEYER_RANGE_CHROMA_U | (0,452) | One-sided range of U where the background shows through the foreground. |
| VL_EV1_KEYER_VALUE_CHROMA_V | (-179,179) | Central V value at which the background shows through the foreground. |
| VL_EV1_KEYER_RANGE_CHROMA_V | (0,358) | One-sided range of V where the background shows through the foreground. |
| VL_EV1_KEYER_DETAIL | (-8,7) | Sharpness of transition between foreground and background |

**Note:** VL_EV1_KEYER_FG_OPACITY has no effect on Indigo$^2$ Video for Indigo$^2$ IMPACT in chroma key mode.

Figure 5-11 diagrams the relationships between these controls.

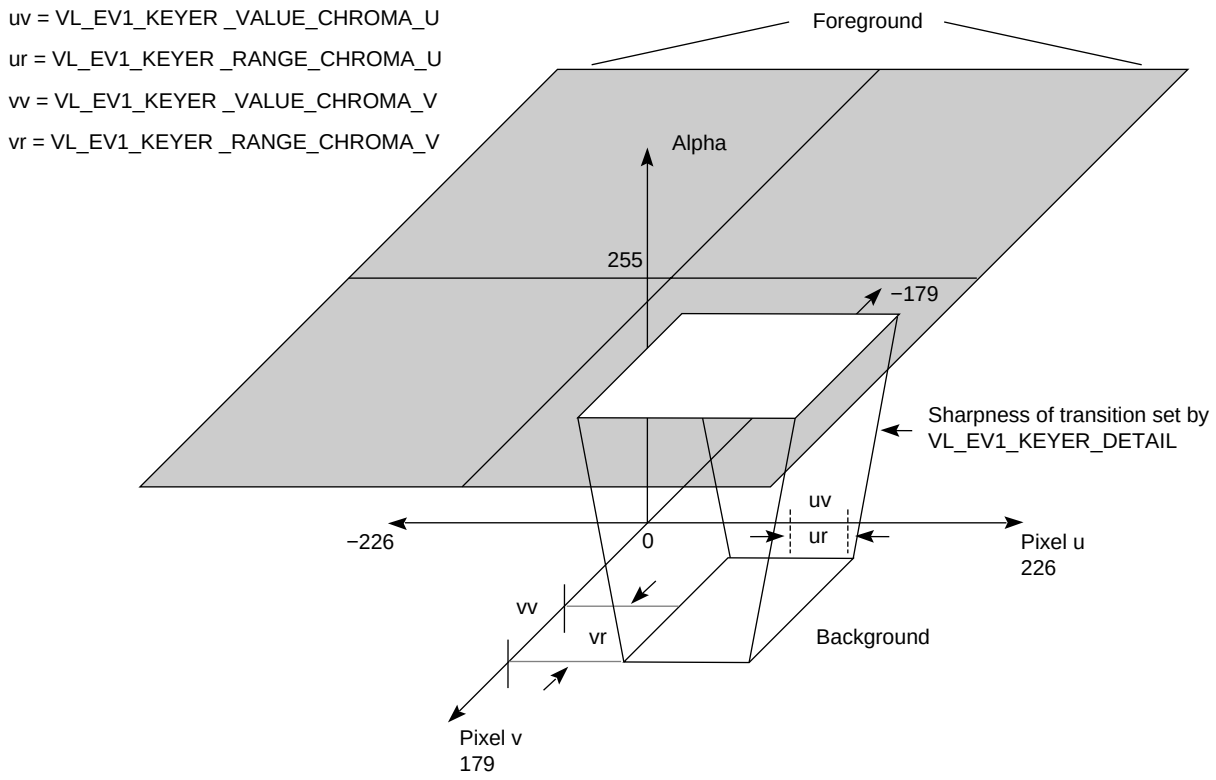



**Figure 5-11** Relationships Between Indigo$^2$ Video for Indigo$^2$ IMPACT Chroma Keying Controls

## Fades, Tiles, and Wipes

The values used with the control VL_EV1_WIPE_TYPE determine the type of blending performed:

- from all-foreground to all-background: VL_EV1_WIPETYPE_FADE
- from all-foreground to all-background by randomly tiling screen with rectangles of a specified size: VL_EV1_WIPETYPE_TILE
- wipe to cross the screen as a vertical, diagonal, or horizontal “front,” with a specified angle: VL_EV1_WIPETYPE_SINGLE

- wipe in two orthogonal directions simultaneously (two single wipes at the same time): VL_EV1_WIPETYPE_DOUBLE
- wipe in two orthogonal directions, with the perpendicular position locked to the normal, or in-line position: VL_EV1_WIPETYPE_CORNER

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

```
VLControlType val;
val.intVal = VL_EV1_WIPETYPE_FADE;
vlSetControl(vlSvr, vlPath, blend_node, VL_EV1_WIPE_TYPE,
             &val);
```

Fades, tiles, and wipes go from all-foreground (VL_EV1_WIPE_POSN=0) to all-background (VL_EV1_WIPE_POSN=1000), unless VL_EV1_WIPE_INVERT control is set, in which case they go from all-background (VL_EV1_WIPE_POSN = 0) to all-foreground (VL_EV1_WIPE_POSN = 1000).

Table 5-6 summarizes controls common to all wipe types.

**Table 5-6** Controls for Fades, Tiles, and Wipes

| Control | Values | Sets |
|---|---|---|
| VL_EV1_WIPE_POSN<br>type *fractVal* | Numerator (0,1000)<br>Denominator (1000) | Amount of progress of wipe, from none (numerator = 0) to full (numerator = 1000). |
| VL_EV1_WIPE_REPT<br>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.<br>Note that this control does not apply to fades. |
| VL_EV1_WIPE_INVERT<br>type *intVal* | (0,1)<br>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).<br>This value is buffered (does not go into effect) until another blending control is set. |

Table 5-7 summarizes the controls specific to wipes or that work differently for wipes. Some of these controls work in conjunction with each other.

**Table 5-7** Indigo$^2$ Video for Indigo$^2$ IMPACT Controls Specific to Wipes

| Control | Values | Sets |
|---|---|---|
| VL_EV1_WIPE_DIRECTION<br>type *intVal* | VL_EV1_WIPEANGLE_E<br>VL_EV1_WIPEANGLE_NE<br>VL_EV1_WIPEANGLE_N<br>VL_EV1_WIPEANGLE_NW<br>VL_EV1_WIPEANGLE_W<br>VL_EV1_WIPEANGLE_SW<br>VL_EV1_WIPEANGLE_S<br>VL_EV1_WIPEANGLE_SE | Wipe vector direction, that is, the direction in which the wipe appears to be proceeding as its position increases.<br>Note: VL_EV1_WIPEANGLE_N and VL_EV1_WIPEANGLE_S do not work for the wipe types VL_EV1_WIPETYPE_DOUBLE and VL_EV1_WIPETYPE_CORNER |
| VL_EV1_WIPE_FUZZ<br>type *intVal* | (-8,7) | Sharpness of wipe transition band. As for VL_EV1_KEYER_DETAIL, -8 is most gradual, +7 is sharpest. |
| VL_EV1_WIPE_SYMMETRY<br>type *intVal* | (0,1)<br>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_EV1_WIPE_CENT for single, double, and corner wipes; enables VL_EV1_WIPE_CENT_PERP control for double and corner wipes. |
| VL_EV1_WIPE_POSN_PERP<br>type *fractVal* | numerator (0,1000)<br>denominator (1000) | Amount of progress of wipe, from none (numerator = 0) to full (numerator = 1000), along a direction perpendicular to normal wipe position VL_EV1_WIPE_POSN. |
| VL_EV1_WIPE_CENT<br>type *fractVal* | numerator (0,1000)<br>denominator (1000) | Offset that is center of a symmetrical wipe along wipe position. 0 means center is where VL_EV1_WIPE_POSN is 0, and 1000 means center is where VL_EV1_WIPE_POSN is 1000. For this control to work for single, double, and corner wipes, VL_EV1_WIPE_SYMMETRY must be on. |
| VL_EV1_WIPE_CENT_PERP<br>type *fractVal* | numerator (0,1000)<br>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_EV1_WIPE_REPT_PERP<br>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. |

In the *vcp* panel, the Blend/Wipe Node-All window is linked to the controls VL_EV1_WIPE_POSN_PERP, VL_EV1_WIPE_CENT, VL_EV1_WIPE_CENT_PERP, and VL_EV1_WIPE_REPT_PERP.

## 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.

In the video panel *vcp*, select the “Blend/Wipe Node” choice in the Pro menu. Select “Keying Controls”; The Blend/Wipe Node-Keying window appears, as shown in Figure 5-12.

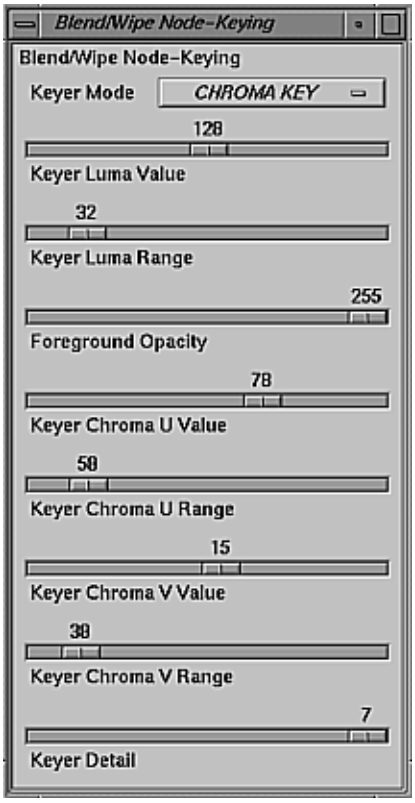



**Figure 5-12** Chroma Keying Controls

In this window, Keyer Mode can be ALPHA MODE, LUMA KEY, CHROMA KEY, or TRANSITION. The keyer mode setting determines the values you can set with the sliders in this window.

The sliders in this Blend/Wipe Node-Keying window have the following purposes:

- Keyer Luma Value sets brightness level
- Keyer Chroma U Value and Keyer Chroma V Value set chrominance
- Keyer Luma Range and Keyer Chroma Range set the dimensions of the foreground area

- Foreground Opacity limits the value of foreground alpha at any point
- Keyer Detail sets the sharpness of the edge between the foreground and background: -8 is the softest transition and +7 is the sharpest transition

Figure 5-13 shows the relationships between value, range, and detail (transition) for a single channel (for example, A).

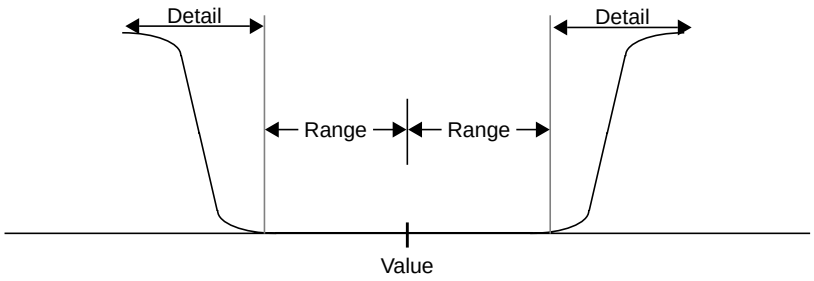



**Figure 5-13** Value, Range, and Transition (Keyer Detail) for a Channel

Use the sliders to set the values you want, based on what you see on the screen. Once you are satisfied with the settings, use the values shown at the sliders to insert into your code.

## VL Blending Examples

This section explains two example programs from */usr/people/4Dgifts/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. You should, however, always check returns.

## Blending Video and Graphics

*simpleblend.c,* which 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, 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()**

*Appendix A*

# 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.

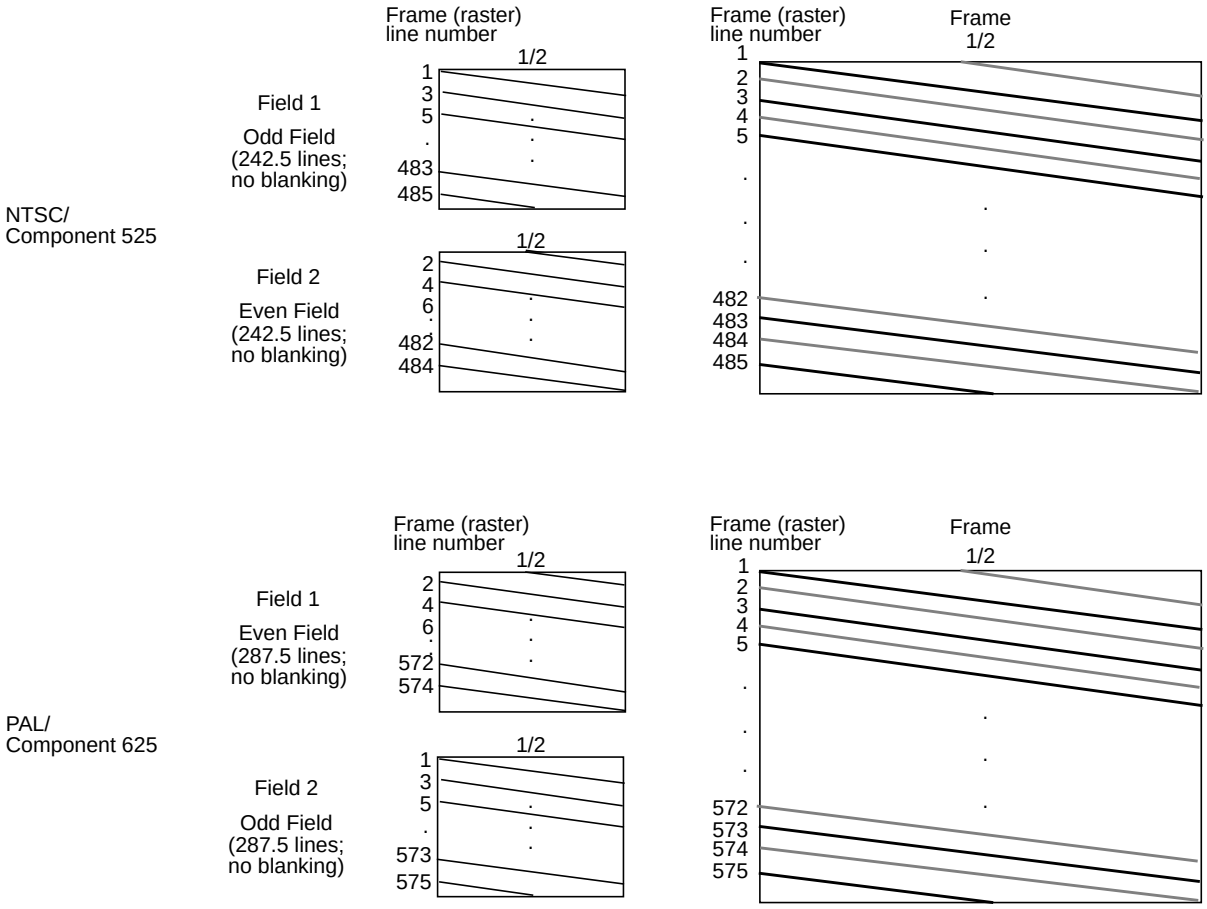



**Figure A-1** Fields and Frame

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-Y
$C_B$ = B-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.

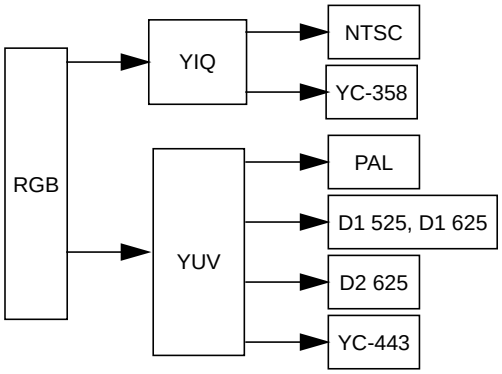



**Figure A-2** Relationships Between Color-encoding Methods and Video Formats

## 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.

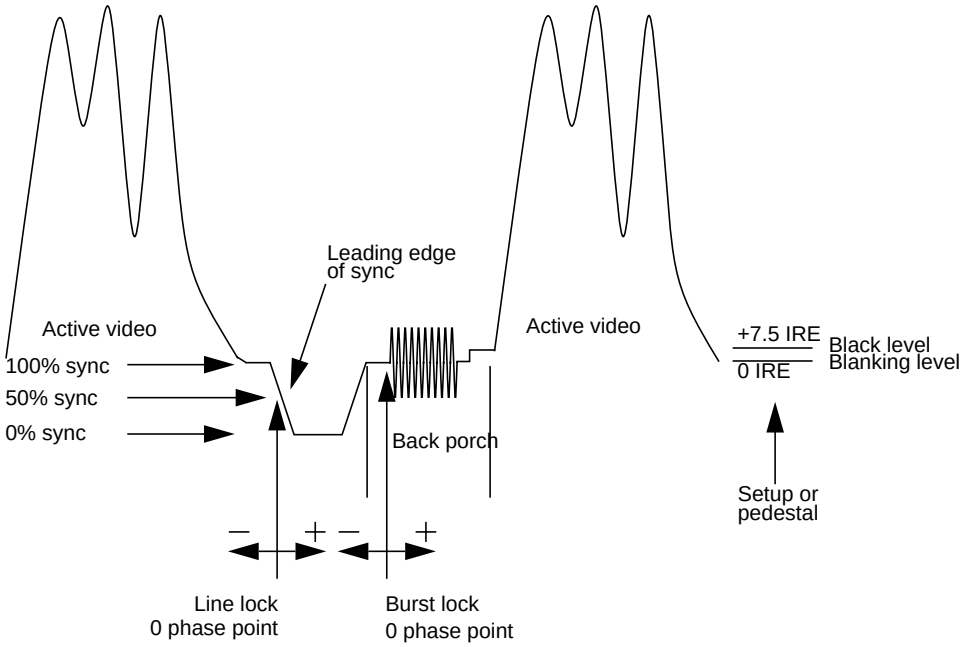



**Figure A-3** Composite Video Waveform

## 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.

# 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 0.5MHz. 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 3.58MHz (NTSC) or 4.43MHz (PAL) subcarrier signal for color in television. SECAM uses two frequency-modulated color subcarriers transmitted on alternate horizontal lines; $SC_R$ is 4.406MHz and $SC_B$ is 4.250MHz.

### 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 (3.58MHz for NTSC, 4.43MHz 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.

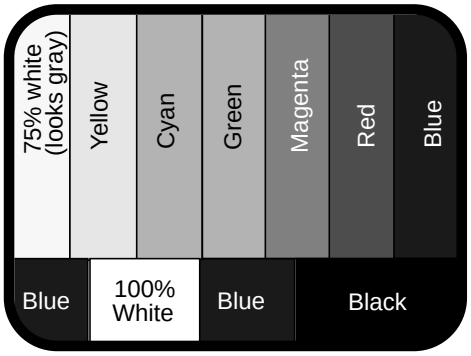



**Figure Gl-1** SMPTE Color Bars (75%)

### 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 3.58MHz 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 3.58MHz 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*.

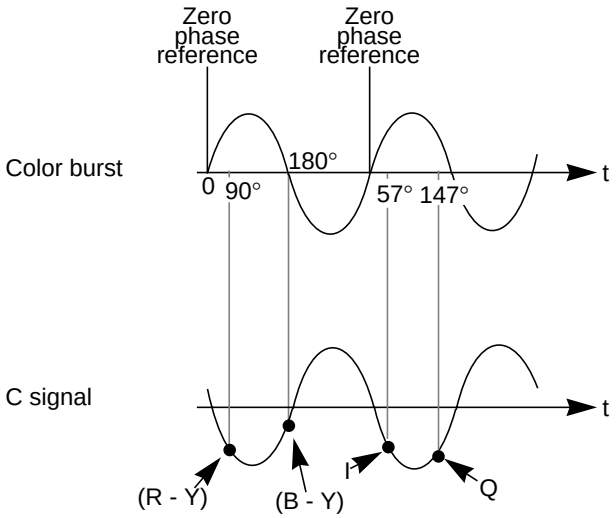



**Figure Gl-2** Color Burst and Chrominance Signal

### 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 space defined by three color components, such as R, G, and B.

### 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 3.58MHz in NTSC and 4.43MHz 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*.

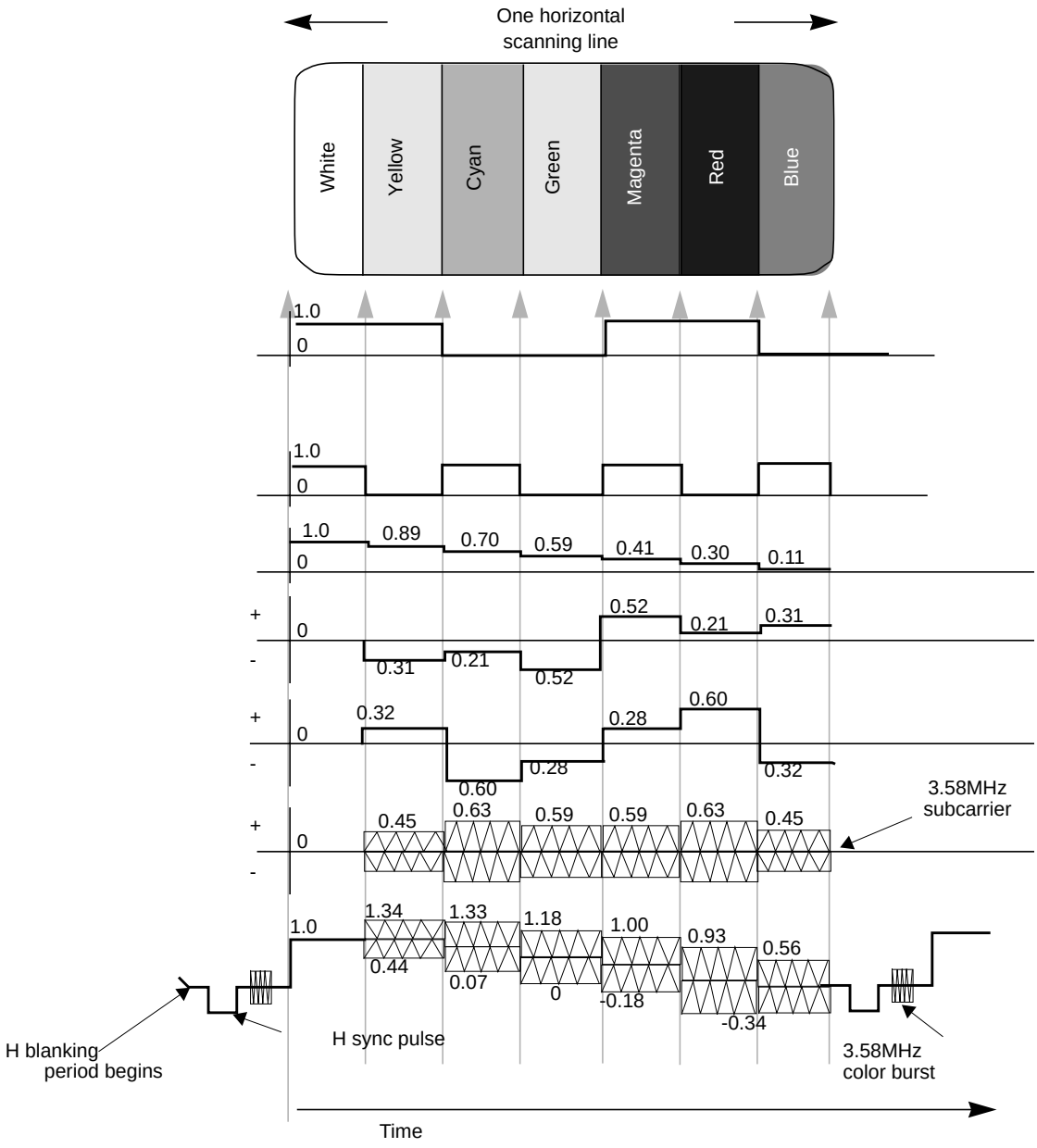



**Figure GI-3** Component Video Signals

### 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 3.579545MHz. 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 0.0MHz to 0.5MHz. 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 30MHz 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).

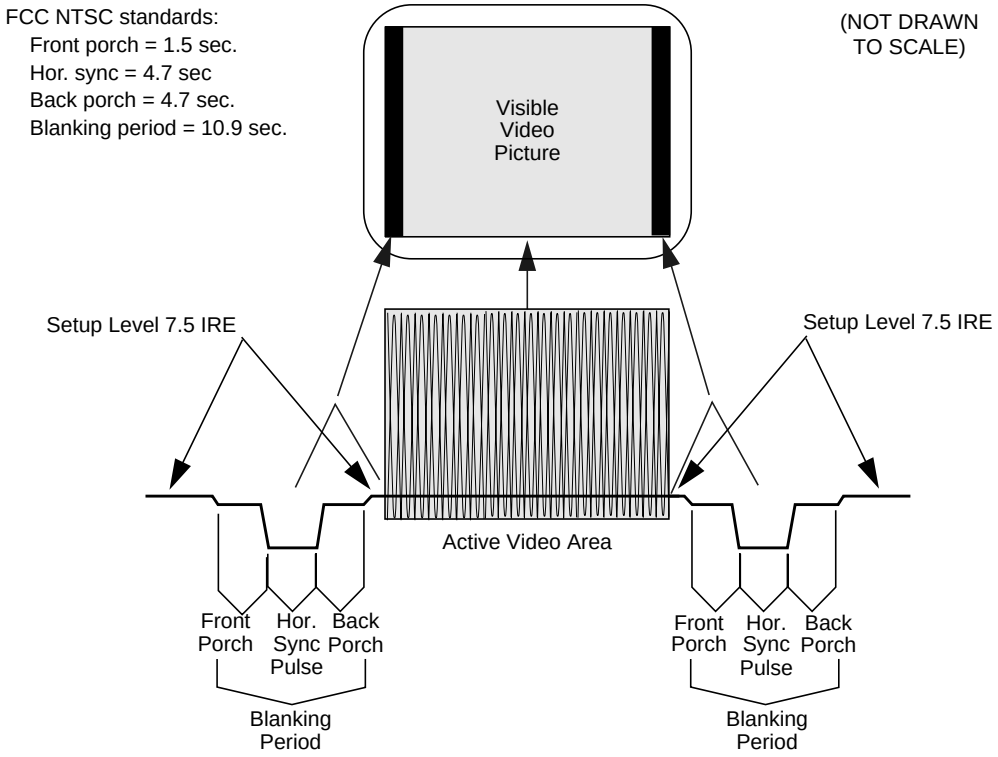



**Figure Gl-4** Horizontal Blanking

### 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*.

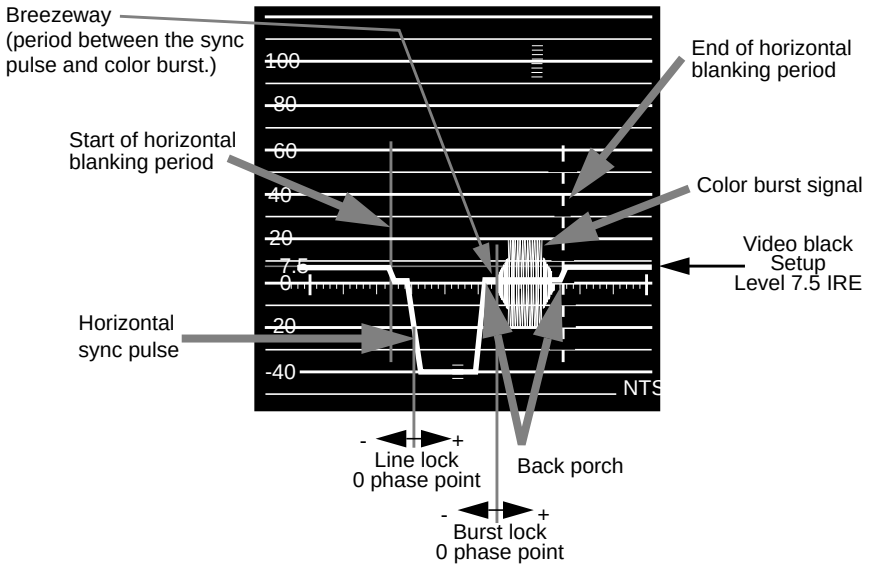



**Figure GI-5** Horizontal Blanking Interval

### 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 3.58MHz (NTSC) or 4.43MHz (PAL) C signal indicate the different hues in the picture information.

### hue-saturation-intensity

A tri-stimulus 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 3.58MHz C signal (NTSC). The hue axis is orange and cyan. This signal is the only color video signal with a bandwidth of 0 to 1.3MHz.

### 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 3.58MHz C signal in quadrature with the I signal. Hues are green and magenta. Bandwidth is 0.0MHz to 0.5MHz. 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 0.0MHz to 0.5MHz. 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 3.58MHz 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*.

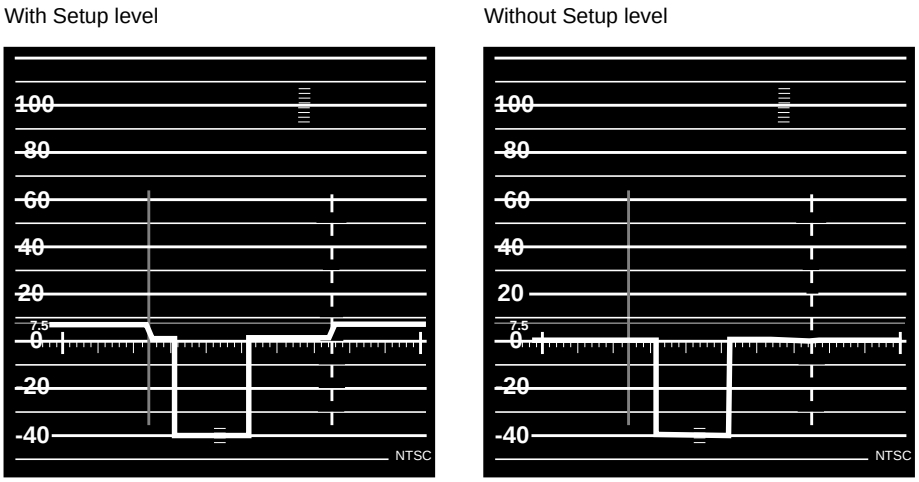



**Figure Gl-6** Waveform Monitor Readings with and without Setup

### 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).

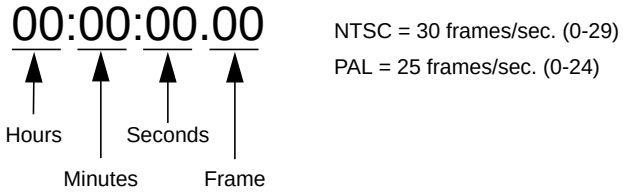



**Figure Gl-7** SMPTE Time Code

**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.

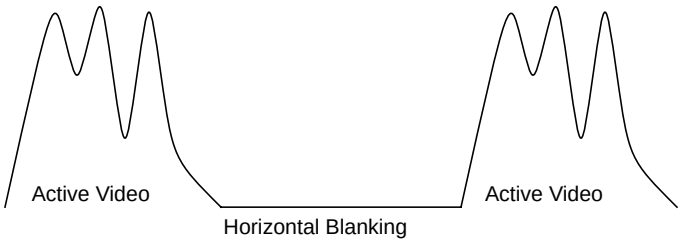



**Figure Gl-8** Red or Blue Signal

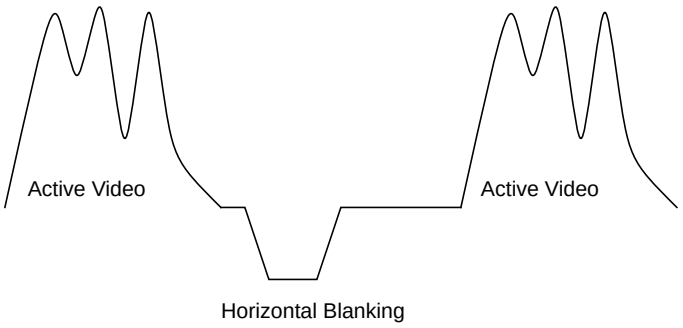



**Figure Gl-9** Y or Green Plus Sync Signal

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.

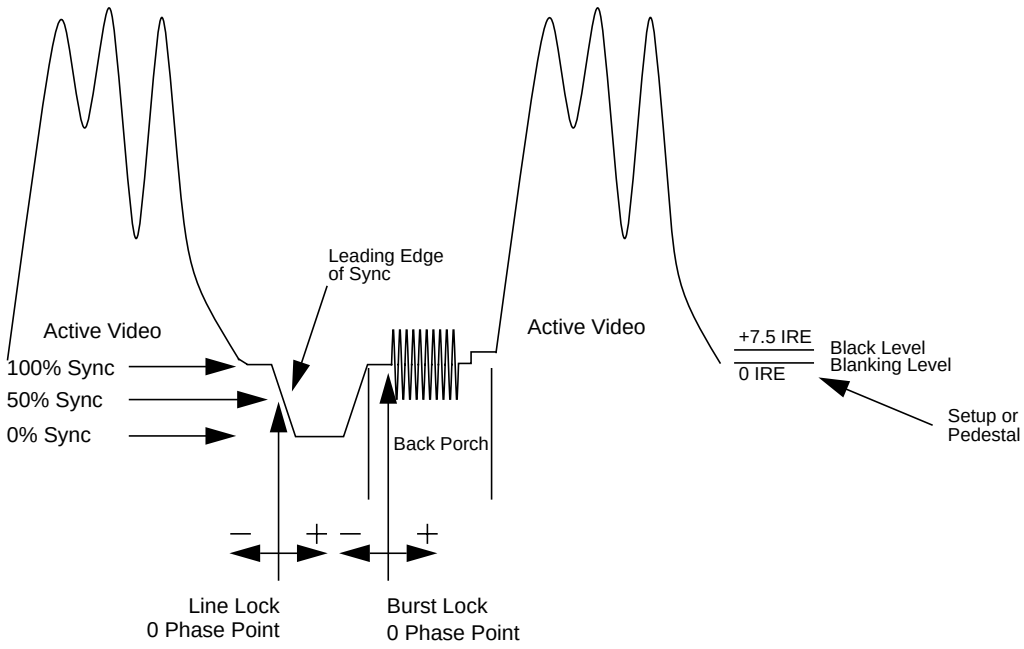



**Figure Gl-10** Video Waveform: Composite Video Signal With Setup (Typical NTSC)

Figure Gl-11 shows a composite video signal without setup.

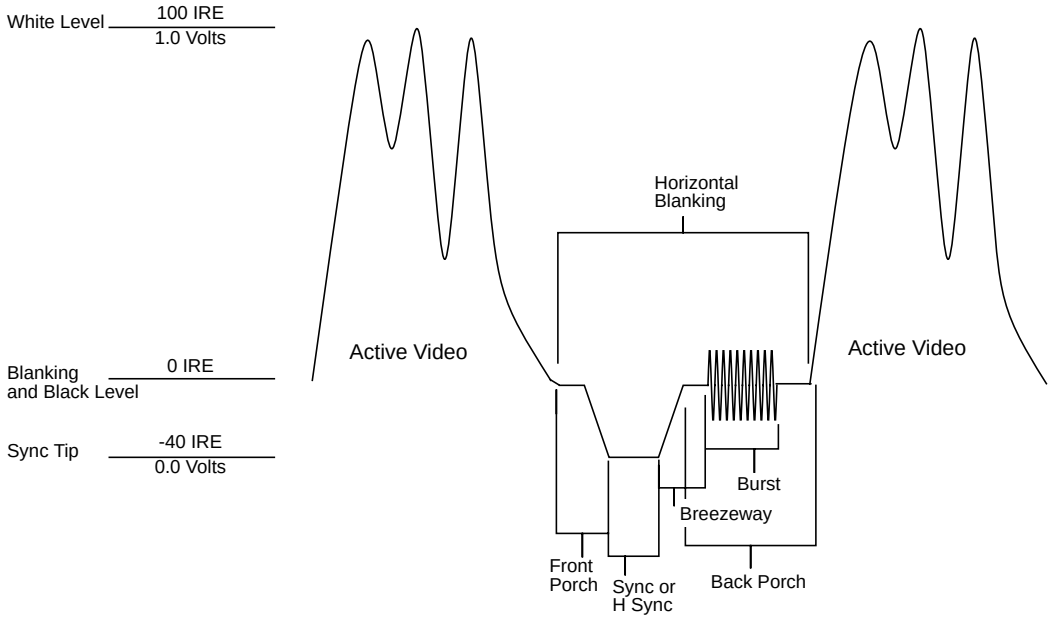



**Figure Gl-11** Video Waveform: Composite Video Signal (Typical PAL)

### 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 fsc 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

## M



## N



## O



## P



## R



## S

## W



## Y



## Z

## We'd Like to Hear From You

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 007-2917-001.

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