

VPro™ for Silicon Graphics®  
Octane® Porting Guide

007-4271-001

Version 001

## CONTRIBUTORS

Written by Tammy Domeier

Illustrated by Chris Wengelski

Edited by Rick Thompson

Production by Susan Gorski

Cover Design By Sarah Bolles, Sarah Bolles Design, and Dany Galgani, SGI Technical Publications

© 2000, Silicon Graphics, Inc. All Rights Reserved; provided portions may be copyright in third parties, as indicated elsewhere herein. No permission is granted to copy, distribute, or create derivative works from the contents of this electronic documentation in any manner, in whole or in part, without the prior written permission of Silicon Graphics, Inc.

## LIMITED RIGHTS LEGEND

The electronic (software) version of this document was developed at private expense; if acquired under an agreement with the USA government or any contractor thereto, it is acquired as "commercial computer software" subject to the provisions of its applicable license agreement, as specified in (a) 48 CFR 12.212 of the FAR; or, if acquired for Department of Defense units, (b) 48 CFR 227-7202 of the DoD FAR Supplement; or sections succeeding thereto. Contractor/manufacturer is Silicon Graphics, Inc., 1600 Amphitheatre Pkwy 2E, Mountain View, CA 94043-1351.

Silicon Graphics, IRIX, and OpenGL are registered trademarks and SGI, VPro, SE, SSE, MXE, Octane, and the SGI logo are trademarks of Silicon Graphics, Inc. All other trademarks mentioned are the property of their respective owners.

---

## Record of Revision

<b>Version</b>	<b>Description</b>
001	April 2000 Original Printing.



---

# Contents

## VPro™ for Silicon Graphics® Octane® Porting Guide

Tables . . . . .	ix
Audience . . . . .	xi
Related Publications . . . . .	xi
Obtaining Publications . . . . .	xi
Conventions . . . . .	xii
Reader Comments . . . . .	xii
<b>1. Product Overview . . . . .</b>	<b>1</b>
Industry-Leading Transform Performance . . . . .	1
Hardware-Accelerated Features . . . . .	2
Customer-Upgradable . . . . .	2
Planned Versions of VPro . . . . .	3
V6 and V8 . . . . .	3
Next Generation VPro . . . . .	3
<b>2. Architectural Overview . . . . .</b>	<b>5</b>
Hardware Features . . . . .	5
Graphics Memory Architecture . . . . .	6
Graphics Memory Usage. . . . .	6
Dual-Channel Display . . . . .	6
Command FIFO and Context Switching . . . . .	7
Supported Visuals . . . . .	7

Rendering Features. . . . .	. 10
Buffer Management . . . . .	. 10
Color Buffers . . . . .	. 11
Accumulation Buffers . . . . .	. 13
Overlay. . . . .	. 13
Stencil and Depth Buffers . . . . .	. 14
Off-Screen Buffers (Pbuffers) . . . . .	. 14
Stereo Support. . . . .	. 14
Buffer I/O . . . . .	. 14
Rendering Techniques Support . . . . .	. 15
Blending . . . . .	. 15
Texture . . . . .	. 17
Shading Support . . . . .	. 19
Anti-Aliasing (AA) and Fog . . . . .	. 19
Instrumentation . . . . .	. 19
Geometry . . . . .	. 20
Geometry Fast Paths . . . . .	. 20
Host Bandwidth . . . . .	. 20
State Changes . . . . .	. 21
Pixel Operations . . . . .	. 22
Buffer Reads and Writes . . . . .	. 22
Data Conversions . . . . .	. 23
Non-Blocking Texture Loads (and Pixel Reads and Draws) . . . . .	. 23
Imaging Operations . . . . .	. 23
Convolutions . . . . .	. 23
YCrCb format . . . . .	. 23
Other . . . . .	. 23
<b>3. Extensions . . . . .</b>	<b>. 25</b>
IMPACT Graphics Extensions Supported by VPro. . . . .	. 25
New Extensions Supported by VPro . . . . .	. 33
IMPACT Graphics Extensions Not Supported by VPro . . . . .	. 40
Other Extensions Not Supported by VPro . . . . .	. 40

---

## Tables

<b>Table 2-1</b>	VPro Blending . . . . .	15
<b>Table 2-2</b>	Texture Features . . . . .	18
<b>Table 2-3</b>	Pixel Transfer Speed . . . . .	22
<b>Table 3-1</b>	IMPACT Graphics Extensions Supported by VPro . . . . .	25
<b>Table 3-2</b>	New Extensions Supported by VPro . . . . .	33





---

## About This Guide

This porting guide describes VPro graphics for Octane2, which is the next-generation graphics set for Silicon Graphics Octane. It offers high polygon and fill performance, excellent image quality and advanced features important to 3D modeling and image processing applications on the desktop. VPro supercedes the SE, SSE, and MXE graphics sets for the Octane.

### Audience

This guide is intended for graphics programmers who use SGI systems. It describes the architectural features of VPro and how VPro supports new and existing extensions to OpenGL.

### Related Publications

The following documents contain additional information that may be helpful:

- *OpenGL Reference Manual*
- *OpenGL Programming Guide*

### Obtaining Publications

To obtain SGI documentation, go to the SGI Technical Publications Library:

<http://techpubs.sgi.com>

## Conventions

The following conventions are used throughout this document:

<b>Convention</b>	<b>Meaning</b>
<code>command</code>	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.
<i>variable</i>	Italic typeface denotes variable entries and words or concepts being defined.
<b>user input</b>	This bold fixed-space font denotes literal items that the user enters in interactive sessions. Output is shown in nonbold, fixed-space font. Also, function names with parentheses following the name—for example, <code>glPolygonMode()</code> —and arguments to command line options.
[ ]	Brackets enclose optional portions of a command or directive line.
...	Ellipses indicate that a preceding element can be repeated.

## Reader Comments

If you have comments about the technical accuracy, content, or organization of this document, please tell us. Be sure to include the title and document number of the manual with your comments. (Online, the document number is located in the front matter of the manual. In printed manuals, the document number can be found on the back cover.)

You can contact us in any of the following ways:

- Send e-mail to the following address:  
`techpubs@sgi.com`
- Use the Feedback option on the Technical Publications Library World Wide Web page:  
`http://techpubs.sgi.com`
- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.

- Send mail to the following address:  
Technical Publications  
SGI  
1600 Amphitheatre Pkwy., M/S 535  
Mountain View, California 94043-1351
- Send a fax to the attention of Technical Publications:  
+1 650 932 0801

We value your comments and will respond to them promptly.



## Product Overview

VPro is the next generation graphics set for Octane. It offers high polygon and fill performance, excellent image quality, and advanced features important to 3D modeling and image processing applications on the desktop. VPro highlights include the following:

- Industry-leading transform performance for the desktop: performance increases over existing Octane products
- Hardware-accelerated features for enhanced image quality and interactivity
- Customer-upgradable on Octane
- Two versions of VPro: V6 with 32MB of graphics memory, V8 with 128MB of graphics memory

### Industry-Leading Transform Performance

VPro brings dramatically higher performance and many new features to the desktop. Some of these features are detailed below as they relate to various markets and industries. For all markets, VPro offers superb transform and fill rate performance for both textured and non-textured data sets.

- VPro has been optimized to deliver high performance transform and lighting critical to the CAD and 3D animation markets so that the end user can interact freely with large, finely tessellated models.
- Industry-leading textured fill-rate performance provides image processing, visual simulation, and volume visualization applications a new level of interactivity.

For 3D modeling and image processing markets, VPro includes a number of differentiated features and meets compliance for OpenGL 1.2 certification.

For 3D modeling markets, new features include the following:

- Per-pixel (per-fragment) lighting for accurate lighting even with simplified geometry
- Deep graphics FIFO for better host and graphics load balancing
- Improved graphics context switching, allowing multiple windows to render at high performance.

## Hardware-Accelerated Features

For graphics and visualization-intensive image processing markets, VPro offers a set of specific hardware-accelerated features. These features enable enhanced performance for applications and guarantee a high level of image quality and accuracy for the end-user.

- 48-bit RGBA (supported on V8 and Next Generation VPro) for applications like film compositing and CAD styling that demand high color precision
- Up to 104 MB of texture memory with non-blocking texture download, for fast handling of very large textures
- 3D textures for very fast rendering of volumetric data sets
- 48-bit texture look-up table for very fast and accurate interaction with volume-rendered data sets
- 24-bit per component hardware accumulation buffer (128MB version only)
- Dual-channel for a two-screen display out of a single graphics card (128MB version only)

## Customer-Upgradable

VPro is a customer-upgradable add-in board on Octane. The upgrade entails replacing the current graphics subsystem and installing the latest IRIX maintenance release. A single VPro occupies the upper left high-speed XIO port in the Octane machine (the card will physically occupy two XIO slots because of physical dimensions).

## Planned Versions of VPro

VPro will supercede the current SE, SSE, and MXE graphics sets available today on Octane and offer significant price-performance advantages. VPro will be available in two versions: V6 and V8.

### V6 and V8

The following are distinguishing features of the V6 and V8 versions:

- V6 includes 32MB of graphics memory (memory shared among framebuffer, texture, pBuffer, accumulation buffer, etc.).
- V8 includes 128MB of graphics memory.

### Next Generation VPro

Next Generation VPro will be a more robust version than V6 and V8:

- improved image processing performance
- improved image copy and image write performance
- support for 12-bit RGBA with 16-bit z
- available with 128MB of graphics memory, and will replace V8





## Architectural Overview

The VPro architecture is a departure from the traditional SIMD large-chip-count graphics systems. VPro consists of only two main chips: Buzz, the transform and rasterizer chip, and PB&J, the back-end video chip. The chips run at high clock rates and in many ways are very similar to a RISC type architecture. The VPro architecture does not have the strip length, context switch, or inter-chip communication drawbacks of traditional SIMD graphics architectures. Buzz implements the full geometry pipeline, including transformation, lighting, and clipping, along with the full OpenGL 1.2 imaging pipeline. Lighting is fully hardware-accelerated for both per-vertex and per-pixel shading. Texturing features include both 2D and 3D textures, borders, post-texture lookup tables, and post-texture specular highlights. The full pipeline runs with 12-bit per-component or greater precision from the geometry stage through rasterization.

This chapter further describes the VPro architecture in two parts:

- hardware features
- rendering features

### Hardware Features

This section describes the following features:

- Graphics memory architecture
- Graphics memory usage
- dual-channel display
- command FIFO and context switching

## Graphics Memory Architecture

On VPro, the commands FIFO, framebuffer, textures, pbuffers, scratch buffers, and all other buffers are allocated out of one large pool of memory. VPro is available in two memory sizes: a 32MB version called V6 and a 128 version called V8. Off-screen rendering (pbuffers) and accumulation buffer usage run at full hardware speeds. Copies between buffers and textures are extremely fast since they are performed on board without needing a read to the host and then back again to the board.

## Graphics Memory Usage

The VPro drawable buffers, having a maximum addressable area of 4K x 4K, allows very large pbuffer and imaging operations. The number of pbuffers is dependent on the size of the buffer (up to the maximum of 4K x 4K), the depth of the buffer, and the available free graphics memory. Any on-board memory that is not directly used for drawable buffers can be used for auxiliary buffers, pbuffers (fully hardware-accelerated), and textures.

Memory usage for on-screen buffers can be calculated by multiplying the screen size in pixels by either 10 or 18 bytes per pixel (actual framebuffer at either 8 or 16 bytes per pixel), plus another 12 bytes per pixel if H/W accelerated accumulation buffers are configured.

Using `xsetmon`, you can configure the graphics memory to allocate and query memory for optimal application configurations. You can evaluate tradeoffs such as pbuffers against textures, and the like.

## Dual-Channel Display

The large addressable framebuffer also enables a high-resolution, dual-channel display option. The left and right channels, positioned side-by-side in a single logical screen, allow windows to be dragged between displays (or rendering into a single viewport that covers both displays). The optional hardware adaptor card provides two digital or analog outputs. The SGI 1600SW (with an adaptor) is compatible with this dual-channel option, allowing completely digital throughput to the monitor.

## Command FIFO and Context Switching

VPro's deep command FIFO allows applications to optimize load balancing between the host and graphics processors. The VPro architecture is also designed for very fast context switching. Commands are sent to a deep command FIFO, which contains approximately one millisecond of commands. The FIFO can hold multiple command streams; so, a context switch does not have to wait for the FIFO to drain. The trade-off for fast context switches is a maximum latency of the size of the FIFO. This latency is only apparent to operations that require a round trip to the graphics, such as `glReadPixels`.

State is also shadowed on the host. The state shadow has an additional benefit that redundant state changes are eliminated on the host and state queries are extremely fast.

## Supported Visuals

Using `xsetmon`, you can configure the framebuffer to be either 16-byte or 8-byte. The memory is allocated at X startup time. The visuals that are advertised by the X server are dependent on the initial size of the framebuffer. Pbuffer visuals are available in all standard formats. Visuals that are too deep for an 8-byte framebuffer (for example, double-buffered RGBA + depth) will not be available if the X Server is started with a 8-byte framebuffer. Likewise, if the server is not started with a hardware accumulation buffer, all accumulation operations will be done in software.

8-Byte Visuals:

visual	x	bf	lv	rg	d	st	r	g	b	a	ax	dp	st	accum	bufs	ms						
id	dep	cl	sp	sz	l	ci	b	ro	sz	sz	sz	sz	bf	th	cl	r	g	b	a	ns	b	
0x20	8	pc	.	8	.	c	.	.	.	.	.	.	.	24	.	.	.	.	.	.	.	.
0x21	8	pc	.	8	.	c	y	.	.	.	.	.	.	24	.	.	.	.	.	.	.	.
0x23	8	pc	y	8	1	c	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0x24	8	pc	.	8	1	c	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0x25	8	pc	.	8	.	r	y	.	8	.	.	.	.	24	.	.	.	.	.	.	.	.
0x26	8	pc	.	8	.	r	y	.	8	.	.	8	.	24	.	.	.	.	.	.	.	.
0x27	8	pc	.	8	.	r	y	.	8	.	.	8	.	24	8	.	.	.	.	.	.	.
0x28	8	pc	.	8	.	c	y	y	.	.	.	.	.	24	.	.	.	.	.	.	.	.
0x29	8	pc	.	8	.	c	y	y	.	.	.	.	.	24	8	.	.	.	.	.	.	.
0x2a	8	pc	.	8	.	r	y	y	8	.	.	.	.	24	.	.	.	.	.	.	.	.
0x2b	8	pc	.	8	.	r	y	y	8	.	.	8	.	.	.	.	.	.	.	.	.	.
0x2c	12	pc	.	12	.	c	.	.	.	.	.	.	.	24	.	.	.	.	.	.	.	.
0x2d	12	pc	.	12	.	c	.	.	.	.	.	.	.	24	8	.	.	.	.	.	.	.
0x2e	12	pc	.	12	.	c	y	.	.	.	.	.	.	24	.	.	.	.	.	.	.	.
0x2f	12	pc	.	12	.	c	y	.	.	.	.	.	.	24	8	.	.	.	.	.	.	.
0x30	12	pc	.	12	.	c	y	y	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0x31	12	pc	.	12	.	r	y	.	12	.	.	.	.	24	.	.	.	.	.	.	.	.
0x32	12	pc	.	12	.	r	y	y	12	.	.	.	.	.	.	.	.	.	.	.	.	.
0x33	12	pc	.	12	.	r	.	.	12	.	.	12	.	24	.	.	.	.	.	.	.	.
0x34	12	pc	.	12	.	r	.	.	12	.	.	12	.	24	8	.	.	.	.	.	.	.
0x35	12	pc	.	12	.	r	y	.	12	.	.	12	.	.	.	.	.	.	.	.	.	.
0x36	12	tc	.	16	.	r	y	.	4	4	4	4	.	24	.	16	16	16	16	.	.	.
0x37	12	tc	.	16	.	r	y	.	4	4	4	4	.	24	8	16	16	16	16	.	.	.
0x38	12	tc	.	16	.	r	y	y	4	4	4	4	.	.	.	16	16	16	16	.	.	.
0x39	15	tc	.	16	.	r	y	.	5	5	5	1	.	24	.	16	16	16	16	.	.	.
0x3a	15	tc	.	16	.	r	y	.	5	5	5	1	.	24	8	16	16	16	16	.	.	.
0x3b	15	tc	.	16	.	r	y	y	5	5	5	1	.	.	.	16	16	16	16	.	.	.
0x3c	24	tc	.	32	.	r	.	.	8	8	8	8	.	24	.	16	16	16	16	.	.	.
0x3d	24	tc	.	32	.	r	.	.	8	8	8	8	.	24	8	16	16	16	16	.	.	.
0x3e	24	tc	.	32	.	r	y	.	8	8	8	8	.	.	.	16	16	16	16	.	.	.

id	dep	cl	xp	bs	lv	rg	d	st	rb	gb	bb	ab	ax	dp	st	ar	ag	ab	aa	ms	b	
0x40	30	tc	.	32	.	r	.	.	10	10	10	2	.	24	.	16	16	16	16	.	.	.
0x41	30	tc	.	32	.	r	.	.	10	10	10	2	.	24	8	16	16	16	16	.	.	.
0x42	30	tc	.	32	.	r	y	.	10	10	10	2	.	.	.	16	16	16	16	.	.	.
0x43	30	tc	.	48	.	r	.	.	12	12	12	12	.	.	.	16	16	16	16	.	.	.
0x44	30	tc	.	48	.	r	.	.	12	12	12	12	.	16	.	16	16	16	16	.	.	.

1

16-Byte Visuals:

visual id	x dep	bf cl	lv sp	rg sz	d l	st ci	r b	g ro	b sz	a sz	ax sz	dp sz	st bf	accum th	bufs cl	ms r	g b	a ns	b	
0x20	8	pc	.	8	.	c	.	.	.	.	.	.	.	24	.	.	.	.	.	.
0x21	8	pc	.	8	.	c	y	.	.	.	.	.	.	24	.	.	.	.	.	.
0x23	8	pc	y	8	1	c	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0x24	8	pc	.	8	1	c	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0x25	8	pc	.	8	.	r	y	.	8	.	.	.	.	24	.	.	.	.	.	.
0x26	8	pc	.	8	.	r	y	.	8	.	.	8	.	24	.	.	.	.	.	.
0x27	8	pc	.	8	.	r	y	.	8	.	.	8	.	24	8	.	.	.	.	.
0x28	8	pc	.	8	.	c	y	y	.	.	.	.	.	24	.	.	.	.	.	.
0x29	8	pc	.	8	.	c	y	y	.	.	.	.	.	24	8	.	.	.	.	.
0x2a	8	pc	.	8	.	r	y	y	8	.	.	.	.	24	.	.	.	.	.	.
0x2b	8	pc	.	8	.	r	y	y	8	.	.	8	.	24	.	.	.	.	.	.
0x2c	12	pc	.	12	.	c	.	.	.	.	.	.	.	24	.	.	.	.	.	.
0x2d	12	pc	.	12	.	c	.	.	.	.	.	.	.	24	8	.	.	.	.	.
0x2e	12	pc	.	12	.	c	y	.	.	.	.	.	.	24	.	.	.	.	.	.
0x2f	12	pc	.	12	.	c	y	.	.	.	.	.	.	24	8	.	.	.	.	.
0x30	12	pc	.	12	.	c	y	y	.	.	.	.	.	24	.	.	.	.	.	.
0x31	12	pc	.	12	.	c	y	y	.	.	.	.	.	24	8	.	.	.	.	.
0x32	12	pc	.	12	.	r	y	.	12	.	.	.	.	24	.	.	.	.	.	.
0x33	12	pc	.	12	.	r	y	y	12	.	.	.	.	24	.	.	.	.	.	.
0x34	12	pc	.	12	.	r	.	.	12	.	.	12	.	24	.	.	.	.	.	.
0x35	12	pc	.	12	.	r	.	.	12	.	.	12	.	24	8	.	.	.	.	.
0x36	12	pc	.	12	.	r	y	.	12	.	.	12	.	24	.	.	.	.	.	.
0x37	12	pc	.	12	.	r	y	.	12	.	.	12	.	24	8	.	.	.	.	.
0x38	12	tc	.	16	.	r	y	.	4	4	4	4	.	24	.	16	16	16	16	.
0x39	12	tc	.	16	.	r	y	.	4	4	4	4	.	24	8	16	16	16	16	.
0x3a	12	tc	.	16	.	r	y	y	4	4	4	4	.	24	.	16	16	16	16	.
0x3b	12	tc	.	16	.	r	y	y	4	4	4	4	.	24	8	16	16	16	16	.
0x3c	15	tc	.	16	.	r	y	.	5	5	5	1	.	24	.	16	16	16	16	.
0x3d	15	tc	.	16	.	r	y	.	5	5	5	1	.	24	8	16	16	16	16	.
0x3e	15	tc	.	16	.	r	y	y	5	5	5	1	.	24	.	16	16	16	16	.

<sup>1</sup> RGBA12 visuals with Z are only available on Next generation VPro.

```

-----
id dep cl xp bs lv rg d st rb gb bb ab ax dp st ar ag ab aa ms,b
-----
0x3f 15 tc . 16 . r y y 5 5 5 1 . 24 8 16 16 16 16 . .
0x40 24 tc . 32 . r . . 8 8 8 8 . 24 . 16 16 16 16 . .
0x41 24 tc . 32 . r . . 8 8 8 8 . 24 8 16 16 16 16 . .
0x42 24 tc . 32 . r y . 8 8 8 8 . 24 . 16 16 16 16 . .
0x43 24 tc . 32 . r y . 8 8 8 8 . 24 8 16 16 16 16 . .
0x45 30 tc . 32 . r . . 10 10 10 2 . 24 . 16 16 16 16 . .
0x46 30 tc . 32 . r . . 10 10 10 2 . 24 8 16 16 16 16 . .
0x47 30 tc . 32 . r y . 10 10 10 2 . 24 . 16 16 16 16 . .
0x48 30 tc . 32 . r y . 10 10 10 2 . 24 8 16 16 16 16 . .
0x49 30 tc . 48 . r . . 12 12 12 12 . . . 16 16 16 16 . .
0x4a 30 tc . 48 . r . . 12 12 12 12 . 16 . 16 16 16 16 . .
0x4b 30 tc . 48 . r y . 12 12 12 12 . . . 16 16 16 16 . .
0x4c 30 tc . 48 . r y . 12 12 12 12 . 16 . 16 16 16 16 . .
-----

```

<sup>2</sup>

## Rendering Features

This section describes the following topics:

- Buffer management
- Rendering techniques support
- Geometry
- Pixel operations
- Imaging operations

### Buffer Management

All OpenGL buffers are allocated out of the same on-board memory pool. The number and size of buffers that are available is dependent on the amount of memory on-board

---

<sup>2</sup> RGBA12 visuals with Z are only available on Next Generation VPro.

(32MB or 128MB), the screen resolution, and the video modes. The maximum possible size for any drawable buffer is 4K x 4K.

VPro does not have a native GLfloat data format for pixel data. Float data that is read or written to the color buffer, depth buffer, stencil buffer, and accumulation buffer is converted on the host. For best performance, do not read or write pixel data as float but use any of the other GL data types, such as [unsigned] integer or [unsigned] long. Performance for pixel read and write operations is best for 32-bit RGBA formats.

VPro supports the following OpenGL buffers:

- Color buffers
- Stencil and depth buffers
- Accumulation buffer
- Overlay buffer
- Off-screen rendering buffer (pbuffer)

## Color Buffers

VPro supports bitmap, CI, L, LA, RGB, RGBA, ABGR, BGRA, and YCrCb (through the subsample and YCrCb\_format specification) in hardware. All color buffers are either single-, double-, or quad-buffered (stereo). The resolution and number of buffers is dependent on the amount of graphics memory. A 32MB system will not be able to do RGBA16 quad buffering.

The following formats are supported <sup>3</sup>:

Data Formats:

```
UNSIGNED_BYTE_3_3_2
UNSIGNED_BYTE_2_3_3_REV
UNSIGNED_BYTE_5_6_5
UNSIGNED_BYTE_5_6_5_REV
UNSIGNED_SHORT_4_4_4_4
UNSIGNED_SHORT_4_4_4_4_REV
UNSIGNED_SHORT_5_5_5_1
UNSIGNED_SHORT_1_5_5_5_REV
```

---

<sup>3</sup> \_REV denotes the ordering of components will be reversed.

UNSIGNED\_INT\_8\_8\_8\_8  
UNSIGNED\_INT\_8\_8\_8\_8\_REV  
UNSIGNED\_INT\_10\_10\_10\_2  
UNSIGNED\_INT\_10\_10\_10\_2\_REV

**Framebuffer Formats:**

R8<sup>4</sup>, G8<sup>4</sup>, B8<sup>4</sup>, A8<sup>4</sup>,  
L8<sup>5</sup>  
I8  
LA8  
R3\_G3\_B2<sup>5</sup>  
B2\_G3\_R3<sup>5</sup>  
R5\_G6\_B5<sup>5</sup>  
B5\_G6\_R5<sup>5</sup>  
RGB8<sup>5</sup>  
RGBA4  
ABGR4  
ARGB4  
BGRA4  
A1\_RGB5  
A1\_BGR5  
BGR5\_A1  
RGBA8  
ABGR8  
ARGB8  
BGRA8  
YCrCb\_444(byte or ubyte)<sup>6</sup>  
YCrCb\_422 (byte or ubyte)<sup>6</sup>  
R16<sup>4</sup>, G16<sup>4</sup>, B16<sup>4</sup>, A16<sup>4</sup>  
R32<sup>4</sup>, G32<sup>4</sup>, B32<sup>4</sup>, A32<sup>4</sup>  
L16<sup>5</sup>  
I16  
LA16  
RGB32<sup>5</sup>  
RGBA32<sup>4</sup>

---

<sup>4</sup> Single components are still expanded to RGBA. The other components are set to 0 and alpha is always set to 0xff for packed and 0xffff for full mode.

<sup>5</sup> Alpha is set to 0xff for packed or 0xffff for full.

<sup>6</sup> For the YCrCb formats, VPro does the proper sampling; that is, replicate or zero-fill.



ABGR32<sup>4</sup>  
 RGB16<sup>5</sup>  
 RGB10\_A2  
 A2\_BGR10  
 BGR10\_A2  
 A2\_RGB10  
 RGBA12  
 ABGR12  
 RGBA16  
 ABGR16  
 YCrCb\_422 (short or ushort)<sup>7</sup>  
 L8, I8  
 CI8  
 L16, I16  
 CI12, CI16  
 L32  
 LA8  
 LA16  
 LA32  
 Bitmap  
 CI8  
 CI16  
 CI32

### Accumulation Buffers

The OpenGL accumulation buffer is fully hardware-accelerated if the X server is configured to advertise hardware-accelerated accumulations buffers. VPro also supports an accumulation buffer within a pbuffer. Using `setmon` or `xsetmon`, you need to do configuration before X server startup time for either software accumulation or shallow hardware accumulation. The software accumulation buffer supports 16-bit precision per component RGBA and the hardware accumulation buffer can support 24-bit precision per component RGBA.

### Overlay

There is one single-buffered 8-bit overlay buffer.

---

<sup>7</sup> For the YCrCb formats, VPro does the proper sampling; that is, replicate or zero-fill.

## Stencil and Depth Buffers

There is one Z/stencil buffer\_per\_color\_buffer\_ for supported formats. The stencil and depth buffers are packed into one 4-byte value on reads and writes. Hence, requesting stencil does not add an additional memory hit over requesting only depth. However, since the stencil planes cannot be cleared using the hardware *fast clear* mode, it should not be requested as part of the visual unless it is going to be used. When using stencil and depth buffers, it is fastest to clear them both simultaneously.

The depth buffer is in eye space instead of the traditional screen space. Depth is calculated before the projection matrix, not afterwards. This allows greater precision in the depth buffer as the space is no longer non-linear due to the perspective divide. Depth buffer readback must be converted from eye space to screen space. Applications need to ensure that the depth buffer is cleared when changing the depth range and other depth buffer state operations.

## Off-Screen Buffers (Pbuffers)

Off-screen rendering areas are known as pbuffers. Pbuffers support all main buffer configurations including color (resolution and format), depth, stencil, and accumulation.

Pbuffers are allocated and destroyed through the `fbconfig` set of commands which become part of the GLX spec as of GLX 1.3. Based upon the video format loaded, buffer allocations are done statically when the X Server activates. Since pbuffers are allocated out of the on-board memory, pbuffer allocation can fail when graphics memory is depleted. This can happen if there are too many pbuffers, deep or wide main buffers (such as for dual-channel mode) or too many textures defined. VPro supports pbuffers as single-buffered entities only.

## Stereo Support

Two modes of stereo are supported: SGI full screen and quad-buffered.

## Buffer I/O

Moving data between buffers is very fast since the buffer-to-buffer copy happens entirely on-board. No round trip to the host is required. Moving data between any buffer (color, depth, stencil, and accumulation) and texture objects is also very fast. Copying data from graphics memory to the host will run at a slower rate. Buffer copies can be accomplished by using the `MakeCurrentRead` GLX extension.

To optimize performance, use the native VPro formats to read and write data. Use this guideline to ensure that the host does not have to convert the data. The host must convert the data when reading and writing in GLfloat format or reading in GLint format.

## Rendering Techniques Support

VPro has many advanced rendering features that provide support for high-quality imaging. There is support for the full OpenGL 1.2 pipeline in addition to many extensions. All paths through the pipeline are 12-bit per-component paths, including blending, texturing, and lighting. Per-pixel shading is supported as well as the Specular after Texture extension.

### Blending

All OpenGL 1.2 blending modes are supported to take incoming RGBA fragments and blend them into the existing framebuffer. Table 2-1 describes VPro blending.

**Table 2-1** VPro Blending

Token	Attribute	Value
Blend Op	source factor	ZERO
		ONE
		DST_COLOR
		ONE_MINUS_DST_COLOR
		SRC_ALPHA
		ONE_MINUS_SRC_ALPHA
		DST_ALPHA
		ONE_MINUS_DST_ALPHA
		SRC_ALPHA_SATURATE
		CONSTANT_COLOR_EXT
	ONE_MINUS_CONSTANT_COLOR_EXT	
	CONSTANT_ALPHA_EXT	
	ONE_MINUS_CONSTANT_ALPHA_EXT	
	destination factor	ZERO
		ONE
		SRC_COLOR
		ONE_MINUS_SRC_COLOR
		SRC_ALPHA
		ONE_MINUS_SRC_ALPHA

**Table 2-1** VPro Blending (continued)

Token	Attribute	Value
		DST_ALPHA
		ONE_MINUS_DST_ALPHA
		CONSTANT_COLOR_EXT
		ONE_MINUS_CONSTANT_COLOR_EXT
		CONSTANT_ALPHA_EXT
		ONE_MINUS_CONSTANT_ALPHA_EXT
	blend equation	FUNC_ADD_EXT
		MIN_EXT
		MAX_EXT
		ALPHA_MIN_SGIX
		ALPHA_MAX_SGIX
		LOGIC_OP
		FUNC_SUBTRACT_EXT
		FUNC_REVERSE_SUBTRACT_EXT
	constant red	[11:0]
	constant green	[11:0]
	constant blue	[11:0]
	constant alpha	[11:0]
Logic Op	logicop	CLEAR
		AND
		AND_REVERSE
		COPY
		AND_INVERTED
		NOOP
		XOR
		OR
		NOR
		EQUIV
		INVERT
		OR_REVERSE
		COPY_INVERTED
		OR_INVERTED
		NAND
		SET

Of interest is a new blend extension, SGIX\_BLEND\_ALPHA\_MINMAX, which is similar to the EXT\_BLEND\_MINMAX extension but uses the minmax comparison result of only the alpha channel to choose all the color components.

## Texture

Texture memory is shared with the on-board framebuffer SDRAM. Since the texture memory size is not constant it is important to use the texture proxy mechanisms to find the maximum texture size.

The following internal texture formats are supported:

- ALPHA4
- ALPHA8
- ALPHA12
- ALPHA16
- LUMINANCE4
- LUMINANCE8
- LUMINANCE12
- LUMINANCE16
- LUMINANCE4\_ALPHA4
- LUMINANCE6\_ALPHA2
- LUMINANCE8\_ALPHA8
- LUMINANCE12\_ALPHA4
- LUMINANCE12\_ALPHA12
- LUMINANCE16\_ALPHA16
- INTENSITY4
- INTENSITY8
- INTENSITY12
- INTENSITY16
- R3\_G3\_B2
- RGB4
- RGB5
- RGB8
- RGB10
- RGB12
- RGB16
- RGBA2
- RGBA4
- RGB5\_A1
- RGBA8

RGB10\_A2  
 RGBA12  
 RGBA16

Table 2-2 summarizes the VPro texture features:

**Table 2-2** Texture Features

Feature	Specifications
Texture types	1D, 2D, 3D, including borders
Filtering	point, bilinear, trilinear
Texture lookup tables	12-bit RGBA post-interpolation tables
Max texture dimension (s or t)	32K (up to board memory size)

The following qualifications apply:

- As textures and other buffers are all resident in the same memory space, the system performs a copy-by-reference when appropriate. Oversubscribing texture memory will cause textures to swap to the host. Copy-by-reference may not be implemented at launch.
- The hardware supports texture borders for all dimensioned textures. Be aware that the default border color is (0,0,0,0), and the edge texels get interpolated with the border if wrap mode is not REPEAT. The CLAMP\_TO\_EDGE\_SGIS can be used to ensure that the border texels are never accessed.
- Post texture lighting is supported to allow lighting highlights to be applied after texturing. For more information, see the EXT\_SEPARATE\_SPECULAR\_COLOR extension in “New Extensions Supported by VPro” on page 33.
- Trilinear filtering is fully hardware-accelerated.
- All textures (including 3D textures) are perspective-correct.
- Mipmapping is not supported with 3D textures. Applications which supply a mipmap pyramid and request a \*\_MIPMAP\_\* mode will automatically use NEAREST or LINEAR, regardless of what minification was requested. Any mipmap filtering will be ignored for 3D textures.
- 24-bit textures are not supported.

- The multi-texture extension is not supported, though the same effect can be accomplished through the accumulation buffer, blending, or a number of other techniques.
- Asynchronous texture download will exist through use of the `SGIX_ASYNC` extension. This feature will only be available on the 128MB Next Generation VPro.. Learn more about this feature in subsection “Pixel Operations” on page 22 and section “New Extensions Supported by VPro” on page 33.

### Shading Support

Both Gouraud (per-vertex) and Phong shading (per-fragment) are supported in hardware. One single per-fragment light and eight per-vertex lights are supported in hardware. The per-fragment light and the per-vertex lights can be enabled simultaneously allowing a total of nine hardware-accelerated lights in a scene. A single light of each type is fastest. Each additional per-vertex light added will incrementally decrease performance. As with all of VPro, shading is done with 12-bit per-component resolution. Two-sided lighting, local lighting, local viewer, and the other standard OpenGL 1.2 lighting properties, are all implemented in hardware.

Use the rescale normal OpenGL 1.2 extension instead of `NORMALIZE` when introducing a scale into the model view matrix. Rescale normal is fully hardware-accelerated in the case where normalized normals are used with a scale in the model view matrix.

Shading is perspective-correct.

### Anti-Aliasing (AA) and Fog

Point, line, and polygon AA is supported in hardware for both RGBA and CI visuals. Non-AA lines and points are supported in hardware up to a width of 10 with a very small incremental performance decrease for each additional width increment. AA points and lines are supported to a width of 2.0; AA widths above 2.0 will be clamped to 2.0. AA lines are hardware-accelerated, although the end caps will be French-cut, not AA. Full-scene AA will be supported using multi-pass to the hardware accumulation buffer.

### Instrumentation

Instrumentation is not yet implemented but will be supported in the future.

## Geometry

Unlike previous graphics systems from SGI, VPro does not have a full geometry engine. Instead, there is a simple transform engine that runs at a very high clock rate. The transform engine performs well even for short strip lengths with even better rates for triangle strips of length 4 or greater.

### Geometry Fast Paths

Texturing and user clip planes both use a shared hardware texturing resource. VPro supports six user clip planes. There are three hardware clip planes that are implemented using the texturing hardware. The remaining three of the total six are implemented in software. Each texture dimension that is enabled moves a hardware-accelerated user clip plane to software. For example, there is only a single hardware-accelerated user clip plane when 2D texturing is enabled.

#### Fast Path

- Points, lines, triangles in strips for display lists and vertex arrays Gouraud, Phong shading, depth, stencil, and accumulation buffer operations
- Alpha Blending and alpha functions (fill hit with blending)
- Enable texturing has little performance impact (textured versus non-textured fill)
- The rescale normal extension is fully implemented in hardware. Use it where possible instead of NORMALIZE. (glnormalize versus rescale\_normal with respect to geometry rates)
- Short normals are accelerated.
- 2 lights, one Gouraud, and one Phong, 4 infinite lights, 2 double-sided lights

### Host Bandwidth

Geometry rates are bounded primarily by the host-to-graphics download rate, although textured geometry can become fill-limited. VPro is efficient with short strips ( 4 to 5 triangles), although long strips are better since they help to reduce the amount of data downloaded. There are three methods of sending data to the VPro board: immediate mode, display lists, and vertex arrays. All geometry is pushed to the graphics board from the CPU while textures and pixels are transferred using a DMA engine.



Immediate-mode rendering pushes each vertex and associated vertex data to the graphics pipe through a function call. This is the slowest method to send data to the pipe. Display lists can pack all data into one list and send it to the pipe more efficiently. Geometry calls within an VPro display list are optimized on the host for more efficient rendering. Adding mode changes or other state manipulation commands to a display list will not affect the performance of the entire display list. This is a change from previous hardware, where certain commands in the display list would be a performance penalty. Additionally, system memory is used to hold all display lists. This ensures that display list performance does not vary with the number or size of display lists.

Vertex arrays also offer an efficient way to send data to the graphics pipe. Packed vertex arrays are fully hardware-accelerated, though slightly slower than display lists. Optimizations are implemented for the commonly used packed vertex arrays and for some of the common separate arrays. The separate arrays will always be slower than the packed arrays due to cache effects. Likewise, the **glDrawElements()** and **glArrayElement()** calls will be slower than the corresponding non-**gl\*Element()** calls.

The following array formats are explicitly tuned:

```
V3F
N3F_V3F
N3S_V3F
V3F_N3F
C3F_V3F
T2F_V3F
```

## State Changes

Much of the VPro state is shadowed on the host. The OpenGL libraries will be able to determine if a state change actually changes hardware state without a need to flush the pipe and query the hardware. The state shadowing allows **glGet\*** operations that occur outside of a **glBegin/End** pair, the equivalent set operations, and **glEnables** and **glDisables** to be very efficient for applications that do a lot of state manipulation. Matrix manipulations are also shadowed.

Color, Normal, and TexCoord calls are not shadowed, nor are the push and pop attribute calls. The **glPushAttrib()** and **glPopAttrib()** calls will be costly compared to a single state change. If possible, applications should carefully monitor and control their own state instead of pushing and popping attributes.

## Pixel Operations

This section describes the following topics:

- buffer reads and writes
- data conversions
- non-blocking texture loads (and pixel reads and draws)

### Buffer Reads and Writes

Buffer-to-buffer copies are fast as all buffers reside in on-board memory. This includes copies to and from pbuffers and textures. Pixel and texture data is transferred to the host by a DMA engine. There is hardware on-board to convert from an internal pixel format to an external pixel format; so, the format that is read or written is not important. When working with YCrCb and color conversions, the color matrix path will introduce a slight performance penalty.

Depth and stencil are stored in an internal floating-point format and packed together. Reading and writing depth and stencil buffers will be slower than the other buffers due to float conversions.

Pixel transfer speed will depend on the data types. The number of bytes transferred per pixel is shown in Table 2-3.

**Table 2-3** Pixel Transfer Speed

Data Type	Number of Bytes Transferred
CI8, L8	1Byte
CI12, L16, LA8, RGBA4, RGB5_A1	2Bytes
RGB8	3Bytes
LA16, RGBA8, RGBA10_A2, Z24_S8	4Bytes
RGBA12	6Bytes
RGBA16	8Bytes

## Data Conversions

Data type conversions are done to any float format and readback of integer formats. ARGB and BGRA short and integer are host-converted unless they are one of the packed formats (for example, 10/10/10/2).

## Non-Blocking Texture Loads (and Pixel Reads and Draws)

VPro supports the `GL_SGIX_async` and `GL_SGIX_async_pixel` set of extensions, which allow non-blocking texture and pixel reads and writes. This means that other graphics operations can proceed immediately after a texture load request. You must properly synchronize commands to guarantee that texture data is resident in the graphics memory before rendering of data using that texture begins.

## Imaging Operations

The VPro imaging pipeline is a full OpenGL 1.2 pipeline, including the *ARB imaging* package. It can be used for 2D image processing operations using `glDrawPixels`, `glReadPixels`, `glCopyPixels`, `glTexImage`, and `glGetTexImage`. The VPro pixel path implements the full OpenGL 1.2 imaging pipeline.

## Convolutions

7 x 7 convolutions will be done in hardware while larger convolutions require multi-pass rendering and will go slower. At some convolution sizes, the hardware speed is slower than a software convolve. The cutoff may be as low as 7 x 7 and is dependent on both kernel size and pixel depth.

## YCrCb format

There is no direct support for YCrCb format. However, VPro supports the subsample and resample extensions in hardware.

## Other

VPro supports pixel textures (1D, 2D, and 3D) and a hardware accumulation buffer.



---

## Extensions

This chapter describes VPro's support of extensions in the following manner:

- IMPACT graphics extensions supported by VPro
- new extensions supported by VPro
- IMPACT graphics extensions not supported by VPro
- other extensions not supported by VPro

---

**Note:** SGI has filed for patent protection for the extensions described in this chapter.

---

### IMPACT Graphics Extensions Supported by VPro

Table 3-1 describes the IMPACT extensions supported by VPro.

**Table 3-1** IMPACT Graphics Extensions Supported by VPro

Extension	Description
EXT abgr	EXT_ABGR extends the list of host-memory color formats. Specifically, it provides a reverse-order alternative to image format RGBA. The ABGR component order matches the cpack Iris GL format on big-endian machines.
EXT blend color	Blending capability is extended by defining a constant color that can be included in blending equations. A typical usage is blending two RGB images. Without the constant blend factor, one image must have an alpha channel with each pixel set to the desired blend factor.
EXT blend logic op	A single additional blending equation is specified using the interface defined by EXT_BLEND_MINMAX. This equation is a simple logical combination of the source and destination colors, where the specific logical operation is as specified by LogicOp. While only the XOR operation may find wide application, the generality of full logical operations is allowed.

**Table 3-1 (continued)** IMPACT Graphics Extensions Supported by VPro

Extension	Description
EXT blend minmax	<p>Blending capability is extended by respecifying the entire blend equation. While this document defines only two new equations, the <code>BlendEquationEXT</code> procedure that it defines will be used by subsequent extensions to define additional blending equations.</p> <p>The two new equations defined by this extension produce the minimum (or maximum) color components of the source and destination colors. Taking the maximum is useful for applications such as maximum projection in medical imaging.</p>
EXT blend subtract	<p>Two additional blending equations are specified using the interface defined by <code>EXT_BLEND_MINMAX</code>. These equations are similar to the default blending equation but produce the difference of its left and right hand sides rather than the sum. Image differences are useful in many image processing applications.</p>
SGI color matrix	<p>This extension adds a 4 x 4 matrix stack to the pixel transfer path. The matrix operates on RGBA pixel groups, using the equation</p> $C' = MC$ <p>where</p> $C = \begin{pmatrix} R \\ G \\ B \\ A \end{pmatrix}$ <p>and M is the 4 x 4 matrix on the top of the color matrix stack. After the matrix multiplication, each resulting color component is scaled and biased by a programmed amount. Color matrix multiplication follows convolution (and the scaling and biasing that are associated with convolution.) The color matrix can be used to reassign and duplicate color components. It can also be used to implement simple color space conversions.</p>
SGI color table	<p>This extension defines a new RGBA-format color lookup mechanism. It does not replace the color lookups defined by the GL specification but rather provides additional lookup capabilities with different operations. The key difference is that the new lookup tables are treated as 1-dimensional images with internal formats like texture images and convolution filter images. From this follows the fact that the new tables can operate on a subset of the components of passing pixel groups. For example, a table with internal format ALPHA modifies only the A component of each pixel group. The table leaves the R, G, and B components unmodified.</p> <p>If <code>EXT_COPY_TEXTURE</code> is implemented, this extension also defines methods to initialize the color lookup tables from the framebuffer in addition to the standard memory source mechanisms.</p>

**Table 3-1 (continued)** IMPACT Graphics Extensions Supported by VPro

Extension	Description
EXT convolution	<p>This extension defines 1- and 2-dimensional convolution operations at a fixed location in the pixel transfer process. Thus, pixel drawing, reading, and copying, as well as texture image definition, are all candidates for convolution. The convolution kernels are themselves treated as 1- and 2-dimensional images, which can be loaded from application memory or from the framebuffer.</p> <p>This extension is designed to accommodate 3D convolution, but the API is left for a future extension.</p>
EXT copy texture	<p>This extension defines methods to load texture images directly from the framebuffer. Methods are defined for both complete and partial replacement of a texture image. Because it is not possible to define an entire 3D texture using a 2D framebuffer image, 3D textures are supported only for partial replacement.</p>
SGIS detail texture	<p>This extension introduces texture magnification filters that blend between the level 0 image and a separately defined <i>detail</i> image. The detail image represents the characteristics of the high frequency subband image above the band-limited level 0 image. The detail image is typically a rectangular portion of the subband image which is modified so that it can be repeated without discontinuities along its edges. Detail blending can be enabled for all color channels, for the alpha channel only, or for the red, green, and blue channels only. It is available only for 2D textures. .</p>
SGIX fbconfig	<p>This extension introduces a new way to describe the capabilities of a GLX drawable (that is, to describe the depth of color buffer components and the type and size of ancillary buffers), removes the <i>similarity</i> requirement when making a context current to a drawable, and supports RGBA rendering to 1- and 2-component windows and GLX pixmaps.</p>
EXT histogram	<p>This extension defines pixel operations that count occurrences of specific color component values (histogram) and that track the minimum and maximum color component values (minmax). An optional mode allows pixel data to be discarded after the histogram and/or minmax operations are completed. Otherwise, the pixel data continue on to the next operation unaffected.</p>
EXT import context	<p>This extension allows multiple X clients to share an indirect rendering context.</p> <p>It also provides additional convenience procedures to get the current Display* bound to a context as well as other context information.</p>
SGI make current read	<p>The association of the current context with a drawable is extended to allow separate write and read drawables. This paves the way for allowing preprocessing of image data in an <i>off-screen</i> window, which is then read into the visible window for final display. Similarly, it sets the framework for direct transfer of video to the GL by treating the video as a special kind of read drawable (readable).</p>

**Table 3-1 (continued)** IMPACT Graphics Extensions Supported by VPro

Extension	Description
EXT packed pixels	<p>This extension provides support for packed pixels in host memory. A packed pixel is represented entirely by one unsigned byte, one unsigned short, or one unsigned integer. The fields with the packed pixel are not proper machine types, but the pixel as a whole is. Thus, the pixel storage modes, including <code>PACK_SKIP_PIXELS</code>, <code>PACK_ROW_LENGTH</code>, <code>PACK_SKIP_ROWS</code>, <code>PACK_IMAGE_HEIGHT_EXT</code>, <code>PACK_SKIP_IMAGES_EXT</code>, <code>PACK_SWAP_BYTES</code>, <code>PACK_ALIGNMENT</code>, and their unpacking counterparts all work correctly with packed pixels.</p>
SGIX pbuffer	<p>This extension defines pixel buffers (<b>GLXPbuffers</b> or pbuffer, for short). <b>GLXPbuffers</b> are additional non-visible rendering buffers for an OpenGL renderer. <b>GLXPbuffers</b> are equivalent to <b>GLXPixmap</b>s with the following exceptions:</p> <ul style="list-style-type: none"> <li>-There is no associated X pixmap. Also, since a <b>GLXPbuffer</b> is a GLX resource, it may not be possible to render to it using X or an X extension other than GLX.</li> <li>-The format of the color buffers and the type and size of any associated ancillary buffers for a <b>GLXPbuffer</b> can only be described with a <b>GLXFBConfig</b>; an X Visual cannot be used.</li> <li>-It is possible to create a <b>GLXPbuffer</b> whose contents may be asynchronously lost at any time.</li> <li>-<b>GLXPbuffers</b> can be rendered to using either direct or indirect rendering contexts.</li> <li>-The allocation of a <b>GLXPbuffer</b> can fail if there are insufficient resources (that is, all the pbuffer memory has been allocated and the implementation does not virtualize pbuffer memory.)</li> </ul> <p>The intent of the pbuffer semantics is to enable implementations to allocate pbuffers in non-visible framebuffer memory. These pbuffers are intended to be static resources in that a program will typically allocate them only once rather than as a part of its rendering loop. However, they should be deallocated when the program is no longer using them (for example, if the program is iconified). The framebuffer resources that are associated with a pbuffer are also static and are deallocated only when the pbuffer is destroyed, or, in the case of a unpreserved pbuffer, as a result of X server activity that changes its framebuffer requirements.</p> <p>The geometry rasterization and pixel pipeline convert-to-fragment stages each produce fragments. The fragments are processed by a unified per-fragment pipeline that begins with the application of the texture to the fragment color. Because the pixel pipeline shares the per-fragment processing with the geometry pipeline, the fragments produced by the pixel pipeline must have the same fields as the ones produced by the geometry pipeline. When pixel groups are being converted to fragments, the parts of the fragment that are not derived from the pixel groups are taken from the associated values in the current raster position.</p> <p>A fragment consists of x and y window coordinates and their associated color value, depth value, and texture coordinates. In the 1.1 OpenGL specification, when the pixel group is RGBA, the fragment color is always derived from the pixel group, and the depth value and texture coordinates always come from the raster position.</p>



**Table 3-1 (continued)** IMPACT Graphics Extensions Supported by VPro

Extension	Description
SGIX pixel texture	<p>This extension provides a way to specify how the texture coordinates of the fragments can be derived from RGBA pixel groups. When this option is enabled, the source of the fragment color value when the pixel group is RGBA can be specified to come from either the raster position or the pixel group.</p> <p>Deriving the fragment texture coordinates from the pixel group effectively converts a color image into a texture coordinate image. The multidimensional texture-mapping lookup logic also makes this extension useful for implementing multidimensional color lookups. Multidimensional color lookups can be used to implement very accurate color space conversions.</p> <p>Deriving texture coordinates from the pixel groups in the pixel pipeline introduces a problem with the lambda parameter in the texture mapping equations. When texture coordinates are being taken from the texture coordinates of the current raster position, the texture coordinate values do not change from pixel to pixel, and the equation for calculating lambda always produces zero. Enabling <code>SGIX_PIXEL_TEXTURE</code> introduces changes in the texture coordinates from pixel to pixel. These changes are not necessarily meaningful for texture lookups. This problem is addressed by specifying that lambda is always set to zero when <code>SGIX_PIXEL_TEXTURE</code> is enabled.</p>
EXT polygon offset	<p>The depth values of fragments generated by rendering polygons are displaced by an amount that is proportional to the maximum absolute value of the depth slope of the polygon, measured and applied in window coordinates. This displacement allows lines (or points) and polygons in the same plane to be rendered without interaction; the lines are rendered either completely in front of or behind the polygons (depending on the sign of the offset factor). It also allows multiple coplanar polygons to be rendered without interaction if different offset factors are used for each polygon. Applications include rendering hidden-line images, rendering solids with highlighted edges, and applying decals to surfaces.</p>
EXT subtexture	<p>This extension allows a contiguous portion of an already existing texture image to be redefined without affecting the remaining portion of the image or any of the other state that describe the texture. No provision is made to query a subregion of a texture.</p> <p>Semantics for null image pointers are defined for <code>glTexImage1D</code>, <code>glTexImage2D</code>, and <code>glTexImage3D</code>. Null image pointers can be used by applications to effectively support texture arrays whose dimensions are not a power of 2.</p>
SGIX swap barrier	<p>This extension provides the capability to synchronize the buffer swaps of different swap groups. A swap group is bound to a <code>_swap_barrier_</code>. The buffer swaps of each swap group using that barrier will wait until every swap group using that barrier is ready to swap (where readiness is defined under extensions SGI swap control and SGIX swap group), after which time all buffer swaps of all groups using that barrier will take place concurrently.</p>
SGI swap control	<p>This extension extends the set of conditions that must be met before a buffer swap can take place. This extension allows an application to specify a minimum period of color buffer swaps, measured in video frame periods.</p>

**Table 3-1** (continued) IMPACT Graphics Extensions Supported by VPro

Extension	Description
SGIX swap group	<p>This extension, like SGI swap control, extends the set of conditions that must be met before a buffer swap can take place. This extension provides the capability to synchronize the buffer swaps of a group of GLX drawables. A swap group is created, and drawables are added as members to the swap group. Buffer swaps to members of the swap group will then take place concurrently.</p>
EXT texture	<p>The original intention of this extension was simply to support various numeric resolutions of color components in texture images. While it accomplishes this, it also accomplishes a larger task, that of formalizing the notion of an internal format for images that corresponds to the external format that already exists for image data in host memory. This notion of an internal image format will be used extensively in later extensions, especially those concerned with pixel manipulation.</p> <p>The idea of an internal format is simple: rather than treating a retained image as having 1, 2, 3, or 4 components, treat it as though it has a specific format, such as LUMINANCE_ALPHA or just ALPHA. Then define the semantics of the use of internal images with these formats in a consistent way. Because texture mapping is already defined in GL, the semantics for internal-format images were chosen to match those of the 1-, 2-, 3- and 4-component internal images that already exist. The new semantics are a superset of the old ones. As such, this extension adds capabilities to the GL as well as allowing internal resolutions to be specified.</p> <p>This extension also defines a robust method for applications to determine what combinations of texture dimensions and resolutions are supported by an implementation. It also introduces a new texture environment: REPLACE_EXT.</p>
EXT texture 3D	<p>This extension defines 3-dimensional texture mapping. In order to define a 3D texture image conveniently, this extension also defines the in-memory formats for 3D images and adds pixel storage modes to support them.</p> <p>One important application of 3D textures is volume rendering.</p>
SGIS texture border clamp	<p>The base OpenGL provides clamping such that the texture coordinates are limited to exactly the range [0,1]. When a texture coordinate is clamped using this algorithm, the texture sampling filter straddles the edge of the texture image, taking 1/2 its sample values from within the texture image and the other 1/2 from the texture border. It is sometimes desirable for a texture to be clamped to the border color rather than to an average of the border and edge colors.</p> <p>This extension defines an additional texture clamping algorithm, CLAMP_TO_BORDER_SGIS. It clamps texture coordinates at all mipmap levels such that NEAREST and LINEAR filters return the color of the border texels. When used with FILTER4 filters, the filter operation of CLAMP_TO_BORDER_SGIS is defined but does not result in a nice clamp-to-border color.</p>

**Table 3-1 (continued)** IMPACT Graphics Extensions Supported by VPro

Extension	Description
SGI texture color table	<p>This extension adds a color lookup table to the texture mechanism. The table is applied to the filtered result of a texture lookup before that result is used in the texture environment equations. The definition and application of the texture color table are similar to those of the color tables defined in SGI_COLOR_TABLE, though it is not necessary for that extension to be implemented. Texture color tables can be used to expand luminance or intensity textures to full RGBA and also to linearize the results of color space conversions implemented by multidimensional texture table lookup.</p>
SGIS texture lod	<p>This extension imposes two constraints related to the texture level of detail parameter LOD, which is represented by the Greek character lambda in the GL specification. One constraint clamps LOD to a specified floating point range. The other limits the selection of mipmap image arrays to a subset of the arrays that would otherwise be considered.</p> <p>Together these constraints allow a large texture to be loaded and used initially at low resolution and to have its resolution raised gradually as more resolution is desired or available. Image array specification is necessarily integral rather than continuous. By providing separate, continuous clamping of the LOD parameter, it is possible to avoid <i>popping</i> artifacts when higher resolution images are provided.</p> <p><b>Note:</b> Because the shape of the mipmap array is always determined by the dimensions of the level 0 array, this array must be loaded for mipmapping to be active. If the level 0 array is specified with a null image pointer, however, no actual data transfer will take place. A sufficiently tuned implementation might not even allocate space for a level 0 array so specified until true image data is presented.</p>
EXT texture object	<p>This extension introduces named texture objects. The only way to name a texture in GL 1.0 is by defining it as a single display list. Because display lists cannot be edited, these objects are static. Yet, it is important to be able to change the images and parameters of a texture.</p>
SGI transparent pixel	
EXT vertex array	<p>This extension adds the ability to specify multiple geometric primitives with very few subroutine calls. Instead of calling an OpenGL procedure to pass each individual vertex, normal, or color; separate arrays of vertexes, normals, and colors are pre-specified and are used to define a sequence of primitives (all of the same type) when a single call is made to <code>glDrawArraysEXT</code>. A stride mechanism is provided so that an application can choose to keep all vertex data staggered in a single array or sparsely in separate arrays. Single-array storage may optimize performance on some implementations.</p> <p>This extension also supports the rendering of individual array elements, each specified as an index into the enabled arrays.</p>

**Table 3-1**      **(continued)**      IMPACT Graphics Extensions Supported by VPro

Extension	Description
SGI video sync	<p>This extension provides a means for synchronization with the video frame rate of a monitor. (In the case of an interlaced monitor, this is typically the rate of displaying both the even and odd fields of a frame.) The kernel maintains a video sync counter for each physical hardware pipe in a system; the counter is incremented upon the completion of the display of each full frame of video data. An OpenGL context always corresponds to a pipe. When an OpenGL process has a current context, it can put itself to sleep until the counter of that pipe reaches a desired value. The process can also query the value of the counter. The counter is an unsigned 32-bit integer.</p> <p>The counter runs as long as the graphics subsystem is running; it is initialized by the <code>/usr/gfx/gfxinit</code> command. However, a process can query or sleep on the counter only when a direct context is current. An error code will be returned if you attempt to use this extension (and associated functions) with an indirect context.</p>
EXT visual info	<p>This extension allows you to request a particular X visual type to be associated with a GLX visual and to query the X visual type underlying a GLX visual.</p> <p>In addition, this extension provides a means to request a visual with a transparent pixel and to query whether a visual supports a transparent pixel value and to query its value. Note that the notions of level and transparent pixels are orthogonal, as both layer 1 and layer 0 visuals may or may not have a transparent pixel values.</p>

## New Extensions Supported by VPro

Table 3-2 describes the new extensions supported by VPro.

**Table 3-2** New Extensions Supported by VPro

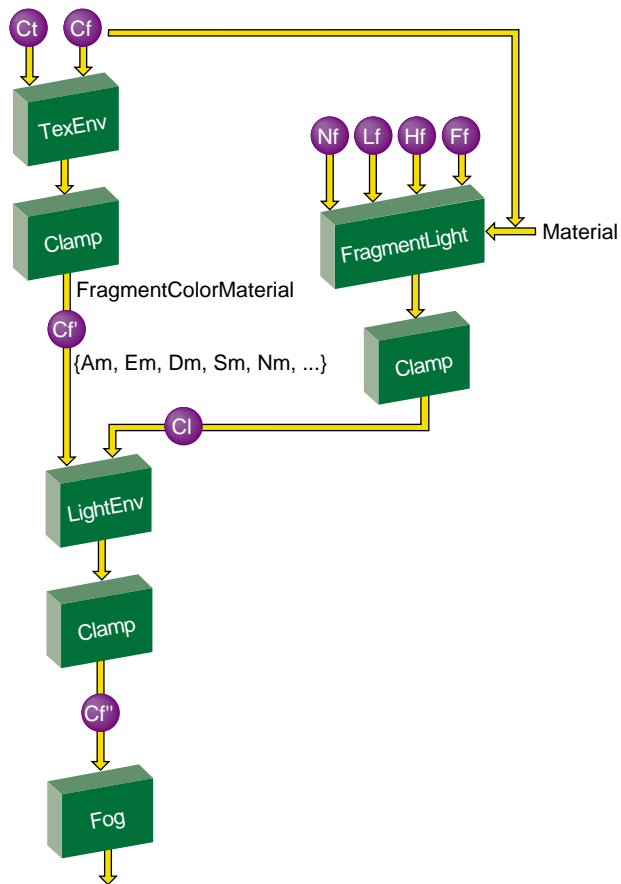
Extension	Description
SGIX async	<p>This extension provides a framework for asynchronous OpenGL commands. It also provides commands allowing a program to wait for the completion of asynchronous commands.</p> <p>Asynchronous commands have two properties:</p> <ul style="list-style-type: none"> <li>- Asynchronous commands are non-blocking. For example, an asynchronous <code>glReadPixels</code> command returns control to the program immediately rather than blocking further program execution until the command completes. This property allows the program to issue other OpenGL commands in parallel with the execution of commands that normally block.</li> <li>- Asynchronous commands may complete out-of-order with respect to other OpenGL commands. For example, an asynchronous <code>glTexImage</code> command may complete after subsequent OpenGL commands issued by the program rather than maintaining the normal serial order of the OpenGL command stream. This property allows the graphics accelerator to execute asynchronous commands in parallel with the normal command stream; for instance, the accelerator can use a secondary path to transfer data from or to the host without doing any dependency checking.</li> </ul> <p>Programs that issue asynchronous commands must also be able to determine when the commands have completed. The completion status may be needed so that results can be retrieved (for example, the image data from a <code>glReadPixels</code> command) or so that dependent commands can be issued (for example, drawing commands that use texture data downloaded by an earlier asynchronous command). This extension provides fine-grain control over asynchronous commands by introducing a mechanism for determining the status of individual commands.</p> <p>Each invocation of an asynchronous command is associated with an integer called a <i>marker</i>. A program specifies a marker before it issues an asynchronous command. The program may later issue a command to query if any asynchronous commands have completed. The query commands return a marker to identify the command that completed. This extension provides both blocking and non-blocking query commands.</p>
SGIX async pixel	<p>This extension introduces a new asynchronous mode for texture download, pixel download, and pixel readback commands. It allows programs to transfer textures or images between the host and the graphics accelerator in parallel with the execution of other graphics commands (possibly taking advantage of a secondary path to the graphics accelerator). It also allows programs to issue non-blocking pixel readback commands that return immediately after they are issued so that the program can issue other commands while the readback takes place.</p> <p>This extension does not define any asynchronous commands. See <code>GL_SGIX_async_pixel</code> documentation for the asynchronous pixel commands.</p>

**Table 3-2 (continued)** New Extensions Supported by VPro

Extension	Description
SGIX blend alpha minmax	Two additional blending equations are specified using the interface defined by <code>GL_EXT_blend_minmax</code> . These equations are similar to the <code>MIN_EXT</code> and <code>MAX_EXT</code> blending equations, but the outcome for all four-color components is determined by a comparison of just the alpha component's source and destination values. These equations are useful in image processing and advanced shading algorithms.
EXT color subtable	This extension allows a portion of a color table to be redefined. If <code>GL_EXT_copy_texture</code> is implemented, this extension also defines a method to load a portion of a color lookup table from the framebuffer.
EXT compiled vertex array	This extension defines an interface which allows static vertex array data to be cached or pre-compiled for more efficient rendering. This is useful for implementations which can cache the transformed results of array data for reuse by several <code>glDrawArrays</code> , <code>glArrayElement</code> , or <code>glDrawElements</code> commands. It is also useful for implementations which can transfer array data to fast memory for more efficient processing.  For example, rendering an $M \times N$ mesh of quadrilaterals can be accomplished by setting up vertex arrays containing all of the vertexes in the mesh and issuing $M$ <code>glDrawElements</code> commands, each of which operate on $2 * N$ vertexes. Each <code>glDrawElements</code> command after the first will share $N$ vertexes with the preceding <code>glDrawElements</code> command. If the vertex array data is locked while the <code>glDrawElements</code> commands are executed, then OpenGL may be able to transform each of these shared vertexes just once.
SGIX compressed textures	This extension defines new host and internal formats for the storage of compressed images. The formats utilize a variant of color cell compression, in which $4 \times 4$ pixel blocks are represented by two-color values and a 2-bit index per pixel. Two additional values are interpolated between the two explicitly stored values, and each pixel's index selects one of these four values. For RGB and RGBA images, two RGB colors and a single index per pixel are used to store the R, G, and B channels. For RGBA and <code>LUMINANCE_ALPHA</code> formats, the alpha channel is encoded independently using two alpha values and an index per pixel.
SGIX decimation	This extension adds a decimation operation to the pixel transfer path. Decimation occurs after convolution and prior to the post-convolution color table.  The operation is controlled by two positive integer parameters, <i>stepx</i> and <i>stepy</i> , that specify the decimation step size in the x and y directions, respectively. During a pixel transfer, the decimation operation passes only those pixels which are at a position $(i*stepx, j*stepy)$ , for integers $(i, j)$ relative to the bottom left corner of the image that is being transferred. All other pixels are discarded. Decimation is applied to the image that results from convolution, which may differ in size from the source image if convolution is enabled.  By default, <i>stepx</i> and <i>stepy</i> are both 1.
EXT draw range elements	

**Table 3-2 (continued)** New Extensions Supported by VPro

Extension	Description
SGIX fragment lighting	<p>This extension adds a new lighting stage to the OpenGL pipeline. This stage occurs during fragment processing after the texture environment has been applied and before fog has been applied. The extension provides a mechanism for computing <i>per-pixel lighting</i>. Fragment lighting applies to fragments generated by all primitives including pixel images. This extension does not eliminate vertex lighting but can be used to complement it. For example, the diffuse contribution can be evaluated at each vertex, and the specular contribution can be evaluated at each fragment with the results summed together to generate the final result.</p>



**Table 3-2 (continued)** New Extensions Supported by VPro

Extension	Description
SGIX texture color mask	<p>This extension implements the same functionality for texture updates that <b>glColorMask</b> implements for color buffer updates. Masks for updating textures with indexed internal formats (the analog for <b>glIndexMask</b>) should be supported by a separate extension.</p> <p>The extension allows an application to update a subset of components in an existing texture. The masks are applied after all pixel transfer operations have been performed, immediately prior to writing the texel value into texture memory. They apply to all texture updates.</p>
SGIS texture edge clamp	<p>This extension defines a new texture clamping algorithm. The base OpenGL provides clamping such that the texture coordinates are limited to exactly the range [0,1]. When a texture coordinate is clamped using this algorithm, the texture sampling filter straddles the edge of the texture image. The filter takes 1/2 its sample values from within the texture image and the other 1/2 from the texture border. It is sometimes desirable to clamp a texture without requiring a border and without using the constant border color.</p> <p>The new texture clamping algorithm, <b>CLAMP_TO_EDGE_SGIS</b>, clamps texture coordinates at all mipmap levels such that the texture filter never samples a border texel. When used with a <b>NEAREST</b> or a <b>LINEAR</b> filter, the color returned when clamping is derived only from texels at the edge of the texture image. When used with <b>FILTER4</b> filters, the filter operations of <b>CLAMP_TO_EDGE_SGIS</b> are defined but do not result in a nice clamp-to-edge color. <b>CLAMP_TO_EDGE_SGIS</b> is supported by 1-, 2-, and 3-dimensional textures only.</p>
SGIX texture env add	<p>A new texture environment function <b>ADD</b> is supported with the following equation:</p> $C_v = C_f + C_c C_t + C_b$ <p>A new function may be specified by calling <b>glTexEnv</b> with <b>GL_ADD</b> token. New parameter <b>C<sub>b</sub></b> (bias) may be specified by calling <b>glTexEnv</b> with <b>TEXTURE_ENV_BIAS_SGIX</b> token.</p>
SGIX texture lod bias	<p>This extension modifies the calculation of texture level of detail parameter <b>LOD</b>, which is represented by the Greek character lambda in the GL specification. The <b>LOD</b> equation assumes that a <math>2^n \times 2^m \times 2^l</math> texture is band-limited at <math>2^{(n-1)}</math>, <math>2^{(m-1)}</math>, <math>2^{(l-1)}</math>. Often a texture is oversampled or filtered such that the texture is band-limited at lower frequencies in one or more dimensions. The result is that texture-mapped primitives appear excessively blurry. This extension provides biases for <b>n</b>, <b>m</b>, and <b>l</b> in the <b>LOD</b> calculation to compensate for under- or over-sampled texture images. Mipmapped textures can be made to appear sharper or blurrier by supplying a negative or positive bias, respectively.</p> <p>Examples of textures which can benefit from this <b>LOD</b> control include the following:</p> <ul style="list-style-type: none"> <li>-video-capture images which are filtered differently horizontally and vertically</li> <li>-a texture which appears blurry because it is mapped with a nonuniform scale, such as a road texture which is repeated hundreds of times in one dimension and only once in the other</li> <li>-textures which had to be magnified to a power of 2 for mipmapping</li> </ul>



**Table 3-2 (continued)** New Extensions Supported by VPro

Extension	Description
SGIX texture scale bias	<p>This extension adds scale, bias, and clamp to [0, 1] operations to the texture pipeline. These operations are applied to the filtered result of a texture lookup before that result is used in the texture environment equations and before the texture color lookup table of <code>GL_SGI_texture_color_table</code>, if that extension exists. These operations are distinct from the scale, bias, and clamp operations that appear in the <code>GL_SGI_color_table</code> extension, which are used to define a color lookup table.</p> <p>Scale and bias operations on texels can be used to better utilize the color resolution of a particular texture internal format (see EXT texture).</p>
SGIX_subsample	<p>Many video image formats and compression techniques utilize various component subsamplings. So, it is necessary to provide a mechanism to specify the up- and down-sampling of components as pixel data is drawn from and read back to the client. Though subsampled components are normally associated with the video color space, YCrCb, use of subsampling in OpenGL does not imply a specific color space.</p> <p>This extension defines new pixel storage modes that are used in the conversion of image data to and from component subsampled formats on the client side. The extension defines a new pixel storage mode to specify these sampling patterns. There are three valid values:</p> <p style="text-align: center;"> <code>SAMPLING_UNIFORM_SGIX</code>  <code>SAMPLING_422_SGIX</code>  <code>SAMPLING_4224_SGIX</code> </p> <p>When pixel data is received from the client and an unpacking upsampling mode other than <code>SAMPLING_UNIFORM_SGIX</code> is specified, the up-sampling is performed by one of two methods: <code>RESAMPLE_REPLICATE_SGIX</code> or <code>RESAMPLE_ZERO_FILL_SGIX</code>. Replicate and zero-fill are provided to give the application greatest performance and control over the filtering process. Similarly, when pixel data is read back to the client and a packing down-sampling mode other than <code>SAMPLING_UNIFORM_SGIX</code> is specified, down-sampling is performed. If either replicate or zero-fill is specified, then the down-sampling is performed by simple component decimation.</p>
EXT bgr, EXT bgra	<p><code>EXT_BGRA</code> extends the list of host-memory color formats. Specifically, it provides formats which match the memory layout of Windows DIBs so that applications can use the same data in both Windows API calls and OpenGL pixel API calls.</p>

**Table 3-2** (continued) New Extensions Supported by VPro

Extension	Description
EXT separate specular color	<p>This extension adds a second color to rasterization when lighting is enabled. Its purpose is to produce textured objects with specular highlights, which are the color of the lights. It applies only to RGBA lighting.</p> <p>The two colors are computed at the vertexes. They are both clamped, flat-shaded, clipped, and converted to fixed point just like the current RGBA color. Rasterization interpolates both colors to fragments. If texture is enabled, the first (or primary) color is the input to the texture environment; the fragment color is the sum of the second color and the color resulting from texture application. If texture is not enabled, the fragment color is the sum of the two colors.</p> <p>A new control to <code>glLightModel*</code>, <code>LIGHT_MODEL_COLOR_CONTROL_EXT</code>, manages the values of the two colors. The new control can have one of two values: <code>SINGLE_COLOR_EXT</code>, a compatibility mode, or <code>SEPARATE_SPECULAR_COLOR_EXT</code>, the object of this extension. In single color mode, the primary color is the current final color and the secondary color is 0.0. In separate specular mode, the primary color is the sum of the ambient, diffuse, and emissive terms of final color and the secondary color is the specular term.</p> <p>There is much concern that this extension may not be compatible with the future direction of OpenGL with regards to better lighting and shading models. Until those impacts are resolved, give this serious consideration before using this extension ( allowing the user to specify a second input color).</p>
HP convolution border modes	<p>This extension provides some additional border modes for the <code>EXT_CONVOLUTION</code> extension.</p>
SGIX flush raster	<p>This extension provides a way to ensure that all raster operations currently in the pipeline will be completed before the next raster operation begins. We define a raster operation as an operation that involves the rasterization stage of the OpenGL pipeline. The implementation is free to decide what constitutes flushing the raster subsystem.</p> <p>The motivation is to allow accurate instrumentation by including this call before stopping rasterization measurements. There are cases where <code>glFinish()</code> is used, but <code>glFlushRaster()</code> would suffice; so, this extension is deliberately kept independent of the instruments extension.</p>
SGIS fog func	<p>This extension allows you to define an application-specific fog blend-factor function. The function is defined by the set of the <i>control</i> points and should be monotonic. A value pair represents each control point: the eye-space distance value and corresponding value of the fog blending factor. The minimum number of control points is one. The maximum number is implementation-dependent.</p>

**Table 3-2 (continued)** New Extensions Supported by VPro

Extension	Description
SGIX fog offset	<p>This extension allows you to make objects look brighter in the foggy environment. FOG_OFFSET_VALUE_SGIX parameter specifies point coordinates in eye space, an offset amount toward viewpoint. Once fog offset is specified and enabled using the FOG_OFFSET_SGIX parameter, it is subtracted from the depth value (to make objects closer to the viewer) right before fog calculation. As a result, objects look less foggy.</p> <p>This extension specifies that the fragment lighting vectors; including the view vector, light vectors, half-angle vectors, and spotlight direction vectors; be transformed into either eye space, object space or tangent space on a per-vertex basis. The default is eye space.</p>
SGIX instruments	<p>This extension allows the gathering and return of performance measurements from within the graphics pipeline by adding instrumentation.</p> <p>There are two reasons to do this. The first is as a part of some type of fixed-frame-rate load management scheme. If we know that the pipeline is stalled or struggling to process the amount of data we have given it so far, we can reduce the level of detail of the remaining objects in the current frame or the next frame, or we can adjust the framebuffer resolution for the next frame if we have a video-zoom capability available. We can call this type of instrumentation <i>load monitoring</i>.</p> <p>The second is for performance tuning and debugging of an application. It might tell us how many triangles were culled or clipped before being rasterized. We can call this simply <i>tuning</i>.</p> <p>Load monitoring requires that the instrumentation and the access of the measurements be efficient; otherwise, the instrumentation itself will reduce performance more than any load management scheme could hope to offset. Tuning does not have the same requirements.</p> <p>The proposed extension adds a call to set up a measurements return buffer similar to <b>glFeedbackBuffer</b> but with an asynchronous behavior to prevent filling the pipeline with NOPs while waiting for the data to be returned.</p> <p>Note that although the extension has been specified without any particular instruments, defining either a device-dependent or device-independent instrument should be as simple as introducing an extension consisting primarily of a new enumerant to identify the instrument.</p>
SGIX list priority	<p>This extension provides a mechanism for specifying the relative importance of display lists. This information can be used by an OpenGL implementation to guide the placement of display list data in a storage hierarchy.</p>
SGIS sharpen texture	<p>This extension introduces texture magnification filters that sharpen the resulting image by extrapolating from the level 1 image to the level 0 image. Sharpening can be enabled for all color channels, for the alpha channel only, or for the red, green, and blue channels only.</p>

**Table 3-2 (continued)** New Extensions Supported by VPro

Extension	Description
EXT rescale normal	<p>When normal rescaling is enabled, a new operation is added to the transformation of the normal vector into eye coordinates. The normal vector is rescaled after it is multiplied by the inverse model view matrix and before it is normalized.</p> <p>The rescale factor is chosen so that in many cases normal vectors with unit length in object coordinates will not need to be normalized as they are transformed into eye coordinates.</p>
SGIX texture coordinate clamp	<p>This extension provides a mechanism to specify the maximum texture coordinate clamping values. Standard OpenGL always clamps the upper bound to 1.0 when the wrap mode is set to CLAMP. This mechanism can be used to guarantee that non-existent texel data will not be accessed when the texture image has dimensions that are not a power of 2.</p>

## IMPACT Graphics Extensions Not Supported by VPro

The following are IMPACT graphics extensions not supported by VPro:

- SGIX impact pixel texture
- SGIX texture multi buffer
- SGIS texture select

## Other Extensions Not Supported by VPro

The following are other extensions not supported by VPro:

- EXT visual rating
- SGIX clipmap
- SGIX cube map
- SGIX texture phase
- SGIX color range
- SGIX complex polar
- SGIX FFT
- SGIX fbconfig types
- SGIS filter4 parameters
- SGIS multisample

SGIS texture filter4  
EXT multitexture  
SGIX light texture  
SGIX cylinder texgen  
SGIX line texgen  
SGIS point line texgen  
EXT misc attribute  
SGI transparent pixel  
IBM vertex array set  
SGIX swap readiness  
SGIX sync swap  
SGIX shadow  
SGIX shadow ambient  
SGIX interlace  
SGIX async histogram  
SGIX convolution accuracy  
SGIX calligraphic fragment  
EXT clip volume hint  
EXT cmyka  
SGIX color matrix accuracy  
SGIX color table index mode  
EXT cull vertex  
SGIX cushion  
SGIX datapipe  
SGIX ffd  
SGIX framezoom  
SGIS generate mipmap  
HP image transform  
EXT index array formats  
EXT index func  
EXT index material  
PGI misc hints  
INTEL parallel arrays  
WIN phong shading  
SGIX pixel texture bits  
SGIX pixel tiles  
IBM rasterpos clip  
SGIX scene marker  
WIN specular fog  
SGIX sprite  
SGIX subdiv patch

SGIX tag sample buffer  
SGIX texture anisotropic mip  
SGIX vertex array object  
HP texture lighting  
INTEL texture scissor  
SUN transparent index  
EXT vertex callback  
PGI vertex hints  
SGIX wait group  
SGIX depth texture  
SGIX dct  
SGIX dvc  
SGIX image compression  
SGIX mpeg  
SGIX mpeg1  
SGIX mpeg2  
SGIX nurbs eval  
EXT nurbs tessellator  
EXT object space tess  
SGIX reference plane  
SGIX pixel texture lod  
SGIS texture4D  
SGIS point parameters  
SGIX video source glx  
SGIX video resize glx  
SGIX dmbuffer  
EXT paletted texture  
EXT index texture

---

## A

AA, 19  
accumulation buffers, 13  
anti-aliasing, 19

## B

bandwidth, 20  
blending, 15  
buffer I/O, 14  
buffers  
  accumulation, 13  
  buffer management, 10  
  color, 11  
  I/O, 14  
  off-screen, 14  
  OpenGL buffers, 11  
  overlay, 13  
  pbuffer, 14  
  reads, 22  
  writes, 22  
Buzz, 5

## C

color buffers, 11  
command FIFO, 7  
context switching, 7  
convolutions, 23

## D

data type conversions, 23  
dual-channel display, 6

## E

extensions  
  IMPACT, 25  
  new, 33  
  not supported, 40  
  overview, 25

## F

fast path, 20  
features  
  hardware, 5  
  hardware-accelerated, 2  
  rendering, 10, 15  
  texture, 18  
  VPro, 1

## G

geometry, 20  
geometry fast path, 20  
Gouraud shading, 19

## H

hardware features, 5  
hardware-accelerated feature, 2  
host bandwidth, 20

## I

imaging pipeline, 23  
instrumentation, 19

---

## O

OCTANE, 1, 3  
off-screen buffers, 14  
overlay buffers, 13

## P

PB&J, 5  
pbuffers, 14  
Phong shading, 19  
pixel operations, 22  
pixel textures, 23  
pixel transfer speed, 22  
product overview, 1

## R

rasterizer chip, 5  
rendering techniques, 15

## S

shading, 19  
state changes, 21  
stereo, 14

## T

Texture, 17  
texture loads, 23  
texturing, 20  
transform chip, 5

## U

user clip planes, 20

## V

video chip, 5  
visuals, 7  
VPro, 1  
VPro architecture, 5

## X

xsetmon, 6, 7

## Y

YCrCb format, 23