3Dlabs®

# GLINT® Gamma

Programmer's Reference Manual

Issue 3

Email: info@3dlabs.com
Web: http://www.3dlabs.com

**3D**labs Inc.
181 Metro Drive, Suite 520
San Jose, CA 95110
United States

Tel: (408) 436 3455
Fax: (408) 436 3458

**3D**labs Ltd.
Meadlake Place
Thorpe Lea Road, Egham
Surrey, TW20 8HE
United Kingdom

Tel: +44 (0) 1784 470555
Fax: +44 (0) 1784 470699

## Change History

| Document | Issue | Date | Change |
|---|---|---|---|
| 149.4.0 | 0 | 2 July 97 | Draft |
| 149.4.0 | 1 | 29 Sept 97 | First issue. |
| 149.4.0 | 2,3 | 26 Nov 97 | Typographical corrections and style changes. |

# Contents

# Tables

# Figures

# 1. Introduction

This is the Programmers Reference Manual for the GLINT Gamma device. GLINT Gamma (subsequently just called Gamma) is a Geometry and Lighting Processor which directly drives the GLINT 500TX and GLINT MX.

A Geometry and Lighting Processor off loads a substantial amount of computation from the host computer (to implement the 3D pipeline), thereby removing a major performance bottleneck and elevates PC graphics cards to new heights. The 3D pipeline is expanded later but broadly includes transformations, lighting, culling, clipping and rasterizer chip set-up.

Gamma replaces the GLINT Delta in a system (it is electrically and mechanically compatible) and can be used as a faster GLINT Delta, but this will only use a small amount of Gamma's potential.

Gamma has been designed to fully support OpenGL and to faithfully implement all its modes and interactions so the OpenGL specification and documentation is an excellent guide to what Gamma does. Gamma does not ignore the needs of the other two main stream graphics APIs, namely Direct3D from Microsoft and QuickDraw3D from Apple. In many respects these two APIs are subsets of OpenGL so are well covered by the OpenGL functionality, however some additional functionality has been incorporated into Gamma specifically to address the needs of these APIs.

Gamma is predominantly oriented to 3D graphics and there is very little of interest for GUI operations. The only two things are rectangle set up and faster image uploads and downloads.

Gamma implements the following parts of the Geometry and Lighting Pipeline. This is not an exhaustive list, but just summarizes the main elements:

- OpenGL Begin/End paradigm for describing primitives.
- Texture coordinate generation.
- Normal and texture coordinate transformation.
- Normalization of the normal after transformation.
- Full OpenGL RGB material and lighting for up to 16 light sources.
- Two sided lighting.
- Polygon Offset.
- OpenGL Polygon mode and Color Material operations.
- Backface culling.
- Fog calculations.
- Full frustum clipping with an optional 6 user defined clipping planes.

- Perspective division and Viewport mapping.
- Render, select and feedback modes.
- Triangle set up (aliased and antialiased).
- Line set up (aliased, antialiased, wide and stippled).
- Point set up (aliased, antialiased and wide).
- Raster Position.

Parts of OpenGL still left to software (excluding system level things such as multiple contexts, etc.):

- Matrix generation, including matrix stack.
- Attribute Push and Pop.
- Display list creation, management and parsing (in the general case).
- Evaluators.
- General state management and Get functions.
- Color index lighting.

In addition to these the following general facilities are available:

- Queued Input DMA.
- Output DMA controller.
- Scatter/Gather operations on DMA.
- Rectangular read and write of host memory.
- Rectangle (2D) set up.
- Hierarchical DMA.

The 2D/GUI support is limited to:

- Rectangle upload DMA controller.
- Rectangle download DMA controller.
- Rectangle set up.

## 1.1 Conventions Used In This Manual

Registers or commands accessed via the input FIFO or DMA (i.e. core registers) are in bold. Registers accessed directly from the PCI bus are in italics. Though Gamma operates either with GLINT 500TX or GLINT MX devices, this manual usually refers to GLINT MX for brevity.

Code snippets are in courier.

## 1.2 Performance

### 1.2.1 Gamma By Itself

If Gamma is in the perfect system so the bottle neck is how fast it can do its work then the expected performance depends on which processing path a primitive takes. The various paths are:

- It is not in view.

The throughput of Gamma in this case is 4.4 M vertices per second or 4.4M connected primitives per second. This rate is independent of any fog, normalization or lighting modes.

- It is in view, but backface culled.

The throughput of Gamma in this case is 4.1 M connected triangles per second. This rate is independent of any fog, normalization or lighting modes.

- It is fully in view, but not backface culled.

The throughput of Gamma in this case is 2.6 M connected primitives per second with one light source. Adding more lights will decrease the throughput. Doubling the number of lights will approximately half the throughput in the worst case, however this depends on the type of lights and other enabled modes such as texture generation.

- It is partially in view, but not backface culled.

It is not possible to give any meaningful figures in this case because the amount of work in the clipping is very variable depending on the exact geometry of the situation. Needless to say it will be slower than the previous 2.6M figure, however this situation is comparatively rare compared to the earlier cases.

A typical scene will have a lot of primitives which are clipped out or backface culled so will benefit from faster throughput in these cases.

The performance figures are for a 66MHz device.

### 1.2.2 Input Data Requirements

*Given the above vertex rates (the triangle rates amount to vertex rates for meshed triangles) what is the input data rate needed to sustain these figures?*

For the purpose of this analysis, each vertex can be considered to consist of a three component Cartesian coordinate and a three component normal. Each component being a single precision floating point number so takes 4 bytes.

The most compact format packs the vertex data into 7 words - one tag word and 6 actual vertex data words. In this case the tag is repeated for every vertex so there are 28 bytes per vertex. This format is useful in display lists where the driver has the time to work out the optimum format for vertices between a glBegin and glEnd.

The least compact format, but the one most suitable for 'on the fly' processing (i.e. dispatching the data as soon as it is given to you) is: normal tag, three components, vertex tag, three components i.e. 8 words or 32 bytes per vertex.

The following table summarizes these formats against the processing rates in Gamma to give the bandwidth (MB/s) needed to sustain each one of them:

| | Clipped | Backface Culled | In view |
|---|---|---|---|
| 7 words per vertex | 184.8 | 120.4 | 72.8 |
| 8 words per vertex | 211.2 | 137.6 | 83.2 |

Note this bandwidth doesn't include any allowance for bus efficiencies, latency times or breaks in burst mode.

The best PCI systems typically have a sustained bandwidth of approximately 80 MB/s so the in view performance should be achievable, but the higher clipped and culled rate are not. Gamma can cope with the peak input bandwidth the PCI bus can deliver until its FIFO fills up (the FIFO depth is greater than 100 words).

### 1.2.3 Output Data Requirements

The data flows from Gamma over the secondary PCI bus to the GLINT rendering device. The bandwidth available on this bus is much closer to the theoretical PCI peak of 132 MB/s because very long single cycle bursts can be guaranteed between the two 3Dlabs devices, and additional (non PCI) sideband signals take care of the hardware flow control.

The original triangle interface used in GLINT 500TX took 34 words to load up the start and gradients for rasterizer, color interpolants and depth interpolants and then initiate the triangle rendering. This places an upper limit of 970K triangles per second.

Gamma with a GLINT MX can use an alternative interface (called Triangle Packet Interface) which reduces the number of words needed to define and render a triangle. This interface removes some of the earlier redundant data and encodes the remaining data into a more compact format. This reduces the number of words per triangle down to 21. This increases the triangle rate to 1.57M per second.

### 1.2.4 PCI (33MHz) Performance Summary

The secondary PCI bus is clearly the limiting factor and with a GLINT MX limits the triangle throughput to 1.5M/s.

At this rate, the demand on the primary PCI bus is 36MB, 42MB or 48MB for 6, 7 and 8 words per triangle respectively. All these rate are easily achievable on current PCI systems.

If Gamma is being throttled to 1.5M triangles/s by the secondary PCI bus, this allows up to 3 lights to be turned on without any drop in system performance.

### 1.2.5 AGP Performance Summary

The main benefit an AGP system brings is in running the secondary PCI bus at 66MHz. This doubles the triangle rate between Gamma and GLINT MX to 3 million triangles per second so this bus is no longer the bottle-neck. The bottle-neck has moved back into Gamma.

Even with the most pessimistic view it is hard to see AGP delivering less than 2X over regular PCI, even with single edge clocking on AGP.

In conclusion an AGP system should be able to allow Gamma to achieve its quoted rates.

## 1.3 Further Reading

This manual is not an introduction or tutorial into 3D graphics or OpenGL. The reader is assumed to be familiar with these subjects. Similarly the details of using and programming the GLINT rasterizer family are not covered here. Suitable books or on-line material which cover these topics are:

- *OpenGL Programming Guide,* Jackie Neider et al, Reading MA: Addison-Wesley
- *OpenGL Reference Manual,* Jackie Neider et al, Reading MA: Addison-Wesley
- *The OpenGL Graphics System: A Specification (Version 1.1)*, Mark Segal and Kurt Akeley, SGI
- *Computer Graphics: Principles and Practice*, James D. Foley et al, Reading MA: Addison-Wesley.

# 2. General Programming Notes

This manual does not try to specify all the interactions between the modes (for example when flat shaded which vertex provides the color) - to do so would result in a substantial portion of the OpenGL specification being included. Gamma follows the OpenGL specification exactly and any deviations are unintentional. 3Dlabs would be very pleased to hear of them so they can be fixed in future products.

Unless otherwise noted all input values are in single precision IEEE floating point format numbers. Denormalized numbers are treated as if they were zero and NaNs are treated as if they are very large numbers.

The remainder of this section describes the programming model for Gamma. It describes the interface conceptually rather than detailing specific registers and their exact usage. In depth descriptions of how to program Gamma for specific operations may be found in later chapters.

A system will consist of a Gamma chip and at least one GLINT rasterizer chip. As far as the programmer is concerned the fact that these are two separate chips is transparent - they are programmed as if they were one.

## 2.1 Gamma as a Register File

The simplest way to view the interface to Gamma is as a flat block of memory-mapped registers (*i.e.* a register file). This register file appears as part of Region 0 of the PCI address map for Gamma. See the Gamma Hardware Reference Manual for details of this address map.

When a Gamma host software driver is initialized it can map the register file into its address space. Each register has an associated address tag, giving its offset from the base of the register file (since all registers reside on a 64-bit boundary, the tag offset is measured in multiples of 8 bytes). The most straightforward way to load a value into a register is to write the data to its mapped address. In reality the chip interface comprises a 32 entry deep FIFO[1], and each write to a register causes the written value and the register's address tag to be written as a new entry in the FIFO.

Programming Gamma and GLINT to draw a primitive consists of writing initial values to the appropriate registers followed by a write to a command register. The last write triggers the start of rendering.

Gamma has several hundred registers. All registers are 32 bits wide and should be 32-bit addressed. Many registers are split into bit fields.

**Note:** bit 0 is the least significant bit**.**

[1]This is, in fact a slight simplification - the full story is presented in the Getting Data into Gamma chapter.

Recall that the graphics registers are shown in the text as bold font (for example: **GeometryMode**). In addition there are registers related to initialization and I/O, which are documented in the Gamma Hardware Reference Manual. Where these registers are referred to in the text of this manual, they are shown in italic font, for example: *InFIFOSpace.*

*In future chip revisions the register file may be extended and currently unused bits in certain registers may be assigned new meanings. Software developers should ensure that only defined registers are written to and that undefined bits in registers are always written as zeros.*

## 2.2 Register Types

Gamma has three main types of register:

- Control and Data Registers
- Command Registers
- Internal Registers

**Control and Data Registers** are updated only by the host - the chip effectively uses them as read-only registers. Examples of control registers are **GeometryMode** and **LightingMode** registers. Example of data registers are **FrontDiffuseColorGreen**, **ViewPortOffsetX** registers. Once initialized by the host, the chip only reads these registers. These registers can be read back at any time, however writing to a register and then immediately reading it is not guaranteed to return the value written as it may not have filtered down the pipeline to this register yet.

**Command Registers** are those which, when written to, typically cause the chip to start rendering or some other type of internal processing. Normally, the host will initialize the appropriate control and data registers and then write to a command register to initiate drawing. Gamma does not have explicit commands to render a triangle, etc., as it uses the fact that a vertex has been written to initiate any action. This slightly modified mechanism is used to reduce the amount of commands sent by the host to render a primitive - this is very important when millions of primitives per second are being processed. Command registers cannot be read back.

**Note:** For convenience in this document we often refer to "sending an **XXX** command to Gamma" rather than saying "the **XXX** Command register is written to, which initiates drawing (or some other action)".

**Internal Registers** are not accessible to host software. They are used internally by the chip to hold intermediate values and share information between adjoining primitives (in a mesh or polyline).

For the most part internal registers can be ignored, however they do need to be saved and restored when a primitive group (demarcated by a **Begin** and **End** pair of commands) can be context switched in the middle. This is covered in a later chapter.

## 2.3 Efficiency

Software developers wishing to write device drivers for Gamma and GLINT should become familiar with the different types of registers. Some registers such as the **Vy** and **Vz** registers have to be updated for almost every primitive whereas other control registers such as the **ViewPortScaleX** or the **TransformMode** can be updated much less frequently. Pre-loading of the appropriate control registers can reduce the amount of data that has to be loaded into the chip for a given primitive thus improving efficiency.

The tables in Appendices B and C list the graphics registers and indicate their type.

## 2.4 DMA Tag Description Format

In general the DMA buffer format consists of a 32-bit address tag description word followed by one or more data words. The DMA buffer consists of one or more sets of these formats. The following paragraphs describe the different types of tag description words that can be used.

When DMA is performed each 32-bit tag description in the DMA buffer conforms to the format in Figure 2.1.

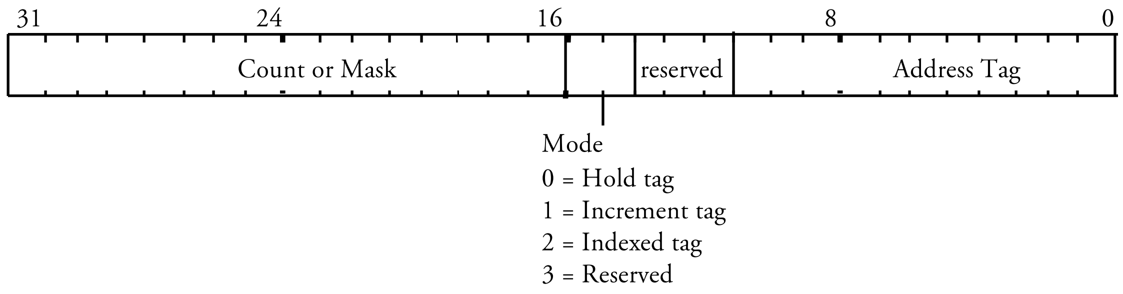

**Figure 2.1 DMA Tag Description Format**

There are 3 different tag addressing modes for DMA: hold, increment and indexed. The different DMA modes are provided to reduce the amount of data which needs to be transferred, hence making better use of the available DMA bandwidth. Each of these is described in the following sections. Each row in the following diagrams represents a 32-bit value in the DMA buffer. The address tag for each register is given in the Graphics Register Reference Appendices B and C.

### 2.4.1 Hold Format

Address-tag with Count = n-1, Mode = 0

value 1

...

value n

In this format the 32-bit tag description contains a tag value and a count specifying the number of data words following in the buffer. The DMA controller writes each of the data words to the same address tag. For example, this is useful for image download where pixel data is continuously written to the **Color** register. The bottom 11 bits specify the register to which the data should be written; the high-order 16 bits specify the number of data words minus 1 which follow in the buffer and which should be written to the address tag

*Note: The 2-bit mode field for this format is zero so a given tag value can simply be loaded into the low order 16 bits).*

A special case of this format is where the top 16 bits are zero indicating that a single data value follows the tag (*i.e.* the 32-bit tag description is simply the address tag value itself). This allows simple DMA buffers to be constructed which consist of tag/data pairs.

### 2.4.2 Increment Format

address-tag with Count = n-1, Mode = 1

value 1

...

value n

This format is similar to the hold format except that as each data value is loaded the address tag is incremented (the value in the DMA buffer is not changed; Gamma updates an internal copy). Thus, this mode allows contiguous Gamma registers to be loaded by specifying a single 32-bit tag value followed by a data word for each register. The low-order 11 bits specify the address tag of the first register to be loaded. The 2 bit mode field is set to 1 and the high-order 16 bits are set to the count minus 1 of the number of registers to update. To enable use of this format, the Gamma register file has been organized so that registers which are frequently loaded together have adjacent address tags.

### 2.4.3 Indexed Format

Gamma address tags are 11 bit values. For the purposes of the Indexed DMA Format they are organized into major groups and within each group there are up to 16 tags. The low-order 4 bits of a tag give its offset within the group. The high-order 7 bits give the major group number. Appendices B and C Register Tables, list the individual registers with their Major Group and Offset.

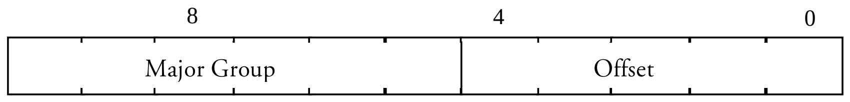

**Figure 2.2 Indexed Format**

This format allows up to 16 registers within a group to be loaded while still only specifying a single address tag description word.

address tag with Mask, Mode=2

value 1

...

value n

If the Mode of the address tag description word is set to indexed mode then the high-order 16 bits are used as a mask to indicate which registers within the group are to be used. The bottom 4 bits of the address tag description word are unused. The group is specified by bits 4 to 10. Each bit in the mask is used to represent a unique tag within the group. If a bit is set then the corresponding register will be loaded. The number of bits set in the mask determines the number of data words that should be following the tag description word in the DMA buffer. The data is stored in order of increasing corresponding address tag. For example,

0x003280F0

value 1

value 2

value 3

The Mode bits are set to 2 so this is indexed mode. The Mask field (0x0032) has 3 bits set so there are three data words following the tag description word. Bits 1, 4 and 5 are set so the tag offsets are 1, 4 and 5. The major group is given by the bits 4-8 which are 0x0F (in indexed mode bits 0-3 are ignored). Thus the actual registers to update have address tags 0x0F1, 0x0F4 and 0x0F5. These are updated with value 1, value 2 and value 3 respectively.

## 2.5 Temporal Ordering

All registers are loaded in the order they are given in and all commands are executed in order as well. A register can be updated at any time and it is guaranteed *never* to corrupt or effect any command which may be in operation and depends on the previous register contents for its correct operation. Updating a register will only effect subsequent commands.

## 2.6 Updating Mode Registers (and/or)

Many of the mode registers have an And and Or variant. For example the **GeometryMode** register can also be written to using **GeometryModeAnd** or **GeometryModeOr**. The And and Or variants combine the new data with the existing register contents using a bitwise AND or a bitwise OR operation respectively. This allows single bit fields to be set or cleared in a single write or multi-bit fields to be set to any value by first clearing the bits in question with the And variant and then using the Or variant to merge in the new value. This mechanism alleviates having to keep a software copy so that the fields we wish to retain are not overwritten. The real benefit of this is that updates to these mode registers can be held in native format display lists (i.e. display lists which don't need to be parsed by software but can be read directly by Gamma). Native display lists are obviously faster than those display lists which must be parsed.

## 2.7 Reading Back Values

Many of the registers can be read back. To read back a particular register read the address you would use to write to the register. Appendices B and C show which registers can be read back.

Reading back a register immediately it has been written to does not guarantee to return the value just written. Recall the writes are buffered in a FIFO so it may take some time before the actual register is updated. The only way to be sure the registers have been updated and are not about to be updated by state held in the FIFO or internal pipeline stages is to synchronize with Gamma and GLINT using the **Sync** command.

## 2.8 Initialization

Very few registers are initialized by power-on reset or by the software reset. It is advisable to set all registers into a well defined state before using Gamma.

# 3. Getting Data into Gamma

The facilities for getting data into Gamma are a super set of those available in the earlier members of the GLINT family. Some of these are aimed at increasing system performance while others introduce totally new functionality.

The input FIFO can still be used to pass commands and data into Gamma and any following GLINT devices, however only a subset of some of the facilities are available via this method.

A general design philosophy is that much of the new functionality is driven via the FIFO (or DMA buffer) rather than dedicated PCI registers. This brings many of the benefits normally associated with core commands such as queuing and asynchronous operation.

*Note: The DMA specific commands must be submitted as tag/data pairs and not part of an indexed tag group. Any DMA commands submitted as part of an indexed group (or indeed in any other form except as tag/data pairs) will be ignored (but may effect the following rasterizer chip) and the IllegalDMA bit in the PCI CommandError register will be set.*

All memory reads and writes instigated by Gamma go via the PCI/AGP interface to reach the host's memory. The PCI Interface implements a layer of byte swapping (enabled by the *DMAControl* register) over and above any byte swapping described in this, or the next chapter.

There are a number of ways of loading Gamma registers or issuing commands:

- The host writes a value to the mapped address of the register. This still goes via the input FIFO even though the register appears to be written directly.
- The host can perform a Block Command Transfer by writing address and data values to the FIFO interface registers.
- The host writes address-tag/data pairs into a host memory buffer and uses the on-chip DMA to do the transfer.

The first two cases will be covered next and the DMA case later.

## 3.1 Command FIFO

In cases where the host writes data values directly to the chip (via the register file) it has to worry about FIFO overflow (unless PCI Disconnect is enabled). The *InFIFOSpace* register indicates how many free entries remain in the FIFO. Before writing to any register the host must ensure that there is enough space left in the FIFO. The values in this register can be read at any time.

### 3.1.1 PCI Disconnect

The PCI bus protocol incorporates a feature known as PCI Disconnect, which is supported by Gamma. PCI Disconnect is enabled by writing to bit zero of the *DisconnectControl* register which is at offset 0x68 in PCI Region0. Once the Gamma is in this mode, if the host processor attempts to write to a full FIFO then instead of the write being lost, Gamma will assert PCI Disconnect which will cause the host processor to keep retrying the write cycle until it succeeds.

This feature allows faster download of data to Gamma, since the host need not poll the *InFIFOSpace* register it should, however be used with care since whenever the PCI Disconnect is asserted the bus is effectively hogged by the host processor until such time as the Gamma frees up an entry in its FIFO. In general this mode should only be used either for operations where it is known that the Gamma can consume data faster than the host can generate it, or where there are no time critical peripherals sharing the PCI bus.

### 3.1.2 FIFO Control

The description above considered the Gamma interface to be a register file. More precisely, when a data value is written to a register this value and the address tag for that register are combined and put into the FIFO as a new entry. The actual register is not updated until Gamma processes this entry. In the case where Gamma or the downstream rasterizer chip is busy performing a time consuming operation, and not draining the FIFO very quickly, it is possible for the FIFO to become full. If a write to a register is performed when the FIFO is full no entry is put into the FIFO and that write is effectively lost (unless PCI Disconnect is enabled as described above).

The input FIFO is 64 entries deep but a tag/data pair takes up two entries, so it has an effective size, when used like this, of 32 entries. More compact forms of tag and data (i.e. an index tag to update several related registers) can be inserted into the FIFO making it appear deeper. The *InFIFOSpace* register can be read to determine how many tag/data pairs can be written. The value returned by this register will never be greater than 32.

The *InFIFOSpace* FIFO control register contains a count of the number of entries currently free in the FIFO. The chip increments this register for each entry it removes from the FIFO and decrements it every time the host puts an entry in the FIFO.

The input FIFO can be written to at any time, even when DMA is in progress. This differs from the earlier GLINT chips where the input FIFO was shared between the DMA controller and regular host use.

The *CommandStatus* PCI register holds a bit (FIFOEmpty) which shows when the FIFO is empty.

## 3.2 Input DMA

Loading registers directly via the FIFO is often an inefficient way to download data to Gamma. Given that the FIFO can accommodate only a small number of entries, Gamma has to be frequently interrogated to determine how much space is left. Also, consider the situation where a given API function requires a large amount of data to be sent to Gamma. If the FIFO is written directly then a return from this function is not possible until almost all the data has been consumed by Gamma. This may take some time depending on the types of primitives being drawn.

To avoid these problems Gamma provides an on-chip DMA controller which can be used to load data from arbitrary sized (< 16M 32-bit words) host buffers into the FIFO. At chip reset the MasterEnable bit in the *CFGCommand* register must be set to allow DMA to operate (see the Gamma Hardware Reference Manual for further details). Then, for the simplest form of DMA, the host software has to prepare a host buffer containing register address tag descriptions and data values. The host then writes the base address of this buffer to the *DMAAddress* register and the count of the number of words to transfer to the *DMACount* register. Writing to the *DMACount* register starts the DMA transfer and the host can now perform other work. In general, if the complete set of rendering commands required by a given call to a driver function can be loaded into a single DMA buffer, then the driver function can return. Meanwhile, in parallel, Gamma is reading data from the host buffer and processing it. The DMA controller uses it's own private FIFO and FIFO overflow never occurs since the DMA controller automatically waits until there is room in the FIFO before doing any transfers.

### 3.2.1 General DMA Modes

The general input DMA mode of operation is controlled by the Operation field in the PCI *CommandMode* register. The options are Single DMA and Queued DMA.

3.2.1.1 Single DMA. This is the default mode of operation after reset and is identical to the normal GLINT DMA controller mode of operation. The start address for a DMA operation is loaded into the *DMAAddress* PCI register and the length of the DMA transfer in the *DMACount* PCI register. The act of loading the *DMACount* register starts the DMA controller and this register should not be written until the current DMA transfer has finished, however reading it will return the amount of data still left to transfer. The end of a DMA transfer can be signaled by an interrupt. The PCI *DMACount* register is decremented as the DMA proceeds.

The only restriction on the use of DMA control registers is that before attempting to reload the *DMACount* register the host software must wait until the previous DMA has completed. It is valid to load the *DMAAddress* register while the previous DMA is in progress since the address is latched internally at the start of the DMA transfer.

*This mode is just for legacy support and none of the new functionality (queued DMA, hierarchical DMA, output DMA) should be used when in this mode.*

The standard behavior of the DMA controller differs from the earlier DMA controller designs in three small ways:

- All the data associated with a tag must be in the DMA buffer and cannot carry on in the next DMA buffer. This means that any tag processing is reset at the start of a DMA buffer. If insufficient data is found in a DMA buffer to meet the needs of an Indexed, Incrementing or Hold tag then the DMAOverrun error bit is set in *CommandError* and an optional interrupt generated.
- Writing zero to the PCI *DMACount* register during a DMA transfer will no longer cause the current DMA transfer to be aborted. The AbortInputDMA bit in the *CommandMode* register is available for this purpose.
- The **DMACount** register has been expanded from 16 bits to 24 bits in size.

3.2.1.2 Queued DMA. In the Queued DMA operation mode the input FIFO is monitored for **DMAAddress** and **DMACount** tags (they cannot be part of an index or incrementing tag). When these are found the DMA address and count register are loaded. The **DMACount** tag initiates the DMA transfer. The end of each DMA transfer can be signaled by an interrupt, but as this is probably not very useful an interrupt can be generated when the last queued DMA has occurred. This interrupt is generated whenever a **CommandInterrupt** tag is found in the input FIFO so it is software's responsibility to write the **CommandInterrupt** tag into the FIFO following the DMA transfer it is interested in being notified about.

Any normal PCI writes to the *DMAAddress* and *DMACount* PCI registers are ignored while in this mode. Reading the *DMAAddress* registers will return the last data written to the register, reading the *DMACount* will return zero. The *DMACount* register will return zero immediately after reset.

Using DMA leaves the host free to return to the application, while in parallel, Gamma is performing the DMA and drawing. This can increase performance significantly over loading a FIFO directly.

There is enough room in the input FIFO to queue up 16 pending DMA operations.

Host software must generate the correct DMA buffer address for the Gamma DMA controller. Gamma includes address translation hardware so the address may be a logical address, in which case it is translated to the correct physical address via a page mapping table. Alternatively if logical addressing isn't being used the address passed to Gamma must be the physical address of the DMA buffer in host memory and the buffer must also reside at contiguous physical addresses as accessed by Gamma. This is expanded section 3.3.

The *CommandStatus* PCI register holds a bit (CommandDMABusy) which shows when the input DMA controller is busy.

### 3.2.2 DMA Interrupts

Gamma provides interrupt support, as an alternative means of determining when a DMA transfer is complete. If enabled, the interrupt is generated whenever the last data item is transferred from the DMA buffer.

To enable the DMA interrupt, the DMAInterruptEnable bit must be set in the *IntEnable* register. The interrupt handler should check the DMAFlag bit in the *IntFlags* register to determine that a DMA interrupt has actually occurred. To clear the interrupt a word should be written to the *IntFlags* register with the DMAFlag bit set to one.

This works fine for the original single DMA mode of operation, however with queued or hierarchical DMA operations this is not very useful and notification when the last DMA has finished may be more desirable. In this case the **CommandInterrupt** command can be written into the input FIFO or into the DMA buffer itself. This will raise an interrupt (if enabled) when this command is read by the DMA controller.

Interrupts are complicated and depend on the facilities provided by the host operating system, however using interrupts frees the processor for other work while DMA is being completed. Since the overhead of handling an interrupt is often quite high for the host processor, the scheme should be used with care, and probably tuned to the target host and operating system.

## 3.3 Address Mapping

In a system which is running with virtual memory the DMA buffer must be locked down to prevent it being swapped out to disk and hence inaccessible directly across the PCI bus. A common restriction[1] is that the locked pages of the DMA buffer are at physically contiguous addresses, however this can be avoided with the Gamma DMA controller design so simplifying the allocation of the DMA buffer. This is particularly true if the allocation of DMA buffers is done after the system has been up and running for some time as very few pages may be physically contiguous anymore.

When running in this Logical Addressing mode (as set by the LogicalAddress bit in the PCI *CommandMode* register) all addresses generated by the DMA controller (and in fact any of the DMA controllers present in Gamma) are translated into physical addresses before the memory read or write is issued on the primary PCI bus. The address translation, or more accurately the page translation from the logical page number to the physical page number is done by indexing into a page table held in the host's memory. This then returns the corresponding physical page address which is merged with low order address bits of the logical address to give the physical address.

The page size is fixed at 4K bytes so the bottom 12 bits of the logical address are the offset within the page and the upper 20 bits are the logical page number used as the index into the page table. It is these 20 bits which are replaced by the physical page number read from the page table.

[1]Imposed, for example, by the earlier members of the GLINT family.

The page table is physically contiguous in memory and holds one 4 byte entry per mapped logical page. If the page table cannot be held in a single 4K page of memory then two or more *contiguous* pages must be used, however this is not as onerous as it sounds (we are, after all trying to avoid finding contiguous physical pages) as a single 4K page is enough to map in 1K logical pages or 4M bytes.

The address translation involves a table lookup and if this always causes a memory read to occur the overall performance would be very poor. Standard memory management units use a technique called Translation Lookaside Buffers (TLB) to hold a cache of the most recent address translations which are checked first and only if the page translation is not in the TLB will an access to the memory resident page table be made. Gamma also uses this technique and because most accesses are sequential a TLB miss will normally occur once every 1024 reads or writes.

The page table physical address and length are held in the *PageTableAddr* and *PageTableLength* PCI registers respectively. The pagetable length (in multiples of 1K entries) is only used to implement some range checking on the logical addresses and if an error is detected an error flag is set and optionally an interrupt (PageMappingError) generated. The page table starts on a 32 bit word boundary and holds one 32 bit word per logical page. The first logical address is zero.

| Bits | Name | Function |
|---|---|---|
| 0 | PageResident | This bit, when set, marks this page as being resident. When a page is found to be non resident the PageFault interrupt is raised and the faulting logical address made available in one of the *FaultingAddress* PCI registers. This interrupt can be used as a notification that an actual error has occurred, or to implement a dynamic paging mechanism. In the latter case the host will make the page available, update the page table and restart the faulting page again by clearing the interrupt. |
| 1 | ReadAccess | This bit, when set, allows read access to this page. Any attempt to read from this page when the bit is clear will cause one of the PageFaultReadAccess error bits to be set and the CommandError interrupt to be raised. The faulting logical address is made available in the corresponding *FaultingAddress* PCI register. |
| 2 | WriteAccess | This bit, when set, allows write access to this page. Any attempt to write to this page when the bit is clear will cause one of the PageFaultWriteAccess error bits to be set and the CommandError interrupt to be raised. The faulting logical address is made available in the corresponding *FaultingAddress* PCI register. |
| 3…11 | | Not used, but should be set to zero as they are reserved for future use. |
| 12…31 | PhysicalPage | The 20 bit physical page addresses associated with this logical entry. Note that if the page is not resident then these bits are not used by the hardware and may be used by the driver to help manage dynamic paging. |

**Table 3.1. Format for the page table entries**

All the addresses are divided up into one of six regions based on what data was being read or written to that address. Each region has its own set of error bits (in the PCI *CommandError* register), interrupt bits and *FaultingAddress* PCI registers to help identify the cause of the error, etc.

| Region | When used |
|---|---|
| **Vertex** | This region is used for any data read as part of a **RectangleRead** command. |
| Write | This region is used for any data written to memory as part of any Output DMA or **RectangleWrite** command. |

**Table 3.2 The address regions**

The *CommandError* PCI register has error bits associated with the address translation process, see Table 3.3.

| Bit No. | Name | Description |
|---|---|---|
| 4 | PageMappingFaultCommand | This is set whenever the logical address exceeds the. translation range of the Page Mapping Table for the appropriate region. |
| 5 | PageMappingFaultVertex | |
| 6 | Reserved | |
| 7 | Reserved | |
| 8 | PageMappingFaultWrite | |
| 9 | Reserved | |
| 10 | PageFaultReadAccessCommand | This is set whenever a read access is made to a page marked as not supporting read accesses. |
| 11 | PageFaultReadAccessVertex | |
| 12 | Reserved | |
| 13 | Reserved | |
| 14 | PageFaultReadAccessWrite | |
| 15 | Reserved | |
| 16 | PageFaultWriteAccessCommand | This is set whenever a write access is made to a page marked as not supporting write accesses. |
| 17 | PageFaultWriteAccessVertex | |
| 18 | Reserved | |
| 19 | Reserved | |
| 20 | PageFaultWriteAccessWrite | |
| 21 | Reserved | |
| 22 | IllegalDMATag | This is set whenever a DMA related tag is detected not in a tag/data pair. |

**Table 3.3 Error bits associated with the address translation process**

The PCI *CommandInterruptEnable* register has fields to enable and disable the various interrupts related to the address translation process , see Table 3.4. The *CommandInterruptStatus* register has the same fields to identify the actual source of the interrupt. Writing a one to a field in the *CommandInterruptStatus* case clears the corresponding interrupt.

| Bit No. | Name | Description |
|---|---|---|
| 4 | CommandError | Enables the occurrence any of the errors detected (including the PageMapping or PageFault errors above) to generate an interrupt. |
| 8 | PageFaultCommand | Enables an interrupt to be generated when a non resident page is accessed in the Command region. |
| 9 | PageFaultVertex | Enables an interrupt to be generated when a non resident page is accessed in the Vertex region. |
| 10 | Reserved | |
| 11 | Reserved | |
| 12 | PageFaultWrite | Enables an interrupt to be generated when a non resident page is accessed in the Write region. |
| 13 | Reserved | |

**Table 3.4 The PCI** *CommandInterruptEnable* **register fields**

The *FaultingAddress* register corresponding to each region hold the logical address for that region at which the error or page fault occurred.

Writing to the *PageTableAddr* PCI register invalidates all the TLB entries and should be done whenever any of the contents of the current page table change. One page table and DMA buffer could be allocated per context or one shared between all contexts. In the former case the *PageTableAddr* and *PageTableLength* registers would be updated on context switches.

## 3.4 Hierarchical DMA

During a DMA transfer, if the **DMAAddr** and **DMACount** commands are encountered in the DMA buffer then the current DMA is temporarily suspended and a new DMA started with the new address and counts This new DMA can similarly be temporarily suspended and an even newer DMA started. When a 'new' DMA finishes the previously suspended DMA is restarted immediately after the **DMACount** command which initiated the new DMA.

As each DMA is suspended the address and count to restart with are pushed onto a stack (much like a subroutine return stack in a general purpose CPU). This stack, called the dmaStack, has eight entries so the maximum nesting level (or hierarchy depth) is set at eight. Any new DMA which would otherwise cause the stack to overflow is ignored and an error bit (StackOverflowError) in *CommandError* is set and this may cause an interrupt, if so enabled.

Transfer of control to another DMA buffer can also be achieved by using the **DMACall** command (with the new DMA buffer address in the data field). The called DMA buffer has a **DMAReturn** command (the data is not used, but must be provided) to indicate the buffer is finished and to transfer control back to the caller. This mechanism is simpler to use as you do not need to know the length of the called DMA buffer. The **DMAReturn** may cause a stack underflow if the stack is empty. In this case the return is ignored, the DMA aborted and, input processing is passed back to the input FIFO and an error bit (StackUnderError) in *CommandError* is set and this may cause an interrupt, if so enabled.

The motivation for hierarchical DMA is that it allows any number of individual DMAs to be queued up, independent of the input FIFO depth, and it supports hierarchical display lists.

Display lists are preferably held in 'native' Gamma format so they can be executed directly without having to be parsed by software first. An OpenGL display list can call other OpenGL display lists and models are typically built as hierarchies of objects. There are three factors which may prevent hierarchical display lists being used:

- Gamma relies on the host to do any matrix generation and concatenation. Future versions of Gamma will include these operations, so removing this barrier.
- Mode changes in the current rasterizer chips may involve changing one or two bits in a register and leaving the rest of the bits unaffected. This requires a software copy of the register to be used to re-generate the bits to keep. The register contents may be different when the display list is executed from when the display list was created and so active involvement from software is needed. All the Gamma mode registers have an And and an Or version to allow individual bits to be changed. Future rasterizer chips will use this same method so changing their mode registers can be done safely in display lists.
- Inter dependencies in mode bits. For example the framebuffer, in general, needs to be enabled for reading when alpha blending is done, however some alpha blending modes don't use the framebuffer contents. Meanwhile, in a different mode register the logical operation may or may not require framebuffer data to be read as well. Future rasterizer chips will do this multi mode register decoding to enable memory reads and writes as appropriate so these can be included in display lists.

## 3.5 Rectangular DMA

The Rectangle DMA mechanism allows image data to be transferred from host memory to GLINT. The image data may be a sub image of a larger image and have any natural alignment or pixel size. Information regarding the rectangle transfer is held in registers loaded from the input FIFO or a DMA buffer.

The pixel data read from host memory is always packed, however when passed to GLINT it can be in packed or unpacked format.

| Command or register | Use |
|---|---|
| DMARectangleRead | This command initiates the image data transfer. See below for a description of the data field. |
| DMARectangleReadAddress | This register provides the byte address of the first pixel in the image or sub image to read. This is treated as a logical or physical address depending on the LogicalAddressing control bit in the *CommandMode* PCI register. The address should be aligned to the natural size of the pixel, except for 24 bit pixels which may be aligned to any byte boundary. |
| DMARectangleReadLinePitch | This register defines the amount, in bytes, to move from one scanline in the image to the next scanline. For a sub image this is based on width of the whole image. The pitch is held as a 32 bit 2's complement number. This is normally an integer multiple of the number of bytes in a pixel. |
| DMARectangleReadTarget | This register holds the 16 bit tag sent to GLINT just before the image data is sent. Normally it would be one of the tags allowed by the rasterizer during a SyncOnHostData or SyncOnBitMask operation with the tag mode set to Hold. The secondary PCI bus traffic is minimized by sending multiple image words with a single tag (with a count). |

*Note: These tags cannot be part of an indexed group.*

**Table 3.5 Relevant commands and registers for Rectangular DMA**

| Bit No. | Name | Description |
|---|---|---|
| 0…11 | Width | Width of the rectangle in pixels. Range 0…4095. |
| 12…23 | Height | Height of the rectangle in pixels. Range 0…4095. |
| 24, 25 | PixelSize | The size of the pixels in the source image to read. The pixel size is used during alignment and packing. The values are:<br>0 = 8 bits<br>1 = 16 bits<br>2 = 24 bits<br>3 = 32 bits. |
| 26 | PackOut | This field, when set, causes the data to be packed into 32 bit words when sent to GLINT, otherwise the data is right justified and any unused bits (in the most significant end of the word) are set to zero. |
| 27, 28 | ByteSwap | These bits control the byte swapping of the data read from the PCI bus before it is aligned and packed/unpacked. If the input bytes are labeled ABCD on input then they are swapped as follows:<br>0 = ABCD (i.e. no swap)<br>1 = BADC<br>2 = CDAB<br>3 = DCBA. |

**Table 3.6 Fields in the DMARectangleRead command**

The minimum number of PCI reads are used to align and pack the image data.

GLINT is set up to rasterize the destination area for the pixel data (depth, stencil, color, etc.) with SyncOnHostData or SyncOnBitMask enabled in the **Render** command (or equivalent if Gamma is doing the rasterizer set up). This is done before the Rectangular DMA is started.

# 4. Getting Data out of Gamma

The Output DMA Controller provides a means to transfer data from the Host Out FIFO in GLINT to the host's memory. If the data originated from Gamma then it still flows into GLINT so it can be picked up from the Host Out FIFO - this ensures temporal ordering is maintained.

The output DMA is useful for image uploads, returning feedback data, returning select data and Gamma context dumps. The DMA is initiated by commands (and not PCI register writes) so can be queued up in the Input FIFO or in input DMA buffers.

The output DMA runs asynchronously to the input DMA so command and register updates continue to be processed. If a second output DMA is initiated before the first one has finished then all subsequent command and register processing is suspended until the first output DMA has finished.

An interrupt may be generated whenever an output DMA transfer finishes (controlled by a bit in the *CommandMode* PCI register). An interrupt can also be generated selectively by placing a **CommandInterrupt** tag into the FIFO or DMA buffer, however this would cause the interrupt to be generated immediately the input DMA controller found it and not when the output DMA had finished. The **CommandInterrupt** command has a bit reserved to force it to wait for the output DMA to finish before generating the interrupt; a side effect of this is to suspend any subsequent commands and register processing until the output DMA also finishes.

The potential exists in the non feedback case for there to be a mismatch between the amount of data the DMA controller expects to read from the Host Out FIFO and how much data GLINT will provide. In this case a lockup is almost inevitable with either Gamma stalling on the next Output DMA command or GLINT stalling because the Host Out FIFO fills up. In the first case the stuck DMA can be aborted by setting the AbortOutputDMA bit in the PCI *CommandMode* register. The *CommandStatus* PCI register holds a bit which shows when the Output DMA controller is busy.

In a multi-GLINT system the **DMAReadGLINTSource** register is used to select which GLINT is to provide the data. The bottom three bits provide the GLINT ID.

Writing to any of the registers while a DMA is in progress will not disrupt the DMA as a local copy of each register is taken at the start of a DMA. This has been highlighted as the output DMA is an asynchronous operation to normal command and register processing.

The dmaStack is not used by either of the two modes of operation.

## 4.1 Linear DMA Transfers

A simple linear DMA transfer is set up by using the **DMAOutputAddress** register and the **DMAOutputCount** command.

The **DMAOutputAddress** register holds the address (logical or physical) where the stream of 32 bit words is to be written. The start address is given as a byte address but the lower two bits are ignored.

The **DMAOutputCount** command holds the number of 32 bit words to transfer and initiates the actual transfer to start, providing the output DMA controller is idle. The count is held as a 24 bit number.

The data to write to memory is read from the GLINT's Host Out FIFO so the **FilterMode** register (in the GLINT) must be set up to allow the required data and/or tag through.

## 4.2 Feedback and Select DMA Transfers

When the output DMA is used for image uploading or context dumps the amount of data to read from GLINT is deterministic, however returning feedback data presents the host with an interesting problem in that it doesn't know how much feedback data is going to be generated[1]. A similar situation also exists for select data.

The Feedback DMA mechanism allows the collection and transfer of an unspecified amount of data from the Host Out FIFO. This can be used for OpenGL feedback and select modes.

The feedback DMA transfer is set up by using the **DMAOutputAddress** register and the **DMAFeedback** command.

The **DMAOutputAddress** holds the address (logical or physical) where the feedback or select data is to be written The start address is given as a byte address but the lower two bits are ignored.

The **DMAFeedback** command with the length of the memory buffer (in words) is sent to start the Output DMA controller. Data is never written beyond the end of the given buffer length.

Once all the data to write to memory has been generated as the result of regular Gamma commands the **EndOfFeedback** command is sent to Gamma. The Output DMA controller, when it reads this tag from the Host Out FIFO, will terminate the DMA transfer and update the *FeedbackSelectCount* PCI register with the number of words written into memory.

During the transfer one of two situations will be detected:

[1]Feedback mode in OpenGL returns the vertex parameters rather than doing the actual rendering these vertex parameters represent. Vertices are returned after polymode, backface culling and clipping so there is no easy way for the host software to predict (without doing all the geometry work in *exactly* the same order and to the same precision as Gamma). For example a single triangle going in may end up producing 0, (1…13) * 3 vertices out. A clipped triangle could, in the worst case, yield a 15 sided polygon and this will be returned as 13 triangles or 13 * 3 vertices.

- The memory buffer will become full before the **EndOfFeedback** tag in Host Out FIFO is detected. In this case the DMA is terminated and the host informed, however the host output FIFO will continue to be read and its contents discarded until the **EndOfFeedback** tag is found. The **EndOfFeedback** tag and its data will also be discarded.
- The **EndOfFeedback** tag is detected before the memory buffer has become full. In this case the DMA is terminated and the host informed.

The *FeedbackSelectCount* PCI register will hold the actual number of words transferred. Bit 31 is set if more data was found before the **EndOfFeedback** tag.

The GLINT **FilterMode** is set up so that both the tag and data are entered into the FIFO - the Output DMA controller is looking for a specific tag (**EndOfFeedback**). All tags are discarded and never written into the buffer. The data for valid tags is also written into the buffer while data for invalid tags is discarded. Valid tags in this mode are:

**FeedbackX**
**FeedbackY**
**FeedbackZ**
**FeedbackW**
**FeedbackRed**
**FeedbackGreen**
**FeedbackBlue**
**FeedbackAlpha**
**FeedbackS**
**FeedbackT**
**FeedbackR**
**FeedbackQ**
**FeedbackToken**
**PassThrough**
**SelectRecord**
**ContextData**
**EndOfFeedback**

All these tags are part of the previously undocumented Remainder group (bits 14 and 15) in the **FilterMode** and both these bits must be set for Feedback DMA to work. The remaining filter groups do not need to be disabled as they are automatically filtered by the Output DMA controller when in feedback mode.

Note that the feedback mode can also be used for context dumps.

## 4.3 Rectangular DMA Transfers

The Rectangle DMA mechanism allows image data to be transferred from GLINT to host memory. The image data written to memory may be a sub image of a larger image and have any natural alignment or pixel size. Information regarding the rectangle transfer is held in registers loaded from the input FIFO or a DMA buffer.

The pixel data written to host memory is always packed, however when read from GLINT it can be in packed or unpacked format.

| Tag | Use |
|---|---|
| DMARectangleWrite | This command initiates the image data transfer. See below for a description of the data field. |
| DMARectangleWriteAddress | This register provides the byte address of the first pixel in the image or sub image to write. This is treated as a logical or physical address depending on the LogicalAddressing control bit in the *CommandMode* PCI register. The address should be aligned to the natural size of the pixel, except for 24 bit pixels which may be aligned to any byte boundary. |
| DMARectangleWriteLinePitch | This register defines the amount, in bytes, to move from one scanline in the image to the next scanline. For a sub image this is based on width of the whole image. The pitch is held as a 32 bit 2's complement number. This is normally an integer multiple of the number of bytes in a pixel. |

*Note: These tags cannot be part of an indexed group.*

**Table 4.1 Relevant Command and Registers for Rectangular DMA**

The **DMARectangleWrite** command.

| Bit No. | Name | Description |
|---|---|---|
| 0…11 | Width | Width of the rectangle in pixels. Range 0…4095. |
| 12…23 | Height | Height of the rectangle in pixels. Range 0…4095. |
| 24, 25 | PixelSize | The size of the pixels in the source image to read. The pixel size is used during alignment and packing. The values are:<br>0 = 8 bits<br>1 = 16 bits<br>2 = 24 bits<br>3 = 32 bits. |
| 26 | PackIn | This field, when set, indicates the image data from GLINT is packed, otherwise there is one pixel per 32 bits read from GLINT. |
| 27, 28 | ByteSwap | These bits control the byte swapping of the data after it is aligned and packed/unpacked just prior to doing the PCI write. If the input bytes are labeled ABCD on input then they are swapped as follows:<br>0 = ABCD (i.e. no swap)<br>1 = BADC<br>2 = CDAB<br>3 = DCBA. |

**Table 4.2 DMARectangleWrite Command Fields**

The minimum number of PCI writes are used to align and pack the image data.

GLINT is set up to rasterize the source area for the pixel data (depth, stencil, color, etc.) with the **FilterMode** set up to allow the appropriate data through (the tag should not be included). The rasterization is best set up before the Rectangular DMA is started, but as this is asynchronous it is not necessary to do things in this order.

# 5. Driver Support

## 5.1 Timer

It is frequently useful to be able generate an interrupt after a certain amount of time has elapsed, maybe to wake up the driver in the absence of regular DMA interrupts, for example, or as a watchdog timer to catch system lockups. Most operating systems provide some way of scheduling a timer interrupt to occur, however the call to the operating system may be prohibitively expensive. Gamma includes a one shot delay timer which can be programmed directly by a driver without any involvement from the operating system.

Writing to the *DelayTimer* PCI register starts the timer. When the timer decrements down to one an interrupt is generated and the timer stops. Writing a zero to *DelayTimer* aborts the timer with no interrupt being generated (unless it has already occurred). Reading this register returns the current timer value. The unit of time is PCI clock / 64, or approximately 2µs for a 33MHz PCI clock and the register is 24 bits wide giving a maximum delay of approximately 32 seconds.

## 5.2 Errors, Interrupts and Status registers

| Bit No. | Name | Description |
|---|---|---|
| 0, 1 | Operation | This field determines the major operation mode of the DMA controller. It has the values:<br>0: Default operation enabled after reset. The DMA is initiated by writes to the PCI *DMAAddress* and *DMACount* registers. Any **DMAAddr** and **DMACount** tags found in the input FIFO are discarded.<br>1: The DMA is initiated by the **DMAAddr** and **DMACount** tags in the input FIFO. Writes to the PCI *DMAAddress* and *DMACount* registers are ignored. |
| 2 | LogicalAddressing | When set causes the addresses generated by the DMA controller to be translated into physical addresses via a page table. |
| 3 | AbortOutputDMA | When this bit is set any current (or future) Output DMA is aborted (linear or rectangular). |
| 6 | AbortInputDMA | When this bit is set any current (or future) Input DMA is aborted (normal DMA, hierarchical or rectangular). |

**Table 5.1 PCI *CommandMode* Register Fields**

The PCI *CommandError* register holds the Gamma core error bits, see Table 5.2.

| Bit No. | Name | Description |
|---|---|---|
| 0 | StackUnderflow | This is set whenever a **DMAReturn** is attempted from the InputFIFO or DMA buffer when the stack is empty. |
| 1 | StackOverflow | This is set whenever a **DMACount** or **DMACall** tag in a DMA buffer are nested more than 8 deep. |
| 2 | DMAOverrun | This is set whenever data beyond the end of the DMA buffer is needed to fulfill the requirements of the last tag in the DMA buffer. |
| 4 | PageMappingFaultCommand | the translation range of the Page Mapping Table |
| 8 | PageMappingFaultWrite | for the appropriate region. |
| 10 | PageFaultReadAccessCommand | This is set whenever a read access is made to a |
| 14 | PageFaultReadAccessWrite | page marked as not supporting read accesses |
| 16 | PageFaultWriteAccessCommand | This is set whenever a write access is made to a |
| 20 | PageFaultWriteAccessWrite | page marked as not supporting write accesses. |
| 22 | IllegalDMATag | This is set whenever a DMA related tag is detected not in a tag/data pair. |

**Table 5.2 PCI *CommandError* Register Error Conditions**

The PCI *CommandInterruptEnable* register has fields to enable and disable the various interrupt sources see Table 5.3. The *CommandInterruptStatus* register has the same fields to identify the actual source of the interrupt. Writing a one to a field in *CommandInterruptStatus* clears the corresponding interrupt.

| Bit No. | Name | Description |
|---|---|---|
| 0 | FIFOQueuedCommandDMA | This field determines when interrupts are generated during the Queued DMA mode of operation. When this bit is set the interrupts occur after every DMA has finished. |
| 1 | OutputDMA | When set enables an interrupt to be generated whenever the Output DMA controller finishes. |
| 2 | Command | Enables the *CommandInterrupt* tag to generate an interrupt. |
| 3 | Timer | Enables the DelayTimer to generate an interrupt when it has counted down to one. |
| 4 | CommandError | Enables the occurrence of any of the errors shown by the *CommandError* register to generate an interrupt. |
| 8 | PageFaultCommand | Enables an interrupt to be generated when a non resident page is accessed in the Command region. |
| 9 | PageFaultVertex | Enables an interrupt to be generated when a non resident page is accessed in the Vertex region. |
| 12 | PageFaultWrite | Enables an interrupt to be generated when a non resident page is accessed in the Write region. |

**Table 5.3 PCI *CommandInterruptEnable* Interrupt Source Fields**

| Bit No. | Name | Description |
|---|---|---|
| 0 | CommandDMABusy | This is set whenever the input DMA controller is busy or a RectangleRead is in progress. |
| 1 | OutputDMABusy | This is set whenever the OutputDMA controller is busy or a RectangleWrite is in progress. |
| 2 | FIFOEmpty | This is set whenever the input FIFO is empty. |

**Table 5.4 PCI *CommandStatus* Register Fields**

# 6. Primitive Assembly

This section looks at how primitives and vertex data are given to Gamma.

## 6.1 Specifying Vertices, Normals, Textures and Colors

The normal input format for the data associated with a vertex is IEEE single precision floating point numbers. All the variants supported by OpenGL such as byte, long, double, etc. must be converted to single precision floats first. The only exceptions to this are colors which can be accepted as unsigned packed bytes.

The vertex parameters are multi valued entities (i.e. vectors) so the vectors are assembled before being processed as they can only be written one component at a time. Frequently one or more of the vector components are left to take a default value, for example a vertex coordinate is commonly specified with a w component set to 1.0. Allowing this component to take its default value saves sending 4 bytes to Gamma which can amount to a substantial saving in bandwidth.

The vector assembly uses the x component (for vertices and normals), the red component (for colors) and the s component (for textures) to indicate the assembled vector is complete and should be processed. The address (or tag) of the trigger component used informs Gamma which of the short hand formats to use. Several short hand formats exist to reduce the amount of information the host needs to provide. The possible formats are:

| | | |
|---|---|---|
| Vertex: | xy, xyz and xyzw | z = 0, w = 1 if missing |
| Raster Position: | xy, xyz and xyzw | z = 0, w = 1 if missing |
| Normal: | xyz | |
| FaceNormal | xyz | |
| Color: | rgb, rgba | a = 1 if missing |
| Texture: | s, st, strq | t = 0, r = 0, q = 1 if missing |

Each short hand format has a unique register for the trigger component (the number, when present, indicates the number of components supplied):

| **Parameter** | **Non-Trigger registers** | **Trigger registers** |
|---|---|---|
| Vertex: | **Vy**, **Vz**, **Vw** | **Vx2**, **Vx3**, **Vx4** |
| Raster Position: | **RPy**, **RPz**, **RPw** | **RPx2**, **RPx3**, **RPx4** |
| Normal: | **Ny**, **Nz** | **Nx** |
| FaceNormal | **FNy**, **FNz** | **FNx** |
| Color: | **Cg**, **Cb**, **Ca** | **Cr3**, **Cr4** |
| Texture: | **Tt**, **Tr**, **Tq** | **Ts1**, **Ts2**, **Ts4** |

The components for a vector are sent in reverse order to ensure the trigger component is always last. The tags are organized so this naturally fits in to the index tag format. The main group for the vertex parameters allows any of the vertex formats, a normal, any of the color formats and the **Ts2** texture format to be encoded into one group. This allows a Gouraud shaded, depth buffered meshed triangle to be described by 7 words (index tag, **Nz**, **Ny**, **Nx**, **Vz**, **Vy**, **Vx3**)[1].

The storage used to assemble a vector is shared by all vectors so it is not possible to mix components from different vectors. Sending all the components associated with a vector together is not an onerous restriction.

Some general rules when sending vertex parameters are:

- Don't mix Vertex, Color, Normal, Face Normal, Raster Position or Texture components up. All the components for a particular vector type must be sent together.
- The components must be sent in reverse order from the 'natural' way. For example send W, then Z, then Y and finally X (to the appropriate register).
- All components are single precision IEEE floating point numbers.
- Short hand formats will fill in some components for you to save having to send W =1, for example. These are analogous to the OpenGL glVertex[2|3|4]f, glColor[3|4]f and glTexCoord[1|2|4]f function calls.
- Only send the vertex coordinate after any color, normal or texture data has been sent. A vertex must be sent to provoke the triangle, line, etc. to be drawn The color, normal and texture update the current color, normal and texture values respectively and are applied to vertices until subsequently changed. This follows the usual OpenGL rules.

These rules are enforced automatically if the parameters are sent as a group using an index tag.

The above registers, except the vertex can be read back. All the trigger registers for a parameter read back the same register but the non trigger registers will not read back the current values until the trigger has been written. Non trigger registers will return the default values if the last trigger register substituted the defaults for absent values.

In addition to defining color information via the **Ca**, **Cb**, **Cg** **Cr3** and **Cr4**]registers two packed color formats are also available. These are **PackedColor3** and **PackedColor4**. The format of each color component is an unsigned 8 bit number with red in the lower byte, then green, then blue and finally alpha in the upper byte. In the **PackedColor3** case the alpha byte is ignored and always interpreted as 255.

[1]This compares very favorably with the 10 words needed by GLINT Delta for the same triangle (index tag, X, Y, Z, R, G, B, A, DrawTag, DrawData).

The current edge flag is updated by writing to the **EdgeFlag** register. The current edge flag is used for edges in polygons, independent triangles and independent quads to determine if they should be drawn when the polygon mode is set to Lines. Connected primitives always have their edges drawn. Interior edges introduced as part of the rendering process (for example decomposing a polygon in to several triangles) are never drawn.

## 6.2 Begin/End Paradigm

The main method in OpenGL for issuing primitives is the Begin/End paradigm and Gamma supports this exactly with no additional control by software. The usual OpenGL commands can be inserted between the **Begin** and **End** commands (in fact Gamma imposes no restrictions on the mix of mode changes between a **Begin** and **End** command) and that is all there is to it. There is no need to manage the vertex store as there was for GLINT Delta.

The **Begin** command starts a sequence of vertices by specifying the primitive type and various enables. The primitive type is held in the upper 4 bits while the lower bits hold the same fields used by the **Render** command in GLINT and allows fine control of the texture mapping, fog, etc. The format of the **Begin** command's data field is shown in Table 6.2. The fields in bits 0 to 18 are the same as those in the **Render** command and only the fields which influence Gamma are identified and described. The effect of these fields on the GLINT rendering device can be found in the appropriate GLINT Programmer's Reference Manual.

| Bit No. | Name | Description |
|---|---|---|
| 0 | AreaStippleEnable | Overridden when incompatible with the primitive type. |
| 1 | LineStippleEnable | Overridden when incompatible with the primitive type. |
| 2 | ResetLineStipple | Ignored and always set by Gamma to meet the OpenGL rules for when the line stipple is reset within primitives. |
| 3 | FastFillEnable | Ignored and forced to be disabled. |
| 4, 5 | Not used | |
| 6, 7 | PrimitiveType | Ignored and always set by Gamma depending on the type field and polymode setting. |
| 8 | AntialiasEnable | Qualifies the AntialiasEnable held for each primitive type in the **PointMode**, **LineMode** and **TriangleMode** registers. If both enables are true then the primitive is antialiased. |
| 9 | AntialiasingQuality | Ignored. This information is held in the **PointMode**, **LineMode** and **TriangleMode** registers. |
| 10 | UsePointTable | Ignored and generated locally when antialiasing points. |
| 11 | SyncOnBitMask | Ignored, and forced to be disabled. |
| 12 | SyncOnHostData | Ignored, and forced to be disabled. |
| 13 | TextureEnable | Passed through but also enables (1) or disables (0) texture calculation to be performed. Note texture transformations and TexGen operations are not influenced by this bit (as required by OpenGL). |
| 14 | FogEnable | Passed through but also enables (1) or disables (0) fog calculation to be performed. |
| 15 | CoverageEnable | Ignored and generated locally when antialiasing is done. |
| 16 | SubPixelCorrectionEnable | This bit is passed through. When enabled (1) subpixel correction is done in the Y direction (the rasterizer does it in the X direction). |
| 17 | | Reserved |
| 18 | SpanOperation | Ignored and forced to be 0. |
| 28…31 | Type | This field sets up the primitive type to process on receiving each new vertex. It has the following values:<br>0 Null<br>1 Points<br>2 Lines<br>3 LineLoop<br>4 LineStrip<br>5 Triangles<br>6 TriangleStrip<br>7 TriangleFan<br>8 Quads<br>9 QuadStrip<br>10 Polygon |

**Table 6.1 Begin/End Paradigm**

Raster position registers should not be loaded within a Begin/End sequence (also specified by OpenGL) as it uses the same temporary working store and would disrupt the Begin/End sequencing.

The **End** command ensures any remaining operations for the primitive are complete.

Vertices sent before a **Begin** command and after an **End** command are ignored. Texture, color, normal and face normals will update the corresponding current values.

# 7. 3D Pipeline

The 3D pipeline defines the normal sequence of operations applied to vertices and primitives and is well covered in the computer graphics literature.

Each of these stages will be described briefly to introduce the register used to control them.

## 7.1 Transformation

Vertices, normals, face normals and texture coordinates are transformed before they are used to define the primitives to ultimately draw. The **TransformMode** and **NormalizeMode** registers define what is to be transformed and how.

### 7.1.1 Vertices

The most involved set of operations are done on vertex coordinates and these pass through the following coordinate systems (in order):

Object Coordinates — This is the coordinate space (suffix o) objects are defined in and what is usually presented to the input of the 3D pipeline. The ModelView matrix converts the object coordinates into eye coordinates (suffix e). The coordinates are four dimensional so the ModelView matrix, M, is 4x4.

$$\begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix} = \mathbf{M} \begin{pmatrix} x_o \\ y_o \\ z_o \\ w_o \end{pmatrix}$$

Eye Coordinates — This is the coordinate space where lighting, fog, etc. operations occur. The Projection matrix converts the eye coordinates into clip coordinates (suffix c). The coordinates are four dimensional so the Projection matrix, P, is 4x4.

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \mathbf{P} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

Clip Coordinates — This coordinate space is arranged to make clipping easier. The perspective division operation converts the clip coordinates (after any necessary clipping has been done) into normalized device coordinates (suffix d).

$$\begin{pmatrix} x_d \\ y_d \\ z_d \end{pmatrix} = \begin{pmatrix} \frac{x_c}{w_c} \\ \frac{y_c}{w_c} \\ \frac{z_c}{w_c} \end{pmatrix}$$

| | |
|---|---|
| Normalized Device Coordinates | This coordinate space defines object to be drawn (or the visible parts, if clipped), but the coordinates are normalized to be in the range ± 1.0. |
| Window Coordinates | The window coordinates (suffix w) are the normalized device coordinates, but now scaled to the size of the window. This scaling is done by the view port mapping defined by the S and O vectors. The x and y components of S and O are related to the window's size and the z components to the depth range. |

$$\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \begin{pmatrix} s_x x_d + o_x \\ s_y y_d + o_y \\ s_z z_d + o_z \end{pmatrix}$$

Gamma holds two matrices used for vertex transformation. The first matrix is the ModelView matrix and this is stored in the **ViewModelMatrix[0…15]** registers. The second matrix is the combined ModelView and Projection matrix (i.e. **M**x**P** matrices from above) and this is stored in the **ModelViewProjectionMatrix[0…15]** registers. Both these matrices are 4x4 and are laid out in the registers as follows:

$$\begin{pmatrix} M_0 & M_4 & M_8 & M_{12} \\ M_1 & M_5 & M_9 & M_{13} \\ M_2 & M_6 & M_{10} & M_{14} \\ M_3 & M_7 & M_{11} & M_{15} \end{pmatrix}$$

The numerical subscripts give the order the elements are stored in the matrix registers (i.e. $M_0$ is stored in **ModelViewMatrix[0]**, for example) and these follow the column-major order convention. Note this is different from the convention C uses which follows the row-major order.

The viewport mapping scale and offset values are held in the **ViewPortScaleX**, **ViewPortScaleY**, **ViewPortScaleZ**, **ViewPortOffsetX**, **ViewPortOffsetY** and **ViewPortOffsetZ** registers.

The transformation of the vertices is controlled by two bits in the **TransformMode** register see Table 7.1.

| Bit No. | Name | Description |
|---|---|---|
| 0 | UseModelViewMatrix | When set causes the incoming vertex to be multiplied by the ModelView matrix. This is only necessary if the vertex in eye space is needed for subsequent processing. A slight gain in performance may be seen when this transformation is disabled. The eye space vertex is used for EyeLinear, TexGen, user clipping planes, fog, lighting, or auto generation of the face normal. |
| 1 | UseModelViewProjectionMatrix | When set causes the incoming vertex to be multiplied by the ModelViewProjection matrix to calculate coordinates in clip space. This bit should normally be set. |

**Table 7.1 TransformMode Register Vertices Control Bits**

### 7.1.2 Normal

The normals are defined in object space and transformed into eye space where the lighting is calculated, or face culling using a supplied face normal occurs. The normal is transformed by the 3x3 Normal matrix held in the **NormalMatrix[0…8]** registers. The Normal matrix is usually the inverse transpose of the upper 3x3 part of the ModelView matrix. The Normal matrix is laid out in the registers as follows:

Similarly for a 3x3 matrix multiplication:

$$\begin{pmatrix} N_0 & N_3 & N_6 \\ N_1 & N_4 & N_7 \\ N_2 & N_5 & N_8 \end{pmatrix}$$

The numerical subscripts give the order the elements are stored in the matrix registers (i.e. $N_0$ is stored in **NormalMatrix[0]**, for example) and these follow the column-major order convention. Note this is different from the convention C uses which follows the row-major order.

The transformation of the normals is controlled by two bits in the **TransformMode** register see Table 7.2

| **Bit No.** | **Name** | **Description** |
|---|---|---|
| 2 | TransformNormal | When set causes any incoming vertex normal to be multiplied by the Normal matrix. This only needs to be set if lighting is used or TexGen SphereMap is selected. |
| 3 | TransformFaceNormal | When set causes any incoming face normal to be multiplied by the Normal matrix. This only needs to be set if face normal lighting is used and/or if face normal backface test is enabled. OpenGL does not use this, whereas QuickDraw3D does for TriMesh primitives. |

**Table 7.2 TransformMode Register Normal Control Bits**

If the incoming normal was not of unit length or the Normal matrix scaled the normal in any way then the transformed normal needs to be normalized for the lighting calculation to work properly. Two bits in the **NormalizeMode** register control this see Table 7.3.

| Bit No. | Name | Description |
|---|---|---|
| 0 | NormalEnable | When set causes any normals to be normalized. |
| 1 | FaceNormalEnable | When set causes any face normals supplied by the user to be normalized. If the face normal is only being used for culling then it never needs to be normalized. |

**Table 7.3 NormalizeMode Register Control Bits**

Enabling normalization does not effect performance in Gamma.

### 7.1.3 Texture

The incoming texture coordinates, or the texture coordinates generated under TexGen control (see later) are transformed by the 4x4 Texture matrix held in the **TextureMatrix[0…15]** registers. This matrix is laid out in the registers as follows:

$$\begin{pmatrix} T_0 & T_4 & T_8 & T_{12} \\ T_1 & T_5 & T_9 & T_{13} \\ T_2 & T_6 & T_{10} & T_{14} \\ T_3 & T_7 & T_{11} & T_{15} \end{pmatrix}$$

The numerical subscripts give the order the elements are stored in the matrix registers (i.e. $T_0$ is stored in **TextureMatrix[0]**, for example) and these follow the column-major order convention. Note this is different from the convention C uses which follows the row-major order.

The transformation of the texture is controlled by one bit in the **TransformMode** register see Table 7.4.

| Bit No. | Name | Description |
|---|---|---|
| 4 | TransformTexture | When set causes the incoming texture or the texture generated from the TexGen operation to be multiplied by the Texture matrix. Frequently the texture matrix will be a unit matrix so the transformation can be preferably avoided. |

**Table 7.4 TransformMode Register Texture Control Bits**

## 7.2 Lighting

Gamma implements the full OpenGL RGB lighting model. The Color Index lighting model is not supported. The ambient, diffuse and specular lighting components are given by the following vector equations (vectors in bold). The subscripts used are

| | |
|---|---|
| *c* | color |
| *p* | position |
| *m* | material |
| *l* | light |
| *i* | light number |

$$\mathbf{lightAmbient} = \sum_{i=0}^{n-1} (atti_i)(spot_i)\mathbf{a}_{cli}$$

$$\mathbf{lightDiffuse} = \sum_{i=0}^{n-1} (atti_i)(spot_i)(\mathbf{n} \bullet \vec{\mathbf{VP}}_{pli})d_{cli}$$

$$\mathbf{lightSpecular} = \sum_{i=0}^{n-1} (atti_i)(spot_i)\left[(f_i)(\mathbf{n} \bullet \hat{\mathbf{h}}_i)^{s_{cm}} \mathbf{s}_{cli}\right]$$

where:

$$atti_i = \begin{cases} \dfrac{1}{k_{0i} + k_{1i}\left\|\mathbf{VP}_{pli}\right\| + k_{2i}\left\|\mathbf{VP}_{pli}\right\|^2}, & \text{if mode.Attenuation is true,} \\ 1.0, & \text{otherwise.} \end{cases}$$

where $k_{0i}$, $k_{1i}$ and $k_{2i}$ are the attenuation coefficients for light $i$.

$$spot_i = \begin{cases} \left(\vec{\mathbf{VP}}_{pli} \bullet \hat{s}_{dli}\right)^{s_{rli}}, & \text{mode.Spotlight is true,} \quad \vec{\mathbf{VP}}_{pli} \bullet \hat{s}_{dli} \geq \cos(c_{rli}), \\ 0.0, & \text{mode.Spotlight is true,} \quad \vec{\mathbf{VP}}_{pli} \bullet \hat{s}_{dli} < \cos(c_{rli}), \\ 1.0, & \text{mode.Spotlight is false.} \end{cases}$$

where $c_{rli}$ is the spotlight cutoff angle for light $i$.

$$f_i = \begin{cases} 1, & \mathbf{n} \bullet \vec{\mathbf{VP}}_{pli} \neq 0, \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathbf{h}_i = \begin{cases} \vec{\mathbf{VP}}_{pli} + \vec{\mathbf{VP}}_e, & \text{local viewer true} \\ \vec{\mathbf{VP}}_{pli} + \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T, & \text{local viewer false} \end{cases}$$

The light and material properties are combined as shown in the following vector equations to calculate the color, diffuse texture (Kd) and specular texture (Ks) values:

$$\mathbf{c} = \mathbf{e}_{cm} + \mathbf{a}_{cm} * \mathbf{a}_{cs} + \mathbf{ambientLight} * \mathbf{a}_{cm} + \mathbf{diffuseLight} * \mathbf{d}_{cm} + \mathbf{specularLight} * \mathbf{s}_{cm}$$

$$\mathbf{diffuseTexture} = \mathbf{e}_{cm} + \mathbf{a}_{cm} + \mathbf{ambientLight} + \mathbf{diffuseLight}$$

$$\mathbf{specularTexture} = \mathbf{specularLight}$$

| | | |
|---|---|---|
| where: | $\mathbf{e}_{\mathrm{cm}}$ | is the emissive material color (front or back) |
| | $\mathbf{a}_{\mathrm{cm}}$ | is the ambient material color (front or back) |
| | $\mathbf{d}_{\mathrm{cm}}$ | is the diffuse material color (front or back) |
| | $\mathbf{s}_{\mathrm{cm}}$ | is the specular material color (front or back) |

The OpenGL specification gives an authoritative description of these equations.

The diffuseTexture and specularTexture are not required for OpenGL, but are used in Direct3D and Quickdraw3D to add lighting effects after texture mapping.

Gamma supports 16 lights (numbered 0 through 15). The registers for light 0 are shown in Table 7.5. For other lights substitute the appropriate number for 0. Note the successive light parameters follow in sequential registers so it is easy to generate the tag number or register address algorithmically rather than always using symbolic tokens.

| **Register** | **Equation symbol** | **Offset from LightMode** | **Description** |
|---|---|---|---|
| Light0Mode | | 0 | Mode control for Light (see later). |
| Light0AmbientIntensityRed | $a_{cli}$ | 1 | Ambient red intensity in floating point format. |
| Light0AmbientIntensityGreen | | 2 | Ambient green intensity in floating point format. |
| Light0AmbientIntensityBlue | | 3 | Ambient blue intensity in floating point format. |
| Light0DiffuseIntensityRed | $d_{cli}$ | 4 | Diffuse red intensity in floating point format. |
| Light0DiffuseIntensityGreen | | 5 | Diffuse green intensity in floating point format. |
| Light0DiffuseIntensityBlue | | 6 | Diffuse blue intensity in floating point format. |
| Light0SpecularIntensityRed | $s_{cli}$ | 7 | Specular red intensity in floating point format. |
| Light0SpecularIntensityGreen | | 8 | Specular green intensity in floating point format. |
| Light0SpecularIntensityBlue | | 9 | Specular blue intensity in floating point format. |
| Light0PositionX | $\mathbf{P}_{pli}$ | 10 | X position of the light if PositionW = 0, otherwise it is the normalized X direction. |
| Light0PositionY | | 11 | Y position of the light if PositionW = 0, otherwise it is the normalized direction. |
| Light0PositionZ | | 12 | Z position of the light if PositionW = 0, otherwise it is the normalized Z direction. |
| Light0PositionW | | 13 | W position of the light. When zero, it changes the meaning of the Position* values to be directions. |
| Light0SpotlightDirectionX | $s_{dli}$ | 14 | Normalized X component for the spotlight direction or the normalized X component of the half vector when the light is not a spotlight. |
| Light0SpotlightDirectionY | | 15 | Normalized Y component for the spotlight direction or the normalized Y component of the half vector when the light is not a spotlight. |
| Light0SpotlightDirectionZ | | 16 | Normalized Z component for the spotlight direction or the normalized Z component of the half vector when the light is not a spotlight. |
| Light0SpotlightExponent | $s_{rli}$ | 17 | Spotlight exponent . This is held as an unsigned 7.4 fixed point number. |
| Light0CosSpotlightCutoffAngle | $c_{rli}$ | 18 | Cosine of the spotlight cut-off angle. Its range is 0.0 to 1.0 inclusive. |
| Light0ConstantAttenuation | $k_{0i}$ | 19 | Constant attenuation factor. |
| Light0LinearAttenuation | $k_{1i}$ | 20 | Linear attenuation factor. |
| Light0QuadraticAttenuation | $k_{2i}$ | 21 | Quadratic attenuation factor. |

**Table 7.5 Light 0 Registers**

Each light has a mode word associated with it and the definition of its control bits are shown in Table 7.6 (these flags are used to optimize the calculations):

| Bit No. | Name | Description |
|---|---|---|
| 0 | LightOn | When set indicates the light is on and contributes illumination to the scene, otherwise it does not. |
| 1 | Spotlight | When set indicates the light is a spotlight. If it is not set then the light is not a spot light and the SpotlightDirection is used to hold the normalized half vector between the viewer and the light. |
| 2 | Attenuation | When set indicates the light is to be attenuated, otherwise no attenuation is done. |
| 3 | LocalLight | When set indicates the light is local and the full lighting equations should be used. This allows a light to be local without it having to be a spotlight or have any attenuation applied to it. |

**Table 7.6 Light Mode Word Control Bits Definitions**

Most of the lighting parameters should be self explanatory, however there are a few which will be clarified:

- The light's position and direction vectors are not transformed automatically when loaded into Gamma and any transformation required to locate them in eye space (the coordinate system where lighting is done, i.e. the view vector is always (0, 0, 1)) should be done in software before they are loaded.
- All direction vectors (i.e. Position vector when the W component is zero, SpotlightDirection and the half vector) must all be normalized.
- The Light mode bits should be set up as necessary. Any special values assumed by OpenGL (for example the spotlight cut-off angle of 180°) are ignored. The preferable state of each bit is zero to avoid some or all of the lighting calculation and the bits are generally set up as shown in Table 7.7 (to follow OpenGL conventions for identifying some light mode or operation that is not required).

| Bit No. | Name | Description |
|---|---|---|
| 1 | Spotlight | Set when the spotlight cut off angle is not 180 degrees. |
| 2 | Attenuation | Set when the light's position has a non zero W (this indicates the position holds a position and not a direction) and the sum of the attenuation factors is not unity are not 1, 0, 0 for constant, linear and quadratic respectively. |
| 3 | LocalLight | Set when the light's position has a non zero W (this indicates the position holds a position and not a direction). |

**Table 7.7 Lighting Calculation Bit Set up**

- When the Light mode indicates the light is not a spotlight the SpotlightDirection vector used to hold the normalized half vector. The half vector between the light's position vector and the eye vector is given by the following equation:

$$\mathbf{h} = \hat{\mathbf{P}}_{pli} + \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$$

- The nominal range for the light's intensity colors (and the material colors) is 0.0…1.0 inclusive. Values outside this range (and even negative ones) can be used and the final vertex colors are clamped (if enabled) to be in the range 0.0…1.0 during the primitive's set up calculations prior to rasterization.

The material parameters are held in the registers shown in Table 7.8. There are two identical sets to hold the front and back materials used during two sided lighting.

| Register | **Equation symbol** |
|---|---|
| FrontEmissiveColorRed | $e_{cm}$ |
| FrontEmissiveColorGreen | |
| FrontEmissiveColorBlue | |
| FrontAmbientColorRed | $a_{cm}$ |
| FrontAmbientColorGreen | |
| FrontAmbientColorBlue | |
| FrontDiffuseColorRed | $d_{cm}$ |
| FrontDiffuseColorGreen | |
| FrontDiffuseColorBlue | |
| FrontAlpha | |
| FrontSpecularColorRed | $s_{cm}$ |
| FrontSpecularColorGreen | |
| FrontSpecularColorBlue | |
| FrontSpecularExponent | $s_{rm}$ |
| BackEmissiveColorRed | $e_{cm}$ |
| BackEmissiveColorGreen | |
| BackEmissiveColorBlue | |
| BackAmbientColorRed | $a_{cm}$ |
| BackAmbientColorGreen | |
| BackAmbientColorBlue | |
| BackDiffuseColorRed | $d_{cm}$ |
| BackDiffuseColorGreen | |
| BackDiffuseColorBlue | |
| BackAlpha | |
| BackSpecularColorRed | $s_{cm}$ |
| BackSpecularColorGreen | |
| BackSpecularColorBlue | |
| BackSpecularExponent | $s_{rm}$ |
| SceneAmbientColorRed | $acs$ |
| SceneAmbientColorGreen | |
| SceneAmbientColorBlue | |

**Table 7.8 Material Parameters Registers**

The OpenGL specification associates an alpha value with each lighting term, however only the diffuse alpha is used. The diffuse alpha value is held in the **FrontAlpha** or **BackAlpha** registers.

The lighting calculations are very expensive and can easily dominate the overall performance so it is very worth while to check the light's parameters to see if a simpler form of the lighting equation can be used (by setting the appropriate bits in the **LightnMode** register) For example if the Constant Attenuation is set to 1.0 and both the Linear and Quadratic Attenuation factors are set to 0.0 then the attenuation part of the calculation can be avoided.

In a similar line if the product of the attenuation and spot values for a light falls below the value given in the **AttenuationCutOff** register then (if suitably enabled) the calculations for this light will be automatically terminated. This optimization allows lights which are becoming too faint to contribute to be terminated early. A suitable value is given by: 1.0/512*n* where *n* is the number of lights. The 512 constant was chosen as it is less than the smallest representable color when converted to a byte integer assuming the light and material colors are restricted to the range 0.0…1.0.

The lighting calculations are controlled by the **LightingMode** and the **MaterialMode** register. The **LightingMode** register fields are shown in Table 7.9.

| Bit No. | Name | Description |
|---|---|---|
| 0 | Enable | When set causes the vertex to be lit using the lighting equations, otherwise the current color is assigned. |
| 1, 2 | TwoSidedLighting | The three options are:<br>0 Use the front side material.<br>1 Use the back side material and invert the normal before it is used in the lighting calculations.<br>2 Use the orientation of the face to select between front or back materials and lighting. The orientation is determined by fields in the **GeometryMode** register. |
| 3 | LocalViewer | When set causes the viewer to be at (0, 0, 1) in eye coordinates, otherwise the viewer is at (0, 0, ). When the viewer is at infinity some of the lighting equations can be simplified so run faster, however the position of the specular highlights are not as correct. |
| 4 | FlipNormal | When set causes the absolute value of the lighting dot products to be taken, otherwise negative dot products are clamped to zero. Clamping is used for OpenGL, but some other APIs allow the normal to be flipped - this gives a cheap form of two sided lighting and is useful when the normals are not consistently facing 'outwards' in the model or scene. |
| 5 | AttenuationTest | When set forces the lighting calculation for the current light to be aborted when the product of *atti* and *spot* (in the lighting equations) for the light falls below the threshold given in the **AttenuationCutOff** register. |
| 6…14 | NumberLights | This 9 bit field holds the number of lights to use. Its legal range is 0…16 inclusive. Numbers greater than 16 are clamped to be 16. |
| 15 | SpecularLightingEnable | When this bit is set the specular part of the lighting calculations are done, otherwise they are skipped. This bit would normally be set for OpenGL, however some APIs allow non specular lighting models to be used. |
| 16 | UseFaceNormal | When this bit is set the face normal is used instead of the vertex normals. The lighting is still evaluated once per vertex so any position dependent effects (i.e. attenuation or spotlight) are still computed correctly. |

**Table 7.9 LightingMode Register Fields**

| Bit No. | Name | Description |
|---|---|---|
| 0 | Enable | When set causes the vertex to be calculated from the lighting equations otherwise the current color is assigned. |
| 1 | DiffuseTextureEnable | When set allows the diffuse texture color to be calculated and sent to GLINT. This is further qualified by the TextureEnable bit in the **Begin** command so is only done when texture mapping is enabled. |
| 2 | SpecularTextureEnable | When set allows the specular texture color to be calculated and sent to GLINT. This is further qualified by the TextureEnable bit in the **Begin** command so is only done when texture mapping is enabled. |
| 3 | MonochromeDiffuseTexture | When set the diffuse texture color is converted to a monochrome value before it is sent to GLINT. This allows the diffuse texture DDA in GLINT 500TX to be set up.<br>When clear the true color value is sent and is used when the target rendering chip is GLINT MX as it has true color diffuse texture DDAs. |
| 4 | MonochromeSpecularTexture | When set the specular texture color is converted to a monochrome value before it is sent to GLINT. This allows the specular texture DDA in GLINT 500TX to be set up.<br>When clear the true color value is sent and is used when the target rendering chip is GLINT MX as it has true color specular texture DDAs. |
| 5 | PremultiplyAlpha | When set premultiplies the diffuse and ambient colors by the material alpha value. |
| 6 | ColorSource | This field selects where the color should be taken from when the Enable field is 0. The options are:<br>0: Current color value.<br>1: Diffuse material value. |
| 7, 8 | TwoSidedLighting | The three options are:<br>0 Use the front side materials.<br>1 Use the back side materials.<br>2 Use the orientation of the face to select between front or back materials and lighting. |

**Table 7.10 MaterialMode Register Fields**

The selection between Gouraud and flat shading is controlled by the FlatShading bit in **GeometryMode**. The SmoothShadingEnable bit in the **DeltaMode** register is ignored.

## 7.3 Clipping

Primitives are clipped to the clip volume. In clip coordinates the view volume is defined by:

$$-w_c \le x_c \le w_c$$
$$-w_c \le y_c \le w_c$$
$$-w_c \le z_c \le w_c$$

This view volume may be further restricted by up to six user defined clip planes to define the clip volume. Each user clip plane defines a half space and it is the intersection of all these half spaces with the view volume which specifies the clip volume. If no user clip planes are enabled then the clip volume is just the view volume.

A point is in view if its clip coordinate falls within the view volume. This is irrespective of the rasterized point size.

A line is in view if both its end vertices (in clip coordinates) fall within the view volume. If either of the vertices, or part of the line segment are within the view volume then the line is clipped to calculate the portion in view. Any parameters (such as depth, color, etc.) for the new vertices produced by clipping are calculated.

A polygon is in view if all its vertices (in clip coordinates) fall within the view volume. If any of the vertices, or part of the polygon's area are within the view volume then the polygon is clipped to calculate the portion in view. Any parameters (such as depth, color, etc.) for the new vertices produced by clipping are calculated.

A rectangle is in view if its raster position (in clip coordinates) falls within the view volume. This is irrespective of the rectangle's width and height.

Clipping is an expensive operation so is only undertaken when a primitive cannot be determined to be totally in view or out of view. Furthermore short lines and small polygons (really triangles) can also avoid being clipped if they don't cross the near, far or any user clipping plane. This relies on it being faster to rasterize a few more fragments and have the window and/or viewport clipping reject them than to spend a lot of time computing new vertices and parameters. The **LineClipLengthThreshold** and **TriangleClipAreaThreshold** registers hold the threshold values used to determine if the line or triangle should be clipped or just rasterized. The line length is measured in screen pixels and the triangle area is twice the actual area required in screen pixels. Note that this technique assumes the viewport is the same size as the window, or that scissoring is enabled for a window which is smaller than the viewport.

A line or polygon whose vertices have $w_c$ values of differing sign do not generate multiple disjoint line segments or polygons after clipping.

The clipping operation is controlled by the **GeometryMode** register.

| Bit No. | Name | Description |
|---|---|---|
| 12 | ClipShortLines | Clipping is an expensive operation and for short lines it is much faster to draw them and rely on the window and/or screen clipping during rasterization. When this bit is set lines below the length given in the **LineClipThreshold** register are not clipped. This does not apply if the line crosses the near, far or user clipping planes. |
| 13 | ClipSmallTriangles | Clipping is an expensive operation and for small triangles it is much faster to draw them and rely on the window and/or screen clipping during rasterization. When this bit is set triangles below the 'area' given in the **TriangleClipThreshold** register are not clipped. This does not apply if the triangle crosses the near, far or user clipping planes. |
| 22…27 | UserClipMask | There is one bit per user defined clipping plane. Clipping against a plane is enabled when the corresponding bit is set.<br>Bit 0 (i.e. bit 22 in register) corresponds to **UserClip0.** |

**Table 7.11 Clipping Operation Control Bits**

The coordinates of the user clipping planes are held in 24 registers:

**UserClip0X**, **UserClip0Y, UserClip0Z** and **UserClip0W** for user clip plane 0,

**UserClip1X**, **UserClip1Y**, **UserClip1Z** and **UserClip1W** for user clip plane 1,

**UserClip2X**, **UserClip2Y**, **UserClip2Z** and **UserClip2W** for user clip plane 2,

**UserClip3X**, **UserClip3Y**, **UserClip3Z** and **UserClip3W** for user clip plane 3,

**UserClip4X**, **UserClip4Y**, **UserClip4Z** and **UserClip4W** for user clip plane 4 and

**UserClip5X**, **UserClip5Y**, **UserClip5Z** and **UserClip5W** for user clip plane 5.

The clip plane coefficients held in the above registers are defined in eye space. For OpenGL the clip plane is supplied at the API level in model coordinates and is transformed into eye space by using the inverse transpose of the 4x4 model view matrix (Gamma does not do this for you).

A point, with eye coordinates $(x_e \ \ y_e \ \ z_e \ \ w_e)^T$ , is visible with respect to a user clip plane if

$$\begin{pmatrix} p_x & p_y & p_z & p_w \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix} \geq 0$$

where *p* is the user clip plane. Note the plane coefficients are frequently labeled A, B, C and D and these correspond to X, Y, Z and W respectively in the above registers.

## 7.4 Culling

Culling is the process of rejecting polygons based on the vertex ordering (either clockwise or counter-clockwise) when projected onto the screen. This can typically reject 50% of the triangles in a scene, however the onus is on the application programmer or modeler to provide the polygons in a consistent order[1].

Gamma provides two methods of doing culling:

- The first method, as done by OpenGL, is to test the sign of the projected area of the polygon.
- The second method is to use a user provided face normal, as done by QuickDraw3D. The sign of the z component of the transformed face normal (assuming its transformation is enabled) is used to determine the orientation. If no face normal is provided then the area based method will be used (until a face normal is provided). Normalization of the face normal is not necessary if the face normal is just used for culling.

[1]A triangle strip, for example, reverses the order for successive triangles however this is automatically allowed for when culling.

These two methods are mutually exclusive so only one form of culling is applied at a time. The orientation of a face (i.e. front facing or not) is still determined even if culling is disabled as this also selects which Polygon Mode to apply and which material to calculate the lighting with for two sided lighting.

The culling is controlled by the **GeometryMode** register.

| Bit No. | Name | Description |
|---|---|---|
| 8 | FrontFaceDirection | This field selects which direction is the 'front' facing direction.<br>0 = Clockwise<br>1 = Counter Clockwise |
| 9 | PolygonCull | This field, when set, enables polygon culling based on the front face direction. It is ignored for points, lines and rectangles. |
| 10, 11 | PolygonCullFace | This field determines which direction of face should be culled (if enabled). It has the following values:<br>0 = Front<br>1 = Back<br>2 = Front and Back |
| 19 | CullUsingFaceNormal | This field, if set will cull using the supplied face normal. The face normal does not have to be of unit length. If no face normal is supplied then the area method of backface culling is used. |
| 31 | InvertFaceNormalCullDirection | This field, if set, causes the supplied Face Normal to be inverted before it is used for backface culling. |

**Table 7.12 Culling Control Bits**

## 7.5 Primitive Set-Up

The general primitive set up operation is controlled by the **DeltaMode** register. Where additional control is needed for individual primitive types this can be found in the **PointMode**, **LineMode**, **TriangleMode** and **RectangleMode** registers.

The **DeltaMode** register fields are shown in Table 7.13 (it is identical to the **DeltaMode** register in GLINT Delta, but with the addition of BiasCoordinates, ColorDiffuse, ColorSpecular and FlatShadingMethod fields).

| Bit No. | Name | Description |
|---|---|---|
| 0, 1 | TargetChip | This two bit field selects which chip the calculations are tailored to. The options are:<br>0 = 300SX<br>1 = 500TX, MX |
| 2, 3 | DepthFormat | This two bit field defines the depth format GLINT is working in and hence the final format of the depth parameters to be written into GLINT. The options are:<br>1 = 16 bits (300SX, 500TX, MX)<br>2 = 24 bits (300SX, 500TX, MX)<br>3 = 32 bits (300SX, 500TX, MX)<br>The depth format is used regardless of any other modes bits. |
| 4 | FogEnable | When set enables the fog calculations. This is qualified by the FogEnable bit in the **Begin** or **Draw*** commands. |
| 5 | TextureEnable | When set enables the texture calculations. This is qualified by the TextureEnable bit in the **Begin** or **Draw*** commands. |
| 6 | SmoothShadingEnable | When set enables the color calculations. This field only has an effect when the **Draw*** commands are used and is ignored when the Gamma 3D pipeline is used. In this case the FlatShading bit in the **GeometryMode** register is used. |
| 7 | DepthEnable | When set enables the depth calculations. |
| 8 | SpecularEnable | When set enables the specular texture calculations. This is qualified by the TextureEnable in the **Begin** or **Draw*** commands. |
| 9 | DiffuseEnable | When set enables the diffuse texture calculations. This is qualified by the TextureEnable in the **Begin** or **Draw*** commands. |
| 10 | SubPixelCorrectionEnable | When set provides the subpixel correction in Y. This is qualified by the SubPixelCorrectionEnable in the **Begin** or **Draw*** commands. |
| 11 | DiamondExit | When set enables the application of the OpenGL 'Diamond-exit' rule to modify the start and end coordinates of lines. |
| 12 | NoDraw | When set prevents any rendering from starting after the set up calculations are done and parameters sent to GLINT. This only effect the **Draw*** commands and is ignored when the Gamma 3D pipeline is used. |
| 13 | ClampEnable | When set causes the input values to be clamped into a parameter specific range. Note that the texture parameters are not included. This should normally be set. |
| 14, 15 | TextureParameterMode | This two bit field causes the texture parameters to be:<br>0 = Used as given<br>1 = Clamped to lie in the range -1.0 to 1.0<br>2 = Normalized to lie in the range -1.0 to 1.0<br>The normal setting for this field is to select texture normalization. |
| 16…18 | reserved | |
| 19 | BiasCoordinates | When set causes the **XBias** and **YBias** registers values to be added to the x and y coordinates respectively. |
| 20 | ColorDiffuse | When set causes the diffuse texture calculations to be done on the red, green and blue components, otherwise the red component (representing monochrome) is done by itself. |
| 21 | ColorSpecular | When set causes specular texture calculations on red, green and blue components, otherwise the red component (representing monochrome) is done by itself. |
| 22 | FlatShadingMethod | This field determines how the ColorDDA unit in GLINT is to do flat shading. The two options are use the **ConstantColor** register (0) or the DDA (1) by setting zero gradients. The rasterization performance is the same in both cases, however the ConstantColor method is faster to set up. Consider the situation when smooth shading is enabled (in the **GeometryMode** register) and a point is to be drawn. The point is always flat shaded. This field would normally be the inverse of the FlatShading field in the **GeometryMode** register. |

**Table 7.13 DeltaMode Register Fields**

How a primitive gets rendered depends on the primitive type given with the **Begin** command and the PolyMode setting (for triangles, quads and polygons) in the **GeometryMode** register.

Antialiasing is controlled by the AntialiasEnable and AntialiasingQuality bits in the individual point, line and triangle mode registers. Antialiasing will only occur for a particular primitive if it is enabled in the corresponding mode register and the AntialiasEnable bit in the **Begin** command is also set.

The x and y coordinates for point, line, triangle and 3D rectangle can optionally have a bias added to them before rendering. This is controlled by the BiasCoordinates bit in the **DeltaMode** register and the biases are held in the **XBias** and **YBias** registers. This biasing facility has several uses:

- The coordinates may be window relative (almost guaranteed with 3D graphics) but the rasterizer may be set up to do screen relative rendering. Adding a bias value will do the conversion.
- The view port mapping may be set up to add a bias to remove any differences in the accuracy of the set up calculations as a function of the primitive's position on the screen. This bias needs to be removed before rendering is actually done.

More consistent (or position independent) set up calculations can be achieved by biasing the x and y coordinates coming out of the view port mapping. This ensures that all the calculations on the x and y coordinates all have the same degree of precision irrespective of their location on the screen. Consider a floating point value of 1.0 - this will have 23 bits of fractional precision whereas a value of 1024.0 will only have 14 bits of fractional precision. Biasing the values by 8K (for example) forces both cases to have 11 bits of fractional precision and hence yield the same rasterized pixels for a given triangle anywhere on the screen.

*Note: This bias will need to be removed from any coordinate information returned during Feedback mode or by reading back the Raster Position because the application will not be expecting biased coordinates to be returned.*

## 7.5.1 Points

The **PointMode**, **PointSize** and **AAPointSize** registers control how points are drawn.

### 7.5.1.1 Aliased Points

For aliased points the **PointSize** register holds the desired point size. The range of actual integral point sizes are 1…255 held in the bottom 8 bits; a 0 point size is treated as a point size of 1. Points are drawn according to the OpenGL rules so wide points are drawn as squares centered on the vertex. Any parameters (such as color, depth, etc.) are held constant for each fragment rasterized as part of the point.

#### 7.5.1.2 Antialiased Points

Antialiased points have their width, as a floating point number, defined by the **AAPointSize** register. In theory any size antialiased points can be defined, however GLINT places some restrictions on what these widths can be. The Point Table in GLINT restricts the diameter of antialiased points to be from 0.5 to 16.0 in steps of 0.25 when the antialiasing quality is 4x4 or 0.25 to 8.0 in steps of 0.125 for 8x8 quality. Gamma does not set up the Point Table. Points with a zero size will draw a single fragment and points with a negative size will draw a point of the same positive size.

It is the user's responsibility to have set up any alpha blending modes. The style of rendering used for a point is determined by the following registers:

The antialiasing quality is held in the **PointMode** register.

| Bit No. | Name | Description |
|---|---|---|
| 0 | AntialiasEnable | This field, when set, enables antialiasing of points. This is qualified by the AntialiasEnable field in the **Begin** command. Note the Point Table in GLINT must be set up for the corresponding point size (held in **AAPointSize** register) and the selected antialiasing quality (next field). |
| 1 | AntialiasingQuality | This field defines the quality of antialiased points:<br>0 = 4x4<br>1 = 8x8<br>The Point Table in GLINT must be set up appropriately for the quality and the **AAPointSize.** |

**Table 7.14 Antialiasing in the PointMode Register**

### 7.5.2 Lines

The **LineMode**, **LineWidth**, **LineWidthOffset** and **AALineWidth** registers control how lines are drawn.

#### 7.5.2.1 Aliased Lines

For aliased lines the **LineWidth** register holds the desired line width. The range of actual integral line widths are 1…255 held in the bottom 8 bits; a 0 line width is treated as a line width of 1. Lines are drawn according to the OpenGL rules so wide lines are drawn as a sequence of lines offset in X or Y depending on whether the line is X major or Y major. The **LineWidthOffset** register is normally set to (line width - 1) / 2. For one pixel wide lines the **LineWidthOffset** is set to 0.

If line stipples are enabled (in the **LineMode** register) then wide aliased lines will be stippled correctly by repeating the line (offset in X or Y), but with the stipple position re-established for each line used to make up the width.

#### 7.5.2.2 Antialiased Lines

Antialiased lines have their width defined as a floating point number by the **AALineWidth** register so can take any width. Lines with a zero width will not be drawn. Lines with a negative width will draw a line of the same positive width. An antialiased line is drawn as rectangle aligned to the direction of the line. This is decomposed into trapezoids so no retained alpha buffer is needed (which would be the case if the line were decomposed into two triangles).

It is the user's responsibility to have set up any alpha blending modes.

If line stipples are enabled (in the **LineMode** register) then antialiased lines will be stippled using the RepeatFactor, StippleMask and Mirror fields in the **LineMode** register. These fields would normally track the fields of the same name in the GLINT **LineStippleMode** register. The stipple pattern is converted into a series of short lines which are drawn as antialiased lines.

The line stipple hardware in GLINT is not used and does not get updated for stippled antialiased lines, however this does not cause a problem because OpenGL does not allow switching between aliased and antialiased lines between polyline segments and the stipple pattern is always reset on a **Begin** command

| Bit No. | Name | Description |
|---|---|---|
| 0 | StippleEnable | This field, when set, enables the stippling of lines. It only effects wide lines or antialiased lines. This will normally be the same value as the Enable field in the **LineStippleMode** GLINT register. |
| 1…9 | RepeatFactor | This 9 bit field holds the repeat factor for antialiased stippled lines. This will normally be the same value as the RepeatFactor field in the **LineStippleMode** GLINT register. The repeat factor stored here is one less than the desired repeat factor. |
| 10…25 | StippleMask | This 16 bit field holds the stipple pattern to use for antialiased lines. This will normally be the same value as the StippleMask field in the **LineStippleMode** GLINT register. |
| 26 | Mirror | This field, when set, will mirror the StippleMask before it is used for antialiased lines. This will normally be the same value as the Mirror field in the **LineStippleMode** GLINT register. |
| 27 | AntialiasEnable | This field, when set, enables antialiasing of lines. This is qualified by the AntialiasEnable field in the **Begin** command. |
| 28 | AntialiasingQuality | This field defines the quality of antialiased lines:<br>0 = 4x4<br>1 = 8x8 |

**Table 7.15. LineMode Register Fields**

### 7.5.3 Polygons

Triangles, quads and polygons are controlled by the **TriangleMode** register.

| Bit No. | Name | Description |
|---|---|---|
| 0 | AntialiasEnable | This field, when set, enables antialiasing of triangles. This is qualified by the AntialiasEnable field in the **Begin** command. |
| 1 | AntialiasingQuality | This field defines the quality of antialiased triangles:<br>0 = 4x4<br>1 = 8x8 |
| 2 | UseTrianglePacketInterface | This field, when set, causes the triangle set up to use the Triangle Packet Interface to send the triangle parameters to GLINT. This is only supported in GLINT MX and provides a higher triangle throughput. |

**Table 7.16 TriangleMode Register Fields**

### 7.5.4 3D Rectangle

OpenGL rectangles are positioned and given depth, color, texture, etc. parameters using the glRasterPos function (which translates into writes to the **RPx3**, **RPy**, etc. registers). The width and height of the rectangle is held in the **RectangleWidth** and **RectangleHeight** registers as floating point numbers and the **RectangleMode** register holds data to pass to GLINT in the **Render** command. When the **RPx2**, **RPx3** or **RPx4** registers are written to the coordinate is transformed, clipped, colored and textured as required by OpenGL and the results saved to be used by the **GeomRectangle** command.

| Bit No. | Name | Description |
|---|---|---|
| 0, 1 | Type | These two bits define the type of rectangle to be inserted into the feedback buffer. They have no effect when not in feedback mode. The options are:<br>0 = Bitmap<br>1 = DrawPixel<br>2 = CopyPixel<br>3 = Don't insert into the feedback buffer. |
| 2 | OffsetEnable | When this bit is set the x and y offset values held in **RasterPosXOffset** and **RasterPosYOffset** registers respectively displace the raster position window coordinates when the rectangle is rendered. This does not update the raster position state. |
| 3 | SelectEnable | When this bit is set the rectangle takes part in the selection process. |

**Table 7.17 GeomRectangle Control Fields**

If the rectangle passes the clip test then it is rendered with the above parameters. The fill direction is always bottom to top (i.e. increasing Y), left to right so any download data or bitmask *must* be provided in this order. If the rectangle is clipped out (this is an all or nothing test, unlike lines, triangles or quads which do geometric clipping of the primitive) then any following image data or bitmasks are automatically discarded.

The floating point **RasterPosXOffset** and **RasterPosYOffset** registers are temporarily added to the current raster position X and Y coordinates respectively before the rectangle is set up in the rasterizer. The original raster position coordinates are not updated. This temporary offset, measured in pixels, is useful when an OpenGL primitive requires several passes or is processed in strips because the color formatting or transfers modes are not handled directly by GLINT.

The floating point **RasterPosXIncrement** and **RasterPosYIncrement** registers are added to the current raster position X and Y coordinates respectively after the rectangle is rendered but not if the clip test fails. This auto increment cannot be disabled so if it is not desired the increment values should be set to zero. The increment is measured in pixels.

### 7.5.5 2D Rectangle

The **DrawRectangle2D** command provides a convenient way to set up the GLINT rasterizer to draw a rectangle.

The origin of the rectangle is supplied as data with the **DrawRectangle2D** command. The least significant 16 bits hold the 2's complement X coordinate while the most significant 16 bits hold the 2's complement Y coordinate.

The width and height of the rectangle, SyncOnHostData, SyncOnBitMask, SpanOperation mode and fill direction are defined by the **Rectangle2DMode** register. The choice of using spans to fill the rectangle is determined by the least significant bit of the **Rectangle2DControl** register.

| Bit No. | Name | Description |
|---|---|---|
| 0…11 | Width | Width of the rectangle. Twelve bit field with range 0…4095 |
| 12…23 | Height | Height of the rectangle. Twelve bit field with range 0…4095 |
| 24 | AreaStippleEnable | Passed to rasterizer in the **Render** command. |
| 25 | SyncOnBitMask | Passed to rasterizer in the **Render** command. |
| 26 | SyncOnHostData | Passed to rasterizer in the **Render** command. |
| 27 | TextureEnable | Passed to rasterizer in the **Render** command. |
| 28 | FogEnable | Passed to rasterizer in the **Render** command. |
| 29 | SpanOperation | Passed to rasterizer in the **Render** command. |
| 30 | HorizontalDirection | Sets the horizontal rasterization direction.<br>0 = Left to Right<br>1 = Right to Left |
| 31 | VerticalDirection | Sets the vertical rasterization direction.<br>0 = Increasing Y<br>1 = Decreasing Y |

**Table 7.18 Rectangle2DMode Fields**

For OpenGL the main use of this is to clear the framebuffer. Note that in this case the ability to use the faster span method of clearing depends if GID window clipping is being used. OpenGL does not necessarily know when a window is clipped so by having the ownership of the **Rectangle2DControl** register reside with the Display Driver the Display Driver can change the rasterization method independent of OpenGL as a function of the window clipping.

The colors and modes set up by OpenGL for the clear operation need to satisfy span and non span clears.

For colors spans use **FBBlockColor** while non spans use **ConstantColor**. Also spans ignore many mode settings, such as Depth compare and Alpha blend. The additional set up is not really a performance issue as clear operations are infrequent and tend to take a comparatively long time.

For Display Driver GUI operations the **Rectangle2D** command provides a very fast way of setting up rectangles - just one register and one command compared to the 6 registers and one command when programming GLINT directly (this is obviously reduced when multiple rectangles are drawn one after the other).

The **XBias** and **YBias** are ignored for this primitive type.

# 8. Context Save and Restore

The natural boundary for context switching in a DMA driven system is on the completion of a DMA buffer. This requires that no internal state is carried over from one DMA buffer to the next as there are no mechanisms for saving and restoring the internal state. This has not been a problem so far because the main time this could arise is between the upper and lower trapezoids in a triangle - a situation which is easy to avoid.

OpenGL places no restrictions on the number of primitives between a Begin/End pair so circumstances will occur when it crosses one or more DMA buffer boundaries. With GLINT Delta this was not a problem as no private internal state was carried from one primitive to the next, however Gamma carries a significant amount of state from one primitive to the next, especially for meshed primitives. The readback mechanism allows all user loadable registers to be read back, however it cannot return any of the internal state.

The software solution to this is for the software to track the last three vertices so a Begin/End sequence can be terminated at the end of a DMA buffer and restarted in the next DMA buffer without any change to the semantics of the original Begin/End sequence. The main objection to this technique is that the overheads of this tracking will seriously effect the system performance, particularly as the Gamma interface only requires the minimal amount of host work to copy (and convert number formats) the vertex and normal arguments to the DMA buffer.

Gamma has a mechanism which allows all the state (public as well as private) to be saved and restored in a consistent manner[1]. The context switch can only happen on a register load or command boundary so cannot happen part way through any internal processing, for example during clipping, so many intermediate results are not required to be saved. This simplifies the hardware while giving the software a convenient level of granularity to work with.

Even with this new context switching method, context switching in the middle of an image or texture download is not handled:

[1]This will also be available in future rasterizer chips, however existing rasterizer chips will need to use the normal read back mechanisms for context switching.

The Gamma context switch is done by sending the **DumpContext** command. The data sent with this command (the context mask) dictates what subset of the full context is to be dumped. The context data (with the **ContextData** tag) will appear in the Host Out FIFO in GLINT. The last tag and data in the FIFO will be the **DumpContext** tag and the context mask. The context data is read from the Host Out FIFO and stored in memory in a context buffer (excluding any tags), the context mask is typically discarded. This context buffer can be restored by prefixing it with the three words: **RestoreContext** tag, context mask (used to generate the buffer in the first place) and the **ContextData** tag and loading it all back into Gamma. The **ContextData** tag has the upper 16 bits set to the number of words of *context data* in the buffer minus one[1]. The layout of the data in the context dump buffer is not important because no massaging of the data is necessary before it can be restored, and in fact, is largely undocumented.

Restoring the context data is easily accomplished by the DMA controller. Saving the context data can also be done by the Output DMA controller, with a probably 10X speed up over software. The Output DMA controller can be used in one of two modes (see Section 4 for a more comprehensive description and details on how to initiate output DMA operations):

- Fixed Count. In this mode the Output DMA controller is given the *exact* number of words of context data to read from GLINT's Host Out FIFO. This count should also include the extra context mask. Setting an inappropriate count for the supplied context mask will likely lead to a lock up situation. The **FilterMode** register should be set up to only allow the context data and not tags through, and the Host Out FIFO should also be empty so as not to interpret any left over contents as context data.
- Variable Count. In this mode the Output DMA controller is placed in Feedback mode so will continue to transfer data from the GLINT's Host Out FIFO until an **EndOfFeedback** tag is found. The **FilterMode** register should be set up to allow both context data and tags through so tags and data inappropriate to this mode can be discarded and the **EndOfFeedback** tag can be identified. The Host Out FIFO does not need to be empty, however this would be preferable. The PCI **FeedbackSelectCount** register will hold the number of words written to memory when the Output DMA has finished. This method relieves the programmer from knowing before hand how much context data will be saved, however there may be a performance penalty in that one extra word (to hold the tag) passes on the secondary PCI bus between Gamma and GLINT. This extra word may or may not effect the overall system performance depending on the primary PCI bus and host write performance.

The sequence of events to do a context switch are:

- Update the **FilterMode** register by setting bits 14 and 15 to enable the context tags and data respectively to be written into the host Output FIFO. These two bits are not documented in the GLINT Programmer Reference manual.

[1]A tag with a count in the upper 16 bits is a hold mode tag so all the subsequent data is automatically given the

- Start the Output FIFO DMA controller transferring *n* words of context data, or in Feedback mode so the context data is terminated by the **EndOfFeedback** command.
- Send the **ContextDump** command to Gamma.
- Send the **EndOfFeedback** command if the Output FIFO DMA controller is being used in feedback mode.
- Send the **ContextRestore** command to Gamma (for the new context).
- Copy the new context data into Gamma.
- The following observations can be made about this:
- No synchronization is necessary to context switch and the queued command DMA controller facility also helps here.
- The Output FIFO DMA controller cannot have transfers queued. In general this is not necessary as it is returning data which the application has asked for so it is already delayed waiting for the data. This does mean that the context switching software cannot just assume the Output FIFO DMA controller is always free.
- In a system with a Gamma and one of the existing rasterizer chips the context save and restore will have to be done in two parts because the **ContextDump** command is not available in the rasterizer. The rasterizer context is saved using the normal **Sync** and readback methods. The Gamma context is saved using the **ContextDump** command.
- Changing the state of the Host Out Unit in GLINT to allow the context data through will mean that the original context of this unit is not being saved. For existing rasterizer chips this will not matter as the context save is done in two parts. Future rasterizer chips will address this problem.
- The context dump method with the Output FIFO DMA controller should be capable of saving the context to host memory at an estimated rate of 40 to 60MB/s (this is *very* host dependent). This contrasts to the readback method of 4 to 6MB/s.
- All the context data is tagged with the **ContextData** tag and this is most easily achieved using a hold tag with a count. Tag and data pairs can be used but this is clearly less bandwidth and memory efficient.

Gamma holds a considerable amount of state (public and private) and future rasterizer chips will add to this. Switching between two 3D contexts requires all (or nearly all) the context to be saved, while switching from a 3D context to a 2D context can get by with far fewer registers. The data associated with the **ContextDump** and **ContextRestore** commands give some control over what is saved and restored and obviously the same setting *must* be used with paired saves and restores.

same tag.

| Bit | Name | Context data includes | Words |
|---|---|---|---|
| 0 | GeneralControl | Mode and general control registers. | 17 |
| 1 | Geometry | Some user geometric state and much of the internal state. | 377 |
| 2 | Matrices | The user defined matrices. | 82 |
| 3 | Material | The user defined material parameters. | 27 |
| 4 | Lights0_7 | The user defined light parameters for lights 0 to 7. | 176 |
| 5 | Lights8_15 | The user defined light parameters for lights 8 to 15. | 176 |
| 6 | RasterPos | The raster position related state. This is expanded below so the current raster position, color, etc. can be read back to satisfy the OpenGL Get calls. | 19 |
| 7 | CurrentState | The current state. This is expanded below so the current texture, color, etc. can be read back to satisfy the OpenGL Get calls. | 12 |
| 8 | TwoD | The 2D related control registers. | 2 |
| 9 | DMA | The DMA related registers. | 7 |
| 10 | Select | The select related registers and name stack. | 67 |

**Table 8.1 Context Mask Fields**

The actual contents of the context buffer is not particularly useful for a programmer to know - the user defined state is readily available via the read back mechanism and the internal state of Gamma is not publicly documented.

The context dump mechanism is also used as a convenient way to report some internal state which can be queried by OpenGL, namely the current state (color, normal and texture) and the raster position. This data is in single precision floating point format unless otherwise noted.

If a context dump is done with only the CurrentState bit set then the resultant context buffer will hold the following information (the tags and context mask are assumed to have been discarded):

| Offset | Data |
|---|---|
| 0 | Current edge flag in bit 5 |
| 1 | Current normal, X component |
| 2 | Current normal, Y component |
| 3 | Current normal, Z component |
| 4 | Current texture, S component |
| 5 | Current texture, T component |
| 6 | Current texture, R component |
| 7 | Current texture, Q component |
| 8 | Current color, Red component |
| 9 | Current color, Green component |
| 10 | Current color, Blue component |
| 11 | Current color, Alpha component |

If a context dump is done with only the RasterPos bit set then the resultant context buffer will hold the following information (the tags and context mask are assumed to have been discarded). Note that some user defined state is also included:

| Offset | Data |
|---|---|
| 0 | Window coordinate, X component |
| 1 | Window coordinate, Y component |
| 2 | Window coordinate, Z component |
| 3 | Eye coordinate, Z component |
| 4 | Clip coordinate, W component |
| 5 | Texture, S component |
| 6 | Texture, T component |
| 7 | Texture, R component |
| 8 | Texture, Q component |
| 9 | Fog |
| 10 | In View (bit 0: 0 = out of view, 1 = in view) |
| 11 | xIncrement (user register) |
| 12 | yIncrement (user register) |
| 13 | xOffset (user register) |
| 14 | yOffset (user register) |
| 15 | Color, Red component |
| 16 | Color, Green component |
| 17 | Color, Blue component |
| 18 | Color, Alpha component |

The **ContextRestore** with the CurrentState bit set can also be used to restore new current values such as might be required during the OpenGL glPopAttrib API function. In this case the new current values must be propagated through out Gamma by using the **TransformCurrent** command. This command takes a four bit mask to specify which parameters are to be refreshed and in this case the simplest thing is to set all four bits (see the reference section for a description of these bits). Note if a full chip context restore is being done then the **TransformCurrent** command is not needed, however it will do no harm if it is sent after the full context restore, or indeed at any time.

The 2D operations do not make much use of Gamma and can be completely switched using the TwoD and DMA settings.

# 9. OpenGL Specific Operations

## 9.1 Polygon Mode

Polygon mode allows primitives submitted as triangles, quads or polygons to be rendered as filled primitives, points at the vertices or lines connecting vertices.

The polygon mode can be different for front facing polygons and back facing polygons.

| Bit No. | Name | Description |
|---|---|---|
| 4, 5 | FrontPolyMode | This field selects how a triangle, quad or polygon should be drawn when its orientation is forwards facing . The options are:<br>0 = Point<br>1 = Line<br>2 = Fill |
| 6, 7 | BackPolyMode | This field selects how a triangle or quad or polygon should be drawn when its orientation is backwards facing. The options are:<br>0 = Point<br>1 = Line<br>2 = Fill |

**Table 9.1 GeometryMode Register: fields that control the Polygon Mode.**

## 9.2 Polygon Offset

Polygon offset provides a mechanism whereby a polygon is offset in Z by an amount given by the following equation:

$$offset = m \times factor + bias$$

where factor and bias are the **PolygonOffsetFactor** and **PolygonOffsetBias** registers respectively and *m* is an approximation to the *z* gradient of the triangle:

$$m = \max\left\{\left|\frac{\partial z_w}{\partial x_w}\right|, \left|\frac{\partial z_w}{\partial y_w}\right|\right\}$$

The bias is the product of two components: a user defined value (called *units* in the OpenGL API) and an implementation constant, *r*. The constant, *r*, is the minimum resolvable difference (i.e. is the smallest difference in window coordinate *z* values that is guaranteed to remain distinct throughout polygon rasterization and in the depth buffer). Typical values for *r* are given by:

$$\frac{3.0}{2^n - 1}$$

where *n* is the number of bits in the depth buffer and 3.0 is a constant found empirically to give good results.

Polygon offset is used to draw co-planar polygons slightly offset so the visual order of the polygons are guaranteed and no depth bleeding occurs between them. Examples of where this is useful is in adding decals to surfaces, shadow polygons or reflection polygons.

Polygon offset only applies to polygons, however a submitted polygon can be drawn as a series of points or lines under control of the Polygon Mode. Polygon offset can be enabled individually in each case and is controlled by three bits in the **GeometryMode** register.

| Bit No. | Name | Description |
|---|---|---|
| 28 | PolygonOffsetPoint | This field, if set, causes the polygon offset to be calculated and applied to the points of a polygon when PolyMode is set to Point. |
| 29 | PolygonOffsetLine | This field, if set, causes the polygon offset to be calculated and applied to the lines of a polygon when PolyMode is set to Line. |
| 30 | PolygonOffsetFill | This field, if set, causes the polygon offset to be calculated and applied to the triangles of a polygon when PolyMode is set to Fill. |

**Table 9.2 Controlled Bits in the GeometryMode Register**

## 9.3 Texture Generation

Texture generation allows some or all of the texture coordinates to be derived automatically from the incoming vertex coordinate or normal information. Each component of the texture (s, t, r and q) can have a different texture generation mode assigned to it. The four options are:

- None. The current texture coordinate is used.
- ObjectLinear. The incoming vertex is converted using the following equation:

$$g = p_a x_o + p_b y_o + p_c z_o + p_d w_o$$

where *p* is the user supplied coefficient held in the **TexGen**[16] registers are shown in Table 9.3

| Target for *g* | $p_a$ | $p_b$ | $p_c$ | $p_d$ |
|---|---|---|---|---|
| s | TexGen0 | TexGen4 | TexGen8 | TexGen12 |
| t | TexGen1 | TexGen5 | TexGen9 | TexGen13 |
| r | TexGen2 | TexGen6 | TexGen10 | TexGen14 |
| q | TexGen3 | TexGen7 | TexGen11 | TexGen15 |

**Table 9.3 TexGen[16] Registers Target for g**

- EyeLinear. The incoming vertex is transformed into eye space and then converted using the following equation:

$$h = p_a x_e + p_b y_e + p_c z_e + p_d w_e$$

where *p* is the user supplied coefficient held in the **TexGen**[16] registers are shown in Table 9.4

| Target for *h* | $p_a$ | $p_b$ | $p_c$ | $p_d$ |
|---|---|---|---|---|
| s | TexGen0 | TexGen4 | TexGen8 | TexGen12 |
| t | TexGen1 | TexGen5 | TexGen9 | TexGen13 |
| r | TexGen2 | TexGen6 | TexGen10 | TexGen14 |
| q | TexGen3 | TexGen7 | TexGen11 | TexGen15 |

**Table 9.4 TexGen[16] Registers Target for h**

- SphereMap. The transformed normal is used in the following equation:

$$\mathbf{r} = \hat{\mathbf{u}} - 2(\hat{\mathbf{n}} \bullet \hat{\mathbf{n}} \bullet \hat{\mathbf{u}})$$

$$m = 2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}$$

$$s = r_x / m + \tfrac{1}{2}$$

$$t = r_y / m + \tfrac{1}{2}$$

where

**r** is the reflection vector,
**u** is the unit vector pointing from the origin to the vertex in eye coordinates,
**n** is the unit transformed normal in eye space
• is a dot product

*Note: SphereMap can only be applied to the s and t components.*

OpenGL maintains separate 16 entry stores for the object linear and eye linear coefficients, where as in Gamma they share a single 16 entry store (**TexGen**[16]). This means that the OpenGL driver needs to hold both sets and build up **TexGen**[16] depending on what the texture generation enables are set to.

| Bit No. | Name | Description |
|---|---|---|
| 5, 6 | TexGenModeS | This field controls the automatic generation of texture coordinates for the S texture component from the vertex or normal information. The TexGen operations are:<br>0 None (use current texture S).<br>1 ObjectLinear.<br>2 EyeLinear.<br>3 SphereMap. |
| 7, 8 | TexGenModeT | This field controls the automatic generation of texture coordinates for the T texture component from the vertex or normal information. The TexGen operations are:<br>0 None (use current texture T).<br>1 ObjectLinear.<br>2 EyeLinear.<br>3 SphereMap. |
| 9, 10 | TexGenModeR | This field controls the automatic generation of texture coordinates for the R texture component from the vertex information. The TexGen operations are:<br>0 None (use current texture R).<br>1 ObjectLinear.<br>2 EyeLinear.<br>3 None (use current texture R, SphereMap is illegal). |
| 11, 12 | TexGenModeQ | This field controls the automatic generation of texture coordinates for the Q texture component from the vertex information. The TexGen operations are:<br>0 None (use current texture Q).<br>1 ObjectLinear.<br>2 EyeLinear.<br>3 None (use current texture Q, SphereMap is illegal). |
| 13 | TexGenS | When this bit is set the S component of the texture coordinate is generated automatically, otherwise it is taken from the current texture S value. This only has an effect when the TexGen operation is ObjectLinear, EyeLinear or SphereMap. |
| 14 | TexGenT | When this bit is set the T component of the texture coordinate is generated automatically, otherwise it is taken from the current texture T value. This only has an effect when the TexGen operation is ObjectLinear, EyeLinear or SphereMap. |
| 15 | TexGenR | When this bit is set the R component of the texture coordinate is generated automatically, otherwise it is taken from the current texture R value. This only has an effect when the TexGen operation is ObjectLinear or EyeLinear. |
| 16 | TexGenQ | When this bit is set the Q component of the texture coordinate is generated automatically, otherwise it is taken from the current texture Q value. This only has an effect when the TexGen operation is ObjectLinear or EyeLinear. |

**Table 9.5 GeometryMode register Bits Controlling Texture Generation**

After any texture generation has been done the resultant texture coordinate is optionally transformed.

## 9.4 Select Mode

The SelectMode is used typically during picking or selection operations of a user interface. Placing Gamma into select mode and specifying the select data to return is controlled by the fields in the **GeometryMode** register as shown in Table 9.6.

| Bit No. | Name | Description |
|---|---|---|
| 14,15 | RenderMode | The RenderMode field controls the action when processing any primitive. The options are:<br>0: Render<br>1: Select<br>2: Feedback |

**Table 9.6 GeometryMode register controlling Field**

In OpenGL the select mode is broken down into two parts:

- All primitives are clipped and culled, but not rendered. If a primitive (or raster position) passed the clipping and culling phases then a hit flag is set and the minimum and maximum Z range grown, if necessary, to include this primitive. Subsequent primitives which also pass the clipping and backface culling may extend the minimum and/or maximum Z values.
- Name stack manipulation. The name stack holds names (as 32 bit integers) the user can push, pop or load to keep track of the model hierarchy. The commands **PushName**, **PopName**, **LoadName** do these actions. **PushName** and **LoadName** update the stack with the 32 bit value in the data field. The name stack is reset with the **InitNames** command. The name stack is 64 entries deep.

If the hit flag is set when a name stack manipulation is done a hit record is written to the host output FIFO (in GLINT). The hit flag is then reset along with the minimum and maximum Z range. The hit record consists of (in order):

- The count of the names (NameCount) on the stack (plus some error flags),
- The minimum Z value as a normalized floating point number,
- The maximum Z value as a normalized floating point number,
- The name stack entries, oldest first (variable number [0…64] words).

Bits 14 and 15 in the **FilterMode** register in GLINT must be set to allow the **SelectRecord** tag and data values to be written in to the FIFO - all the select record data uses the same tag.

The name stack manipulations commands are ignored when not in Select mode.

The hit record data is almost in the correct format for OpenGL. The only thing the software needs to do is to convert the minimum and maximum Z values from the floating point format (normalized to be in the range 0.0…1.0) to the 32 bit integer format required by OpenGL.

| Bit | Name | Description |
|---|---|---|
| 0…6 | Count | This field holds the number of names on the name stack. |
| 7…28 | | Not used. |
| 29 | InvalidOperation | A **LoadName** operation was attempted on an empty name stack when this hit record was being collected. This is cleared for subsequent hit records (unless they manifest this error) however the stack may no longer be totally valid. |
| 30 | StackUnderflow | The name stack was popped while empty when this hit record was being collected. This is cleared for subsequent hit records (unless they manifest this error) however the stack may no longer be totally valid. |
| 31 | StackOverflow | The name stack was pushed while full when this hit record was being collected. This is cleared for subsequent hit records (unless they manifest this error) however the stack may no longer be totally valid. |

**Table 9.7 NameCount Value Fields**

The **SelectResult** command can be used to flush out a hit record (but only if the hit flag has been set) without having to do a name stack manipulation. This provides a convenient method when re-entering or leaving the select mode. The **SelectResult** command will also reset the hit flag.

During Select mode the lighting is still calculated even though it is never used so a useful optimization is to disable lighting. Similarly for texture and fog modes. Also it is advisable to disable short line and small triangle threshold testing and always do a full clip otherwise there is a chance that primitives which would have been clipped out actually get included in the selection process.

The amount of data generated in select mode is not easily determined by the host as it depends on how primitives are clipped and backface culled, and the name stack depth when the hit record is written. The **EndOfFeedback** command can be used by the host as a marker to indicate the end of the select stream. When this is found in the Host Out FIFO all the select data will have been read from GLINT.

Host software can read the Host Out FIFO, analyze the tags and build up an OpenGL conformant select buffer. This action of the host reading the FIFO must be done simultaneously with sending user commands or the Host Out FIFO will fill up and GLINT, and then Gamma will stall. Alternatively the Output DMA controller can be placed in feedback mode to simplify and speed up this operation.

The Output DMA controller feedback mode:

- Discards the tags (these are still necessary in the Host Out FIFO so the type of data can be ascertained).
- Discard any surplus select data when the buffer is full.
- Terminate the transfer when the end of the select data is found (as indicated by the **EndOfFeedback** tag).
- Discard any invalid tag and data pairs while in select mode.

The overall method for implementing select with the Output DMA controller is as follows:

- The GLINT **FilterMode** is set up so that the tag and data for the Remainder group of tags (bits 14 and 15 set) are written to the host out FIFO and this FIFO is assumed to be empty after the filter mode has been set up.
- The **DMAOutputAddress** holds the address (logical or physical) where the select data is to be written The start address is given as a byte address but the lower two bits are ignored.
- The **DMAFeedback** command with the length of the memory buffer (in words) is sent to start the Output DMA controller.
- The RenderMode in the **GeometryMode** register is set to Select .
- The user rendering is done.
- The **EndOfFeedback** command is sent to mark the end of the select mode.
- Wait for all the select data to be transferred. This can be done by polling the *CommandStatus* PCI register or via an interrupt.
- Read the count of the number of words transferred from the *FeedbackSelectCount* PCI register. If the most significant bit is set then the buffer is full and there was more data to append to it (but this has been discarded).

If the select buffer becomes full before the **EndOfFeedback** tag in the tag/data stream is detected the Output DMA is terminated and the host informed, however the host output FIFO will continue to be read and its contents discarded until the **EndOfFeedback** is found. The **EndOfFeedback** tag and its data will also be discarded. The *FeedbackSelectCount* PCI register will hold the actual number of words transferred. Bit 31 is set if more data was found before the **EndOfFeedback** tag.

Once the select buffer has been updated the host software needs to parse the buffer to:

- Convert the minimum and maximum Z values from floating point format (in the range 0.0…1.0) to the integer format needed by OpenGL.
- Remove the status bits from the hit record (in the most significant byte of the name stack depth).
- Count the number of hit records present in the buffer.

## 9.5 Feedback

The OpenGL Feedback mode returns the vertex data for the primitives which have passed clipping and culling. Placing Gamma into feedback mode and specifying the vertex data to return is controlled by the two fields in the **GeometryMode** register shown in Table 9.8.

| Bit No. | Name | Description |
|---|---|---|
| 14,15 | RenderMode | The RenderMode field controls the action when processing any primitive. The options are:<br>0: Render<br>1: Select<br>2: Feedback |
| 16…18 | FeedbackType | This field only has any effect if the RenderMode is Feedback. In this case it determines the parameters to be returned for every primitive. The options are:<br>0: X, Y 2D<br>1: X, Y, Z 3D<br>2: X, Y, Z, R, G, B, A 3DColor<br>3: X, Y, Z, R, G, B, A, S, T, R, Q 3DColorTexture<br>4: X, Y, Z, W, R, G, B, A, S, T, R, Q 4DColorTexture |

**Table 9.8 Controlling Fields in the GeometryMode Register**

While in Feedback mode no rendering is done, however details about what would have been rendered are returned.

The vertex data will appear in the host output FIFO in the order given here, i.e. the X value will appear first, followed by the Y value, etc. Bits 14 and 15 in the **FilterMode** register in GLINT must be set to allow the feedback tag and data values to be written in to the FIFO.

The data is identified in the FIFO by a tag starting with 'Feedback' with the appropriate name appended. For example the X value will have a tag of **FeedbackX**, the Red value a tag of **FeedbackRed**, etc.[1]. All the data is returned in floating point format.

The vertex data is associated with a particular primitive and before any of the vertex data is written to the FIFO the **FeedbackToken** tag will be written. The data with this tag defines what primitive would have been drawn and implicitly how many vertices worth of vertex data are in the FIFO.

[1]The full set of feedback tags is: **FeedbackX**, **FeedbackY**, **FeedbackZ**, **FeedbackW**, **FeedbackRed**, **FeedbackGreen**, **FeedbackBlue**, **FeedbackAlpha**, **FeedbackS**, **FeedbackT**, **FeedbackR**, **FeedbackQ**.

| Data value | Primitive Type | Number of vertices |
|---|---|---|
| 0x44e02000 | Point | 1 |
| 0x44e04000 | Line | 2 |
| 0x44e06000 | Triangle | 3 |
| 0x44e08000 | Bitmap | 1 |
| 0x44e0a000 | DrawPixel | 1 |
| 0x44e0c000 | CopyPixel | 1 |
| 0x44e0e000 | LineReset | 2 |
| 0x44e00000 | PassThrough | 0 |

**Table 9.9 Data Field Possibilities**

The LineReset is the same as a Line but the stipple pattern was reset for this line.

The hex values supplied match up with the tokens defined by OpenGL. Note that the Triangle token is the same as the Polygon token in OpenGL, however the vertex count is fixed at three and not included in the tag and data stream.

The Bitmap, DrawPixel and CopyPixel are all the Rectangle primitive with bits 13 and 14 (of the data value) set from the low two bits of the **GeomRectangle** command. These two bits are only used to generate the correct OpenGL feedback token. Note a value of 3 is used to indicate this rectangle shouldn't have an entry in the feedback buffer and this is used to prevent multiple feedback entries when the **GeomRectangle** is used several times to implement a single API call.

The number and type of primitives returned during feedback depends on what primitives are sent in the first place, the Polymode setting and the result of any clipping operation. Polygons and Quads are always decomposed into triangles and clipping a triangle can give anywhere between 0 and 14 triangles depending on the number of clipping planes (frustum and user) the original triangle cuts.

For some FeedbackType settings the color and/or texture values are not required. In these cases these can be disabled for higher performance (this does assume that the host reading back the feedback tags and data will not be the limiting factor). Fog can always be disabled. It is also advisable to disable short line and small triangle threshold testing and always do a full clip otherwise there is a chance that primitives which would have been clipped out actually get included in the feedback process.

It may be useful to inject markers while doing feedback (OpenGL has a function call glPassThrough to do this) to help keep track of which part of the model you are in. This is done by using the **PassThrough** command - the tag and data are written directly into the Host Out FIFO without changing any internal state in Gamma or GLINT.

The amount of data generated in feedback mode is not easily determined by the host as it depends on how primitives are clipped, polygon mode and backface culling. The **EndOfFeedback** command can be used by the host as a marker to indicate the end of the feedback stream. When this is found in the Host Out FIFO all the feedback data will have been read from GLINT.

Host software can read the Host Out FIFO, analyze the tags and build up an OpenGL conferment feedback buffer. This action of the host reading the FIFO must be done simultaneously with sending user commands or the Host Out FIFO will fill up and GLINT, and then Gamma will stall. Alternatively the Output DMA controller can be placed in feedback mode to simplify and speed up this operation.

The Output DMA controller feedback mode does the following:

- Formats the feedback data from GLINT into the exact format required by OpenGL and write it into a memory buffer. This includes adding in the vertex count (3.0) for triangles (a.k.a. polygons), otherwise just involves discarding the tags (these are still necessary in the Host Out FIFO so the type of data can be ascertained).
- Converts the **PassThrough** tag to the PassThrough token and also append its associated data.
- Discards any surplus feedback data when the buffer is full.
- Terminates the transfer when the end of the feedback data is found (as indicated by the **EndOfFeedback** tag).
- Discards any invalid tag and data pairs while in feedback mode.

The overall method for implementing feedback with the Output DMA controller is as follows:

- The GLINT **FilterMode** is set up so that the tag and data for the Remainder group of tags (bits 14 and 15 set) are written to the host out FIFO and this FIFO is assumed to be empty after the filter mode has been set up.
- The **DMAOutputAddress** holds the address (logical or physical) where the feedback data is to be written The start address is given as a byte address but the lower two bits are ignored.
- The **DMAFeedback** command with the length of the memory buffer (in words) is sent to start the Output DMA controller.
- The RenderMode in the **GeometryMode** register is set to Feedback .
- The user rendering is done.
- The **EndOfFeedback** command is sent to mark the end of the feedback mode.
- Wait for all the feedback data to be transferred. This can be done by polling the *CommandStatus* PCI register or via an interrupt.
- Read the count of the number of words transferred from the *FeedbackSelectCount* PCI register. If the most significant bit is set then the buffer is full and there was more data to append to it (but this has been discarded).

If the feedback buffer becomes full before the **EndOfFeedback** tag in the tag/data stream is detected the Output DMA is terminated and the host informed, however the host output FIFO will continue to be read and its contents discarded until the **EndOfFeedback** is found. The **EndOfFeedback** tag and its data will also be discarded. The *FeedbackSelectCount* PCI register will hold the actual number of words transferred. Bit 31 is set if more data was found before the **EndOfFeedback** tag.

*Note: If the viewport mapping includes any additional biasing normally removed during the primitive set up operation (see section 7.5) using the XBias and YBias register values then the X and Y coordinates in the feedback buffer need to have the XBias and YBias values added to. This will restore the coordinates to the number range the application is expecting. This is not done automatically by Gamma and is a post processing operation by software once the feedback buffer has been created.*

## 9.6 Raster Position

The raster position in OpenGL is set by writing to the RP* registers (**RPy**, **RPz**, **RPw** and one of **RPx2**, **RPx3** or **RPx4**). The raster position is transformed and lit as a normal vertex would be and the results saved away.

OpenGL has a rectangle primitive (glReadPixels, glWritePixels, etc.) where the raster position (previously established as above) defines the rectangle origin. The **RectangleWidth** and **RectangleHeight** registers define the width and height respectively. The texture, normal, fog, color, etc. values are calculated and stored with the raster position when the raster position is first defined.

The **GeomRectangle** command is used to render the rectangle. If the raster position is not in view then all **GeomRectangle** command are ignored until a new raster position is established. If the raster position is in view then the operation is controlled by the data field.

| Bit | Name | Description |
|---|---|---|
| 0, 1 | Type | These two bits define the type of rectangle to be inserted into the feedback buffer. They have no effect when not in feedback mode. The options are:<br>0 = Bitmap<br>1 = DrawPixel<br>2 = CopyPixel<br>3 = Don't insert into the feedback buffer. |
| 2 | OffsetEnable | When this bit is set the x and y offset values held in **RasterPosXOffset** and **RasterPosYOffset** respectively displace the raster position window coordinates when the rectangle is rendered. This does not update the raster position state. |
| 3 | SelectEnable | When this bit is set the rectangle takes part in the selection process. |

**Table 9.10 GeomRectangle Data Field**

After every rectangle is submitted using the **GeomRectangle** command (in any RenderMode) the window coordinate x and y components are updated by the amount held in the **RasterPosXIncrement** and **RasterPosYIncrement** registers respectively. This occurs irrespective of the raster position being in view either before or after the update. If the initial raster position was in view then all subsequent raster positions updated via the increment will be in view. The converse also holds. The texture, fog and color values are only updated when the raster position is changed by an update to one of the **RPx2**, **RPx3** or **RPx4** registers.

The width and height of the rectangle is held in the **RectangleWidth** and **RectangleHeight** registers as floating point numbers. The **RectangleMode** holds the low level enables to control TextureEnable, FogEnable, etc. and has the same format as the GLINT **Render** command.

A raster position should not be sent between a Begin/End sequence as it will corrupt the vertex data being carried from one primitive to the next (an OpenGL requirement as well).

For OpenGL bitmap operations the supplied **RasterPosOffsetX** and **RasterPosOffsetY** should be set from the -origin values given in the API call. Also 0.4999 (nearly half) should be subtracted as well to get the required floor operation the OpenGL specification required for bit maps.

All the state associated with the raster position can be saved and restored via the context dump and context restore mechanisms (see the context saving section for details). This mechanism is also used to get the current raster position when it is queried by the glGet function in OpenGL.

If a **ContextDump** is done with only the RasterPos bit set then the resultant context buffer will hold the following information (the tags and context mask are assumed to have been discarded). Note there is some user defined state also included:

| **Offset** | **Data** | **Offset** | **Data** |
|---|---|---|---|
| 0 | Window coordinate, X component | 10 | In View (bit 0: 0 = out of view, 1 = in view) |
| 1 | Window coordinate, Y component | | |
| 2 | Window coordinate, Z component | 11 | xIncrement (user register) |
| 3 | Eye coordinate, Z component | 12 | yIncrement (user register) |
| 4 | Clip coordinate, W component | 13 | xOffset (user register) |
| 5 | Texture, S component | 14 | yOffset (user register) |
| 6 | Texture, T component | 15 | Color, Red component |
| 7 | Texture, R component | 16 | Color, Green component |
| 8 | Texture, Q component | 17 | Color, Blue component |
| 9 | Fog | 18 | Color, Alpha component |

This data is in single precision floating point format unless otherwise noted.

## 9.7 Current Texture, Normal and Color values

The current values for texture, normal and color are updated when texture coordinates, normals and colors are written to Gamma. OpenGL allows these to be queried and these can be read back from Gamma or the **ContextDump** command with the CurrentState context mask bit set can be used.

If a context dump is done with only the CurrentState bit set then the resultant context buffer will hold the following information (the tags and context mask are assumed to have been discarded):

| **Offset** | **Data** | **Offset** | **Data** |
|---|---|---|---|
| 0 | Current edge flag in bit 5 | 6 | Current texture, R component |
| 1 | Current normal, X component | 7 | Current texture, Q component |
| 2 | Current normal, Y component | 8 | Current color, Red component |
| 3 | Current normal, Z component | 9 | Current color, Green component |
| 4 | Current texture, S component | 10 | Current color, Blue component |
| 5 | Current texture, T component | 11 | Current color, Alpha component |

This data is in single precision floating point format unless otherwise noted.

The current data can be restored by writing to their assigned registers (as would normally be done to pass this data to Gamma in the first place) or by using the **ContextRestore** command with the CurrentState context mask bit set. In this case the new current values must be propagated through out Gamma by using the **TransformCurrent** command. This command takes a four bit mask to specify which parameters are to be refreshed and in this case the simplest thing is to set all four bits (see the reference section for a description of these bits).

The current values can also be saved and restored using a set of internal registers. This is controlled by the **SaveCurrent** and **RestoreCurrent** commands. There is only one set of registers so the **SaveCurrent** command will overwrite the data saved by the previous **SaveCurrent**. This mechanism is much faster and simpler to use than using the **DumpContext** and **RestoreContext** commands.

OpenGL evaluators (used for curved surfaces, NURBS, etc.) calculate texture, normal and color values (depending on what is enabled) and they are treated as if the programmer had supplied them directly, but with one important exception. The current values (or any derived state such as material parameters edited using ColorMaterial) are not changed. The **SaveCurrent** and **RestoreCurrent** commands are used to bracket the evaluator operations. All derived state is also refreshed by the **TransformCurrent** command.

## 9.8 Window Clipping Support

Gamma does not have any support for window clipping. The method of repeating a DMA buffer used by the earlier 3Dlabs drivers can be used, however a *full* Gamma context dump is required before the DMA buffer is read for the first time. On subsequent repeats of the DMA buffer the previous Gamma state is restored. Clearly this is unlikely to be a high performance solution.

Window clipping should be done using the Graphics ID (GID) facility in GLINT. The GID is stored in the GLINT's local buffer and provides a pixel ownership test. See one of the GLINT Programmer's Reference manuals for more details on using GIDs for window clipping.

The OpenGL driver does not know when one of its windows is clipped and even if there were a callback to notify of this situation this does not help any DMA buffers already generated, and waiting to be read.

GID testing is controlled by the **Window** register and the **LBReadMode** register (to enable local buffer reads), unfortunately the display (GUI) driver needs to own some bits in these registers while the OpenGL driver need to own others. To prevent some sort of software handshaking or locking being necessary to control access to these register Gamma intercepts writes to these registers and keeps a local copy. The local copies of these registers can be modified using the **WindowAnd**, **WindowOr**, **LBReadModeAnd** and **LBReadModeOr** commands. Recall that the data associated with the command is logically combined with the existing data so individual bits can be modified in isolation. Once the local copy is updated the GLINT copy is updated. One further modification to the **LBReadMode** is done and this is to enable destination reads if GID testing is required.

The **Rectangle2DControl** register holds a single bit which is set when window clipping is needed. This bit is used to force local buffer reads to be done and this register is owned by the display driver.

GID testing, for the most part on the GLINT MX, is a free test, however it is not compatible with the span operations. Normally these are used by 2D GUI operations which are pre clipped to the window so don't need to use GIDs. There is one situation where OpenGL would use spans and this is to clear the color buffer (i.e. as part of the glClear function). When GID testing is being used OpenGL can't use spans to implement the clear. To avoid OpenGL needing to know which style of clears to use, or using a driver call to do the clear it sets up the clear for both methods and lets the display driver choose depending if window clipping is necessary.

OpenGL does the set up for clears assuming a single pixel at a time rendering will be used. In addition the clear color (in framebuffer raw format) is written to the **FBBlockColor** register. The **Rectangle2D** command is used to render the rectangle. The **Rectangle2DControl** register, under control of the driver, will select the appropriate rasterization method.

## 9.9 Color Material Support

OpenGL has the facility to allow the current color or a color sent to the graphics pipeline to change one or more selected material parameters. In Gamma there is no performance penalty for changing a material's parameter directly rather than via the color material mechanism. The **ColorMaterialMode** register controls the color material mechanism.

| Bit No. | Name | Description |
|---|---|---|
| 0 | Enable | When set causes a vertex color to update the material parameter(s) for the given face(s). |
| 1, 2 | Face | This field selects which face(s) any material changes should be made to by the updating color. The values are:<br>0 = front material<br>1 = back material<br>2 = front and back material |
| 3…5 | Parameter | This three bit field selects which material parameter(s) should be updated by the updating color. The values are:<br>0 = Emissive<br>1 = Ambient<br>2 = Diffuse<br>3 = Specular<br>4 = Ambient and diffuse |

**Table 9.11 ColorMaterialMode Register Fields**

When color material is first enabled the existing current color is immediately applied to the selected material parameter(s).

The OpenGL specification states that material edits using glMaterial do not affect any material properties selected for update by **ColorMaterial.** Gamma does not enforce this behavior.

Note: *The OpenGL specification stipulates that when ColorMaterial is enabled the target material parameter(s) are updated by calls to glColor. This is handled automatically by Gamma. The OpenGL specification also goes on to state that the target material parameters are no longer updated by calls to glMaterial. This is not done automatically by Gamma and the OpenGL driver must do this filtering.*

## 9.10 Get Operations

OpenGL provides a query mechanism. The parameters which can be queried broadly fall into three categories:

- User defined state. User defined state is state the application sets directly, for example a light's color or position. The OpenGL driver can choose to track this state itself or read it back from the hardware. If the state is to be read back from the hardware any pending DMA must be finished and Gamma and GLINT synchronized with first.
- Derived state. Derived state is state which could be tracked by the OpenGL, but will impact performance too much to do so. State in this category is the current texture, normal and color values. The material state may be included in this if Color Material is enabled.
- Transformed state. This is state which undergoes significant processing before the results are queried. An example of this is raster position, and unless a software geometry and lighting pipeline is maintained there is no alternative but to read back the state from Gamma.

Keeping a software copy of state looks like it is a big performance win, however it does have one big drawback - state changes cannot be included in native display lists as these would not be reflected by the software copy when the display list was executed.

## 9.11 Display Lists

Display lists can be held in two formats: some internal OpenGL format which is parsed into Gamma commands when executed, or in native Gamma format. Clearly the native format will be much faster and is the preferred format.

The native format suffers from three drawbacks:

- Gamma relies on the host to do any matrix generation and concatenation so OpenGL commands for this cannot exist in a display list.
- Mode changes in the current rasterizer chips may involve changing one or two bits in a register and leaving the rest of the bits unaffected. This requires a software copy of the register to be used to re-generate the bits to keep. The register contents may be different when the display list is executed from when the display list was created so needs active involvement from software. All the Gamma mode registers have an And and an Or version to allow individual bits to be changed.

- Inter dependencies in mode bits. For example the framebuffer, in general, needs to be enabled for reading when alpha blending is done, however some alpha blending modes don't use the framebuffer contents. Meanwhile, in a different mode register the logical operation may or may not require framebuffer data to be read as well.

Future Gamma devices and new rasterizers will address these problems so most operations can be included in display lists. Meanwhile a composite scheme of mixing native and parsed display lists based on their contents is a good compromise.

# 10. Direct3D and QuickDraw3D Functionality

## 10.1 Face Normals

QuickDraw3D uses face normals for culling and for flat shading. Face normals are written using **FNx**, **FNy** and **FNz** registers (recall **FNx** must be sent last) and this must be sent before the last vertex for the triangle, or the last vertex of the first triangle a quad or polygon is decomposed into. If a face normal is not received in time then the area based calculation may be used for culling, however the same face normal is used for subsequent triangles a quad or polygon is decomposed into unless a new one is received.

The face normals are transformed (if enabled by the TransformFaceNormal bit in the **TransformMode** register) using the **NormalMatrix**[9] used by the vertex normals. Face normals never need to be of unit length if they are just going to be used for culling.

The CullUsingFaceNormal bit in the **GeometryMode** register enables culling using the face normal. If the sign of the Z component of the transformed face normal is positive the vertices are assumed to be ordered counter clockwise. This sense can be inverted by the InvertFaceNormalCullDirection bit in **GeometryMode**.

The association of front facing to vertex order is controlled by the FrontFaceDirection bit in **GeometryMode** and the facing direction to cull is set by PolygonCullFace in **GeometryMode**. The PolygonCull bit in **GeometryMode** enables polygon culling, however this is ignored when a face normal is provided.

If CullUsingFaceNormal is true and no face normal has been provided then the area based method will be used in which case the PolygonCull bit is taken into account. This allows polygons to be culled if and only if a face normal is provided.

The face normal can be used for lighting rather than the vertex normal and in this case it can be normalized (enabled by the FaceNormalEnable in **NormalizeMode**) if, after transformation, it will not be of unit length. If no face normal is provided then setting the AutoGenerateFaceNormal bit in **GeometryMode** automatically calculates the normalized face normal (the setting of FaceNormalEnable in **NormalizeMode** is ignored). The automatically generated face normal can be optionally inverted (by setting the InvertAutoFaceNormal bit in **NormalizeMode**) in case the cross product order of the two edges doesn't follow the application's convention.

Setting the UseFaceNormal bit in the **LightingMode** register forces the lighting to be calculated with the face normals rather than the vertex normals. The lighting is still evaluated once per vertex so any position dependent effects (i.e. attenuation or spotlight) are computed correctly.

## 10.2 Diffuse Textures

QuickDraw3D allows a diffuse lighting component to be applied after texture mapping. The diffuse values (monochrome for GLINT 500TX , color for GLINT MX) are interpolated across a triangle and are derived from the diffuse light value at each vertex (see the earlier lighting equations).

Diffuse textures are enabled by the DiffuseTextureEnable bit in **MaterialMode,** DiffuseEnable bit in **DeltaMode** and the TextureEnable bit in the **Begin** command. The diffuse texture (at each vertex) is calculated using:

$$\mathbf{diffuseTexture} = \mathbf{e}_{cm} + \mathbf{a}_{cm} + \mathbf{ambientLight} + \mathbf{diffuseLight}$$

where

$\mathbf{e}_{cm}$ is the emissive material color

$\mathbf{a}_{cm}$ is the ambient material color

The monochrome or color is selected by the MonochromeDiffuseTexture bit in **MaterialMode** and ColorDiffuse bit in **DeltaMode**.

## 10.3 Specular Textures

Direct3D and QuickDraw3D allows a specular lighting component to be applied after texture mapping. The specular values (monochrome for GLINT 500TX , color for GLINT MX) are interpolated across a triangle and are derived from the specular light value at each vertex (see the earlier lighting equations).

Specular textures are enabled by the SpecularTextureEnable bit in **MaterialMode,** SpecularEnable bit in **DeltaMode** and the TextureEnable bit in the **Begin** command. The specular texture (at each vertex) is calculated using:

$$\mathbf{specularTexture} = \mathbf{specularLight}$$

The monochrome or color is selected by the MonochromeSpecularTexture bit in **MaterialMode** and ColorSpecular bit in **DeltaMode**.

# 11. Compatibility with GLINT Delta

Gamma can be programmed like GLINT Delta where the host software provides the coordinate, color, etc. information for one or more vertices and then issues one of the **Draw*** commands. Obviously this forgoes all the benefits of using the Gamma specific commands and facilities, but will give a significant performance increase over GLINT Delta.

The contents of any of the vertex stores *must* not be changed while the rest of Gamma is being used, or more precisely between the **Begin** and **End** commands.

The software interface and behavior is basically compatible with GLINT Delta (once the new mode registers have been initialized to zero) but 100% software compatibility was not a design goal. The differences are:

- Antialiased triangles are enabled by the AntialiasEnable bit in the **TriangleMode** register in addition to the AntialiasEnable bit in the **DrawTriangle** command.
- The antialiasing quality is taken from the **TriangleMode** register and not the **DrawTriangle** command.
- The Render command issued by Gamma to control the rasterizer chip has the data field set to sensible values rather than using the given value in the **Draw*** command directly.
- All texture, specular texture, diffuse texture, fog and subpixel correction operations are qualified by *both* the corresponding bits in the **DeltaMode** register and the **Draw*** command.

The vertex information is held in three areas V0, V1 and V2, each area has storage for the 16 words assigned to the usual parameters. The input format for each parameter is also shown in fixed point format (*us* is unsigned and *s* is 2's complement) and floating point format:

| Offset | Category | Parameter | Fixed Point Format | Floating Point Range |
|---|---|---|---|---|
| 0 | | s | 2.30 s[1] | -1.0…1.0[2] |
| 1 | Texture | t | 2.30 s | -1.0…1.0 |
| 2 | | q | 2.30 s | -1.0…1.0 |
| 3 | | Ks | 2.22 us | 0.0…2.0 |
| 4 | | Kd | 2.22 us | 0.0…2.0 |
| 5 | | red | 1.30 us | 0.0…1.0 |
| 6 | Color | green | 1.30 us | 0.0…1.0 |
| 7 | | blue | 1.30 us | 0.0…1.0 |
| 8 | | alpha | 1.30 us | 0.0…1.0 |
| 9 | Fog | f | 10.22 s | -512.0…512.0 |
| 10 | | x | 16.16 s | -32K…+32K[3] [4] |
| 11 | Coordinate | y | 16.16 s | -32K…+32K |
| 12 | | z | 1.30us | 0.0…1.0 |
| 13 | | | | |
| 14 | Reserved | | | |
| 15 | | | | |

**Table 11.1 Compatibility with GLINT Delta**

Each value can be written to two addresses:

- The **V*n*Fixed*** address treats the data value as a fixed point number with the format associated with the parameter. The fixed point value is converted to IEEE floating point format and clamped (if enabled) to lie in the range assigned to this parameter.

The unused bits in some of the formats are zero extended when the format is unsigned or sign extended when the format is signed, prior to converting to floating point format.

- The **V*n*Float*** address treats the data value as a IEEE floating point number and clamp it (if enabled) to lie in the range assigned to this parameter.

The values read back from the vertex store via the readback mechanism are the clamped floating point version of the number written (in either format).

No parameters are corrupted by the calculations so parameter sharing between primitives is simply achieved by not re-loading those parameters. For example if the first triangle in a tri-strip is loaded into V0, V1 and V2, then the next triangle will load V0, the next V1, etc..

[1]This is the range when Normalize is not used. When Normalize is enabled the fixed point format can be anything, providing it is the same for the s, t and q parameters. The numbers will be interpreted as if they had 2.30 format for the purpose of conversion to floating point. If the fixed point format (2.30) is different from what the user had in mind then the input values are just pre-scaled by a fixed amount (i.e. the difference in binary point positions) prior to conversion.

[2]This is the range when Normalize is not used. When Normalize is enabled the range is extended to $2^{\pm 32}$ approximately. This also applies to the t and q values as well.

[3]The normal range here is limited by the size of the screen.

[4]K = 1024.

Lines are handled slightly differently in that only V0 and V1 are used. The direction the line is drawn is defined as part of the command so the line runs from V0 to V1 (**DrawLine01**), or V1 to V0 (**DrawLine10**). To draw a polyline the initial segment starts as V0 to V1 and then V1 to V0 (new parameters loaded), etc..

The texture parameters (S, T and Q) are handled differently to the other parameters as their dynamic range is much more variable from the application's view point, but must be constrained to get the best out of the finite precision DDA and perspective division hardware available in GLINT. Any operation on the texture coordinates before they are used is controlled by the TextureParameterMode in the **DeltaMode** register. The options are NoClamp, Clamp or Normalize . The NoClamp and Clamp work the same as for the other parameters. The Normalize option finds the maximum absolute value of the texture S, T and Q values for the primitive and normalizes all the value to lie in the range -1.0 … 1.0 inclusive prior to being used in the set-up calculations. Note that the texture values in the vertex store are *not* changed by the Normalize option.

In addition to the **DeltaMode** register the unit's operation is also influenced by the bits in the data field associated with one of the **Draw*** commands. The data field for the **Draw*** command is identical to that defined for the **Render** command, and provides the data field whenever a **Render** command is sent to GLINT to initiate rendering (nonsensical combinations default to reasonable combinations).

See the GLINT Delta documentation for more specific programming details.

# 12. Multi-GLINT Support

Gamma can support up to two GLINTs and broadcast the commands and register loads to any combination of the two GLINTs at the same time.

The selection of which GLINT to direct the broadcast at is determined by the **BroadcastMask** register. The data field has bit 0 assigned to GLINT0, bit 1 to GLINT1, etc. When one of these bits are set the corresponding GLINT is included in the broadcast.

The assumed mode for the two GLINTs is scan line interleaved so it is very natural for both GLINTs to get the same command stream. The obvious exception is the initial scan line ownership set up.

A **BroadcastMask** value of zero results in all subsequent commands being discarded until the broadcast mask is changed.

# Appendix A PCI Related Registers

This Appendix summarizes the PCI registers which directly effect programming Gamma rather than those registers concerned with set up and configuration. The Gamma Hardware Reference Manual gives more details.

| Address | PCI Register | Description |
|---|---|---|
| 0x0000. 0C00 | PageTableAddr | This register holds the physical base address of the page table used during the logical to physical mapping. The base address is given as a byte address but must be on a 32 bit word boundary. |
| 0x0000. 0C08 | PageTableLength | This register holds the length, in multiples of 1K entries of the page table. It is only used for some basic range checking. The register is 24 bits wide. |
| 0x0000. 0C38 | DelayTimer | This register, when written to, starts a timer. When the timer decrements down to one an interrupt is generated and the timer stops. Writing zero aborts the timer with no interrupt being generated (unless it has already occurred). Reading this register returns the current timer value. |
| 0x0000. 0C40 | CommandMode | This register holds some basic mode information. It is described in a later table. |
| 0x0000. 0028 | DMAAddress | This register holds the address a DMA will start from when the Operation mode (held in the *CommandMode* register) is 'default'. When the mode is not 'default' then this register can be read and written as normal, but is not used internally. |
| 0x0000. 0030 | DMACount | This register holds the number of words in the DMA transfer. The transfer size is held as a 24 bit number. When the Operation mode (held in the *CommandMode* register) is 'default' a write to this register will start a DMA transfer. When the mode is not 'default' then this register can be read and written as normal, but is not used internally and does not start a DMA transfer. |
| 0x0000. 0C58 | CommandError | This register holds the error bits for each type of error which can be detected by the Gamma core. |
| 0x0000. 0C48 | CommandInterruptEnable | This register holds the enables for each type of interrupt the Gamma core can generate. |
| 0x0000. 0C50 | CommandInterruptStatus | This register reports on which interrupt(s) have been generated. |
| 0x0000. 0C60 | CommandStatus | This register holds various status information such as when a DMA is busy. |
| 0x0000. 0C68… 0C90 | FaultingAddress[6] | These register holds the logical page which caused the page fault interrupt or the one of the three errors detected during logical to physical address translation for each region. |
| 0x0000. 0C98 | FeedbackSelectCount | This register holds the number of words written to memory when the Feedback output mode is used, or the number of hit records when in Select output mode is used. The most significant 8 bits are reserved/used as error flags. |
| 0x0000. 0CB8 | GammaProcessorMode | This register enables (0) or disables (1) the use of the Delta Unit in Gamma. For the GLINT 500TX and GLINT MX this will be set to 0. |

**Table A1 PCI Related Registers**

| Bit No. | Name | Description |
|---|---|---|
| 0, 1 | Operation | This field determines the major operation mode of the DMA controller. It has the values:<br>0: Default operation enabled after reset. The DMA is initiated by writes to the PCI *DMAAddress* and *DMACount* registers. Any **DMAAddr** and **DMACount** tags found in the input FIFO are discarded.<br>1: The DMA is initiated by the **DMAAddr** and **DMACount** tags in the input FIFO. Writes to the PCI *DMAAddress* and *DMACount* registers are ignored. |
| 2 | LogicalAddressing | When set causes the addresses generated by the DMA controller to be translated into physical addresses via a page table. |
| 3 | AbortOutputDMA | When this bit is set any current (or future) Output DMA is aborted (linear or rectangular). |
| 4 | Reserved | |
| 5 | Reserved | |
| 6 | AbortInputDMA | When this bit is set any current (or future) Input DMA is aborted (normal DMA, hierarchical or rectangular). |

**Table A2 PCI *CommandMode* Register Fields**

| Bit No. | Name | Description |
|---|---|---|
| 0 | StackUnderflow | This is set whenever a **DMAReturn** is attempted from the InputFIFO |
| 1 | StackOverflow | This is set whenever a **DMACount** or **DMACall** tag in a DMA buffer are nested more than 8 deep. |
| 2 | DMAOverrun | This is set whenever data beyond the DMA buffer is needed to fulfill the requirements of the last tag in the DMA buffer. |
| 3 | Reserved | |
| 4 | PageMappingFaultCommand | This is set whenever the logical address exceeds the translation range of the Page Mapping Table for the appropriate region. |
| 5 | PageMappingFaultVertex | |
| 6 | Reserved | |
| 7 | Reserved | |
| 8 | PageMappingFaultWrite | |
| 9 | Reserved | |
| 10 | PageFaultReadAccessCommand | This is set whenever a read access is made to a page marked as not supporting read accesses. |
| 11 | PageFaultReadAccessVertex | |
| 12 | Reserved | |
| 13 | Reserved | |
| 14 | PageFaultReadAccessWrite | |
| 15 | Reserved | |
| 16 | PageFaultWriteAccessComman d | This is set whenever a write access is made to a page marked as not supporting write accesses. |
| 17 | PageFaultWriteAccessVertex | |
| 18 | Reserved | |
| 19 | Reserved | |
| 20 | PageFaultWriteAccessWrite | |
| 21 | Reserved | |
| 22 | IllegalDMATag | This is set whenever a DMA related tag is detected not in a tag/data pair. |

**Table A3 PCI *CommandError* Register Fields**

The PCI *CommandInterruptEnable* register has the following fields to enable and disable the various interrupt sources. The *CommandInterruptStatus* register has the same fields to identify the actual source of the interrupt. Writing a one to a field in the *CommandInterruptStatus* clears the corresponding interrupt.

| Bit No. | Name | Description |
|---|---|---|
| 0 | FIFOQueuedCommandDMA | This field determines when interrupts are generated during the Queued DMA mode of operation. When this bit is set the interrupts occur after every DMA has finished. |
| 1 | OutputDMA | When set causes an interrupt to be generated whenever the Output DMA controller finishes. |
| 2 | Command | Enables the **CommandInterrupt** tag to generate an interrupt. |
| 3 | Timer | Enables the DelayTimer to generate an interrupt when it has counted down to one. |
| 4 | CommandError | Enables the occurrence any of the errors detected by the command unit to generate an interrupt. |
| 5 | Reserved | |
| 6 | Reserved | |
| 7 | Reserved | |
| 8 | PageFaultCommand | Enables an interrupt to be generated when a non resident page is accessed in the Command region. |
| 9 | PageFaultVertex | Enables an interrupt to be generated when a non resident page is accessed in the Vertex region. |
| 10 | Reserved | |
| 11 | Reserved | |
| 12 | PageFaultWrite | Enables an interrupt to be generated when a non resident page is accessed in the Write region. |
| 13 | Reserved | |

**Table A4 PCI** ***CommandInterruptEnable*** **Register Fields**

| Bit No. | Name | Description |
|---|---|---|
| 0 | CommandDMABusy | This is set whenever the command stream DMA controller is busy or a RectangleRead is in progress. |
| 1 | OutputDMABusy | This is set whenever the OutputDMA controller is busy or a RectangleWrite is in progress. |
| 2 | Reserved | |
| 3 | Reserved | |

**Table A5 PCI** ***CommandStatus*** **Register Fields**

# Appendix B Tag Values (Numerical Order)

| Tags | Major Group (Hex) | Offset (Hex) | Readback | Notes |
|---|---|---|---|---|
| ContextDump | 1B | 8 | | Command |
| ContextRestore | 1B | 9 | | Command |
| ContextData | 1B | A | | |
| FeedbackToken | 1F | 0 | | |
| FeedbackX | 1F | 1 | | |
| FeedbackY | 1F | 2 | | |
| FeedbackZ | 1F | 3 | | |
| FeedbackW | 1F | 4 | | |
| FeedbackRed | 1F | 5 | | |
| FeedbackGreen | 1F | 6 | | |
| FeedbackBlue | 1F | 7 | | |
| FeedbackAlpha | 1F | 8 | | |
| FeedbackS | 1F | 9 | | |
| FeedbackT | 1F | A | | |
| FeedbackR | 1F | B | | |
| FeedbackQ | 1F | C | | |
| SelectRecord | 1F | D | | Command |
| PassThrough | 1F | E | | Command |
| EndOfFeedback | 1F | F | | Command |
| V0FixedS | 20 | 0 | • | |
| V0FixedT | 20 | 1 | • | |
| V0FixedQ | 20 | 2 | • | |
| V0FixedKs | 20 | 3 | • | |
| V0FixedKd | 20 | 4 | • | |
| V0FixedR | 20 | 5 | • | |
| V0FixedG | 20 | 6 | • | |
| V0FixedB | 20 | 7 | • | |
| V0FixedA | 20 | 8 | • | |
| V0FixedF | 20 | 9 | • | |
| V0FixedX | 20 | A | • | |
| V0FixedY | 20 | B | • | |
| V0FixedZ | 20 | C | • | |
| V1FixedS | 21 | 0 | • | |
| V1FixedT | 21 | 1 | • | |
| V1FixedQ | 21 | 2 | • | |
| V1FixedKs | 21 | 3 | • | |
| V1FixedKd | 21 | 4 | • | |
| V1FixedR | 21 | 5 | • | |
| V1FixedG | 21 | 6 | • | |
| V1FixedB | 21 | 7 | • | |
| V1FixedA | 21 | 8 | • | |
| V1FixedF | 21 | 9 | • | |
| V1FixedX | 21 | A | • | |
| V1FixedY | 21 | B | • | |
| V1FixedZ | 21 | C | • | |
| V2FixedS | 22 | 0 | • | |
| V2FixedT | 22 | 1 | • | |
| V2FixedQ | 22 | 2 | • | |
| V2FixedKs | 22 | 3 | • | |
| V2FixedKd | 22 | 4 | • | |
| V2FixedR | 22 | 5 | • | |

| | | | | |
|---|---|---|---|---|
| V2FixedG | 22 | 6 | • | |
| V2FixedB | 22 | 7 | • | |
| V2FixedA | 22 | 8 | • | |
| V2FixedF | 22 | 9 | • | |
| V2FixedX | 22 | A | • | |
| V2FixedY | 22 | B | • | |
| V2FixedZ | 22 | C | • | |
| V0FloatS | 23 | 0 | • | |
| V0FloatT | 23 | 1 | • | |
| V0FloatQ | 23 | 2 | • | |
| V0FloatKs | 23 | 3 | • | |
| V0FloatKd | 23 | 4 | • | |
| V0FloatR | 23 | 5 | • | |
| V0FloatG | 23 | 6 | • | |
| V0FloatB | 23 | 7 | • | |
| V0FloatA | 23 | 8 | • | |
| V0FloatF | 23 | 9 | • | |
| V0FloatX | 23 | A | • | |
| V0FloatY | 23 | B | • | |
| V0FloatZ | 23 | C | • | |
| V1FloatS | 24 | 0 | • | |
| V1FloatT | 24 | 1 | • | |
| V1FloatQ | 24 | 2 | • | |
| V1FloatKs | 24 | 3 | • | |
| V1FloatKd | 24 | 4 | • | |
| V1FloatR | 24 | 5 | • | |
| V1FloatG | 24 | 6 | • | |
| V1FloatB | 24 | 7 | • | |
| V1FloatA | 24 | 8 | • | |
| V1FloatF | 24 | 9 | • | |
| V1FloatX | 24 | A | • | |
| V1FloatY | 24 | B | • | |
| V1FloatZ | 24 | C | • | |
| V2FloatS | 25 | 0 | • | |
| V2FloatT | 25 | 1 | • | |
| V2FloatQ | 25 | 2 | • | |
| V2FloatKs | 25 | 3 | • | |
| V2FloatKd | 25 | 4 | • | |
| V2FloatR | 25 | 5 | • | |
| V2FloatG | 25 | 6 | • | |
| V2FloatB | 25 | 7 | • | |
| V2FloatA | 25 | 8 | • | |
| V2FloatF | 25 | 9 | • | |
| V2FloatX | 25 | A | • | |
| V2FloatY | 25 | B | • | |
| V2FloatZ | 25 | C | • | |
| DeltaMode | 26 | 0 | • | |
| DrawTriangle | 26 | 1 | | Command |
| RepeatTriangle | 26 | 2 | | Command |
| DrawLine01 | 26 | 3 | | Command |
| DrawLine10 | 26 | 4 | | Command |
| RepeatLine | 26 | 5 | | Command |
| EpilogueTag | 26 | D | • | |
| EpilogueData | 26 | E | • | |

| | | | | |
|---|---|---|---|---|
| BroadcastMask | 26 | F | • | |
| XBias | 29 | 0 | • | |
| YBias | 29 | 1 | • | |
| PointMode | 29 | 2 | • | |
| PointSize | 29 | 3 | • | |
| AAPointSize | 29 | 4 | • | |
| LineMode | 29 | 5 | • | |
| LineWidth | 29 | 6 | • | |
| LineWidthOffset | 29 | 7 | • | |
| AALineWidth | 29 | 8 | • | |
| TriangleMode | 29 | 9 | • | |
| RectangleMode | 29 | A | • | |
| RectangleWidth | 29 | B | • | |
| RectangleHeight | 29 | C | • | |
| Rectangle2DMode | 29 | D | • | |
| Rectangle2DControl | 29 | E | • | |
| TransformMode | 2A | 1 | • | |
| GeometryMode | 2A | 2 | • | |
| NormalizeMode | 2A | 3 | • | |
| LightingMode | 2A | 4 | • | |
| ColorMaterialMode | 2A | 5 | • | |
| MaterialMode | 2A | 6 | • | |
| SelectResult | 2B | 0 | | Command |
| Begin | 2B | 2 | | Command |
| End | 2B | 3 | | Command |
| EdgeFlag | 2B | 4 | • | |
| ObjectIDValue | 2B | 5 | • | |
| IncrementObjectID | 2B | 6 | | Command |
| TransformCurrent | 2B | 7 | | Command |
| SaveCurrent | 2B | 9 | | Command |
| RestoreCurrent | 2B | A | | Command |
| InitNames | 2B | B | | Command |
| PushName | 2B | C | | Command |
| PopName | 2B | D | | Command |
| LoadName | 2B | E | | Command |
| GeomRectangle | 2D | 4 | | Command |
| DrawRectangle2D | 2F | 4 | | Command |
| Nz | 30 | 0 | • | |
| Ny | 30 | 1 | • | |
| Nx | 30 | 2 | • | Trigger |
| Ca | 30 | 3 | • | |
| Cb | 30 | 4 | • | |
| Cg | 30 | 5 | • | |
| Cr3 | 30 | 6 | • | Trigger |
| Cr4 | 30 | 7 | • | Trigger |
| Tt2 | 30 | 8 | • | |
| Ts2 | 30 | 9 | • | Trigger |
| Vw | 30 | A | | |
| Vz | 30 | B | | |
| Vy | 30 | C | | |
| Vx2 | 30 | D | | Trigger |
| Vx3 | 30 | E | | Trigger |
| Vx4 | 30 | F | | Trigger |
| FNz | 31 | 0 | • | |

| | | | | |
|---|---|---|---|---|
| FNy | 31 | 1 | • | |
| FNx | 31 | 2 | • | Trigger |
| PackedColor3 | 31 | 3 | | Trigger |
| PackedColor4 | 31 | 4 | | Trigger |
| Tq4 | 31 | 5 | • | |
| Tr4 | 31 | 6 | • | |
| Tt4 | 31 | 7 | • | |
| Ts4 | 31 | 8 | • | Trigger |
| RPw | 31 | 9 | | |
| RPz | 31 | A | | |
| RPy | 31 | B | | |
| RPx2 | 31 | C | | Trigger |
| RPx3 | 31 | D | | Trigger |
| RPx4 | 31 | E | | Trigger |
| Ts1 | 31 | F | • | Trigger |
| ModelViewMatrix[16] | 32 | 0…F | • | |
| ModelViewProjectionMatrix[16] | 33 | 0…F | • | |
| NormalMatrix[9] | 34 | 0…8 | • | |
| TextureMatrix[16] | 35 | 0…F | • | |
| TexGen[16] | 36 | 0…F | • | |
| ViewPortScaleX | 37 | 0 | • | |
| ViewPortScaleY | 37 | 1 | • | |
| ViewPortScaleZ | 37 | 2 | • | |
| ViewPortOffsetX | 37 | 3 | • | |
| ViewPortOffsetY | 37 | 4 | • | |
| ViewPortOffsetZ | 37 | 5 | • | |
| FogDensity | 37 | 6 | • | |
| FogScale | 37 | 7 | • | |
| FogEnd | 37 | 8 | • | |
| PolygonOffsetFactor | 37 | 9 | • | |
| PolygonOffsetBias | 37 | A | • | |
| LineClipLengthThreshold | 37 | B | • | |
| TriangleClipAreaThreshold | 37 | C | • | |
| RasterPosXIncrement | 37 | D | • | |
| RasterPosYIncrement | 37 | E | • | |
| UserClip0X | 38 | 0 | • | |
| UserClip0Y | 38 | 1 | • | |
| UserClip0Z | 38 | 2 | • | |
| UserClip0W | 38 | 3 | • | |
| UserClip1X | 38 | 4 | • | |
| UserClip1Y | 38 | 5 | • | |
| UserClip1Z | 38 | 6 | • | |
| UserClip1W | 38 | 7 | • | |
| UserClip2X | 38 | 8 | • | |
| UserClip2Y | 38 | 9 | • | |
| UserClip2Z | 38 | A | • | |
| UserClip2W | 38 | B | • | |
| UserClip3X | 38 | C | • | |
| UserClip3Y | 38 | D | • | |
| UserClip3Z | 38 | E | • | |
| UserClip3W | 38 | F | • | |
| UserClip4X | 39 | 0 | • | |
| UserClip4Y | 39 | 1 | • | |

| | | | | |
|---|---|---|---|---|
| UserClip4Z | 39 | 2 | • | |
| UserClip4W | 39 | 3 | • | |
| UserClip5X | 39 | 4 | • | |
| UserClip5Y | 39 | 5 | • | |
| UserClip5Z | 39 | 6 | • | |
| UserClip5W | 39 | 7 | • | |
| RasterPosXOffset | 39 | D | • | |
| RasterPosYOffset | 39 | E | • | |
| AttenuationCutOff | 39 | F | • | |
| Light0Mode | 3A | 0 | • | |
| Light0AmbientIntensityRed | 3A | 1 | • | |
| Light0AmbientIntensityGreen | 3A | 2 | • | |
| Light0AmbientIntensityBlue | 3A | 3 | • | |
| Light0DiffuseIntensityRed | 3A | 4 | • | |
| Light0DiffuseIntensityGreen | 3A | 5 | • | |
| Light0DiffuseIntensityBlue | 3A | 6 | • | |
| Light0SpecularIntensityRed | 3A | 7 | • | |
| Light0SpecularIntensityGreen | 3A | 8 | • | |
| Light0SpecularIntensityBlue | 3A | 9 | • | |
| Light0PositionX | 3A | A | • | |
| Light0PositionY | 3A | B | • | |
| Light0PositionZ | 3A | C | • | |
| Light0PositionW | 3A | D | • | |
| Light0SpotlightDirectionX | 3A | E | • | |
| Light0SpotlightDirectionY | 3A | F | • | |
| Light0SpotlightDirectionZ | 3B | 0 | • | |
| Light0SpotlightExponent | 3B | 1 | • | |
| Light0CosSpotlightCutoffAngle | 3B | 2 | • | |
| Light0ConstantAttenuation | 3B | 3 | • | |
| Light0LinearAttenuation | 3B | 4 | • | |
| Light0QuadraticAttenuation | 3B | 5 | • | |
| Light1Mode | 3B | 6 | • | |
| Light1AmbientIntensityRed | 3B | 7 | • | |
| Light1AmbientIntensityGreen | 3B | 8 | • | |
| Light1AmbientIntensityBlue | 3B | 9 | • | |
| Light1DiffuseIntensityRed | 3B | A | • | |
| Light1DiffuseIntensityGreen | 3B | B | • | |
| Light1DiffuseIntensityBlue | 3B | C | • | |
| Light1SpecularIntensityRed | 3B | D | • | |
| Light1SpecularIntensityGreen | 3B | E | • | |
| Light1SpecularIntensityBlue | 3B | F | • | |
| Light1PositionX | 3C | 0 | • | |
| Light1PositionY | 3C | 1 | • | |
| Light1PositionZ | 3C | 2 | • | |
| Light1PositionW | 3C | 3 | • | |
| Light1SpotlightDirectionX | 3C | 4 | • | |
| Light1SpotlightDirectionY | 3C | 5 | • | |
| Light1SpotlightDirectionZ | 3C | 6 | • | |
| Light1SpotlightExponent | 3C | 7 | • | |
| Light1CosSpotlightCutoffAngle | 3C | 8 | • | |
| Light1ConstantAttenuation | 3C | 9 | • | |
| Light1LinearAttenuation | 3C | A | • | |
| Light1QuadraticAttenuation | 3C | B | • | |
| Light2Mode | 3C | C | • | |

| | | | | |
|---|---|---|---|---|
| Light2AmbientIntensityRed | 3C | D | • | |
| Light2AmbientIntensityGreen | 3C | E | • | |
| Light2AmbientIntensityBlue | 3C | F | • | |
| Light2DiffuseIntensityRed | 3D | 0 | • | |
| Light2DiffuseIntensityGreen | 3D | 1 | • | |
| Light2DiffuseIntensityBlue | 3D | 2 | • | |
| Light2SpecularIntensityRed | 3D | 3 | • | |
| Light2SpecularIntensityGreen | 3D | 4 | • | |
| Light2SpecularIntensityBlue | 3D | 5 | • | |
| Light2PositionX | 3D | 6 | • | |
| Light2PositionY | 3D | 7 | • | |
| Light2PositionZ | 3D | 8 | • | |
| Light2PositionW | 3D | 9 | • | |
| Light2SpotlightDirectionX | 3D | A | • | |
| Light2SpotlightDirectionY | 3D | B | • | |
| Light2SpotlightDirectionZ | 3D | C | • | |
| Light2SpotlightExponent | 3D | D | • | |
| Light2CosSpotlightCutoffAngle | 3D | E | • | |
| Light2ConstantAttenuation | 3D | F | • | |
| Light2LinearAttenuation | 3E | 0 | • | |
| Light2QuadraticAttenuation | 3E | 1 | • | |
| Light3Mode | 3E | 2 | • | |
| Light3AmbientIntensityRed | 3E | 3 | • | |
| Light3AmbientIntensityGreen | 3E | 4 | • | |
| Light3AmbientIntensityBlue | 3E | 5 | • | |
| Light3DiffuseIntensityRed | 3E | 6 | • | |
| Light3DiffuseIntensityGreen | 3E | 7 | • | |
| Light3DiffuseIntensityBlue | 3E | 8 | • | |
| Light3SpecularIntensityRed | 3E | 9 | • | |
| Light3SpecularIntensityGreen | 3E | A | • | |
| Light3SpecularIntensityBlue | 3E | B | • | |
| Light3PositionX | 3E | C | • | |
| Light3PositionY | 3E | D | • | |
| Light3PositionZ | 3E | E | • | |
| Light3PositionW | 3E | F | • | |
| Light3SpotlightDirectionX | 3F | 0 | • | |
| Light3SpotlightDirectionY | 3F | 1 | • | |
| Light3SpotlightDirectionZ | 3F | 2 | • | |
| Light3SpotlightExponent | 3F | 3 | • | |
| Light3CosSpotlightCutoffAngle | 3F | 4 | • | |
| Light3ConstantAttenuation | 3F | 5 | • | |
| Light3LinearAttenuation | 3F | 6 | • | |
| Light3QuadraticAttenuation | 3F | 7 | • | |
| Light4Mode | 3F | 8 | • | |
| Light4AmbientIntensityRed | 3F | 9 | • | |
| Light4AmbientIntensityGreen | 3F | A | • | |
| Light4AmbientIntensityBlue | 3F | B | • | |
| Light4DiffuseIntensityRed | 3F | C | • | |
| Light4DiffuseIntensityGreen | 3F | D | • | |
| Light4DiffuseIntensityBlue | 3F | E | • | |
| Light4SpecularIntensityRed | 3F | F | • | |
| Light4SpecularIntensityGreen | 40 | 0 | • | |
| Light4SpecularIntensityBlue | 40 | 1 | • | |
| Light4PositionX | 40 | 2 | • | |

| | | | | |
|---|---|---|---|---|
| Light4PositionY | 40 | 3 | • | |
| Light4PositionZ | 40 | 4 | • | |
| Light4PositionW | 40 | 5 | • | |
| Light4SpotlightDirectionX | 40 | 6 | • | |
| Light4SpotlightDirectionY | 40 | 7 | • | |
| Light4SpotlightDirectionZ | 40 | 8 | • | |
| Light4SpotlightExponent | 40 | 9 | • | |
| Light4CosSpotlightCutoffAngle | 40 | A | • | |
| Light4ConstantAttenuation | 40 | B | • | |
| Light4LinearAttenuation | 40 | C | • | |
| Light4QuadraticAttenuation | 40 | D | • | |
| Light5Mode | 40 | E | • | |
| Light5AmbientIntensityRed | 40 | F | • | |
| Light5AmbientIntensityGreen | 41 | 0 | • | |
| Light5AmbientIntensityBlue | 41 | 1 | • | |
| Light5DiffuseIntensityRed | 41 | 2 | • | |
| Light5DiffuseIntensityGreen | 41 | 3 | • | |
| Light5DiffuseIntensityBlue | 41 | 4 | • | |
| Light5SpecularIntensityRed | 41 | 5 | • | |
| Light5SpecularIntensityGreen | 41 | 6 | • | |
| Light5SpecularIntensityBlue | 41 | 7 | • | |
| Light5PositionX | 41 | 8 | • | |
| Light5PositionY | 41 | 9 | • | |
| Light5PositionZ | 41 | A | • | |
| Light5PositionW | 41 | B | • | |
| Light5SpotlightDirectionX | 41 | C | • | |
| Light5SpotlightDirectionY | 41 | D | • | |
| Light5SpotlightDirectionZ | 41 | E | • | |
| Light5SpotlightExponent | 41 | F | • | |
| Light5CosSpotlightCutoffAngle | 42 | 0 | • | |
| Light5ConstantAttenuation | 42 | 1 | • | |
| Light5LinearAttenuation | 42 | 2 | • | |
| Light5QuadraticAttenuation | 42 | 3 | • | |
| Light6Mode | 42 | 4 | • | |
| Light6AmbientIntensityRed | 42 | 5 | • | |
| Light6AmbientIntensityGreen | 42 | 6 | • | |
| Light6AmbientIntensityBlue | 42 | 7 | • | |
| Light6DiffuseIntensityRed | 42 | 8 | • | |
| Light6DiffuseIntensityGreen | 42 | 9 | • | |
| Light6DiffuseIntensityBlue | 42 | A | • | |
| Light6SpecularIntensityRed | 42 | B | • | |
| Light6SpecularIntensityGreen | 42 | C | • | |
| Light6SpecularIntensityBlue | 42 | D | • | |
| Light6PositionX | 42 | E | • | |
| Light6PositionY | 42 | F | • | |
| Light6PositionZ | 43 | 0 | • | |
| Light6PositionW | 43 | 1 | • | |
| Light6SpotlightDirectionX | 43 | 2 | • | |
| Light6SpotlightDirectionY | 43 | 3 | • | |
| Light6SpotlightDirectionZ | 43 | 4 | • | |
| Light6SpotlightExponent | 43 | 5 | • | |
| Light6CosSpotlightCutoffAngle | 43 | 6 | • | |
| Light6ConstantAttenuation | 43 | 7 | • | |
| Light6LinearAttenuation | 43 | 8 | • | |

| | | | | |
|---|---|---|---|---|
| Light6QuadraticAttenuation | 43 | 9 | • | |
| Light7Mode | 43 | A | • | |
| Light7AmbientIntensityRed | 43 | B | • | |
| Light7AmbientIntensityGreen | 43 | C | • | |
| Light7AmbientIntensityBlue | 43 | D | • | |
| Light7DiffuseIntensityRed | 43 | E | • | |
| Light7DiffuseIntensityGreen | 43 | F | • | |
| Light7DiffuseIntensityBlue | 44 | 0 | • | |
| Light7SpecularIntensityRed | 44 | 1 | • | |
| Light7SpecularIntensityGreen | 44 | 2 | • | |
| Light7SpecularIntensityBlue | 44 | 3 | • | |
| Light7PositionX | 44 | 4 | • | |
| Light7PositionY | 44 | 5 | • | |
| Light7PositionZ | 44 | 6 | • | |
| Light7PositionW | 44 | 7 | • | |
| Light7SpotlightDirectionX | 44 | 8 | • | |
| Light7SpotlightDirectionY | 44 | 9 | • | |
| Light7SpotlightDirectionZ | 44 | A | • | |
| Light7SpotlightExponent | 44 | B | • | |
| Light7CosSpotlightCutoffAngle | 44 | C | • | |
| Light7ConstantAttenuation | 44 | D | • | |
| Light7LinearAttenuation | 44 | E | • | |
| Light7QuadraticAttenuation | 44 | F | • | |
| Light8Mode | 45 | 0 | • | |
| Light8AmbientIntensityRed | 45 | 1 | • | |
| Light8AmbientIntensityGreen | 45 | 2 | • | |
| Light8AmbientIntensityBlue | 45 | 3 | • | |
| Light8DiffuseIntensityRed | 45 | 4 | • | |
| Light8DiffuseIntensityGreen | 45 | 5 | • | |
| Light8DiffuseIntensityBlue | 45 | 6 | • | |
| Light8SpecularIntensityRed | 45 | 7 | • | |
| Light8SpecularIntensityGreen | 45 | 8 | • | |
| Light8SpecularIntensityBlue | 45 | 9 | • | |
| Light8PositionX | 45 | A | • | |
| Light8PositionY | 45 | B | • | |
| Light8PositionZ | 45 | C | • | |
| Light8PositionW | 45 | D | • | |
| Light8SpotlightDirectionX | 45 | E | • | |
| Light8SpotlightDirectionY | 45 | F | • | |
| Light8SpotlightDirectionZ | 46 | 0 | • | |
| Light8SpotlightExponent | 46 | 1 | • | |
| Light8CosSpotlightCutoffAngle | 46 | 2 | • | |
| Light8ConstantAttenuation | 46 | 3 | • | |
| Light8LinearAttenuation | 46 | 4 | • | |
| Light8QuadraticAttenuation | 46 | 5 | • | |
| Light9Mode | 46 | 6 | • | |
| Light9AmbientIntensityRed | 46 | 7 | • | |
| Light9AmbientIntensityGreen | 46 | 8 | • | |
| Light9AmbientIntensityBlue | 46 | 9 | • | |
| Light9DiffuseIntensityRed | 46 | A | • | |
| Light9DiffuseIntensityGreen | 46 | B | • | |
| Light9DiffuseIntensityBlue | 46 | C | • | |
| Light9SpecularIntensityRed | 46 | D | • | |
| Light9SpecularIntensityGreen | 46 | E | • | |

| | | | | |
|---|---|---|---|---|
| Light9SpecularIntensityBlue | 46 | F | • | |
| Light9PositionX | 47 | 0 | • | |
| Light9PositionY | 47 | 1 | • | |
| Light9PositionZ | 47 | 2 | • | |
| Light9PositionW | 47 | 3 | • | |
| Light9SpotlightDirectionX | 47 | 4 | • | |
| Light9SpotlightDirectionY | 47 | 5 | • | |
| Light9SpotlightDirectionZ | 47 | 6 | • | |
| Light9SpotlightExponent | 47 | 7 | • | |
| Light9CosSpotlightCutoffAngle | 47 | 8 | • | |
| Light9ConstantAttenuation | 47 | 9 | • | |
| Light9LinearAttenuation | 47 | A | • | |
| Light9QuadraticAttenuation | 47 | B | • | |
| Light10Mode | 47 | C | • | |
| Light10AmbientIntensityRed | 47 | D | • | |
| Light10AmbientIntensityGreen | 47 | E | • | |
| Light10AmbientIntensityBlue | 47 | F | • | |
| Light10DiffuseIntensityRed | 48 | 0 | • | |
| Light10DiffuseIntensityGreen | 48 | 1 | • | |
| Light10DiffuseIntensityBlue | 48 | 2 | • | |
| Light10SpecularIntensityRed | 48 | 3 | • | |
| Light10SpecularIntensityGreen | 48 | 4 | • | |
| Light10SpecularIntensityBlue | 48 | 5 | • | |
| Light10PositionX | 48 | 6 | • | |
| Light10PositionY | 48 | 7 | • | |
| Light10PositionZ | 48 | 8 | • | |
| Light10PositionW | 48 | 9 | • | |
| Light10SpotlightDirectionX | 48 | A | • | |
| Light10SpotlightDirectionY | 48 | B | • | |
| Light10SpotlightDirectionZ | 48 | C | • | |
| Light10SpotlightExponent | 48 | D | • | |
| Light10CosSpotlightCutoffAngle | 48 | E | • | |
| Light10ConstantAttenuation | 48 | F | • | |
| Light10LinearAttenuation | 49 | 0 | • | |
| Light10QuadraticAttenuation | 49 | 1 | • | |
| Light11Mode | 49 | 2 | • | |
| Light11AmbientIntensityRed | 49 | 3 | • | |
| Light11AmbientIntensityGreen | 49 | 4 | • | |
| Light11AmbientIntensityBlue | 49 | 5 | • | |
| Light11DiffuseIntensityRed | 49 | 6 | • | |
| Light11DiffuseIntensityGreen | 49 | 7 | • | |
| Light11DiffuseIntensityBlue | 49 | 8 | • | |
| Light11SpecularIntensityRed | 49 | 9 | • | |
| Light11SpecularIntensityGreen | 49 | A | • | |
| Light11SpecularIntensityBlue | 49 | B | • | |
| Light11PositionX | 49 | C | • | |
| Light11PositionY | 49 | D | • | |
| Light11PositionZ | 49 | E | • | |
| Light11PositionW | 49 | F | • | |
| Light11SpotlightDirectionX | 4A | 0 | • | |
| Light11SpotlightDirectionY | 4A | 1 | • | |
| Light11SpotlightDirectionZ | 4A | 2 | • | |
| Light11SpotlightExponent | 4A | 3 | • | |
| Light11CosSpotlightCutoffAngle | 4A | 4 | • | |

| | | | | |
|---|---|---|---|---|
| Light11ConstantAttenuation | 4A | 5 | • | |
| Light11LinearAttenuation | 4A | 6 | • | |
| Light11QuadraticAttenuation | 4A | 7 | • | |
| Light12Mode | 4A | 8 | • | |
| Light12AmbientIntensityRed | 4A | 9 | • | |
| Light12AmbientIntensityGreen | 4A | A | • | |
| Light12AmbientIntensityBlue | 4A | B | • | |
| Light12DiffuseIntensityRed | 4A | C | • | |
| Light12DiffuseIntensityGreen | 4A | D | • | |
| Light12DiffuseIntensityBlue | 4A | E | • | |
| Light12SpecularIntensityRed | 4A | F | • | |
| Light12SpecularIntensityGreen | 4B | 0 | • | |
| Light12SpecularIntensityBlue | 4B | 1 | • | |
| Light12PositionX | 4B | 2 | • | |
| Light12PositionY | 4B | 3 | • | |
| Light12PositionZ | 4B | 4 | • | |
| Light12PositionW | 4B | 5 | • | |
| Light12SpotlightDirectionX | 4B | 6 | • | |
| Light12SpotlightDirectionY | 4B | 7 | • | |
| Light12SpotlightDirectionZ | 4B | 8 | • | |
| Light12SpotlightExponent | 4B | 9 | • | |
| Light12CosSpotlightCutoffAngle | 4B | A | • | |
| Light12ConstantAttenuation | 4B | B | • | |
| Light12LinearAttenuation | 4B | C | • | |
| Light12QuadraticAttenuation | 4B | D | • | |
| Light13Mode | 4B | E | • | |
| Light13AmbientIntensityRed | 4B | F | • | |
| Light13AmbientIntensityGreen | 4C | 0 | • | |
| Light13AmbientIntensityBlue | 4C | 1 | • | |
| Light13DiffuseIntensityRed | 4C | 2 | • | |
| Light13DiffuseIntensityGreen | 4C | 3 | • | |
| Light13DiffuseIntensityBlue | 4C | 4 | • | |
| Light13SpecularIntensityRed | 4C | 5 | • | |
| Light13SpecularIntensityGreen | 4C | 6 | • | |
| Light13SpecularIntensityBlue | 4C | 7 | • | |
| Light13PositionX | 4C | 8 | • | |
| Light13PositionY | 4C | 9 | • | |
| Light13PositionZ | 4C | A | • | |
| Light13PositionW | 4C | B | • | |
| Light13SpotlightDirectionX | 4C | C | • | |
| Light13SpotlightDirectionY | 4C | D | • | |
| Light13SpotlightDirectionZ | 4C | E | • | |
| Light13SpotlightExponent | 4C | F | • | |
| Light13CosSpotlightCutoffAngle | 4D | 0 | • | |
| Light13ConstantAttenuation | 4D | 1 | • | |
| Light13LinearAttenuation | 4D | 2 | • | |
| Light13QuadraticAttenuation | 4D | 3 | • | |
| Light14Mode | 4D | 4 | • | |
| Light14AmbientIntensityRed | 4D | 5 | • | |
| Light14AmbientIntensityGreen | 4D | 6 | • | |
| Light14AmbientIntensityBlue | 4D | 7 | • | |
| Light14DiffuseIntensityRed | 4D | 8 | • | |
| Light14DiffuseIntensityGreen | 4D | 9 | • | |
| Light14DiffuseIntensityBlue | 4D | A | • | |

| | | | | |
|---|---|---|---|---|
| Light14SpecularIntensityRed | 4D | B | • | |
| Light14SpecularIntensityGreen | 4D | C | • | |
| Light14SpecularIntensityBlue | 4D | D | • | |
| Light14PositionX | 4D | E | • | |
| Light14PositionY | 4D | F | • | |
| Light14PositionZ | 4E | 0 | • | |
| Light14PositionW | 4E | 1 | • | |
| Light14SpotlightDirectionX | 4E | 2 | • | |
| Light14SpotlightDirectionY | 4E | 3 | • | |
| Light14SpotlightDirectionZ | 4E | 4 | • | |
| Light14SpotlightExponent | 4E | 5 | • | |
| Light14CosSpotlightCutoffAngle | 4E | 6 | • | |
| Light14ConstantAttenuation | 4E | 7 | • | |
| Light14LinearAttenuation | 4E | 8 | • | |
| Light14QuadraticAttenuation | 4E | 9 | • | |
| Light15Mode | 4E | A | • | |
| Light15AmbientIntensityRed | 4E | B | • | |
| Light15AmbientIntensityGreen | 4E | C | • | |
| Light15AmbientIntensityBlue | 4E | D | • | |
| Light15DiffuseIntensityRed | 4E | E | • | |
| Light15DiffuseIntensityGreen | 4E | F | • | |
| Light15DiffuseIntensityBlue | 4F | 0 | • | |
| Light15SpecularIntensityRed | 4F | 1 | • | |
| Light15SpecularIntensityGreen | 4F | 2 | • | |
| Light15SpecularIntensityBlue | 4F | 3 | • | |
| Light15PositionX | 4F | 4 | • | |
| Light15PositionY | 4F | 5 | • | |
| Light15PositionZ | 4F | 6 | • | |
| Light15PositionW | 4F | 7 | • | |
| Light15SpotlightDirectionX | 4F | 8 | • | |
| Light15SpotlightDirectionY | 4F | 9 | • | |
| Light15SpotlightDirectionZ | 4F | A | • | |
| Light15SpotlightExponent | 4F | B | • | |
| Light15CosSpotlightCutoffAngle | 4F | C | • | |
| Light15ConstantAttenuation | 4F | D | • | |
| Light15LinearAttenuation | 4F | E | • | |
| Light15QuadraticAttenuation | 4F | F | • | |
| SceneAmbientColorRed | 50 | 0 | • | |
| SceneAmbientColorGreen | 50 | 1 | • | |
| SceneAmbientColorBlue | 50 | 2 | • | |
| FrontAmbientColorRed | 51 | 0 | • | |
| FrontAmbientColorGreen | 51 | 1 | • | |
| FrontAmbientColorBlue | 51 | 2 | • | |
| FrontDiffuseColorRed | 51 | 3 | • | |
| FrontDiffuseColorGreen | 51 | 4 | • | |
| FrontDiffuseColorBlue | 51 | 5 | • | |
| FrontAlpha | 51 | 6 | • | |
| FrontSpecularColorRed | 51 | 7 | • | |
| FrontSpecularColorGreen | 51 | 8 | • | |
| FrontSpecularColorBlue | 51 | 9 | • | |
| FrontEmissiveColorRed | 51 | A | • | |
| FrontEmissiveColorGreen | 51 | B | • | |
| FrontEmissiveColorBlue | 51 | C | • | |
| FrontSpecularExponent | 51 | D | • | |

| | | | | |
|---|---|---|---|---|
| BackAmbientColorRed | 52 | 0 | • | |
| BackAmbientColorGreen | 52 | 1 | • | |
| BackAmbientColorBlue | 52 | 2 | • | |
| BackDiffuseColorRed | 52 | 3 | • | |
| BackDiffuseColorGreen | 52 | 4 | • | |
| BackDiffuseColorBlue | 52 | 5 | • | |
| BackAlpha | 52 | 6 | • | |
| BackSpecularColorRed | 52 | 7 | • | |
| BackSpecularColorGreen | 52 | 8 | • | |
| BackSpecularColorBlue | 52 | 9 | • | |
| BackEmissiveColorRed | 52 | A | • | |
| BackEmissiveColorGreen | 52 | B | • | |
| BackEmissiveColorBlue | 52 | C | • | |
| BackSpecularExponent | 52 | D | • | |
| DMAAddr | 53 | 0 | | Tag/data pair format only. |
| DMACount | 53 | 1 | | Command. Tag/data pair format only. |
| CommandInterrupt | 53 | 2 | | Command. Tag/data pair format only. |
| DMACall | 53 | 3 | | Command. Tag/data pair format only. |
| DMAReturn | 53 | 4 | | Command. Tag/data pair format only. |
| DMARectangleRead | 53 | 5 | | Command. Tag/data pair format only. |
| DMARectangleReadAddress | 53 | 6 | • | Tag/data pair format only. |
| DMARectangleReadLinePitch | 53 | 7 | • | Tag/data pair format only. |
| DMARectangleReadTarget | 53 | 8 | • | Tag/data pair format only. |
| DMARectangleWrite | 53 | 9 | | Command. Tag/data pair format only. |
| DMARectangleWriteAddress | 53 | A | • | Tag/data pair format only. |
| DMARectangleWriteLinePitch | 53 | B | • | Tag/data pair format only. |
| DMAOutputAddress | 53 | C | | Tag/data pair format only. |
| DMAOutputCount | 53 | D | | Command. Tag/data pair format only. |
| DMAReadGLINTSource | 53 | E | • | Tag/data pair format only. |
| DMAFeedback | 54 | 2 | | Command. Tag/data pair format only. |
| TransformModeAnd | 55 | 0 | | |
| TransformModeOr | 55 | 1 | | |
| GeometryModeAnd | 55 | 2 | | |
| GeometryModeOr | 55 | 3 | | |
| NormalizeModeAnd | 55 | 4 | | |
| NormalizeModeOr | 55 | 5 | | |
| LightingModeAnd | 55 | 6 | | |
| LightingModeOr | 55 | 7 | | |
| ColorMaterialModeAnd | 55 | 8 | | |
| ColorMaterialModeOr | 55 | 9 | | |
| DeltaModeAnd | 55 | A | | |
| DeltaModeOr | 55 | B | | |
| PointModeAnd | 55 | C | | |
| PointModeOr | 55 | D | | |
| LineModeAnd | 55 | E | | |
| LineModeOr | 55 | F | | |
| TriangleModeAnd | 56 | 0 | | |
| TriangleModeOr | 56 | 1 | | |
| MaterialModeAnd | 56 | 2 | | |
| MaterialModeOr | 56 | 3 | | |
| WindowAnd | 57 | 0 | | |
| WindowOr | 57 | 1 | | |
| LBReadModeAnd | 57 | 2 | | |
| LBReadModeOr | 57 | 3 | | |

# Appendix C Tag Values (Alphabetical Order)

| Tags | Major Group (Hex) | Offset (Hex) | Readback | Notes |
|---|---|---|---|---|
| AALineWidth | 29 | 8 | • | |
| AAPointSize | 29 | 4 | • | |
| AttenuationCutOff | 39 | F | • | |
| BackAlpha | 52 | 6 | • | |
| BackAmbientColorBlue | 52 | 2 | • | |
| BackAmbientColorGreen | 52 | 1 | • | |
| BackAmbientColorRed | 52 | 0 | • | |
| BackDiffuseColorBlue | 52 | 5 | • | |
| BackDiffuseColorGreen | 52 | 4 | • | |
| BackDiffuseColorRed | 52 | 3 | • | |
| BackEmissiveColorBlue | 52 | C | • | |
| BackEmissiveColorGreen | 52 | B | • | |
| BackEmissiveColorRed | 52 | A | • | |
| BackSpecularColorBlue | 52 | 9 | • | |
| BackSpecularColorGreen | 52 | 8 | • | |
| BackSpecularColorRed | 52 | 7 | • | |
| BackSpecularExponent | 52 | D | • | |
| Begin | 2B | 2 | | Command |
| BroadcastMask | 26 | F | • | |
| Ca | 30 | 3 | • | |
| Cb | 30 | 4 | • | |
| Cg | 30 | 5 | • | |
| ColorMaterialMode | 2A | 5 | • | |
| ColorMaterialModeAnd | 55 | 8 | | |
| ColorMaterialModeOr | 55 | 9 | | |
| CommandInterrupt | 53 | 2 | | Command |
| ContextData | 1B | A | | |
| ContextDump | 1B | 8 | | Command |
| ContextRestore | 1B | 9 | | Command |
| Cr3 | 30 | 6 | • | Trigger |
| Cr4 | 30 | 7 | • | Trigger |
| DeltaMode | 26 | 0 | • | |
| DeltaModeAnd | 55 | A | | |
| DeltaModeOr | 55 | B | | |
| DMAAddr | 53 | 0 | | Tag/data pair format only. |
| DMACall | 53 | 3 | | Command. Tag/data pair format only. |
| DMACount | 53 | 1 | | Command Tag/data pair format only. |
| DMAFeedback | 54 | 2 | | Command. Tag/data pair format only. |
| DMAOutputAddress | 53 | C | | Tag/data pair format only. |
| DMAOutputCount | 53 | D | | Command. Tag/data pair format only. |
| DMAReadGLINTSource | 53 | E | • | Tag/data pair format only. |
| DMARectangleRead | 53 | 5 | | Command. Tag/data pair format only. |
| DMARectangleReadAddress | 53 | 6 | • | Tag/data pair format only. |
| DMARectangleReadLinePitch | 53 | 7 | • | Tag/data pair format only. |
| DMARectangleReadTarget | 53 | 8 | • | Tag/data pair format only. |
| DMARectangleWrite | 53 | 9 | | Command. Tag/data pair format only. |
| DMARectangleWriteAddress | 53 | A | • | Tag/data pair format only. |
| DMARectangleWriteLinePitch | 53 | B | • | Tag/data pair format only. |
| DMAReturn | 53 | 4 | | Command. Tag/data pair format only. |

| | | | | |
|---|---|---|---|---|
| DrawLine01 | 26 | 3 | | Command |
| DrawLine10 | 26 | 4 | | Command |
| DrawRectangle2D | 2F | 4 | | Command |
| DrawTriangle | 26 | 1 | | Command |
| EdgeFlag | 2B | 4 | • | |
| End | 2B | 3 | | Command |
| EndOfFeedback | 1F | F | | Command |
| EpilogueData | 26 | E | • | |
| EpilogueTag | 26 | D | • | |
| FeedbackAlpha | 1F | 8 | | |
| FeedbackBlue | 1F | 7 | | |
| FeedbackGreen | 1F | 6 | | |
| FeedbackQ | 1F | C | | |
| FeedbackR | 1F | B | | |
| FeedbackRed | 1F | 5 | | |
| FeedbackS | 1F | 9 | | |
| FeedbackT | 1F | A | | |
| FeedbackToken | 1F | 0 | | |
| FeedbackW | 1F | 4 | | |
| FeedbackX | 1F | 1 | | |
| FeedbackY | 1F | 2 | | |
| FeedbackZ | 1F | 3 | | |
| FNx | 31 | 2 | • | Trigger |
| FNy | 31 | 1 | • | |
| FNz | 31 | 0 | • | |
| FogDensity | 37 | 6 | • | |
| FogEnd | 37 | 8 | • | |
| FogScale | 37 | 7 | • | |
| FrontAlpha | 51 | 6 | • | |
| FrontAmbientColorBlue | 51 | 2 | • | |
| FrontAmbientColorGreen | 51 | 1 | • | |
| FrontAmbientColorRed | 51 | 0 | • | |
| FrontDiffuseColorBlue | 51 | 5 | • | |
| FrontDiffuseColorGreen | 51 | 4 | • | |
| FrontDiffuseColorRed | 51 | 3 | • | |
| FrontEmissiveColorBlue | 51 | C | • | |
| FrontEmissiveColorGreen | 51 | B | • | |
| FrontEmissiveColorRed | 51 | A | • | |
| FrontSpecularColorBlue | 51 | 9 | • | |
| FrontSpecularColorGreen | 51 | 8 | • | |
| FrontSpecularColorRed | 51 | 7 | • | |
| FrontSpecularExponent | 51 | D | • | |
| GeometryMode | 2A | 2 | • | |
| GeometryModeAnd | 55 | 2 | | |
| GeometryModeOr | 55 | 3 | | |
| GeomRectangle | 2D | 4 | | Command |
| IncrementObjectID | 2B | 6 | | Command |
| InitNames | 2B | B | | Command |
| LBReadModeAnd | 57 | 2 | | |
| LBReadModeOr | 57 | 3 | | |
| Light0AmbientIntensityBlue | 3A | 3 | • | |
| Light0AmbientIntensityGreen | 3A | 2 | • | |

| | | | | |
|---|---|---|---|---|
| Light0AmbientIntensityRed | 3A | 1 | • | |
| Light0ConstantAttenuation | 3B | 3 | • | |
| Light0CosSpotlightCutoffAngle | 3B | 2 | • | |
| Light0DiffuseIntensityBlue | 3A | 6 | • | |
| Light0DiffuseIntensityGreen | 3A | 5 | • | |
| Light0DiffuseIntensityRed | 3A | 4 | • | |
| Light0LinearAttenuation | 3B | 4 | • | |
| Light0Mode | 3A | 0 | • | |
| Light0PositionW | 3A | D | • | |
| Light0PositionX | 3A | A | • | |
| Light0PositionY | 3A | B | • | |
| Light0PositionZ | 3A | C | • | |
| Light0QuadraticAttenuation | 3B | 5 | • | |
| Light0SpecularIntensityBlue | 3A | 9 | • | |
| Light0SpecularIntensityGreen | 3A | 8 | • | |
| Light0SpecularIntensityRed | 3A | 7 | • | |
| Light0SpotlightDirectionX | 3A | E | • | |
| Light0SpotlightDirectionY | 3A | F | • | |
| Light0SpotlightDirectionZ | 3B | 0 | • | |
| Light0SpotlightExponent | 3B | 1 | • | |
| Light10AmbientIntensityBlue | 47 | F | • | |
| Light10AmbientIntensityGreen | 47 | E | • | |
| Light10AmbientIntensityRed | 47 | D | • | |
| Light10ConstantAttenuation | 48 | F | • | |
| Light10CosSpotlightCutoffAngle | 48 | E | • | |
| Light10DiffuseIntensityBlue | 48 | 2 | • | |
| Light10DiffuseIntensityGreen | 48 | 1 | • | |
| Light10DiffuseIntensityRed | 48 | 0 | • | |
| Light10LinearAttenuation | 49 | 0 | • | |
| Light10Mode | 47 | C | • | |
| Light10PositionW | 48 | 9 | • | |
| Light10PositionX | 48 | 6 | • | |
| Light10PositionY | 48 | 7 | • | |
| Light10PositionZ | 48 | 8 | • | |
| Light10QuadraticAttenuation | 49 | 1 | • | |
| Light10SpecularIntensityBlue | 48 | 5 | • | |
| Light10SpecularIntensityGreen | 48 | 4 | • | |
| Light10SpecularIntensityRed | 48 | 3 | • | |
| Light10SpotlightDirectionX | 48 | A | • | |
| Light10SpotlightDirectionY | 48 | B | • | |
| Light10SpotlightDirectionZ | 48 | C | • | |
| Light10SpotlightExponent | 48 | D | • | |
| Light11AmbientIntensityBlue | 49 | 5 | • | |
| Light11AmbientIntensityGreen | 49 | 4 | • | |
| Light11AmbientIntensityRed | 49 | 3 | • | |
| Light11ConstantAttenuation | 4A | 5 | • | |
| Light11CosSpotlightCutoffAngle | 4A | 4 | • | |
| Light11DiffuseIntensityBlue | 49 | 8 | • | |
| Light11DiffuseIntensityGreen | 49 | 7 | • | |
| Light11DiffuseIntensityRed | 49 | 6 | • | |
| Light11LinearAttenuation | 4A | 6 | • | |
| Light11Mode | 49 | 2 | • | |

| | | | | |
|---|---|---|---|---|
| Light11PositionW | 49 | F | • | |
| Light11PositionX | 49 | C | • | |
| Light11PositionY | 49 | D | • | |
| Light11PositionZ | 49 | E | • | |
| Light11QuadraticAttenuation | 4A | 7 | • | |
| Light11SpecularIntensityBlue | 49 | B | • | |
| Light11SpecularIntensityGreen | 49 | A | • | |
| Light11SpecularIntensityRed | 49 | 9 | • | |
| Light11SpotlightDirectionX | 4A | 0 | • | |
| Light11SpotlightDirectionY | 4A | 1 | • | |
| Light11SpotlightDirectionZ | 4A | 2 | • | |
| Light11SpotlightExponent | 4A | 3 | • | |
| Light12AmbientIntensityBlue | 4A | B | • | |
| Light12AmbientIntensityGreen | 4A | A | • | |
| Light12AmbientIntensityRed | 4A | 9 | • | |
| Light12ConstantAttenuation | 4B | B | • | |
| Light12CosSpotlightCutoffAngle | 4B | A | • | |
| Light12DiffuseIntensityBlue | 4A | E | • | |
| Light12DiffuseIntensityGreen | 4A | D | • | |
| Light12DiffuseIntensityRed | 4A | C | • | |
| Light12LinearAttenuation | 4B | C | • | |
| Light12Mode | 4A | 8 | • | |
| Light12PositionW | 4B | 5 | • | |
| Light12PositionX | 4B | 2 | • | |
| Light12PositionY | 4B | 3 | • | |
| Light12PositionZ | 4B | 4 | • | |
| Light12QuadraticAttenuation | 4B | D | • | |
| Light12SpecularIntensityBlue | 4B | 1 | • | |
| Light12SpecularIntensityGreen | 4B | 0 | • | |
| Light12SpecularIntensityRed | 4A | F | • | |
| Light12SpotlightDirectionX | 4B | 6 | • | |
| Light12SpotlightDirectionY | 4B | 7 | • | |
| Light12SpotlightDirectionZ | 4B | 8 | • | |
| Light12SpotlightExponent | 4B | 9 | • | |
| Light13AmbientIntensityBlue | 4C | 1 | • | |
| Light13AmbientIntensityGreen | 4C | 0 | • | |
| Light13AmbientIntensityRed | 4B | F | • | |
| Light13ConstantAttenuation | 4D | 1 | • | |
| Light13CosSpotlightCutoffAngle | 4D | 0 | • | |
| Light13DiffuseIntensityBlue | 4C | 4 | • | |
| Light13DiffuseIntensityGreen | 4C | 3 | • | |
| Light13DiffuseIntensityRed | 4C | 2 | • | |
| Light13LinearAttenuation | 4D | 2 | • | |
| Light13Mode | 4B | E | • | |
| Light13PositionW | 4C | B | • | |
| Light13PositionX | 4C | 8 | • | |
| Light13PositionY | 4C | 9 | • | |
| Light13PositionZ | 4C | A | • | |
| Light13QuadraticAttenuation | 4D | 3 | • | |
| Light13SpecularIntensityBlue | 4C | 7 | • | |
| Light13SpecularIntensityGreen | 4C | 6 | • | |
| Light13SpecularIntensityRed | 4C | 5 | • | |

| | | | | |
|---|---|---|---|---|
| Light13SpotlightDirectionX | 4C | C | • | |
| Light13SpotlightDirectionY | 4C | D | • | |
| Light13SpotlightDirectionZ | 4C | E | • | |
| Light13SpotlightExponent | 4C | F | • | |
| Light14AmbientIntensityBlue | 4D | 7 | • | |
| Light14AmbientIntensityGreen | 4D | 6 | • | |
| Light14AmbientIntensityRed | 4D | 5 | • | |
| Light14ConstantAttenuation | 4E | 7 | • | |
| Light14CosSpotlightCutoffAngle | 4E | 6 | • | |
| Light14DiffuseIntensityBlue | 4D | A | • | |
| Light14DiffuseIntensityGreen | 4D | 9 | • | |
| Light14DiffuseIntensityRed | 4D | 8 | • | |
| Light14LinearAttenuation | 4E | 8 | • | |
| Light14Mode | 4D | 4 | • | |
| Light14PositionW | 4E | 1 | • | |
| Light14PositionX | 4D | E | • | |
| Light14PositionY | 4D | F | • | |
| Light14PositionZ | 4E | 0 | • | |
| Light14QuadraticAttenuation | 4E | 9 | • | |
| Light14SpecularIntensityBlue | 4D | D | • | |
| Light14SpecularIntensityGreen | 4D | C | • | |
| Light14SpecularIntensityRed | 4D | B | • | |
| Light14SpotlightDirectionX | 4E | 2 | • | |
| Light14SpotlightDirectionY | 4E | 3 | • | |
| Light14SpotlightDirectionZ | 4E | 4 | • | |
| Light14SpotlightExponent | 4E | 5 | • | |
| Light15AmbientIntensityBlue | 4E | D | • | |
| Light15AmbientIntensityGreen | 4E | C | • | |
| Light15AmbientIntensityRed | 4E | B | • | |
| Light15ConstantAttenuation | 4F | D | • | |
| Light15CosSpotlightCutoffAngle | 4F | C | • | |
| Light15DiffuseIntensityBlue | 4F | 0 | • | |
| Light15DiffuseIntensityGreen | 4E | F | • | |
| Light15DiffuseIntensityRed | 4E | E | • | |
| Light15LinearAttenuation | 4F | E | • | |
| Light15Mode | 4E | A | • | |
| Light15PositionW | 4F | 7 | • | |
| Light15PositionX | 4F | 4 | • | |
| Light15PositionY | 4F | 5 | • | |
| Light15PositionZ | 4F | 6 | • | |
| Light15QuadraticAttenuation | 4F | F | • | |
| Light15SpecularIntensityBlue | 4F | 3 | • | |
| Light15SpecularIntensityGreen | 4F | 2 | • | |
| Light15SpecularIntensityRed | 4F | 1 | • | |
| Light15SpotlightDirectionX | 4F | 8 | • | |
| Light15SpotlightDirectionY | 4F | 9 | • | |
| Light15SpotlightDirectionZ | 4F | A | • | |
| Light15SpotlightExponent | 4F | B | • | |
| Light1AmbientIntensityBlue | 3B | 9 | • | |
| Light1AmbientIntensityGreen | 3B | 8 | • | |
| Light1AmbientIntensityRed | 3B | 7 | • | |
| Light1ConstantAttenuation | 3C | 9 | • | |

| | | | | |
|---|---|---|---|---|
| Light1CosSpotlightCutoffAngle | 3C | 8 | • | |
| Light1DiffuseIntensityBlue | 3B | C | • | |
| Light1DiffuseIntensityGreen | 3B | B | • | |
| Light1DiffuseIntensityRed | 3B | A | • | |
| Light1LinearAttenuation | 3C | A | • | |
| Light1Mode | 3B | 6 | • | |
| Light1PositionW | 3C | 3 | • | |
| Light1PositionX | 3C | 0 | • | |
| Light1PositionY | 3C | 1 | • | |
| Light1PositionZ | 3C | 2 | • | |
| Light1QuadraticAttenuation | 3C | B | • | |
| Light1SpecularIntensityBlue | 3B | F | • | |
| Light1SpecularIntensityGreen | 3B | E | • | |
| Light1SpecularIntensityRed | 3B | D | • | |
| Light1SpotlightDirectionX | 3C | 4 | • | |
| Light1SpotlightDirectionY | 3C | 5 | • | |
| Light1SpotlightDirectionZ | 3C | 6 | • | |
| Light1SpotlightExponent | 3C | 7 | • | |
| Light2AmbientIntensityBlue | 3C | F | • | |
| Light2AmbientIntensityGreen | 3C | E | • | |
| Light2AmbientIntensityRed | 3C | D | • | |
| Light2ConstantAttenuation | 3D | F | • | |
| Light2CosSpotlightCutoffAngle | 3D | E | • | |
| Light2DiffuseIntensityBlue | 3D | 2 | • | |
| Light2DiffuseIntensityGreen | 3D | 1 | • | |
| Light2DiffuseIntensityRed | 3D | 0 | • | |
| Light2LinearAttenuation | 3E | 0 | • | |
| Light2Mode | 3C | C | • | |
| Light2PositionW | 3D | 9 | • | |
| Light2PositionX | 3D | 6 | • | |
| Light2PositionY | 3D | 7 | • | |
| Light2PositionZ | 3D | 8 | • | |
| Light2QuadraticAttenuation | 3E | 1 | • | |
| Light2SpecularIntensityBlue | 3D | 5 | • | |
| Light2SpecularIntensityGreen | 3D | 4 | • | |
| Light2SpecularIntensityRed | 3D | 3 | • | |
| Light2SpotlightDirectionX | 3D | A | • | |
| Light2SpotlightDirectionY | 3D | B | • | |
| Light2SpotlightDirectionZ | 3D | C | • | |
| Light2SpotlightExponent | 3D | D | • | |
| Light3AmbientIntensityBlue | 3E | 5 | • | |
| Light3AmbientIntensityGreen | 3E | 4 | • | |
| Light3AmbientIntensityRed | 3E | 3 | • | |
| Light3ConstantAttenuation | 3F | 5 | • | |
| Light3CosSpotlightCutoffAngle | 3F | 4 | • | |
| Light3DiffuseIntensityBlue | 3E | 8 | • | |
| Light3DiffuseIntensityGreen | 3E | 7 | • | |
| Light3DiffuseIntensityRed | 3E | 6 | • | |
| Light3LinearAttenuation | 3F | 6 | • | |
| Light3Mode | 3E | 2 | • | |
| Light3PositionW | 3E | F | • | |
| Light3PositionX | 3E | C | • | |

| | | | | |
|---|---|---|---|---|
| Light3PositionY | 3E | D | • | |
| Light3PositionZ | 3E | E | • | |
| Light3QuadraticAttenuation | 3F | 7 | • | |
| Light3SpecularIntensityBlue | 3E | B | • | |
| Light3SpecularIntensityGreen | 3E | A | • | |
| Light3SpecularIntensityRed | 3E | 9 | • | |
| Light3SpotlightDirectionX | 3F | 0 | • | |
| Light3SpotlightDirectionY | 3F | 1 | • | |
| Light3SpotlightDirectionZ | 3F | 2 | • | |
| Light3SpotlightExponent | 3F | 3 | • | |
| Light4AmbientIntensityBlue | 3F | B | • | |
| Light4AmbientIntensityGreen | 3F | A | • | |
| Light4AmbientIntensityRed | 3F | 9 | • | |
| Light4ConstantAttenuation | 40 | B | • | |
| Light4CosSpotlightCutoffAngle | 40 | A | • | |
| Light4DiffuseIntensityBlue | 3F | E | • | |
| Light4DiffuseIntensityGreen | 3F | D | • | |
| Light4DiffuseIntensityRed | 3F | C | • | |
| Light4LinearAttenuation | 40 | C | • | |
| Light4Mode | 3F | 8 | • | |
| Light4PositionW | 40 | 5 | • | |
| Light4PositionX | 40 | 2 | • | |
| Light4PositionY | 40 | 3 | • | |
| Light4PositionZ | 40 | 4 | • | |
| Light4QuadraticAttenuation | 40 | D | • | |
| Light4SpecularIntensityBlue | 40 | 1 | • | |
| Light4SpecularIntensityGreen | 40 | 0 | • | |
| Light4SpecularIntensityRed | 3F | F | • | |
| Light4SpotlightDirectionX | 40 | 6 | • | |
| Light4SpotlightDirectionY | 40 | 7 | • | |
| Light4SpotlightDirectionZ | 40 | 8 | • | |
| Light4SpotlightExponent | 40 | 9 | • | |
| Light5AmbientIntensityBlue | 41 | 1 | • | |
| Light5AmbientIntensityGreen | 41 | 0 | • | |
| Light5AmbientIntensityRed | 40 | F | • | |
| Light5ConstantAttenuation | 42 | 1 | • | |
| Light5CosSpotlightCutoffAngle | 42 | 0 | • | |
| Light5DiffuseIntensityBlue | 41 | 4 | • | |
| Light5DiffuseIntensityGreen | 41 | 3 | • | |
| Light5DiffuseIntensityRed | 41 | 2 | • | |
| Light5LinearAttenuation | 42 | 2 | • | |
| Light5Mode | 40 | E | • | |
| Light5PositionW | 41 | B | • | |
| Light5PositionX | 41 | 8 | • | |
| Light5PositionY | 41 | 9 | • | |
| Light5PositionZ | 41 | A | • | |
| Light5QuadraticAttenuation | 42 | 3 | • | |
| Light5SpecularIntensityBlue | 41 | 7 | • | |
| Light5SpecularIntensityGreen | 41 | 6 | • | |
| Light5SpecularIntensityRed | 41 | 5 | • | |
| Light5SpotlightDirectionX | 41 | C | • | |
| Light5SpotlightDirectionY | 41 | D | • | |

| | | | | |
|---|---|---|---|---|
| Light5SpotlightDirectionZ | 41 | E | • | |
| Light5SpotlightExponent | 41 | F | • | |
| Light6AmbientIntensityBlue | 42 | 7 | • | |
| Light6AmbientIntensityGreen | 42 | 6 | • | |
| Light6AmbientIntensityRed | 42 | 5 | • | |
| Light6ConstantAttenuation | 43 | 7 | • | |
| Light6CosSpotlightCutoffAngle | 43 | 6 | • | |
| Light6DiffuseIntensityBlue | 42 | A | • | |
| Light6DiffuseIntensityGreen | 42 | 9 | • | |
| Light6DiffuseIntensityRed | 42 | 8 | • | |
| Light6LinearAttenuation | 43 | 8 | • | |
| Light6Mode | 42 | 4 | • | |
| Light6PositionW | 43 | 1 | • | |
| Light6PositionX | 42 | E | • | |
| Light6PositionY | 42 | F | • | |
| Light6PositionZ | 43 | 0 | • | |
| Light6QuadraticAttenuation | 43 | 9 | • | |
| Light6SpecularIntensityBlue | 42 | D | • | |
| Light6SpecularIntensityGreen | 42 | C | • | |
| Light6SpecularIntensityRed | 42 | B | • | |
| Light6SpotlightDirectionX | 43 | 2 | • | |
| Light6SpotlightDirectionY | 43 | 3 | • | |
| Light6SpotlightDirectionZ | 43 | 4 | • | |
| Light6SpotlightExponent | 43 | 5 | • | |
| Light7AmbientIntensityBlue | 43 | D | • | |
| Light7AmbientIntensityGreen | 43 | C | • | |
| Light7AmbientIntensityRed | 43 | B | • | |
| Light7ConstantAttenuation | 44 | D | • | |
| Light7CosSpotlightCutoffAngle | 44 | C | • | |
| Light7DiffuseIntensityBlue | 44 | 0 | • | |
| Light7DiffuseIntensityGreen | 43 | F | • | |
| Light7DiffuseIntensityRed | 43 | E | • | |
| Light7LinearAttenuation | 44 | E | • | |
| Light7Mode | 43 | A | • | |
| Light7PositionW | 44 | 7 | • | |
| Light7PositionX | 44 | 4 | • | |
| Light7PositionY | 44 | 5 | • | |
| Light7PositionZ | 44 | 6 | • | |
| Light7QuadraticAttenuation | 44 | F | • | |
| Light7SpecularIntensityBlue | 44 | 3 | • | |
| Light7SpecularIntensityGreen | 44 | 2 | • | |
| Light7SpecularIntensityRed | 44 | 1 | • | |
| Light7SpotlightDirectionX | 44 | 8 | • | |
| Light7SpotlightDirectionY | 44 | 9 | • | |
| Light7SpotlightDirectionZ | 44 | A | • | |
| Light7SpotlightExponent | 44 | B | • | |
| Light8AmbientIntensityBlue | 45 | 3 | • | |
| Light8AmbientIntensityGreen | 45 | 2 | • | |
| Light8AmbientIntensityRed | 45 | 1 | • | |
| Light8ConstantAttenuation | 46 | 3 | • | |
| Light8CosSpotlightCutoffAngle | 46 | 2 | • | |
| Light8DiffuseIntensityBlue | 45 | 6 | • | |

| | | | | |
|---|---|---|---|---|
| Light8DiffuseIntensityGreen | 45 | 5 | • | |
| Light8DiffuseIntensityRed | 45 | 4 | • | |
| Light8LinearAttenuation | 46 | 4 | • | |
| Light8Mode | 45 | 0 | • | |
| Light8PositionW | 45 | D | • | |
| Light8PositionX | 45 | A | • | |
| Light8PositionY | 45 | B | • | |
| Light8PositionZ | 45 | C | • | |
| Light8QuadraticAttenuation | 46 | 5 | • | |
| Light8SpecularIntensityBlue | 45 | 9 | • | |
| Light8SpecularIntensityGreen | 45 | 8 | • | |
| Light8SpecularIntensityRed | 45 | 7 | • | |
| Light8SpotlightDirectionX | 45 | E | • | |
| Light8SpotlightDirectionY | 45 | F | • | |
| Light8SpotlightDirectionZ | 46 | 0 | • | |
| Light8SpotlightExponent | 46 | 1 | • | |
| Light9AmbientIntensityBlue | 46 | 9 | • | |
| Light9AmbientIntensityGreen | 46 | 8 | • | |
| Light9AmbientIntensityRed | 46 | 7 | • | |
| Light9ConstantAttenuation | 47 | 9 | • | |
| Light9CosSpotlightCutoffAngle | 47 | 8 | • | |
| Light9DiffuseIntensityBlue | 46 | C | • | |
| Light9DiffuseIntensityGreen | 46 | B | • | |
| Light9DiffuseIntensityRed | 46 | A | • | |
| Light9LinearAttenuation | 47 | A | • | |
| Light9Mode | 46 | 6 | • | |
| Light9PositionW | 47 | 3 | • | |
| Light9PositionX | 47 | 0 | • | |
| Light9PositionY | 47 | 1 | • | |
| Light9PositionZ | 47 | 2 | • | |
| Light9QuadraticAttenuation | 47 | B | • | |
| Light9SpecularIntensityBlue | 46 | F | • | |
| Light9SpecularIntensityGreen | 46 | E | • | |
| Light9SpecularIntensityRed | 46 | D | • | |
| Light9SpotlightDirectionX | 47 | 4 | • | |
| Light9SpotlightDirectionY | 47 | 5 | • | |
| Light9SpotlightDirectionZ | 47 | 6 | • | |
| Light9SpotlightExponent | 47 | 7 | • | |
| LightingMode | 2A | 4 | • | |
| LightingModeAnd | 55 | 6 | | |
| LightingModeOr | 55 | 7 | | |
| LineClipLengthThreshold | 37 | B | • | |
| LineMode | 29 | 5 | • | |
| LineModeAnd | 55 | E | | |
| LineModeOr | 55 | F | | |
| LineWidth | 29 | 6 | • | |
| LineWidthOffset | 29 | 7 | • | |
| LoadName | 2B | E | | Command |
| MaterialMode | 2A | 6 | • | |
| MaterialModeAnd | 56 | 2 | | |
| MaterialModeOr | 56 | 3 | | |
| ModelViewMatrix[16] | 32 | 0…F | • | |

| | | | | |
|---|---|---|---|---|
| ModelViewProjectionMatrix[16] | 33 | 0…F | • | |
| NormalizeMode | 2A | 3 | • | |
| NormalizeModeAnd | 55 | 4 | | |
| NormalizeModeOr | 55 | 5 | | |
| NormalMatrix[9] | 34 | 0…8 | • | |
| Nx | 30 | 2 | • | Trigger |
| Ny | 30 | 1 | • | |
| Nz | 30 | 0 | • | |
| ObjectIDValue | 2B | 5 | • | |
| PackedColor3 | 31 | 3 | | Trigger |
| PackedColor4 | 31 | 4 | | Trigger |
| PassThrough | 1F | E | | Command |
| PointMode | 29 | 2 | • | |
| PointModeAnd | 55 | C | | |
| PointModeOr | 55 | D | | |
| PointSize | 29 | 3 | • | |
| PolygonOffsetBias | 37 | A | • | |
| PolygonOffsetFactor | 37 | 9 | • | |
| PopName | 2B | D | | Command |
| PushName | 2B | C | | Command |
| RasterPosXIncrement | 37 | D | • | |
| RasterPosXOffset | 39 | D | • | |
| RasterPosYIncrement | 37 | E | • | |
| RasterPosYOffset | 39 | E | • | |
| Rectangle2DControl | 29 | E | • | |
| Rectangle2DMode | 29 | D | • | |
| RectangleHeight | 29 | C | • | |
| RectangleMode | 29 | A | • | |
| RectangleWidth | 29 | B | • | |
| RepeatLine | 26 | 5 | | Command |
| RepeatTriangle | 26 | 2 | | Command |
| RestoreCurrent | 2B | A | | Command |
| RPw | 31 | 9 | | |
| RPx2 | 31 | C | | Trigger |
| RPx3 | 31 | D | | Trigger |
| RPx4 | 31 | E | | Trigger |
| RPy | 31 | B | | |
| RPz | 31 | A | | |
| SaveCurrent | 2B | 9 | | Command |
| SceneAmbientColorBlue | 50 | 2 | • | |
| SceneAmbientColorGreen | 50 | 1 | • | |
| SceneAmbientColorRed | 50 | 0 | • | |
| SelectRecord | 1F | D | | Command |
| SelectResult | 2B | 0 | | Command |
| TexGen[16] | 36 | 0…F | • | |
| TextureMatrix[16] | 35 | 0…F | • | |
| Tq4 | 31 | 5 | • | |
| Tr4 | 31 | 6 | • | |
| TransformCurrent | 2B | 7 | | Command |
| TransformMode | 2A | 1 | • | |
| TransformModeAnd | 55 | 0 | | |
| TransformModeOr | 55 | 1 | | |

| | | | | |
|---|---|---|---|---|
| TriangleClipAreaThreshold | 37 | C | • | |
| TriangleMode | 29 | 9 | • | |
| TriangleModeAnd | 56 | 0 | | |
| TriangleModeOr | 56 | 1 | | |
| Ts1 | 31 | F | • | Trigger |
| Ts2 | 30 | 9 | • | Trigger |
| Ts4 | 31 | 8 | • | Trigger |
| Tt2 | 30 | 8 | • | |
| Tt4 | 31 | 7 | • | |
| UserClip0W | 38 | 3 | • | |
| UserClip0X | 38 | 0 | • | |
| UserClip0Y | 38 | 1 | • | |
| UserClip0Z | 38 | 2 | • | |
| UserClip1W | 38 | 7 | • | |
| UserClip1X | 38 | 4 | • | |
| UserClip1Y | 38 | 5 | • | |
| UserClip1Z | 38 | 6 | • | |
| UserClip2W | 38 | B | • | |
| UserClip2X | 38 | 8 | • | |
| UserClip2Y | 38 | 9 | • | |
| UserClip2Z | 38 | A | • | |
| UserClip3W | 38 | F | • | |
| UserClip3X | 38 | C | • | |
| UserClip3Y | 38 | D | • | |
| UserClip3Z | 38 | E | • | |
| UserClip4W | 39 | 3 | • | |
| UserClip4X | 39 | 0 | • | |
| UserClip4Y | 39 | 1 | • | |
| UserClip4Z | 39 | 2 | • | |
| UserClip5W | 39 | 7 | • | |
| UserClip5X | 39 | 4 | • | |
| UserClip5Y | 39 | 5 | • | |
| UserClip5Z | 39 | 6 | • | |
| V0FixedA | 20 | 8 | • | Reads back as a float |
| V0FixedB | 20 | 7 | • | Reads back as a float |
| V0FixedF | 20 | 9 | • | Reads back as a float |
| V0FixedG | 20 | 6 | • | Reads back as a float |
| V0FixedKd | 20 | 4 | • | Reads back as a float |
| V0FixedKs | 20 | 3 | • | Reads back as a float |
| V0FixedQ | 20 | 2 | • | Reads back as a float |
| V0FixedR | 20 | 5 | • | Reads back as a float |
| V0FixedS | 20 | 0 | • | Reads back as a float |
| V0FixedT | 20 | 1 | • | Reads back as a float |
| V0FixedX | 20 | A | • | Reads back as a float |
| V0FixedY | 20 | B | • | Reads back as a float |
| V0FixedZ | 20 | C | • | Reads back as a float |
| V0FloatA | 23 | 8 | • | |
| V0FloatB | 23 | 7 | • | |
| V0FloatF | 23 | 9 | • | |
| V0FloatG | 23 | 6 | • | |
| V0FloatKd | 23 | 4 | • | |
| V0FloatKs | 23 | 3 | • | |

| | | | | |
|---|---|---|---|---|
| V0FloatQ | 23 | 2 | • | |
| V0FloatR | 23 | 5 | • | |
| V0FloatS | 23 | 0 | • | |
| V0FloatT | 23 | 1 | • | |
| V0FloatX | 23 | A | • | |
| V0FloatY | 23 | B | • | |
| V0FloatZ | 23 | C | • | |
| V1FixedA | 21 | 8 | • | Reads back as a float |
| V1FixedB | 21 | 7 | • | Reads back as a float |
| V1FixedF | 21 | 9 | • | Reads back as a float |
| V1FixedG | 21 | 6 | • | Reads back as a float |
| V1FixedKd | 21 | 4 | • | Reads back as a float |
| V1FixedKs | 21 | 3 | • | Reads back as a float |
| V1FixedQ | 21 | 2 | • | Reads back as a float |
| V1FixedR | 21 | 5 | • | Reads back as a float |
| V1FixedS | 21 | 0 | • | Reads back as a float |
| V1FixedT | 21 | 1 | • | Reads back as a float |
| V1FixedX | 21 | A | • | Reads back as a float |
| V1FixedY | 21 | B | • | Reads back as a float |
| V1FixedZ | 21 | C | • | Reads back as a float |
| V1FloatA | 24 | 8 | • | |
| V1FloatB | 24 | 7 | • | |
| V1FloatF | 24 | 9 | • | |
| V1FloatG | 24 | 6 | • | |
| V1FloatKd | 24 | 4 | • | |
| V1FloatKs | 24 | 3 | • | |
| V1FloatQ | 24 | 2 | • | |
| V1FloatR | 24 | 5 | • | |
| V1FloatS | 24 | 0 | • | |
| V1FloatT | 24 | 1 | • | |
| V1FloatX | 24 | A | • | |
| V1FloatY | 24 | B | • | |
| V1FloatZ | 24 | C | • | |
| V2FixedA | 22 | 8 | • | Reads back as a float |
| V2FixedB | 22 | 7 | • | Reads back as a float |
| V2FixedF | 22 | 9 | • | Reads back as a float |
| V2FixedG | 22 | 6 | • | Reads back as a float |
| V2FixedKd | 22 | 4 | • | Reads back as a float |
| V2FixedKs | 22 | 3 | • | Reads back as a float |
| V2FixedQ | 22 | 2 | • | Reads back as a float |
| V2FixedR | 22 | 5 | • | Reads back as a float |
| V2FixedS | 22 | 0 | • | Reads back as a float |
| V2FixedT | 22 | 1 | • | Reads back as a float |
| V2FixedX | 22 | A | • | Reads back as a float |
| V2FixedY | 22 | B | • | Reads back as a float |
| V2FixedZ | 22 | C | • | Reads back as a float |
| V2FloatA | 25 | 8 | • | |
| V2FloatB | 25 | 7 | • | |
| V2FloatF | 25 | 9 | • | |
| V2FloatG | 25 | 6 | • | |
| V2FloatKd | 25 | 4 | • | |
| V2FloatKs | 25 | 3 | • | |

| | | | | |
|---|---|---|---|---|
| V2FloatQ | 25 | 2 | • | |
| V2FloatR | 25 | 5 | • | |
| V2FloatS | 25 | 0 | • | |
| V2FloatT | 25 | 1 | • | |
| V2FloatX | 25 | A | • | |
| V2FloatY | 25 | B | • | |
| V2FloatZ | 25 | C | • | |
| ViewPortOffsetX | 37 | 3 | • | |
| ViewPortOffsetY | 37 | 4 | • | |
| ViewPortOffsetZ | 37 | 5 | • | |
| ViewPortScaleX | 37 | 0 | • | |
| ViewPortScaleY | 37 | 1 | • | |
| ViewPortScaleZ | 37 | 2 | • | |
| Vw | 30 | A | | |
| Vx2 | 30 | D | | Trigger |
| Vx3 | 30 | E | | Trigger |
| Vx4 | 30 | F | | Trigger |
| Vy | 30 | C | | |
| Vz | 30 | B | | |
| WindowAnd | 57 | 0 | | |
| WindowOr | 57 | 1 | | |
| XBias | 29 | 0 | • | |
| YBias | 29 | 1 | • | |

# Appendix D Register and Command Reference

## AALineWidth

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| AALineWidth | 0x298 | 0x0000.14C0 | Read/Write | Undefined | Float |

This register sets the width of antialiased lines. Lines with a zero width will not be drawn and lines with a negative width will draw a line of the same positive width. The width is measured in pixels. Lines with a width less than 1.0 can be drawn, however they may contain gaps due to sampling errors.

## AAPointSize

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| AAPointSize | 0x294 | 0x0000.14A0 | Read/Write | Undefined | Float |

This register sets the point size (diameter) for antialiased points. In theory any size antialiased points can be defined, however GLINT places some restrictions on what these widths can be. The Point Table in GLINT restricts the diameter of antialiased points to be from 0.5 to 16.0 in steps of 0.25 when the antialiasing quality is 4x4 or 0.25 to 8.0 in steps of 0.125 for 8x8 quality. Gamma does not set up the Point Table. Points with a zero size will draw a single fragment and points with a negative size will draw a point of the same positive size.

## AttenuationCutOff

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| AttenuationCutOff | 0x39F | 0x0000.1CF8 | Read/Write | Undefined | Float |

A positional light has its intensity attenuated by its distance and the constant, linear and quadratic attenuation factors. A spot light has its intensity attenuated by the angle the spot light makes with the vertex being lit. If the product of the distance attenuation and the spot attenuation for a light falls below the value in this register then this light doesn't contribute to this vertex. This optimization allows lights which are becoming too faint to contribute to be terminated early.

A suitable value is given by: 1.0/512*n* where *n* is the number of lights. The 512 constant was chosen as it is less than the smallest representable color when converted to a byte integer assuming the light and material colors are restricted to the range 0.0…1.0.

This optimization is enabled by the AttenuationTest field in the **LightingMode** register.

## BackAlpha

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| BackAlpha | 0x526 | 0x0000.2930 | Read/Write | Undefined | Float |

This register holds the alpha value for the back material. Its normal range of values is 0.0…1.0, however any value can be used.

This register would normally take the back material diffuse alpha in OpenGL.

## BackAmbientColorBlue BackAmbientColorGreen BackAmbientColorRed

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| BackAmbientColorBlue | 0x522 | 0x0000.2910 | Read/Write | Undefined | Float |
| BackAmbientColorGreen | 0x521 | 0x0000.2908 | Read/Write | Undefined | Float |
| BackAmbientColorRed | 0x520 | 0x0000.2900 | Read/Write | Undefined | Float |

These registers hold the red, green and blue ambient colors for the back material. Their normal range of values is 0.0…1.0, however any value can be used.

## BackDiffuseColorBlue BackDiffuseColorGreen BackDiffuseColorRed

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| BackDiffuseColorBlue | 0x525 | 0x0000.2928 | Read/Write | Undefined | Float |
| BackDiffuseColorGreen | 0x524 | 0x0000.2920 | Read/Write | Undefined | Float |
| BackDiffuseColorRed | 0x523 | 0x0000.2918 | Read/Write | Undefined | Float |

These registers hold the red, green and blue diffuse colors for the back material. Their normal range of values is 0.0…1.0, however any value can be used.

## BackEmissiveColorBlue BackEmissiveColorGreen BackEmissiveColorRed

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| BackEmissiveColorBlue | 0x52C | 0x0000.2960 | Read/Write | Undefined | Float |
| BackEmissiveColorGreen | 0x52B | 0x0000.2958 | Read/Write | Undefined | Float |
| BackEmissiveColorRed | 0x52A | 0x0000.2950 | Read/Write | Undefined | Float |

These registers hold the red, green and blue emissive colors for the back material. Their normal range of values is 0.0…1.0, however any value can be used.

## BackSpecularColorBlue BackSpecularColorGreen BackSpecularColorRed

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| BackSpecularColorBlue | 0x529 | 0x0000.2948 | Read/Write | Undefined | Float |
| BackSpecularColorGreen | 0x528 | 0x0000.2940 | Read/Write | Undefined | Float |
| BackSpecularColorRed | 0x527 | 0x0000.2938 | Read/Write | Undefined | Float |

These registers hold the red, green and blue specular colors for the back material. Their normal range of values is 0.0…1.0, however any value can be used. Setting the red, green and blue color to 0.0 is detected and the specular part of the calculations are avoided.

## BackSpecularExponent

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| BackSpecularExponent | 0x52D | 0x0000.2968 | Read/Write | Undefined | Fixed point |

This register holds back material specular exponent as an unsigned 7.4 fixed point format.

## Begin

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| Begin | 0x2B2 | 0x0000.1590 | Write | Undefined | Bitfield |

This command identifies the start of a sequence of primitives. As each new vertex is received the next point, line, triangle is generated, if sufficient vertices have been received. The **Begin** command should have a corresponding **End** command to ensure the primitive sequence is tidied up.

The format of the **Begin** command's data field is shown overleaf. The fields in bits 0 to 18 are the same as those in the **Render** command and only the fields which influence Gamma are identified and described. The effect of the fields on GLINT can be found in the relevant GLINT Programmer's Reference Manual.

| Bit No. | Name | Description |
|---|---|---|
| 0 | AreaStippleEnable | Overridden when incompatible with the primitive type. |
| 1 | LineStippleEnable | Overridden when incompatible with the primitive type. |
| 2 | ResetLineStipple | Ignored and always set by Gamma to meet the OpenGL rules for when the line stipple is reset within primitives. |
| 3 | FastFillEnable | Ignored and forced to be disabled. |
| 4, 5 | Not used | |
| 6, 7 | PrimitiveType | Ignored and always set by Gamma depending on the type field and polymode setting. |
| 8 | AntialiasEnable | Qualifies the AntialiasEnable held for each primitive type in the **PointMode**, **LineMode** and **TriangleMode** registers. If both enables are true then the primitive is antialiased. |
| 9 | AntialiasingQuality | Ignored. This information is held in the **PointMode**, **LineMode** and **TriangleMode** registers. |
| 10 | UsePointTable | Ignored and generated locally when antialiasing points. |
| 11 | SyncOnBitMask | Ignored, and forced to be disabled. |
| 12 | SyncOnHostData | Ignored, and forced to be disabled. |
| 13 | TextureEnable | Passed through but also enables (1) or disables (0) texture calculation to be performed. Note texture transformations and TexGen operations are not influenced by this bit (as required by OpenGL). |
| 14 | FogEnable | Passed through but also enables (1) or disables (0) fog calculation to be performed. |
| 15 | CoverageEnable | Ignored and generated locally when antialiasing is done. |
| 16 | SubPixelCorrectionEnable | This bit is passed through. When enabled (1) subpixel correction is done in the Y direction (the rasterizer does it in the X direction). |
| 17 | | Reserved |
| 18 | SpanOperation | Ignored and forced to be 0. |
| 19…27 | Not used | |
| 28…31 | Type | This field sets up the primitive type to process on receiving each new vertex. It has the following values:<br>0 Null<br>1 Points<br>2 Lines<br>3 LineLoop<br>4 LineStrip<br>5 Triangles<br>6 TriangleStrip<br>7 TriangleFan<br>8 Quads<br>9 QuadStrip<br>10 Polygon |

## BroadcastMask

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| BroadcastMask | 0x26F | 0x0000.1378 | Read/Write | Undefined | Bitfield |

The broadcast mask is used to specify to which GLINT rendering device the command and data stream should be written. The possible values are:

| | |
|---|---|
| 0 | No GLINT is written to. All the GLINT commands and data are discarded so nothing gets rendered. Not really useful except to explore performance bottle-necks. |
| 1 | GLINT 0 is written to. |
| 2 | GLINT 1 is written to. |
| 3 | GLINT 0 and GLINT 1 are written to. The writes are done simultaneously so there is no loss in performance. |

## Ca
## Cb
## Cg
## Cr3
## Cr4

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| Ca | 0x303 | 0x0000.1818 | Read/Write | Undefined | Float |
| Cb | 0x304 | 0x0000.1820 | Read/Write | Undefined | Float |
| Cg | 0x305 | 0x0000.1828 | Read/Write | Undefined | Float |
| Cr3 | 0x306 | 0x0000.1830 | Read/Write/Trigger | Undefined | Float |
| Cr4 | 0x307 | 0x0000.1838 | Read/Write/Trigger | Undefined | Float |

These registers hold the red, green, blue and alpha color components. The nominal range for these values are 0.0…1.0, however this is not enforced. The **Cr3** or **Cr4** registers must be written last as these write will trigger the color to be entered into Gamma. With **Cr3** the alpha value provided (if any) is ignored and set to 1.0. All the color components should be written together and not interleaved with writes to the normal (N*), face normal (FN*), texture (T*) or vertex (V*) registers.

The color may be used as the vertex color (if lighting is disabled), or to change a material property if color material is enabled.

See also: **ColorMaterialMode**, **LightingMode**, **MaterialMode**.

## ColorMaterialMode
## ColorMaterialModeAnd
## ColorMaterialModeOr

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| ColorMaterialMode | 0x2A5 | 0x0000.1528 | Read/Write | Undefined | Bitfield |
| ColorMaterialModeAnd | 0x558 | 0x0000.2AC0 | Write | Undefined | Bitfield |
| ColorMaterialModeOr | 0x559 | 0x0000.2AC8 | Write | Undefined | Bitfield |

The **ColorMaterialMode** has the following fields:

| Bit No. | Name | Description |
|---|---|---|
| 0 | ColorMaterialEnable | When set causes a write to the color registers to update the material parameter(s) for the given face(s). |
| 1…2 | Face | This field selects which face(s) any material changes should be made to by the color writes. The values are:<br>0 = front material<br>1 = back material<br>2 = front and back material |
| 3…5 | Parameter | This field selects which material parameter(s) should be updated by the color writes The values are:<br>0 = Emissive<br>1 = Ambient<br>2 = Diffuse<br>3 = Specular<br>4 = Ambient and diffuse |

Writing to the **ColorMaterialModeAnd** and **ColorMaterialModeOr** registers logically combine the new value with the existing values in the **ColorMaterialMode** register rather than replacing its contents.

## CommandInterrupt

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| CommandInterrupt | 0x532 | 0x0000.2990 | Write | Undefined | Bitfield |

This command will generate an interrupt whenever it is first processed by Gamma. If the least significant bit is set the interrupt is delayed until the Output DMA controller has finished. A consequence of this is that command processing is also suspended while waiting.

## ContextData

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| ContextData | 0x1BA | 0x0000.0DD0 | Write/Output | Undefined | Variable |

The context data generated by the **ContextDump** command has this tag on output. When the context data is restored it is sent with this tag, although using the hold mode tag format is more economical.

## ContextDump
## ContextRestore

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| ContextDump | 0x1B8 | 0x0000.0DC0 | Write | Undefined | Bitfield |
| ContextRestore | 0x1B9 | 0x0000.0DC8 | Write | Undefined | Bitfield |

The context mask sent with the **ContextDump** or **ContextRestore** commands have the following fields:

| Bit | Name | Context data includes | Words |
|---|---|---|---|
| 0 | GeneralControl | Mode and general control registers. | 19 |
| 1 | Geometry | Some user geometric state and much of the internal state. | 381 |
| 2 | Matrices | The user defined matrices. | 82 |
| 3 | Material | The user defined material parameters. | 27 |
| 4 | Lights0_7 | The user defined light parameters for lights 0 to 7. | 176 |
| 5 | Lights8_15 | The user defined light parameters for lights 8 to 15. | 176 |
| 6 | RasterPos | The raster position related state. This is expanded below so the current raster position, color, etc. can be read back to satisfy the OpenGL Get calls. | 19 |
| 7 | CurrentState | The current state. This is expanded below so the current texture, color, etc. can be read back to satisfy the OpenGL Get calls. | 12 |
| 8 | TwoD | The 2D related control registers. | 2 |
| 9 | DMA | The DMA related registers. | 7 |
| 10 | Select | The select related registers and name stack. | 67 |

The **ContextDump** command forces Gamma to dump the selected context out. This appears in the Host Output FIFO in GLINT where it can be read from (by the host of the Output DMA controller). The amount of data placed in the FIFO depends on the context mask sent with the command (shown in the table). The context mask is also in the FIFO after the context data, but is not included in the table count. The **FilterMode** register (in GLINT) has bit 14 set to allow the ContextData tag to be written into the FIFO and bit 15 the context data.

The context data is largely undocumented, however the CurrentState and RasterPos selections, when used by themselves, allow state which OpenGL can query to be provided.

If a context dump is done with only the CurrentState bit set then the resultant context buffer will hold the following information (the tags and context mask are assumed to have been discarded):

| Offset | Data |
|---|---|
| 0 | Current edge flag in bit 5 |
| 1 | Current normal, X component |
| 2 | Current normal, Y component |
| 3 | Current normal, Z component |
| 4 | Current texture, S component |
| 5 | Current texture, T component |
| 6 | Current texture, R component |
| 7 | Current texture, Q component |
| 8 | Current color, Red component |
| 9 | Current color, Green component |
| 10 | Current color, Blue component |
| 11 | Current color, Alpha component |

If a context dump is done with only the RasterPos bit set then the resultant context buffer will hold the following information (the tags and context mask are assumed to have been discarded). Note that some user defined state is also included:

| Offset | Data |
|---|---|
| 0 | Window coordinate, X component |
| 1 | Window coordinate, Y component |
| 2 | Window coordinate, Z component |
| 3 | Eye coordinate, Z component |
| 4 | Clip coordinate, W component |
| 5 | Texture, S component |
| 6 | Texture, T component |
| 7 | Texture, R component |
| 8 | Texture, Q component |
| 9 | Fog |
| 10 | In View (bit 0: 0 = out of view, 1 = in view) |
| 11 | xIncrement (user register) |
| 12 | yIncrement (user register) |
| 13 | xOffset (user register) |
| 14 | yOffset (user register) |
| 15 | Color, Red component |
| 16 | Color, Green component |
| 17 | Color, Blue component |
| 18 | Color, Alpha component |

The **ContextRestore** command prepares Gamma to receive context data. The context mask should be the same as was used to generate the context data in the first place. The context data in the buffer is written to the **ContextData** register.

## Cr3
## Cr4

See: **Ca/Cb/Cg**

## DeltaMode
## DeltaModeAnd
## DeltaModeOr

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| DeltaMode | 0x260 | 0x0000.1300 | Read/Write | Undefined | Bitfield |
| DeltaModeAnd | 0x55A | 0x0000.2AD0 | Write | Undefined | Bitfield |
| DeltaModeOr | 0x55B | 0x0000.2AD8 | Write | Undefined | Bitfield |

The **DeltaMode** register has the following fields (it is identical to the **DeltaMode** register in GLINT Delta, but with the addition of BiasCoordinates, ColorDiffuse, ColorSpecular, FlatShadingMethod, EpilogueEnable and EpilogueCount fields):

| Bit No. | Name | Description |
|---|---|---|
| 0, 1 | TargetChip | This two bit field selects which chip the calculations are tailored to. The options are:<br>0 = 300SX<br>1 = 500TX, MX |
| 2, 3 | DepthFormat | This two bit field defines the depth format GLINT is working in and hence the final format of the depth parameters to be written into GLINT. The options are:<br>1 = 16 bits (300SX, 500TX, MX)<br>2 = 24 bits (300SX, 500TX, MX)<br>3 = 32 bits (300SX, 500TX, MX)<br>The depth format is used regardless of any other modes bits. |
| 4 | FogEnable | When set enables the fog calculations. This is qualified by the FogEnable bit in the **Begin** or **Draw*** commands. |
| 5 | TextureEnable | When set enables the texture calculations. This is qualified by the TextureEnable bit in the **Begin** or **Draw*** commands. |
| 6 | SmoothShadingEnable | When set enables the color calculations. This field only has an effect when the **Draw*** commands are used and is ignored when the Gamma 3D pipeline is used. In this case the FlatShading bit in the **GeometryMode** register is used. |
| 7 | DepthEnable | When set enables the depth calculations. |
| 8 | SpecularEnable | When set enables the specular texture calculations. This is qualified by the TextureEnable in the **Begin** or **Draw*** commands. |
| 9 | DiffuseEnable | When set enables the diffuse texture calculations. This is qualified by the TextureEnable in the **Begin** or **Draw*** commands. |
| 10 | SubPixelCorrectionEnable | When set provides the subpixel correction in Y. This is qualified by the SubPixelCorrectionEnable in the **Begin** or **Draw*** commands. |
| 11 | DiamondExit | When set enables the application of the OpenGL 'Diamond-exit' rule to modify the start and end coordinates of lines. |
| 12 | NoDraw | When set prevents any rendering from starting after the set up calculations are done and parameters sent to GLINT. This only effect the **Draw*** commands and is ignored when the Gamma 3D pipeline is used. |
| 13 | ClampEnable | When set causes the input values to be clamped into a parameter specific range. Note that the texture parameters are not included. This should normally be set. |

| | | |
|---|---|---|
| 14, 15 | TextureParameterMode | This two bit field causes the texture parameters to be:<br>0 = Used as given<br>1 = Clamped to lie in the range -1.0 to 1.0<br>2 = Normalized to lie in the range -1.0 to 1.0<br>The normal setting for this field is to select texture normalization. |
| 16…18 | reserved | |
| 19 | BiasCoordinates | When set causes the **XBias** and **YBias** registers values to be added to the x and y coordinates respectively. |
| 20 | ColorDiffuse | When set causes the diffuse texture calculations to be done on the red, green and blue components, otherwise the red component (representing monochrome) is done by itself. |
| 21 | ColorSpecular | When set causes the specular texture calculations to be done on the red, green and blue components, otherwise the red component (representing monochrome) is done by itself. |
| 22 | FlatShadingMethod | This field determines how the ColorDDA unit in GLINT is to do flat shading. The two options are use the **ConstantColor** register (0) or the DDA (1) by setting zero gradients. The rasterization performance is the same in both cases, however the ConstantColor method is faster to set up. Consider the situation when smooth shading is enabled (in the **GeometryMode** register) and a point is to be drawn. The point is always flat shaded. This field would normally be the inverse of the FlatShading field in the **GeometryMode** register. |
| 23 | EpilogueEnable | |
| 24…25 | EpilogueCount | |

Writing to the **DeltaModeAnd** and **DeltaModeOr** registers logically combine the new value with the existing values in the **DeltaMode** register rather than replacing its contents.

## DMAAddr

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| DMAAddr | 0x530 | 0x0000.2980 | Write | Undefined | Integer |

This register holds the byte address of the next DMA buffer to read from (reading doesn't start until the **DMACount** command). The bottom two bits of the address are ignored.

This register should not be confused by the PCI register of the same name.

This register must be loaded by itself and not as part of any increment, hold or indexed group.

See also: **DMACount**.

## DMACall

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| DMACall | 0x533 | 0x0000.2998 | Write | Undefined | Integer |

This command starts a new DMA buffer being read. The length of the DMA buffer is set to 0xffffff as the **DMAReturn** command (in the called DMA buffer) is expected to be used to end the DMA buffer. **DMACall**s can be nested 8 deep. If the **DMACall** were to exceed the maximum nesting levels then the **DMACall** is ignored.

This register must be loaded by itself and not as part of any increment, hold or indexed group.

See also **DMAReturn**.

## DMACount

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| DMACount | 0x531 | 0x0000.2988 | Write | Undefined | Integer |

This command starts a new DMA buffer being read starting at the previously loaded address in **DMAAddr**. The length of the DMA buffer is given in the bottom 24 bits of the command. New DMAs can be nested 8 deep. If the **DMACount** were to exceed the maximum nesting levels then the **DMACount** is ignored.

This register should not be confused by the PCI register of the same name.

This register must be loaded by itself and not as part of any increment, hold or indexed group.

See also: **DMAAddr.**

## DMAFeedback

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| DMAFeedback | 0x542 | 0x0000.2A10 | Write | Undefined | Integer |

This command starts a new output DMA if the output DMA controller is idle, otherwise it will block until the output DMA controller becomes available suspending all subsequent commands and register loads.

The length in words of the destination memory buffer is given in the least significant 24 bits of the command, and the memory buffer address will have previously been set up in the **DMAOutputAddress** register.

The output DMA expects to read tags and data from the Host Out FIFO in GLINT so bits 14 and 15 of **FilterMode** register must be set. All tags are discarded and not written into memory and only the data for the following tags is written into memory:
**FeedbackX**
**FeedbackY**
**FeedbackZ**
**FeedbackW**
**FeedbackRed**
**FeedbackGreen**
**FeedbackBlue**
**FeedbackAlpha**
**FeedbackS**
**FeedbackT**
**FeedbackR**
**FeedbackQ**
**FeedbackToken**
**PassThrough**
**SelectRecord**
**ContextData**

The output DMA terminates when the **EndOfFeedback** tag is found or when the buffer is full. The Host Out FIFO is read until the EndOfFeedback tag is found and any data which would overflow the memory buffer is discarded.

The **DMAReadGLINTSource** register identifies the GLINT to read from in a multi-GLINT system.

The actual number of words written to memory is loaded into the *FeedbackSelectCount* PCI register. Overflows are indicated in bit 31.

This register must be loaded by itself and not as part of any increment, hold or indexed group.

See also: **DMAOutputAddress**, **DMAReadGLINTSource**.

## DMAOutputAddress

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| DMAOutputAddress | 0x53C | 0x0000.29E0 | Write | Undefined | Integer |

This register holds the byte address where the output DMA controller will write to. The lower two bits of the address are ignored. The address can be logical or physical. If logical addresses are used then the corresponding page table entries must be set up to allow write access.

This register must be loaded by itself and not as part of any increment, hold or indexed group.

See also: **DMAFeedback**, **DMAOutputCount**.

## DMAOutputCount

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| DMAOutputCount | 0x53D | 0x0000.29E8 | Write | Undefined | Integer |

This command starts a new output DMA if the output DMA controller is idle, otherwise it will block until the output DMA controller becomes available and all subsequent commands and register loads are suspended.

The number of words to read from the GLINT Host Out FIFO is given in the bottom 24 bits of the command, and the memory buffer address will have previously been set up in the **DMAOutputAddress** register.

The GLINT **FilterMode** register must have been set up to allow the required tags and/or data to be written in to the FIFO.

The **DMAReadGLINTSource** register identifies the GLINT to read from in a multi-GLINT system.

This register must be loaded by itself and not as part of any increment, hold or indexed group.

See also: **DMAOutputAddress**, **DMAReadGLINTSource**.

## DMAReadGLINTSource

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| DMAReadGLINTSource | 0x53E | 0x0000.29F0 | Read/Write | Undefined | Integer |

This register identifies which GLINT's Host Out FIFO is to be read in a multi-GLINT system by the output DMA controller (in linear, feedback or rectangle modes). The bottom three bits provide the GLINT ID.

This register must be loaded by itself and not as part of any increment, hold or indexed group.

| | |
|---|---|
| See also | **DMARectangleWriteAddress** |
| | **DMARectangleWriteLinePitch** |

## DMARectangleRead

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| DMARectangleRead | 0x535 | 0x0000.29A8 | Write | Undefined | Bitfield |

The **DMARectangleRead** command has the following fields:

| Bit No. | Name | Description |
|---|---|---|
| 0…11 | Width | Width of the rectangle in pixels. Range 0…4095 |
| 12…23 | Height | Height of the rectangle in pixels. Range 0…4095 |
| 24, 25 | PixelSize | The size of the pixels in the source image to read. The pixel size is used during alignment and packing. The values are:<br>0 = 8 bits<br>1 = 16 bits<br>2 = 24 bits<br>3 = 32 bits |
| 26 | PackOut | This field, when set, causes the data to be packed into 32 bit words when sent to GLINT, otherwise the data is right justified and any unused bits (in the most significant end of the word) are set to zero |
| 27, 28 | ByteSwap | These bits control the byte swapping of the data read from the PCI bus before it is aligned and packed/unpacked. If the input bytes are labeled ABCD on input then they are swapped as follows:<br>0 = ABCD (i.e. no swap)<br>1 = BADC<br>2 = CDAB<br>3 = DCBA |

The Rectangle DMA mechanism allows image data to be transferred from host memory to GLINT. The image data may be a sub image of a larger image and have any natural alignment or pixel size. Information regarding the rectangle transfer is held in registers loaded from the input FIFO or a DMA buffer.

The pixel data read from host memory is always packed, however when passed to GLINT it can be in packed or unpacked format.

The minimum number of PCI reads are used to align and pack the image data.

GLINT is set up to rasterize the destination area for the pixel data (depth, stencil, color, etc.) with SyncOnHostData or SyncOnBitMask enabled in the **Render** command (or equivalent if Gamma is doing the rasterizer set up). This is done before the Rectangular DMA is started.

This register must be loaded by itself and not as part of any increment, hold or indexed group.

See also **DMARectangleReadAddress**
**DMARectangleReadLinePitch**
**DMARectangleReadTarget**

## DMARectangleReadAddress

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| DMARectangleReadAddress | 0x536 | 0x0000.29B0 | Read/Write | Undefined | Integer |

This register provides the byte address of the first pixel in the image or sub image to read during a rectangular DMA transfer from host memory to GLINT. This is treated as a logical or physical address depending on the LogicalAddressing control bit in the *CommandMode* PCI register. The address should be aligned to the natural size of the pixel, except for 24 bit pixels which may be aligned to any byte boundary.

This register must be loaded by itself and not as part of any increment, hold or indexed group.

| | |
|---|---|
| See also | **DMARectangleRead** |
| | **DMARectangleReadLinePitch** |
| | **DMARectangleReadTarget** |

## DMARectangleReadLinePitch

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| DMARectangleReadLinePitch | 0x537 | 0x0000.29B8 | Read/Write | Undefined | Integer |

This register defines the amount, in bytes, to move from one scanline in the image to the next scanline during a rectangular DMA transfer from host memory to GLINT. For a sub image this is based on width of the whole image. The pitch is held as a 32 bit 2's complement number. This is normally an integer multiple of the number of bytes in a pixel.

This register must be loaded by itself and not as part of any increment, hold or indexed group.

| | |
|---|---|
| See also | **DMARectangleReadAddress** |
| | **DMARectangleRead** |
| | **DMARectangleReadTarget** |

## DMARectangleReadTarget

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| DMARectangleReadTarget | 0x538 | 0x0000.29C0 | Read/Write | Undefined | Integer |

This register holds the 16 bit tag sent to GLINT just before the image data is sent during a rectangular DMA transfer from host memory to GLINT. Normally it would be one of the tags allowed by the rasterizer during a SyncOnHostData or SyncOnBitMask operation with the tag mode set to Hold. The secondary PCI bus traffic is minimized by sending multiple image words with a single tag (with a count).

This register must be loaded by itself and not as part of any increment, hold or indexed group.

| | |
|---|---|
| See also | **DMARectangleReadAddress** |
| | **DMARectangleReadLinePitch** |
| | **DMARectangleRead** |

## DMARectangleWrite

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| DMARectangleWrite | 0x539 | 0x0000.29C8 | Write | Undefined | Bitfield |

The **DMARectangleWrite** command has the following fields:

| Bit No. | Name | Description |
|---|---|---|
| 0…11 | Width | Width of the rectangle in pixels. Range 0…4095 |
| 12…23 | Height | Height of the rectangle in pixels. Range 0…4095 |
| 24, 25 | PixelSize | The size of the pixels in the source image to read. The pixel size is used during alignment and packing. The values are:<br>0 = 8 bits<br>1 = 16 bits<br>2 = 24 bits<br>3 = 32 bits |
| 26 | PackIn | This field, when set, indicates the image data from GLINT is packed, otherwise there is one pixel per 32 bits read from GLINT. |
| 27, 28 | ByteSwap | These bits control the byte swapping of the data after it is aligned and packed/unpacked just prior to doing the PCI write. If the input bytes are labeled ABCD on input then they are swapped as follows:<br>0 = ABCD (i.e. no swap)<br>1 = BADC<br>2 = CDAB<br>3 = DCBA |

The Rectangle DMA mechanism allows image data to be transferred from GLINT to host memory. The image data written to memory may be a sub image of a larger image and have any natural alignment or pixel size. Information regarding the rectangle transfer is held in registers loaded from the input FIFO or a DMA buffer.

The pixel data written to host memory is always packed, however when read from GLINT it can be in packed or unpacked format.

The minimum number of PCI writes are used to align and pack the image data.

GLINT is set up to rasterize the source area for the pixel data (depth, stencil, color, etc.) with the **FilterMode** set up to allow the appropriate data through (the tag should not be included). The rasterization is best set up before the Rectangular DMA is started, but as this is asynchronous it is not necessary to do things in this order.

This register must be loaded by itself and not as part of any increment, hold or indexed group.

See also **DMARectangleWriteAddress**
**DMARectangleWriteLinePitch**
**DMAReadGLINTSource**

## DMARectangleWriteAddress

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| DMARectangleWriteAddress | 0x53A | 0x0000.29D0 | Read/Write | Undefined | Integer |

This register provides the byte address of the first pixel in the image or sub image to read during a rectangular DMA transfer from GLINT to host memory. This is treated as a logical or physical address depending on the LogicalAddressing control bit in the *CommandMode* PCI register. The address should be aligned to the natural size of the pixel, except for 24 bit pixels which may be aligned to any byte boundary.

If logical addresses are used then the corresponding page table entries must be set up to allow write access.

This register must be loaded by itself and not as part of any increment, hold or indexed group.

See also **DMARectangleWrite**
**DMARectangleWriteLinePitch**
**DMAReadGLINTSource**

## DMARectangleWriteLinePitch

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| DMARectangleWriteLinePitch | 0x53B | 0x0000.29D8 | Read/Write | Undefined | Integer |

This register defines the amount, in bytes, to move from one scanline in the image to the next scanline during a rectangular DMA transfer from GLINT to host memory. For a sub image this is based on width of the whole image. The pitch is held as a 32 bit 2's complement number. This is normally an integer multiple of the number of bytes in a group.

See also **DMARectangleWriteAddress**
**DMARectangleWrite**
**DMAReadGLINTSource**

## DMAReturn

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| DMAReturn | 0x534 | 0x0000.29A0 | Write | Undefined | Unused |

This command causes the current DMA transfer to be terminated and control is returned back to whoever initiated the DMA transfer. This can be used with **DMACall** or **DMACount** commands. The **DMAReturn** will cause a stack underflow if the stack is empty and it was loaded into the stack FIFO. An error bit (StackUnderError) in *CommandError* is set and this will cause an interrupt, if so enabled.

This register must be loaded by itself and not as part of any increment, hold or indexed group.

## DrawLine01 DrawLine10

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| DrawLine01 | 0x263 | 0x0000.1318 | Write | Undefined | Bitfield |
| DrawLine10 | 0x264 | 0x0000.1320 | Write | Undefined | Bitfield |

These commands draw a line from vertex store 0 to vertex store 1 (**DrawLine01**) or from vertex store 1 to vertex store 0 (**DrawLine10**). The data with this command is identical to the data with the **Render** command.

These are legacy commands and should only be used if Gamma is just being used as a faster GLINT Delta and none of the additional Gamma functionality is being used.

## DrawRectangle2D

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| DrawRectangle2D | 0x2F4 | 0x0000.17A0 | Write | Undefined | Bitfield |

This command causes GLINT to rasterize a rectangle using the width, height and various modes found in the **Rectangle2DMode** register. The data supplied with this command has the 2's complement X coordinate in the lower 16 bits and the 2's complement Y coordinate in the upper 16 bits.

See also **Rectangle2DMode**, **Rectangle2DControl**

## DrawTriangle

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| DrawTriangle | 0x261 | 0x0000.1308 | Write | Undefined | Bitfield |

This command draws a triangle from vertex store 0, 1 and 2. The data with this command is identical to the data with the **Render** command.

This is a legacy command and should only be used if Gamma is just being used as a faster GLINT Delta and none of the additional Gamma functionality is being used.

## EdgeFlag

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| EdgeFlag | 0x2B4 | 0x0000.15A0 | Read/Write | Undefined | Boolean |

This register holds the current edge flag. The edge flag is associated with the following vertex (it starts the next edge) and is used to disable edges being drawn when Polygon Mode (set in **GeometryMode** register) is lines or points. It only has an effect when the primitive type is Triangles, Quads or Polygons. The edge is always set for TriangleStrip, TriangleFan and QuadStrip.

The data value (0) disables the edge being drawn (for the appropriate primitive) and the value (1) enables the edge being drawn.

## End

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| End | 0x2B3 | 0x0000.1598 | Write | Undefined | Unused |

This command marks the end of a sequence of vertices and causes any residual lines or triangles to be rendered. This action is dependent on the primitive type defined by the **Begin** command and the number of vertices received (since the **Begin**).

See also: **Begin**.

## EndOfFeedback

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| EndOfFeedback | 0x1FF | 0x0000.0FF8 | Write | Undefined | Unused |

This command is sent to mark the end of data the output DMA controller is collecting when it is in feedback mode. When the output DMA controller (or host software) detects this tag in the GLINT Host Out FIFO it knows all the preceding feedback, select or context data has been read from the FIFO.

## FeedbackAlpha
## FeedbackBlue
## FeedbackGreen
## FeedbackQ
## FeedbackR
## FeedbackRed
## FeedbackS
## FeedbackT
## FeedbackToken
## FeedbackW
## FeedbackX
## FeedbackY
## FeedbackZ

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| FeedbackAlpha | 0x1F8 | 0x0000.0FC0 | Output | Undefined | Float |
| FeedbackBlue | 0x1F7 | 0x0000.0FB8 | Output | Undefined | Float |
| FeedbackGreen | 0x1F6 | 0x0000.0FB0 | Output | Undefined | Float |
| FeedbackQ | 0x1FC | 0x0000.0FE0 | Output | Undefined | Float |
| FeedbackR | 0x1FB | 0x0000.0FD8 | Output | Undefined | Float |
| FeedbackRed | 0x1F5 | 0x0000.0FA8 | Output | Undefined | Float |
| FeedbackS | 0x1F9 | 0x0000.0FC8 | Output | Undefined | Float |
| FeedbackT | 0x1FA | 0x0000.0FD0 | Output | Undefined | Float |
| FeedbackToken | 0x1F0 | 0x0000.0F80 | Output | Undefined | Integer |
| FeedbackW | 0x1F4 | 0x0000.0FA0 | Output | Undefined | Float |
| FeedbackX | 0x1F1 | 0x0000.0F88 | Output | Undefined | Float |
| FeedbackY | 0x1F2 | 0x0000.0F90 | Output | Undefined | Float |
| FeedbackZ | 0x1F3 | 0x0000.0F98 | Output | Undefined | Float |

These tags will be read from GLINT's Host Out FIFO when Gamma is in feedback mode (RenderMode field in **GeometryMode** is set to feedback). The actual tags returned depend on what parameters have been selected.

The data with the **FeedbackToken** tag defines what primitive would have been drawn and implicitly how many vertices worth of vertex data are in the FIFO. The data field can be one of the following:

| Data value (0x44e*****) | 02000 | 04000 | 06000 | 08000 | 0a000 | 0c000 | 0e000 | 00000 |
|---|---|---|---|---|---|---|---|---|
| Primitive Type | Point | Line | Triangle | Bitmap | DrawPixel | CopyPixel | LineReset | PassThrough |
| Number of vertices | 1 | 2 | 3 | 1 | 2 | 3 | 2 | 0 |

The LineReset is the same as a Line but the stipple pattern was reset to the start for this line. The hex values supplied match up with the tokens defined by OpenGL.

*Note: The Triangle token is the same as the Polygon token in OpenGL, however the vertex count is fixed at three and not included in the tag and data stream.*

## FNx
## FNy
## FNz

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| FNx | 0x312 | 0x0000.1890 | Read/Write/Trigger | Undefined | Float |
| FNy | 0x311 | 0x0000.1888 | Read/Write | Undefined | Float |
| FNz | 0x310 | 0x0000.1880 | Read/Write | Undefined | Float |

These registers hold the x, y and z face normal components. The **FNx** register must be written last as this write will trigger the face normal to be entered into Gamma. All the face normal components should be written together and not interleaved with writes to the color (C*), normal (N*), texture (T*) or vertex (V*) registers.

The face normals can be used for culling or lighting and it is optionally transformed by the **NormalMatrix**. If the face normal is used for culling only then it does not need to have unit length, however if it is used for lighting then it must have a unit length. The face normal can be optionally normalized, if necessary.

See also: **NormalMatrix**, **TransformMode**, **NormalizeMode**, **GeometryMode**, **LightingMode**.

## FogDensity
## FogEnd
## FogScale

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| FogDensity | 0x376 | 0x0000.1BB0 | Read/Write | Undefined | Float |
| FogEnd | 0x378 | 0x0000.1BC0 | Read/Write | Undefined | Float |
| FogScale | 0x377 | 0x0000.1BB8 | Read/Write | Undefined | Float |

These registers hold the parameters used during the fog calculations. The fog calculation takes the vertex z value in eye coordinates and applies one of the selected equations:

| | |
|---|---|
| $f = e^{-d \times z}$ | Exponential |
| $f = e^{-(d \times z)^2}$ | Exponential squared |
| $f = (\varepsilon - z)\lambda$ where $\lambda = 1/(\varepsilon - s)$ | Linear |

where:

| | |
|---|---|
| $d$ | is the fog density |
| $s$ | is the fog start value |
| $\varepsilon$ | is the fog end value. |

The values $d$ , $\lambda$, and $\varepsilon$ are held in the **FogDensity**, **FogScale** and **FogEnd** registers respectively.

## FrontAlpha

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| FrontAlpha | 0x516 | 0x0000.28B0 | Read/Write | Undefined | Float |

This register holds the alpha value for the front material. Its normal range of values is 0.0…1.0, however any value can be used.

This register would normally take the front material diffuse alpha in OpenGL.

## FrontAmbientColorBlue FrontAmbientColorGreen FrontAmbientColorRed

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| FrontAmbientColorRed | 0x510 | 0x0000.2880 | Read/Write | Undefined | Float |
| FrontAmbientColorGreen | 0x511 | 0x0000.2888 | Read/Write | Undefined | Float |
| FrontAmbientColorBlue | 0x512 | 0x0000.2890 | Read/Write | Undefined | Float |

These registers hold the red, green and blue ambient colors for the front material. Their normal range of values is 0.0…1.0, however any value can be used.

## FrontDiffuseColorBlue
## FrontDiffuseColorGreen
## FrontDiffuseColorRed

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| FrontDiffuseColorBlue | 0x515 | 0x0000.28A8 | Read/Write | Undefined | Float |
| FrontDiffuseColorGreen | 0x514 | 0x0000.28A0 | Read/Write | Undefined | Float |
| FrontDiffuseColorRed | 0x513 | 0x0000.2898 | Read/Write | Undefined | Float |

These registers hold the red, green and blue diffuse colors for the front material. Their normal range of values is 0.0…1.0, however any value can be used.

## FrontEmissiveColorBlue
## FrontEmissiveColorGreen
## FrontEmissiveColorRed

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| FrontEmissiveColorBlue | 0x51C | 0x0000.28E0 | Read/Write | Undefined | Float |
| FrontEmissiveColorGreen | 0x51B | 0x0000.28D8 | Read/Write | Undefined | Float |
| FrontEmissiveColorRed | 0x51A | 0x0000.28D0 | Read/Write | Undefined | Float |

These registers hold the red, green and blue emissive colors for the front material. Their normal range of values is 0.0…1.0, however any value can be used.

## FrontSpecularColorBlue FrontSpecularColorGreen FrontSpecularColorRed

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| FrontSpecularColorBlue | 0x519 | 0x0000.28C8 | Read/Write | Undefined | Float |
| FrontSpecularColorGreen | 0x518 | 0x0000.28C0 | Read/Write | Undefined | Float |
| FrontSpecularColorRed | 0x517 | 0x0000.28B8 | Read/Write | Undefined | Float |

These registers hold the red, green and blue specular colors for the front material. Their normal range of values is 0.0…1.0, however any value can be used. Setting the red, green and blue color to 0.0 is detected and the specular part of the calculations are avoided.

## FrontSpecularExponent

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| FrontSpecularExponent | 0x51D | 0x0000.28E8 | Read/Write | Undefined | Fixed point |

This register holds front material specular exponent as an unsigned 7.4 fixed point format.

## GeometryMode
## GeometryModeAnd
## GeometryModeOr

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| GeometryMode | 0x2A2 | 0x0000.1510 | Read/Write | Undefined | Bitfield |
| GeometryModeAnd | 0x552 | 0x0000.2A90 | Write | Undefined | Bitfield |
| GeometryModeOr | 0x553 | 0x0000.2A98 | Write | Undefined | Bitfield |

This register has the following fields:

| Bit No. | Name | Description |
|---|---|---|
| 0 | TextureEnable | When set causes the texture value associated with a vertex to be calculated. This calculation is just the division by the homogeneous coordinate w. It is qualified by the TextureEnable bit in the **Begin** command. |
| 1 | FogEnable | When set causes the fog value associated with the vertex to be calculated. It is qualified by the FogEnable bit in the **Begin** command. The actual method of calculating the fog value is given by the FogFunction field. |
| 2, 3 | FogFunction | This field selects the function used to calculate the fog value as a function of the vertex's Z value. The options are:<br>0 = Linear<br>1 = Exponential<br>2 = ExponentialSquared |
| 4, 5 | FrontPolyMode | This field selects the how a triangle, quad or polygon should be drawn when its orientation is facing forwards. The options are:<br>0 = Point<br>1 = Line<br>2 = Fill |
| 6, 7 | BackPolyMode | This field selects the how a triangle or quad or polygon should be drawn when its orientation is facing backwards. The options are:<br>0 = Point<br>1 = Line<br>2 = Fill |
| 8 | FrontFaceDirection | This field selects which direction is the 'front' facing direction. The direction is important as it is used to determine if a triangle, etc. should be culled (if enabled), the material to use during lighting, and the PolyMode to use.<br>0 = Clockwise<br>1 = Counter Clockwise |
| 9 | PolygonCull | This field, when set, enables polygon culling based on the front face direction. It is ignored for points, lines and rectangles. |
| 10, 11 | PolygonCullFace | This field determines which direction of face should be culled (if enabled). It has the following values:<br>0 = Front<br>1 = Back<br>2 = Front and Back |

| | | |
|---|---|---|
| 12 | ClipShortLines | Clipping is an expensive operation and for short lines, it is much faster to draw them and rely on the window and/or screen clipping during rasterization. When this bit is set lines below the length given in the **LineClipThreshold** register are not clipped. This does not apply if the line crosses the near, far or user clipping planes. |
| 13 | ClipSmallTriangles | Clipping is an expensive operation and for small triangles it is much faster to draw them and rely on the window and/or screen clipping during rasterization. When this bit is set triangles below the 'area' given in the **TriangleClipThreshold** register are not clipped. This does not apply if the triangle crosses the near, far or user clipping planes. |
| 14, 15 | RenderMode | The RenderMode field controls the action when processing any primitive. The options are:<br>0 = Render<br>1 = Select<br>2 = Feedback |
| 16 … 18 | FeedbackType | This field only has any effect if the RenderMode is Feedback. In this case it determines the parameters to be returned for every primitive. The options are:<br>0 = X, Y 2D<br>1 = X, Y, Z 3D<br>2 = X, Y, Z, R, G, B, A 3DColor<br>3 = X, Y, Z, R, G, B, A, S, T, R, Q 3DColorTexture<br>4 = X, Y, Z, W, R, G, B, A, S, T, R, Q 4DColorTexture |
| 19 | CullUsingFaceNormal | This field, if set will cull using the supplied face normal. The face normal does not have to be of unit length. If no face normal is supplied then the area method of backface culling is used. |
| 20 | AutoGenerateFaceNormal | This field, if set, will calculate the face normal for a triangle or quad to be used in the lighting calculations if one has not been provided. It will be normalized automatically. |
| 21 | FlatShading | When set selects flat shading to be used, otherwise Gouraud shading will be used. |
| 22 … 27 | UserClipMask | There is one bit per user defined clipping plane. Clipping against a plane is enabled when the corresponding bit is set. The clipping plane is defined in eye space by<br>**UserClip*n***{X \| Y \| Z \| W}.<br>Bit 0 (i.e. bit 22 in register) corresponds to UserClip0. |
| 28 | PolygonOffsetPoint | This field, if set, causes the polygon offset to be calculated and applied to the points of a polygon when PolyMode is set to Point. |
| 29 | PolygonOffsetLine | This field, if set, causes the polygon offset to be calculated and applied to the lines of a polygon when PolyMode is set to Line. |
| 30 | PolygonOffsetFill | This field, if set, causes the polygon offset to be calculated and applied to the triangles of a polygon when PolyMode is set to Fill. |
| 31 | InvertFaceNormalCullDirection | This field, if set, causes the supplied Face Normal to be inverted before it is used for backface culling. |

Writing to the **GeometryModeAnd** and **GeometryModeOr** registers logically combine the new value with the existing values in the **GeometryMode** register rather than replacing its contents.

## GeomRectangle

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| GeomRectangle | 0x2D4 | 0x0000.16A0 | Write | Undefined | Bitfield |

The **GeomRectangle** command is used to render the rectangle, based on the position, color, etc. established with the write to the **RPx2**, **RPx3** or **RPx4** registers. If the raster position is not in view then all **GeomRectangle** commands are ignored. If the raster position is in view then the operation is controlled by the data field:

| Bit No. | Name | Description |
|---|---|---|
| 0, 1 | Type | These two bits define the type of rectangle to be inserted into the feedback buffer. They have no effect when not in feedback mode. The options are:<br>0 = Bitmap<br>1 = DrawPixel<br>2 = CopyPixel<br>3 = Don't insert into the feedback buffer. |
| 2 | OffsetEnable | When this bit is set the x and y offset values held in **RasterPosXOffset** and **RasterPosYOffset** respectively displace the raster position window coordinates when the rectangle is rendered. This does not update the raster position state. |
| 3 | SelectEnable | When this bit is set the rectangle takes part in the selection process. |

After every rectangle is submitted using the **GeomRectangle** command (in any RenderMode) the window coordinate x and y components are updated by the amount held in the **RasterPosXIncrement** and **RasterPosYIncrement** registers, respectively. This occurs irrespective of the raster position being in view either before or after the update. If the initial raster position was in view then all subsequent raster positions updated via the increment will be in view. The converse also holds.

The width and height of the rectangle are held in the **RectangleWidth** and **RectangleHeight** registers. The **RectangleMode** register holds the low level enables to control TextureEnable, FogEnable, etc. and has the same format as the **Render** command.

## IncrementObjectID

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| IncrementObjectID | 0x2B6 | 0x0000.15B0 | Write | Undefined | Unused |

This command increments the object ID register. This is for future use.

See also: **ObjectIDValue**.

## InitNames

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| InitNames | 0x2BB | 0x0000.15D8 | Write | Undefined | Unused |

This command resets the name stack and hit flag when the RenderMode field in **GeometryMode** register is set to select mode, otherwise it is ignored.

See also: **PushName**, **PopName**, **LoadName**.

## LBReadModeAnd
## LBReadModeOr

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| LBReadModeAnd | 0x572 | 0x0000.2B90 | Write | Undefined | Bitfield |
| LBReadModeOr | 0x573 | 0x0000.2B98 | Write | Undefined | Bitfield |

See the GLINT documentation for the field definition of the **LBReadMode** and these registers.

Gamma tracks GLINT's **LBReadMode** to allow its functionality to be extended to better support window clipping using GIDs.

Writing to the **LBReadModeAnd** and **LBReadModeOr** registers logically combine the new value with the existing values in the **LBReadMode** register rather than replacing its contents.

The DestinationReadEnable field is OR'ed with the **Rectangle2DControl** register to always force local buffer reads (where the GID is held) when window clipping is enabled.

## LightingMode
## LightingModeAnd
## LightingModeOr

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| LightingMode | 0x2A4 | 0x0000.1520 | Read/Write | Undefined | Bitfield |
| LightingModeAnd | 0x556 | 0x0000.2AB0 | Write | Undefined | Bitfield |
| LightingModeOr | 0x557 | 0x0000.2AB8 | Write | Undefined | Bitfield |

This register has the following fields:

| Bit No. | Name | Description |
|---|---|---|
| 0 | Enable | When set causes the vertex to be lit using the lighting equations, otherwise the current color is assigned. |
| 1,2 | TwoSidedLighting | The three options are:<br>0 Use the front side materials.<br>1 Use the back side materials and invert the normal before it is used in the lighting calculations.<br>2 Use the orientation of the face to select between front or back materials and lighting. The orientation is determined by the geometry processing. Fields in the **GeometryMode** register control the association between a front face and the vertex ordering. |
| 3 | LocalViewer | When set causes the viewer to be at (0, 0, 1) in eye coordinates, otherwise the viewer is at (0, 0, ). When the viewer is at infinity some of the lighting equations can be simplified and so run faster, however the position of the specular highlights are not as correct. |
| 4 | FlipNormal | When set causes the absolute value of the lighting dot products to be taken, otherwise negative dot products are clamped. Clamping is used for OpenGL, but some other APIs allow the normal to be flipped - this gives a cheap form of two sided lighting and is useful when the normals are not consistently facing 'outwards' in the model or scene. |
| 5 | AttenuationTest | When set forces the lighting calculation for the current light to be aborted when the product of *atti* and *spot* (in the lighting equations) for the light falls below the threshold given in the **AttenuationCutOff** register. This optimization allows lights which are becoming too faint to contribute to be terminated early. A suitable value is given by: 1.0/512*n* where *n* is the number of lights. The 512 constant was chosen as it is less than the smallest representable color when converted to a byte integer assuming the light and material colors are restricted to the range 0.0…1.0. |
| 6…14 | NumberLights | This 9 bit field holds the number of lights to use. Its legal range is 0…16 inclusive. Numbers greater than 16 are clamped to be 16. |
| 15 | SpecularLightingEnable | When this bit is set the specular part of the lighting calculations are done, otherwise they are skipped. |
| 16 | UseFaceNormal | When this bit is set the face normal is used instead of the vertex normals. The lighting is still evaluated once per vertex so any position dependent effects (i.e. attenuation or spotlight) are still computed correctly. |

Writing to the **LightingModeAnd** and **LightingModeOr** registers logically combine the new value with the existing values in the **LightingMode** register rather than replacing its contents.

## Light*n*AmbientIntensityBlue
## Light*n*AmbientIntensityGreen
## Light*n*AmbientIntensityRed

| Light | Red Tag | Red Offset | Green Tag | Green Offset | Blue Tag | Blue Offset |
|---|---|---|---|---|---|---|
| 0 | 0x3A1 | 0x0000.1D08 | 0x3A2 | 0x0000.1D10 | 0x3A3 | 0x0000.1D18 |
| 1 | 0x3B7 | 0x0000.1DB8 | 0x3B8 | 0x0000.1DC0 | 0x3B9 | 0x0000.1DC8 |
| 2 | 0x3CD | 0x0000.1E68 | 0x3CE | 0x0000.1E70 | 0x3CF | 0x0000.1E78 |
| 3 | 0x3E3 | 0x0000.1F18 | 0x3E4 | 0x0000.1F20 | 0x3E5 | 0x0000.1F28 |
| 4 | 0x3F9 | 0x0000.1FC8 | 0x3FA | 0x0000.1FD0 | 0x3FB | 0x0000.1FD8 |
| 5 | 0x40F | 0x0000.2078 | 0x410 | 0x0000.2080 | 0x411 | 0x0000.2088 |
| 6 | 0x425 | 0x0000.2128 | 0x426 | 0x0000.2130 | 0x427 | 0x0000.2138 |
| 7 | 0x43B | 0x0000.21D8 | 0x43C | 0x0000.21E0 | 0x43D | 0x0000.21E8 |
| 8 | 0x451 | 0x0000.2288 | 0x452 | 0x0000.2290 | 0x453 | 0x0000.2298 |
| 9 | 0x467 | 0x0000.2338 | 0x468 | 0x0000.2340 | 0x469 | 0x0000.2348 |
| 10 | 0x47D | 0x0000.23E8 | 0x47E | 0x0000.23F0 | 0x47F | 0x0000.23F8 |
| 11 | 0x493 | 0x0000.2498 | 0x494 | 0x0000.24A0 | 0x495 | 0x0000.24A8 |
| 12 | 0x4A9 | 0x0000.2548 | 0x4AA | 0x0000.2550 | 0x4AB | 0x0000.2558 |
| 13 | 0x4BF | 0x0000.25F8 | 0x4C0 | 0x0000.2600 | 0x4C1 | 0x0000.2608 |
| 14 | 0x4D5 | 0x0000.26A8 | 0x4D6 | 0x0000.26B0 | 0x4D7 | 0x0000.26B8 |
| 15 | 0x4EB | 0x0000.2758 | 0x4EC | 0x0000.2760 | 0x4ED | 0x0000.2768 |

| Name | Access | Reset | Format |
|---|---|---|---|
| *all*Read/Write | Undefined | Float | |

These registers hold the ambient intensity per color component the light contributes to the scene. The nominal range of values is 0.0 to 1.0, although this is not enforced.

The register name is formed by inserting the light number for *n* and appending Red, Green or Blue as appropriate.

## Light*n*ConstantAttenuation

| **Light** | **Tag** | **Offset** |
|---|---|---|
| 0 | 0x3B3 | 0x0000.1D98 |
| 1 | 0x3C9 | 0x0000.1E48 |
| 2 | 0x3DF | 0x0000.1EF8 |
| 3 | 0x3F5 | 0x0000.1FA8 |
| 4 | 0x40B | 0x0000.2058 |
| 5 | 0x421 | 0x0000.2108 |
| 6 | 0x437 | 0x0000.21B8 |
| 7 | 0x44D | 0x0000.2268 |
| 8 | 0x463 | 0x0000.2318 |
| 9 | 0x479 | 0x0000.23C8 |
| 10 | 0x48F | 0x0000.2478 |
| 11 | 0x4A5 | 0x0000.2528 |
| 12 | 0x4BB | 0x0000.25D8 |
| 13 | 0x4D1 | 0x0000.2688 |
| 14 | 0x4E7 | 0x0000.2738 |
| 15 | 0x4FD | 0x0000.27E8 |

| **Name** | **Access** | **Reset** | **Format** |
|---|---|---|---|
| *all*Read/Write | Undefined | Float | |

These registers hold the constant attenuation factor the lights intensity is decreased by .

The register name is formed by inserting the light number for *n* .

## Light*n*CosSpotlightCutoffAngle

| Light | Tag | Offset |
|---|---|---|
| 0 | 0x3B2 | 0x0000.1D90 |
| 1 | 0x3C8 | 0x0000.1E40 |
| 2 | 0x3DE | 0x0000.1EF0 |
| 3 | 0x3F4 | 0x0000.1FA0 |
| 4 | 0x40A | 0x0000.2050 |
| 5 | 0x420 | 0x0000.2100 |
| 6 | 0x436 | 0x0000.21B0 |
| 7 | 0x44C | 0x0000.2260 |
| 8 | 0x462 | 0x0000.2310 |
| 9 | 0x478 | 0x0000.23C0 |
| 10 | 0x48E | 0x0000.2470 |
| 11 | 0x4A4 | 0x0000.2520 |
| 12 | 0x4BA | 0x0000.25D0 |
| 13 | 0x4D0 | 0x0000.2680 |
| 14 | 0x4E6 | 0x0000.2730 |
| 15 | 0x4FC | 0x0000.27E0 |

| Name | Access | Reset | Format |
|---|---|---|---|
| *all*Read/Write | Undefined | Float | |

These registers hold the cosine of the spot light cut off angle.

The register name is formed by inserting the light number for *n* .

## Light*n*DiffuseIntensityBlue
## Light*n*DiffuseIntensityGreen
## Light*n*DiffuseIntensityRed

| Light | Red Tag | Red Offset | Green Tag | Green Offset | Blue Tag | Blue Offset |
|---|---|---|---|---|---|---|
| 0 | 0x3A4 | 0x0000.1D20 | 0x3A5 | 0x0000.1D28 | 0x3A6 | 0x0000.1D30 |
| 1 | 0x3BA | 0x0000.1DD0 | 0x3BB | 0x0000.1DD8 | 0x3BC | 0x0000.1DE0 |
| 2 | 0x3D0 | 0x0000.1E80 | 0x3D1 | 0x0000.1E88 | 0x3D2 | 0x0000.1E90 |
| 3 | 0x3E6 | 0x0000.1F30 | 0x3E7 | 0x0000.1F38 | 0x3E8 | 0x0000.1F40 |
| 4 | 0x3FC | 0x0000.1FE0 | 0x3FD | 0x0000.1FE8 | 0x3FE | 0x0000.1FF0 |
| 5 | 0x412 | 0x0000.2090 | 0x413 | 0x0000.2098 | 0x414 | 0x0000.20A0 |
| 6 | 0x428 | 0x0000.2140 | 0x429 | 0x0000.2148 | 0x42A | 0x0000.2150 |
| 7 | 0x43E | 0x0000.21F0 | 0x43F | 0x0000.21F8 | 0x440 | 0x0000.2200 |
| 8 | 0x454 | 0x0000.22A0 | 0x455 | 0x0000.22A8 | 0x456 | 0x0000.22B0 |
| 9 | 0x46A | 0x0000.2350 | 0x46B | 0x0000.2358 | 0x46C | 0x0000.2360 |
| 10 | 0x480 | 0x0000.2400 | 0x481 | 0x0000.2408 | 0x482 | 0x0000.2410 |
| 11 | 0x496 | 0x0000.24B0 | 0x497 | 0x0000.24B8 | 0x498 | 0x0000.24C0 |
| 12 | 0x4AC | 0x0000.2560 | 0x4AD | 0x0000.2568 | 0x4AE | 0x0000.2570 |
| 13 | 0x4C2 | 0x0000.2610 | 0x4C3 | 0x0000.2618 | 0x4C4 | 0x0000.2620 |
| 14 | 0x4D8 | 0x0000.26C0 | 0x4D9 | 0x0000.26C8 | 0x4DA | 0x0000.26D0 |
| 15 | 0x4EE | 0x0000.2770 | 0x4EF | 0x0000.2778 | 0x4F0 | 0x0000.2780 |

| Name | Access | Reset | Format |
|---|---|---|---|
| *all* | Read/Write | Undefined | Float |

These registers hold the diffuse intensity per color component the light contributes to the scene. The nominal range of values is 0.0 to 1.0, although this is not enforced.

The register name is formed by inserting the light number for *n* and appending Red, Green or Blue as appropriate.

## Light*n*LinearAttenuation

| Light | Tag | Offset |
|---|---|---|
| 0 | 0x3B4 | 0x0000.1DA0 |
| 1 | 0x3CA | 0x0000.1E50 |
| 2 | 0x3E0 | 0x0000.1F00 |
| 3 | 0x3F6 | 0x0000.1FB0 |
| 4 | 0x40C | 0x0000.2060 |
| 5 | 0x422 | 0x0000.2110 |
| 6 | 0x438 | 0x0000.21C0 |
| 7 | 0x44E | 0x0000.2270 |
| 8 | 0x464 | 0x0000.2320 |
| 9 | 0x47A | 0x0000.23D0 |
| 10 | 0x490 | 0x0000.2480 |
| 11 | 0x4A6 | 0x0000.2530 |
| 12 | 0x4BC | 0x0000.25E0 |
| 13 | 0x4D2 | 0x0000.2690 |
| 14 | 0x4E8 | 0x0000.2740 |
| 15 | 0x4FE | 0x0000.27F0 |

| Name | Access | Reset | Format |
|---|---|---|---|
| *all*Read/Write | Undefined | Float | |

These registers hold the linear attenuation factor the lights intensity is decreased by .

The register name is formed by inserting the light number for *n* .

## Light*n*Mode

| Light | Tag | Offset |
|---|---|---|
| 0 | 0x3A0 | 0x0000.1D00 |
| 1 | 0x3B6 | 0x0000.1DB0 |
| 2 | 0x3CC | 0x0000.1E60 |
| 3 | 0x3E2 | 0x0000.1F10 |
| 4 | 0x3F8 | 0x0000.1FC0 |
| 5 | 0x40E | 0x0000.2070 |
| 6 | 0x424 | 0x0000.2120 |
| 7 | 0x43A | 0x0000.21D0 |
| 8 | 0x450 | 0x0000.2280 |
| 9 | 0x466 | 0x0000.2330 |
| 10 | 0x47C | 0x0000.23E0 |
| 11 | 0x492 | 0x0000.2490 |
| 12 | 0x4A8 | 0x0000.2540 |
| 13 | 0x4BE | 0x0000.25F0 |
| 14 | 0x4D4 | 0x0000.26A0 |
| 15 | 0x4EA | 0x0000.2750 |

| Name | Access | Reset | Format |
|---|---|---|---|
| *all*Read/Write | Undefined | Bitfield | |

These registers hold the individual mode words for each light. The mode as the following fields:

| Bit No. | Name | Description |
|---|---|---|
| 0 | LightOn | When set indicates the light is on and contributes illumination to the scene, otherwise it does not. |
| 1 | Spotlight | When set indicates the light is a spotlight. If it is not set then the light is not a spot light and the SpotlightDirection is used to hold the normalized half vector between the viewer and the light. OpenGL would set this bit when the spot light's cut off angle is not 180 degrees. |
| 2 | Attenuation | When set indicates the light is to be attenuated, otherwise no attenuation is done. OpenGL would set this when the W component of the light's position is non zero. |
| 3 | LocalLight | When set indicates the light is local and the full lighting equations should be used. This allows a light to be local without it having to be a spotlight or have any attenuation applied to it. |

The register name is formed by inserting the light number for *n*.

## Light*n*PositionW
## Light*n*PositionX
## Light*n*PositionY
## Light*n*PositionZ

| Light | X Tag | X Offset | Y Tag | Y Offset | Z Tag | Z Offset | W Tag | W Offset |
|---|---|---|---|---|---|---|---|---|
| 0 | 0x3AA | 0x0000.1D50 | 0x3AB | 0x0000.1D58 | 0x3AC | 0x0000.1D60 | 0x3AD | 0x0000.1D68 |
| 1 | 0x3C0 | 0x0000.1E00 | 0x3C1 | 0x0000.1E08 | 0x3C2 | 0x0000.1E10 | 0x3C3 | 0x0000.1E18 |
| 2 | 0x3D6 | 0x0000.1EB0 | 0x3D7 | 0x0000.1EB8 | 0x3D8 | 0x0000.1EC0 | 0x3D9 | 0x0000.1EC8 |
| 3 | 0x3EC | 0x0000.1F60 | 0x3ED | 0x0000.1F68 | 0x3EE | 0x0000.1F70 | 0x3EF | 0x0000.1F78 |
| 4 | 0x402 | 0x0000.2010 | 0x403 | 0x0000.2018 | 0x404 | 0x0000.2020 | 0x405 | 0x0000.2028 |
| 5 | 0x418 | 0x0000.20C0 | 0x419 | 0x0000.20C8 | 0x41A | 0x0000.20D0 | 0x41B | 0x0000.20D8 |
| 6 | 0x42E | 0x0000.2170 | 0x42F | 0x0000.2178 | 0x430 | 0x0000.2180 | 0x431 | 0x0000.2188 |
| 7 | 0x444 | 0x0000.2220 | 0x445 | 0x0000.2228 | 0x446 | 0x0000.2230 | 0x447 | 0x0000.2238 |
| 8 | 0x45A | 0x0000.22D0 | 0x45B | 0x0000.22D8 | 0x45C | 0x0000.22E0 | 0x45D | 0x0000.22E8 |
| 9 | 0x470 | 0x0000.2380 | 0x471 | 0x0000.2388 | 0x472 | 0x0000.2390 | 0x473 | 0x0000.2398 |
| 10 | 0x486 | 0x0000.2430 | 0x487 | 0x0000.2438 | 0x488 | 0x0000.2440 | 0x489 | 0x0000.2448 |
| 11 | 0x49C | 0x0000.24E0 | 0x49D | 0x0000.24E8 | 0x49E | 0x0000.24F0 | 0x49F | 0x0000.24F8 |
| 12 | 0x4B2 | 0x0000.2590 | 0x4B3 | 0x0000.2598 | 0x4B4 | 0x0000.25A0 | 0x4B5 | 0x0000.25A8 |
| 13 | 0x4C8 | 0x0000.2640 | 0x4C9 | 0x0000.2648 | 0x4CA | 0x0000.2650 | 0x4CB | 0x0000.2658 |
| 14 | 0x4DE | 0x0000.26F0 | 0x4DF | 0x0000.26F8 | 0x4E0 | 0x0000.2700 | 0x4E1 | 0x0000.2708 |
| 15 | 0x4F4 | 0x0000.27A0 | 0x4F5 | 0x0000.27A8 | 0x4F6 | 0x0000.27B0 | 0x4F7 | 0x0000.27B8 |

| Name | Access | Reset | Format |
|---|---|---|---|
| *all*Read/Write | Undefined | Float | |

These registers hold the position or direction of the light. If the direction is being held then the W component should be set to zero and the direction is normalized. The Attenuation bit in the light's mode should be 0 for a direction light or 1 for a position light.

The register name is formed by inserting the light number for *n* and appending X, Y, Z or W as appropriate.

## Light*n*QuadraticAttenuation

| Light | Tag | Offset |
|---|---|---|
| 0 | 0x3B5 | 0x0000.1DA8 |
| 1 | 0x3CB | 0x0000.1E58 |
| 2 | 0x3E1 | 0x0000.1F08 |
| 3 | 0x3F7 | 0x0000.1FB8 |
| 4 | 0x40D | 0x0000.2068 |
| 5 | 0x423 | 0x0000.2118 |
| 6 | 0x439 | 0x0000.21C8 |
| 7 | 0x44F | 0x0000.2278 |
| 8 | 0x465 | 0x0000.2328 |
| 9 | 0x47B | 0x0000.23D8 |
| 10 | 0x491 | 0x0000.2488 |
| 11 | 0x4A7 | 0x0000.2538 |
| 12 | 0x4BD | 0x0000.25E8 |
| 13 | 0x4D3 | 0x0000.2698 |
| 14 | 0x4E9 | 0x0000.2748 |
| 15 | 0x4FF | 0x0000.27F8 |

| Name | Access | Reset | Format |
|---|---|---|---|
| *all*Read/Write | Undefined | Float | |

These registers hold the quadratic attenuation factor corresponding to the decrease in a lights intensity.

The register name is formed by inserting the light number for *n* .

## Light*n*SpecularIntensityBlue
## Light*n*SpecularIntensityGreen
## Light*n*SpecularIntensityRed

| Light | Red Tag | Red Offset | Green Tag | Green Offset | Blue Tag | Blue Offset |
|---|---|---|---|---|---|---|
| 0 | 0x3A7 | 0x0000.1D38 | 0x3A8 | 0x0000.1D40 | 0x3A9 | 0x0000.1D48 |
| 1 | 0x3BD | 0x0000.1DE8 | 0x3BE | 0x0000.1DF0 | 0x3BF | 0x0000.1DF8 |
| 2 | 0x3D3 | 0x0000.1E98 | 0x3D4 | 0x0000.1EA0 | 0x3D5 | 0x0000.1EA8 |
| 3 | 0x3E9 | 0x0000.1F48 | 0x3EA | 0x0000.1F50 | 0x3EB | 0x0000.1F58 |
| 4 | 0x3FF | 0x0000.1FF8 | 0x400 | 0x0000.2000 | 0x401 | 0x0000.2008 |
| 5 | 0x415 | 0x0000.20A8 | 0x416 | 0x0000.20B0 | 0x417 | 0x0000.20B8 |
| 6 | 0x42B | 0x0000.2158 | 0x42C | 0x0000.2160 | 0x42D | 0x0000.2168 |
| 7 | 0x441 | 0x0000.2208 | 0x442 | 0x0000.2210 | 0x443 | 0x0000.2218 |
| 8 | 0x457 | 0x0000.22B8 | 0x458 | 0x0000.22C0 | 0x459 | 0x0000.22C8 |
| 9 | 0x46D | 0x0000.2368 | 0x46E | 0x0000.2370 | 0x46F | 0x0000.2378 |
| 10 | 0x483 | 0x0000.2418 | 0x484 | 0x0000.2420 | 0x485 | 0x0000.2428 |
| 11 | 0x499 | 0x0000.24C8 | 0x49A | 0x0000.24D0 | 0x49B | 0x0000.24D8 |
| 12 | 0x4AF | 0x0000.2578 | 0x4B0 | 0x0000.2580 | 0x4B1 | 0x0000.2588 |
| 13 | 0x4C5 | 0x0000.2628 | 0x4C6 | 0x0000.2630 | 0x4C7 | 0x0000.2638 |
| 14 | 0x4DB | 0x0000.26D8 | 0x4DC | 0x0000.26E0 | 0x4DD | 0x0000.26E8 |
| 15 | 0x4F1 | 0x0000.2788 | 0x4F2 | 0x0000.2790 | 0x4F3 | 0x0000.2798 |

| Name | Access | Reset | Format |
|---|---|---|---|
| *all* | Read/Write | Undefined | Float |

These registers hold the specular intensity per color component the light contributes to the scene. The nominal range of values is 0.0 to 1.0, although this is not enforced.

The register name is formed by inserting the light number for *n* and appending Red, Green or Blue as appropriate.

## Light*n*SpotlightDirectionW
## Light*n*SpotlightDirectionX
## Light*n*SpotlightDirectionY
## Light*n*SpotlightDirectionZ

| Light | X Tag | X Offset | Y Tag | Y Offset | Z Tag | Z Offset |
|---|---|---|---|---|---|---|
| 0 | 0x3AE | 0x0000.1D70 | 0x3AF | 0x0000.1D78 | 0x3B0 | 0x0000.1D80 |
| 1 | 0x3C4 | 0x0000.1E20 | 0x3C5 | 0x0000.1E28 | 0x3C6 | 0x0000.1E30 |
| 2 | 0x3DA | 0x0000.1ED0 | 0x3DB | 0x0000.1ED8 | 0x3DC | 0x0000.1EE0 |
| 3 | 0x3F0 | 0x0000.1F80 | 0x3F1 | 0x0000.1F88 | 0x3F2 | 0x0000.1F90 |
| 4 | 0x406 | 0x0000.2030 | 0x407 | 0x0000.2038 | 0x408 | 0x0000.2040 |
| 5 | 0x41C | 0x0000.20E0 | 0x41D | 0x0000.20E8 | 0x41E | 0x0000.20F0 |
| 6 | 0x432 | 0x0000.2190 | 0x433 | 0x0000.2198 | 0x434 | 0x0000.21A0 |
| 7 | 0x448 | 0x0000.2240 | 0x449 | 0x0000.2248 | 0x44A | 0x0000.2250 |
| 8 | 0x45E | 0x0000.22F0 | 0x45F | 0x0000.22F8 | 0x460 | 0x0000.2300 |
| 9 | 0x474 | 0x0000.23A0 | 0x475 | 0x0000.23A8 | 0x476 | 0x0000.23B0 |
| 10 | 0x48A | 0x0000.2450 | 0x48B | 0x0000.2458 | 0x48C | 0x0000.2460 |
| 11 | 0x4A0 | 0x0000.2500 | 0x4A1 | 0x0000.2508 | 0x4A2 | 0x0000.2510 |
| 12 | 0x4B6 | 0x0000.25B0 | 0x4B7 | 0x0000.25B8 | 0x4B8 | 0x0000.25C0 |
| 13 | 0x4CC | 0x0000.2660 | 0x4CD | 0x0000.2668 | 0x4CE | 0x0000.2670 |
| 14 | 0x4E2 | 0x0000.2710 | 0x4E3 | 0x0000.2718 | 0x4E4 | 0x0000.2720 |
| 15 | 0x4F8 | 0x0000.27C0 | 0x4F9 | 0x0000.27C8 | 0x4FA | 0x0000.27D0 |

| Name | Access | Reset | Format |
|---|---|---|---|
| *all*Read/Write | Undefined | Float | |

When the Spotlight bit in the light's mode indicates the light is a spotlight these registers hold the normalized spot lights direction.

When the Spotlight bit in the light's mode indicates the light is not a spotlight these registers hold the normalized half vector. The half vector between the light's position vector and the eye vector is given by the following equation:

$$\mathbf{h} = \hat{\mathbf{P}}_{pli} + (0 \quad 0 \quad 1)$$

The register name is formed by inserting the light number for *n* and appending X, Y or Z as appropriate.

## Light*n*SpotlightExponent

| Light | Tag | Offset |
|---|---|---|
| 0 | 0x3B1 | 0x0000.1D88 |
| 1 | 0x3C7 | 0x0000.1E38 |
| 2 | 0x3DD | 0x0000.1EE8 |
| 3 | 0x3F3 | 0x0000.1F98 |
| 4 | 0x409 | 0x0000.2048 |
| 5 | 0x41F | 0x0000.20F8 |
| 6 | 0x435 | 0x0000.21A8 |
| 7 | 0x44B | 0x0000.2258 |
| 8 | 0x461 | 0x0000.2308 |
| 9 | 0x477 | 0x0000.23B8 |
| 10 | 0x48D | 0x0000.2468 |
| 11 | 0x4A3 | 0x0000.2518 |
| 12 | 0x4B9 | 0x0000.25C8 |
| 13 | 0x4CF | 0x0000.2678 |
| 14 | 0x4E5 | 0x0000.2728 |
| 15 | 0x4FB | 0x0000.27D8 |

| Name | Access | Reset | Format |
|---|---|---|---|
| *all* | Read/Write | Undefined | Fixed point |

These registers hold the spot light exponent as an unsigned 7.4 fixed point number.

The register name is formed by inserting the light number for *n* .

## LineClipLengthThreshold

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| LineClipLengthThreshold | 0x37B | 0x0000.1BD8 | Read/Write | Undefined | Float |

This register holds the length (in pixels) below which lines should not be clipped as it is faster to use screen or window clipping during rasterization . This is enabled in the **GeometryMode** register and automatically disabled if the line crossed the near, far or user clipping planes.

It is best to disable this if any of the following conditions arise:

- The viewport is smaller than the window and user scissoring isn't enabled. The rasterization level clipping cannot be relied upon to do the clipping.
- Select or Feedback modes are enabled. Short lines which would otherwise have been clipped out will be included in the select or feedback data.

## LineMode
## LineModeAnd
## LineModeOr

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| LineMode | 0x295 | 0x0000.14A8 | Read/Write | Undefined | Bitfield |
| LineModeAnd | 0x55E | 0x0000.2AF0 | Write | Undefined | Bitfield |
| LineModeOr | 0x55F | 0x0000.2AF8 | Write | Undefined | Bitfield |

The **LineMode** register defines how lines are to be rendered and has the following fields:

| Bit No. | Name | Description |
|---|---|---|
| 0 | StippleEnable | This field, when set, enables the stippling of lines. It only effects wide lines or antialiased lines. This will normally be the same value as the Enable field in the **LineStippleMode** GLINT register. |
| 1…9 | RepeatFactor | This 9 bit field holds the repeat factor for antialiased stippled lines. This will normally be the same value as the RepeatFactor field in the **LineStippleMode** GLINT register. The repeat factor stored here is one less than the desired repeat factor. |
| 10…25 | StippleMask | This 16 bit field holds the stipple pattern to use for antialiased lines. This will normally be the same value as the StippleMask field in the **LineStippleMode** GLINT register. |
| 26 | Mirror | This field, when set, will mirror the StippleMask before it is used for antialiased lines. This will normally be the same value as the Mirror field in the **LineStippleMode** GLINT register. |
| 27 | AntialiasEnable | This field, when set, enables antialiasing of lines. This is qualified by the AntialiasEnable field in the **Begin** command. |
| 28 | AntialiasingQuality | This field defines the quality of antialiased lines:<br>0 = 4x4<br>1 = 8x8 |

Writing to the **LineModeAnd** and **LineModeOr** registers logically combine the new value with the existing values in the **LineMode** register rather than replacing its contents.

## LineWidth

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| LineWidth | 0x296 | 0x0000.14B0 | Read/Write | Undefined | Integer |

This register holds the 8 bit unsigned integer width for aliased lines. The width is measured in pixels. A zero width is treated as if the width were one. The **LineWidthOffset** register must also be set up as well.

## LineWidthOffset

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| LineWidthOffset | 0x297 | 0x0000.14B8 | Read/Write | Undefined | Integer |

The **LineWidthOffset** register is normally set to (**LineWidth** - 1) / 2. For one pixel wide lines the **LineWidthOffset** is set to 0.

This sets up the initial offset subtracted from the line's mathematical X (for Y major lines) or Y (for X major lines) before the line is stepped and repeated **LineWidth** times.

## LoadName

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| LoadName | 0x2BE | 0x0000.15F0 | Write | Undefined | Integer |

This command replaces the name at the top of the name stack with the given name (a 32 bit integer) when in select mode (RenderMode field in **GeometryMode** register).

If the hit flag is set and select mode is enabled then the hit record is written to the GLINT's Host Out FIFO and the hit flag reset.

The hit record consists of (in order):

The count of the names (NameCount) on the stack (plus some error flags),

The minimum Z value as a normalized floating point number,

The maximum Z value as a normalized floating point number,

The name stack entries, oldest first (variable number [0…64] words.

Bits 14 and 15 in the **FilterMode** register in GLINT must be set to allow the **SelectRecord** tag and data values to be written in to the FIFO - all the select record data uses the same tag.

The NameCount value has the following fields:

| Bit | Name | Description |
|---|---|---|
| 0…6 | Count | This field holds the number of names on the name stack. |
| 7…28 | | Not used. |
| 29 | InvalidOperation | A **LoadName** operation was attempted on an empty name stack when this hit record was being collected. This is cleared for subsequent hit records (unless they manifest this error) however the stack may no longer be totally valid. |
| 30 | StackUnderflow | The name stack was popped while empty when this hit record was being collected. This is cleared for subsequent hit records (unless they manifest this error) however the stack may no longer be totally valid. |
| 31 | StackOverflow | The name stack was pushed while full when this hit record was being collected. This is cleared for subsequent hit records (unless they manifest this error) however the stack may no longer be totally valid. |

The **LoadName** stack manipulation command is ignored when not in Select mode.

## MaterialMode
## MaterialModeAnd
## MaterialModeOr

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| MaterialMode | 0x2A6 | 0x0000.1530 | Read/Write | Undefined | Bitfield |
| MaterialModeAnd | 0x562 | 0x0000.2B10 | Write | Undefined | Bitfield |
| MaterialModeOr | 0x563 | 0x0000.2B18 | Write | Undefined | Bitfield |

This register has the following fields:

| Bit No. | Name | Description |
|---|---|---|
| 0 | Enable | When set causes the vertex to be calculated from the lighting equations otherwise the current color is assigned. |
| 1 | DiffuseTextureEnable | When set allows the diffuse texture color to be calculated and sent to GLINT. This is further qualified by the EnableTexture bit in the **Begin** command so is only done when texture mapping is enabled. |
| 2 | SpecularTextureEnable | When set allows the specular texture color to be calculated and sent to GLINT. This is further qualified by the EnableTexture bit in the **Begin** command so is only done when texture mapping is enabled. |
| 3 | MonochromeDiffuseTexture | When set the diffuse texture color is converted to a monochrome value before it is sent to GLINT. This allows the diffuse texture DDA in GLINT 500TX to be set up.<br>When clear the true color value is sent and is used when the target rendering chip is GLINT MX as it has true color diffuse texture DDAs. |
| 4 | MonochromeSpecularTexture | When set the specular texture color is converted to a monochrome value before it is sent to GLINT. This allows the specular texture DDA in GLINT 500TX to be set up.<br>When clear the true color value is sent and is used when the target rendering chip is GLINT MX as it has true color specular texture DDAs. |
| 5 | PremultiplyAlpha | When set premultiplies the diffuse and ambient colors by the material alpha value. |
| 6 | ColorSource | This field selects where the color should be taken from when the Enable field is 0. The options are:<br>0 Current color value.<br>1 Diffuse material value. |
| 7, 8 | TwoSidedLighting | The three options are:<br>0 Use the front side materials.<br>1 Use the back side materials.<br>2 Use the orientation of the face to select between front or back materials and lighting. |

The selection between Gouraud and flat shading is controlled by the FlatShading bit in **GeometryMode**. The SmoothShadingEnable bit in the **DeltaMode** register is ignored.

Writing to the **MaterialModeAnd** and **MaterialModeOr** registers logically combine the new value with the existing values in the **MaterialMode** register rather than replacing its contents.

## ModelViewMatrix0 … ModelViewMatrix15

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| ModelViewMatrix0 | 0x320 | 0x0000.1900 | Read/Write | Undefined | Float |
| ModelViewMatrix1 | 0x321 | 0x0000.1908 | Read/Write | Undefined | Float |
| ModelViewMatrix2 | 0x322 | 0x0000.1910 | Read/Write | Undefined | Float |
| ModelViewMatrix3 | 0x323 | 0x0000.1918 | Read/Write | Undefined | Float |
| ModelViewMatrix4 | 0x324 | 0x0000.1920 | Read/Write | Undefined | Float |
| ModelViewMatrix5 | 0x325 | 0x0000.1928 | Read/Write | Undefined | Float |
| ModelViewMatrix6 | 0x326 | 0x0000.1930 | Read/Write | Undefined | Float |
| ModelViewMatrix7 | 0x327 | 0x0000.1938 | Read/Write | Undefined | Float |
| ModelViewMatrix8 | 0x328 | 0x0000.1940 | Read/Write | Undefined | Float |
| ModelViewMatrix9 | 0x329 | 0x0000.1948 | Read/Write | Undefined | Float |
| ModelViewMatrix10 | 0x32A | 0x0000.1950 | Read/Write | Undefined | Float |
| ModelViewMatrix11 | 0x32B | 0x0000.1958 | Read/Write | Undefined | Float |
| ModelViewMatrix12 | 0x32C | 0x0000.1960 | Read/Write | Undefined | Float |
| ModelViewMatrix13 | 0x32D | 0x0000.1968 | Read/Write | Undefined | Float |
| ModelViewMatrix14 | 0x32E | 0x0000.1970 | Read/Write | Undefined | Float |
| ModelViewMatrix15 | 0x32F | 0x0000.1978 | Read/Write | Undefined | Float |

These 16 registers hold the model view matrix used to multiply the input vertex coordinates (as a column vector). If the matrix is subscripted:

$$\begin{pmatrix} M_0 & M_4 & M_8 & M_{12} \\ M_1 & M_5 & M_9 & M_{13} \\ M_2 & M_6 & M_{10} & M_{14} \\ M_3 & M_7 & M_{11} & M_{15} \end{pmatrix}$$

then the numerical subscripts give the order the elements are stored in the matrix registers (i.e. $M_0$ is stored in **ModelViewMatrix0**, for example) and these follow the column-major order convention. Note this is different from the convention C uses which follows the row-major order.

This matrix is only used if enabled by the **TransformMode** register and can be avoided if the vertex in eye space is not needed for fog, lighting, texture generation or user clipping.

## ModelViewProjectionMatrix0… ModelViewProjectionMatrix15

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| ModelViewProjectionMatrix0 | 0x330 | 0x0000.1980 | Read/Write | Undefined | Float |
| ModelViewProjectionMatrix1 | 0x331 | 0x0000.1988 | Read/Write | Undefined | Float |
| ModelViewProjectionMatrix2 | 0x332 | 0x0000.1990 | Read/Write | Undefined | Float |
| ModelViewProjectionMatrix3 | 0x333 | 0x0000.1998 | Read/Write | Undefined | Float |
| ModelViewProjectionMatrix4 | 0x334 | 0x0000.19A0 | Read/Write | Undefined | Float |
| ModelViewProjectionMatrix5 | 0x335 | 0x0000.19A8 | Read/Write | Undefined | Float |
| ModelViewProjectionMatrix6 | 0x336 | 0x0000.19B0 | Read/Write | Undefined | Float |
| ModelViewProjectionMatrix7 | 0x337 | 0x0000.19B8 | Read/Write | Undefined | Float |
| ModelViewProjectionMatrix8 | 0x338 | 0x0000.19C0 | Read/Write | Undefined | Float |
| ModelViewProjectionMatrix9 | 0x339 | 0x0000.19C8 | Read/Write | Undefined | Float |
| ModelViewProjectionMatrix10 | 0x33A | 0x0000.19D0 | Read/Write | Undefined | Float |
| ModelViewProjectionMatrix11 | 0x33B | 0x0000.19D8 | Read/Write | Undefined | Float |
| ModelViewProjectionMatrix12 | 0x33C | 0x0000.19E0 | Read/Write | Undefined | Float |
| ModelViewProjectionMatrix13 | 0x33D | 0x0000.19E8 | Read/Write | Undefined | Float |
| ModelViewProjectionMatrix14 | 0x33E | 0x0000.19F0 | Read/Write | Undefined | Float |
| ModelViewProjectionMatrix15 | 0x33F | 0x0000.19F8 | Read/Write | Undefined | Float |

These 16 registers hold the concatenated model view matrix and projection matrix used to multiply the input vertex coordinates (as a column vector). If the matrix is subscripted:

$$\begin{pmatrix} M_0 & M_4 & M_8 & M_{12} \\ M_1 & M_5 & M_9 & M_{13} \\ M_2 & M_6 & M_{10} & M_{14} \\ M_3 & M_7 & M_{11} & M_{15} \end{pmatrix}$$

then the numerical subscripts give the order the elements are stored in the matrix registers (i.e. $M_0$ is stored in **ModelViewProjectionMatrix0** for example) and these follow the column-major order convention. Note this is different from the convention C uses which follows the row-major order.

This matrix should always be enabled by the **TransformMode** register.

## NormalizeMode
## NormalizeModeAnd
## NormalizeModeOr

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| NormalizeMode | 0x2A3 | 0x0000.1518 | Read/Write | Undefined | Bitfield |
| NormalizeModeAnd | 0x554 | 0x0000.2AA0 | Write | Undefined | Bitfield |
| NormalizeModeOr | 0x555 | 0x0000.2AA8 | Write | Undefined | Bitfield |

This register has the following fields:

| Bit No. | Name | Description |
|---|---|---|
| 0 | NormalEnable | When set causes any normals to be normalized. |
| 1 | FaceNormalEnable | When set causes any face normals supplied by the user to be normalized. If the face normal is only being used for culling then it never needs to be normalized. |
| 2 | InvertAutoFaceNormal | When set causes the automatically generated face normal to have its direction reversed. |

Writing to the **NormalizeModeAnd** and **NormalizeModeOr** registers logically combine the new value with the existing values in the **NormalizeMode** register rather than replacing its contents.

These registers are also spelt as **NormaliseMode**, **NormaliseModeAnd** and **NormaliseModeOr**

## NormalMatrix0 … NormalMatrix8

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| NormalMatrix0 | 0x340 | 0x0000.1A00 | Read/Write | Undefined | Float |
| NormalMatrix1 | 0x341 | 0x0000.1A08 | Read/Write | Undefined | Float |
| NormalMatrix2 | 0x342 | 0x0000.1A10 | Read/Write | Undefined | Float |
| NormalMatrix3 | 0x343 | 0x0000.1A18 | Read/Write | Undefined | Float |
| NormalMatrix4 | 0x344 | 0x0000.1A20 | Read/Write | Undefined | Float |
| NormalMatrix5 | 0x345 | 0x0000.1A28 | Read/Write | Undefined | Float |
| NormalMatrix6 | 0x346 | 0x0000.1A30 | Read/Write | Undefined | Float |
| NormalMatrix7 | 0x347 | 0x0000.1A38 | Read/Write | Undefined | Float |
| NormalMatrix8 | 0x348 | 0x0000.1A40 | Read/Write | Undefined | Float |

These 9 registers hold the normal matrix used to transform the vertex normal or face normal into eye space (where lighting is done). If the matrix is subscripted:

$$\begin{pmatrix} N_0 & N_3 & N_6 \\ N_1 & N_4 & N_7 \\ N_2 & N_5 & N_8 \end{pmatrix}$$

then the numerical subscripts give the order the elements are stored in the matrix registers (i.e. $N_0$ is stored in **NormalMatrix0**, for example) and these follow the column-major order convention. Note this is different from the convention C uses which follows the row-major order.

The Normal matrix is usually the inverse transpose of the upper 3x3 part of the ModelView matrix.

This matrix is only used if enabled by the **TransformMode** register and can be avoided if lighting, sphere map in texture generation or face culling (using the face normal) are not required. There are separate enables for vertex normals and face normals.

## Nx
## Ny
## Nz

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| Nx | 0x302 | 0x0000.1810 | Read/Write/Trigger | Undefined | Float |
| Ny | 0x301 | 0x0000.1808 | Read/Write | Undefined | Float |
| Nz | 0x300 | 0x0000.1800 | Read/Write | Undefined | Float |

These registers hold the x, y and z normal components. The **Nx** register must be written last as this write will trigger the normal to be entered into Gamma. All the normal components should be written together and not interleaved with writes to the color (C*), face normal (N*), texture (T*) or vertex (V*) registers.

The normal is used for lighting and it is optionally transformed by the **NormalMatrix**. It must have a unit length for lighting to work properly. The normal can be optionally normalized, if necessary.

See also: **NormalMatrix**, **TransformMode**, **NormalizeMode**, **GeometryMode**, **LightingMode**.

## ObjectIDValue

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| ObjectIDValue | 0x2B5 | 0x0000.15A8 | Read/Write | Undefined | Integer |

This is for future use.

## PackedColor3
## PackedColor4

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| PackedColor3 | 0x313 | 0x0000.1898 | Write | Undefined | Bitfield |
| PackedColor4 | 0x314 | 0x0000.18A0 | Write | Undefined | Bitfield |

These registers provide a more compact way of getting colors into Gamma. The color information is packed as unsigned bytes with red in the lowest byte, then green, then blue and finally alpha in the most significant byte.

The unsigned byte color components are divided by 255.0 to convert them into floats for internal use.

In **PackedColor3** the alpha value is ignored and is always set to 1.0.

## PassThrough

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| PassThrough | 0x1FE | 0x0000.0FF0 | Output | Undefined | Integer |

This command passes straight through Gamma and GLINT without changing any state or causing any other action. The tag and data are written into the GLINT Host Out FIFO (assuming bits 14 and 15 in the GLINT **FilterMode** register are set). This would typically be used to inject marker tags into a feedback buffer.

## PointMode
## PointModeAnd
## PointModeOr

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| PointMode | 0x292 | 0x0000.1490 | Read/Write | Undefined | Bitfield |
| PointModeAnd | 0x55C | 0x0000.2AE0 | Write | Undefined | Bitfield |
| PointModeOr | 0x55D | 0x0000.2AE8 | Write | Undefined | Bitfield |

This register has the following fields:

| Bit No. | Name | Description |
|---|---|---|
| 0 | AntialiasEnable | This field, when set, enables antialiasing of points. This is qualified by the AntialiasEnable field in the **Begin** command. Note the Point Table in GLINT must be set up for the corresponding point size (held in **AAPointSize** register) and the selected antialiasing quality (next field). |
| 1 | AntialiasingQuality | This field defines the quality of antialiased points:<br>0 = 4x4<br>1 = 8x8<br>The Point Table in GLINT must be set up appropriately for the quality and the **AAPointSize.** |

Writing to the **PointModeAnd** and **PointModeOr** registers logically combine the new value with the existing values in the **PointMode** register rather than replacing its contents.

## PointSize

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| PointSize | 0x293 | 0x0000.1498 | Read/Write | Undefined | Integer |

This register holds the point size (diameter) used for aliased points. The size is held as an unsigned 8 bit value and a size of zero is treated as a size of one.

## PolygonOffsetBias
## PolygonOffsetFactor

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| PolygonOffsetBias | 0x37A | 0x0000.1BD0 | Read/Write | Undefined | Float |
| PolygonOffsetFactor | 0x379 | 0x0000.1BC8 | Read/Write | Undefined | Float |

These registers hold the polygon offset and bias values used during polygon offset processing (enabled in the **GeometryMode** register).

## PopName

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| PopName | 0x2BD | 0x0000.15E8 | Write | Undefined | Unused |

This command removes the name at the top of the name stack when in select mode (RenderMode field in **GeometryMode** register).

If the hit flag is set and select mode is enabled then the hit record is written to the GLINT's Host Out FIFO and the hit flag reset.

The hit record consists of (in order):

The count of the names (NameCount) on the stack (plus some error flags),

The minimum Z value as a normalized floating point number,

The maximum Z value as a normalized floating point number,

The name stack entries, oldest first (variable number [0…64] words.

Bits 14 and 15 in the **FilterMode** register in GLINT must be set to allow the **SelectRecord** tag and data values to be written in to the FIFO - all the select record data uses the same tag.

The NameCount value has the following fields:

| Bit | Name | Description |
|---|---|---|
| 0…6 | Count | This field holds the number of names on the name stack. |
| 7…28 | | Not used. |
| 29 | InvalidOperation | A **LoadName** operation was attempted on an empty name stack when this hit record was being collected. This is cleared for subsequent hit records (unless they manifest this error) however the stack may no longer be totally valid. |
| 30 | StackUnderflow | The name stack was popped while empty when this hit record was being collected. This is cleared for subsequent hit records (unless they manifest this error) however the stack may no longer be totally valid. |
| 31 | StackOverflow | The name stack was pushed while full when this hit record was being collected. This is cleared for subsequent hit records (unless they manifest this error) however the stack may no longer be totally valid. |

The **PopName** stack manipulation command is ignored when not in Select mode.

## PushName

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| PushName | 0x2BC | 0x0000.15E0 | Write | Undefined | Integer |

This command pushes the given name (a 32 bit integer) onto the name stack when in select mode (RenderMode field in **GeometryMode** register).

If the hit flag is set and select mode is enabled then the hit record is written to the GLINT's Host Out FIFO and the hit flag reset.

The hit record consists of (in order):

The count of the names (NameCount) on the stack (plus some error flags),

The minimum Z value as a normalized floating point number,

The maximum Z value as a normalized floating point number,

The name stack entries, oldest first (variable number [0…64] words.

Bits 14 and 15 in the **FilterMode** register in GLINT must be set to allow the **SelectRecord** tag and data values to be written in to the FIFO - all the select record data uses the same tag.

The NameCount value has the following fields:

| Bit | Name | Description |
|---|---|---|
| 0…6 | Count | This field holds the number of names on the name stack. |
| 7…28 | | Not used. |
| 29 | InvalidOperation | A **LoadName** operation was attempted on an empty name stack when this hit record was being collected. This is cleared for subsequent hit records (unless they manifest this error) however the stack may no longer be totally valid. |
| 30 | StackUnderflow | The name stack was popped while empty when this hit record was being collected. This is cleared for subsequent hit records (unless they manifest this error) however the stack may no longer be totally valid. |
| 31 | StackOverflow | The name stack was pushed while full when this hit record was being collected. This is cleared for subsequent hit records (unless they manifest this error) however the stack may no longer be totally valid. |

The **PushName** stack manipulation command is ignored when not in Select mode.

## RasterPosXIncrement RasterPosYIncrement

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| RasterPosXIncrement | 0x37D | 0x0000.1BE8 | Read/Write | Undefined | Float |
| RasterPosYIncrement | 0x37E | 0x0000.1BF0 | Read/Write | Undefined | Float |

These registers hold the value to add to the raster pointer's coordinate after it is rendered (only if it passed the chip test). The increment is measured in pixels.

## RasterPosXOffset RasterPosYOffset

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| RasterPosXOffset | 0x39D | 0x0000.1CE8 | Read/Write | Undefined | Float |
| RasterPosYOffset | 0x39E | 0x0000.1CF0 | Read/Write | Undefined | Float |

These registers hold an offset added to the raster pointer's coordinate prior to rendering. The raster pointer is not updated. The offset is measured in pixels.

# Rectangle2DControl

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| Rectangle2DControl | 0x29E | 0x0000.14F0 | Read/Write | Undefined | Boolean |

This register has a single bit to specify whether the **DrawRectangle2D** command is to use GID window clipping (1) or not (0). The side effects of this are:

| Window clipping | Side effect |
|---|---|
| 0 | Use spans and no local buffer reads (GIDs are not used/needed). |
| 1 | Don't use spans and the local buffer is read. |

## Rectangle2DMode

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| Rectangle2DMode | 0x29D | 0x0000.14E8 | Read/Write | Undefined | Bitfield |

This register has the following fields and this is used by the **DrawRectangle2D** command to rasterize rectangles.

| Bit No. | Name | Description |
|---|---|---|
| 0…11 | Width | Width of the rectangle. Twelve bit field with range 0…4095 |
| 12…23 | Height | Height of the rectangle. Twelve bit field with range 0…4095 |
| 24 | AreaStippleEnable | Passed to rasterizer in the **Render** command. |
| 25 | SyncOnBitMask | Passed to rasterizer in the **Render** command. |
| 26 | SyncOnHostData | Passed to rasterizer in the **Render** command. |
| 27 | TextureEnable | Passed to rasterizer in the **Render** command. |
| 28 | FogEnable | Passed to rasterizer in the **Render** command. |
| 29 | SpanOperation | Passed to rasterizer in the **Render** command. |
| 30 | HorizontalDirection | Sets the horizontal rasterization direction.<br>0 = Left to Right<br>1 = Right to Left |
| 31 | VerticalDirection | Sets the vertical rasterization direction.<br>0 = Increasing Y<br>1 = Decreasing Y |

See also **DrawRectangle2D**, **Rectangle2DControl**.

## RectangleHeight

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| RectangleHeight | 0x29C | 0x0000.14E0 | Read/Write | Undefined | Float |

This register holds the height of the rectangle when **GeomRectangle** command is used. It should be a positive number. The origin (color, texture, etc.) of the rectangle is set by the raster position and the opposite vertical edge is set by adding on the height value.

See also: **GeomRectangle**, **RectangleWidth**, **RP***

## RectangleMode

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| RectangleMode | 0x29A | 0x0000.14D0 | Read/Write | Undefined | Bitfield |

This register holds the rasterizer mode to use for the **GeomRectangle** command. This mode is identical to the data provided with the GLINT **Render** command.

## RectangleWidth

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| RectangleWidth | 0x29B | 0x0000.14D8 | Read/Write | Undefined | Float |

This register holds the width of the rectangle when **GeomRectangle** command is used. It should be a positive number. The origin (color, texture, etc.) of the rectangle is set by the raster position and the opposite horizontal edge is set by adding on the width value.

See also: **GeomRectangle**, **RectangleHeight**, **RP***

## RepeatLine

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| RepeatLine | 0x265 | 0x0000.1328 | Write | Undefined | Unused |

This command causes the previous line drawn with **DrawLine01** or **DrawLine10** to be repeated. It would be normal for some mode or other state information to have been changed before the line is repeated. An example of this is to use scissor clipping with the line being repeated for each clip rectangle.

This is a legacy command and should only be used if Gamma is just being used as a faster GLINT Delta and none of the additional Gamma functionality is being used.

## RepeatTriangle

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| RepeatTriangle | 0x262 | 0x0000.1310 | Write | Undefined | Unused |

This command causes the previous triangle drawn with **DrawTriangle** to be repeated. It would be normal for some mode or other state information to have been changed before the triangle is repeated. An example of this is to use scissor clipping with the triangle being repeated for each clip rectangle.

This is a legacy command and should only be used if Gamma is just being used as a faster GLINT Delta and none of the additional Gamma functionality is being used.

## RestoreCurrent

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| RestoreCurrent | 0x2BA | 0x0000.15D0 | Write | Undefined | Unused |

This command causes the current color, current normal and current texture values to be restored from internal registers. This is useful for preserving the current values across evaluator calls in OpenGL.

See also: **SaveCurrent**.

## RPw
## RPx2
## RPx3
## RPx4
## RPy
## RPz

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| RPw | 0x319 | 0x0000.18C8 | Write | Undefined | Float |
| RPx2 | 0x31C | 0x0000.18E0 | Write/Trigger | Undefined | Float |
| RPx3 | 0x31D | 0x0000.18E8 | Write/Trigger | Undefined | Float |
| RPx4 | 0x31E | 0x0000.18F0 | Write/Trigger | Undefined | Float |
| RPy | 0x31B | 0x0000.18D8 | Write | Undefined | Float |
| RPz | 0x31A | 0x0000.18D0 | Write | Undefined | Float |

These registers hold the x, y and z raster position components. The **RPx2, RPx3** or **RPx4** registers must be written to last as the write will trigger the raster position to be entered into Gamma. All the raster position components should be written together and not interleaved with writes to the color (C*), face normal (FN*), normal (N*) or texture (T*) registers.

Writing to **RPx2** will ignore any supplied values for **RPz** and **RPw** and set them to 0.0 and 1.0 respectively.

Writing to **RPx3** will ignore any supplied values for **RPw** and set it to 1.0.

A raster position is optionally transformed by the **ModelViewMatrix** and/or the **ModelViewProjectionMatrix**.

Once the raster position has been entered into Gamma it is transformed and lit like a regular vertex and causes the resultant color, texture, fog and coordinate to be saved for later use by the **GeomRectangle** command. No primitive is rendered, however, if Gamma is in Select mode (set in RenderMode field in the **GeometryMode** register) a select hit may occur.

The raster position should not be sent between the **Begin** and **End** commands as it will corrupt vertex data.

See also: **TransformMode**, **GeomRectangle**.

## SaveCurrent

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| SaveCurrent | 0x2B9 | 0x0000.15C8 | Write | Undefined | Unused |

This command causes the current color, current normal and current texture values to be saved in internal registers. Only one level of saving can be done. This is useful for preserving the current values across evaluator calls in OpenGL.

See also: **RestoreCurrent**.

## SceneAmbientColorBlue SceneAmbientColorGreen SceneAmbientColorRed

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| SceneAmbientColorBlue | 0x502 | 0x0000.2810 | Read/Write | Undefined | Float |
| SceneAmbientColorGreen | 0x501 | 0x0000.2808 | Read/Write | Undefined | Float |
| SceneAmbientColorRed | 0x500 | 0x0000.2800 | Read/Write | Undefined | Float |

These registers hold the scene ambient color values used in lighting the equations. The normal range of values is 0.0 … 1.0, however any value can be used.

## SelectRecord

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| SelectRecord | 0x1FD | 0x0000.0FE8 | Output | Undefined | Variable |

The select record is written into GLINT's Host Out FIFO when select mode is enabled (RenderMode field in **GeometryMode**) and a name stack operation (**LoadName**, **PushName**, **PopName**) occurs after a hit has been found, or on a **SelectResult** when a hit record has been found.

All the words in a hit record are given the **SelectRecord** tag and the select record consists of (in order):

The count of the names (NameCount) on the stack (plus some error flags),
The minimum Z value as a normalized floating point number,
The maximum Z value as a normalized floating point number,
The name stack entries, oldest first (variable number [0…64] words.

Bits 14 and 15 in the **FilterMode** register in GLINT must be set to allow the **SelectRecord** tag and data values to be written in to the FIFO - all the select record data uses the same tag.

The NameCount value has the following fields:

| Bit No. | Name | Description |
|---|---|---|
| 0…6 | Count | This field holds the number of names on the name stack. |
| 7…28 | | Not used. |
| 29 | InvalidOperation | A **LoadName** operation was attempted on an empty name stack when this hit record was being collected. This is cleared for subsequent hit records (unless they manifest this error) however the stack may no longer be totally valid. |
| 30 | StackUnderflow | The name stack was popped while empty when this hit record was being collected. This is cleared for subsequent hit records (unless they manifest this error) however the stack may no longer be totally valid. |
| 31 | StackOverflow | The name stack was pushed while full when this hit record was being collected. This is cleared for subsequent hit records (unless they manifest this error) however the stack may no longer be totally valid. |

## SelectResult

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| SelectResult | 0x2B0 | 0x0000.1580 | Write | Undefined | Unused |

The **SelectResult** command flushes out the select record if a hit had occurred since the last time the select record was written out. This avoids doing a name stack manipulation to get the select record out which may result in a stack under or over flow.

## TexGen0…TexGen15

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| TexGen0 | 0x360 | 0x0000.1B00 | Read/Write | Undefined | Float |
| TexGen1 | 0x361 | 0x0000.1B08 | Read/Write | Undefined | Float |
| TexGen2 | 0x362 | 0x0000.1B10 | Read/Write | Undefined | Float |
| TexGen3 | 0x363 | 0x0000.1B18 | Read/Write | Undefined | Float |
| TexGen4 | 0x364 | 0x0000.1B20 | Read/Write | Undefined | Float |
| TexGen5 | 0x365 | 0x0000.1B28 | Read/Write | Undefined | Float |
| TexGen6 | 0x366 | 0x0000.1B30 | Read/Write | Undefined | Float |
| TexGen7 | 0x367 | 0x0000.1B38 | Read/Write | Undefined | Float |
| TexGen8 | 0x368 | 0x0000.1B40 | Read/Write | Undefined | Float |
| TexGen9 | 0x369 | 0x0000.1B48 | Read/Write | Undefined | Float |
| TexGen10 | 0x36A | 0x0000.1B50 | Read/Write | Undefined | Float |
| TexGen11 | 0x36B | 0x0000.1B58 | Read/Write | Undefined | Float |
| TexGen12 | 0x36C | 0x0000.1B60 | Read/Write | Undefined | Float |
| TexGen13 | 0x36D | 0x0000.1B68 | Read/Write | Undefined | Float |
| TexGen14 | 0x36E | 0x0000.1B70 | Read/Write | Undefined | Float |
| TexGen15 | 0x36F | 0x0000.1B78 | Read/Write | Undefined | Float |

These registers hold the texture generation coefficients to be used for object and eye linear texture generation. Each texture component has 4 registers assigned to hold the coefficients for object linear or eye linear operation for that component. The registers are assigned as follows:

| Texture component | Registers |
|---|---|
| S | **TexGen0**, **TexGen1**, **TexGen2**, **TexGen3** |
| T | **TexGen4**, **TexGen5**, **TexGen6**, **TexGen7** |
| R | **TexGen8**, **TexGen9**, **TexGen10**, **TexGen11** |
| Q | **TexGen12**, **TexGen13**, **TexGen14**, **TexGen15** |

The texture generation modes are controlled by the **TransformMode** register.

## TextureMatrix0…TextureMatrix15

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| TextureMatrix0 | 0x350 | 0x0000.1A80 | Read/Write | Undefined | Float |
| TextureMatrix1 | 0x351 | 0x0000.1A88 | Read/Write | Undefined | Float |
| TextureMatrix2 | 0x352 | 0x0000.1A90 | Read/Write | Undefined | Float |
| TextureMatrix3 | 0x353 | 0x0000.1A98 | Read/Write | Undefined | Float |
| TextureMatrix4 | 0x354 | 0x0000.1AA0 | Read/Write | Undefined | Float |
| TextureMatrix5 | 0x355 | 0x0000.1AA8 | Read/Write | Undefined | Float |
| TextureMatrix6 | 0x356 | 0x0000.1AB0 | Read/Write | Undefined | Float |
| TextureMatrix7 | 0x357 | 0x0000.1AB8 | Read/Write | Undefined | Float |
| TextureMatrix8 | 0x358 | 0x0000.1AC0 | Read/Write | Undefined | Float |
| TextureMatrix9 | 0x359 | 0x0000.1AC8 | Read/Write | Undefined | Float |
| TextureMatrix10 | 0x35A | 0x0000.1AD0 | Read/Write | Undefined | Float |
| TextureMatrix11 | 0x35B | 0x0000.1AD8 | Read/Write | Undefined | Float |
| TextureMatrix12 | 0x35C | 0x0000.1AE0 | Read/Write | Undefined | Float |
| TextureMatrix13 | 0x35D | 0x0000.1AE8 | Read/Write | Undefined | Float |
| TextureMatrix14 | 0x35E | 0x0000.1AF0 | Read/Write | Undefined | Float |
| TextureMatrix15 | 0x35F | 0x0000.1AF8 | Read/Write | Undefined | Float |

These 16 registers hold the texture matrix used to multiply the input vertex coordinates (as a column vector). If the matrix is subscripted:

$$\begin{pmatrix} M_0 & M_4 & M_8 & M_{12} \\ M_1 & M_5 & M_9 & M_{13} \\ M_2 & M_6 & M_{10} & M_{14} \\ M_3 & M_7 & M_{11} & M_{15} \end{pmatrix}$$

then the numerical subscripts give the order the elements are stored in the matrix registers (i.e. $M_0$ is stored in **TextureMatrix0**, for example) and these follow the column-major order convention. Note this is different from the convention C uses which follows the row-major order.

This matrix is only used if enabled by the **TransformMode** register and can be avoided if the texture matrix is a unit matrix, or the API doesn't support texture matrices.

## TransformCurrent

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| TransformCurrent | 0x2B7 | 0x0000.15B8 | Write | Undefined | Bitfield |

This command is used to refresh the current values when they have been restored using the **ContextRestore** command with the CurrentState bit set. It has the following fields:

| Bit No. | Name | Description |
|---|---|---|
| 0 | Normal | When set refreshes the vertex normal. |
| 1 | FaceNormal | When set refreshes the face normal. |
| 2 | Texture | When set refreshes the texture coordinates. |
| 3 | Color | When set refreshes the color. |

## TransformMode
## TransformModeAnd
## TransformModeOr

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| TransformMode | 0x2A1 | 0x0000.1508 | Read/Write | Undefined | Bitfield |
| TransformModeAnd | 0x550 | 0x0000.2A80 | Write | Undefined | Bitfield |
| TransformModeOr | 0x551 | 0x0000.2A88 | Write | Undefined | Bitfield |

This register has the following fields:

| Bit No. | Name | Description |
|---|---|---|
| 0 | UseModelViewMatrix | When set causes the incoming vertex to be multiplied by the ModelView matrix. This is only necessary if the vertex in eye space is needed for subsequent processing. A slight gain in performance will be seen when this transformation is disabled. The eye space vertex is used for EyeLinear TexGen, user clipping planes, fog, lighting, or auto generation of the face normal.. |
| 1 | UseModelViewProjectionMatrix | When set causes the incoming vertex to be multiplied by the ModelViewProjection matrix to calculate coordinates in clip space. This bit should normally be set. |
| 2 | TransformNormal | When set causes any incoming vertex normal to be multiplied by the Normal matrix. This only needs to be set if lighting is used or TexGen SphereMap is selected. |
| 3 | TransformFaceNormal | When set causes any incoming face normal to be multiplied by the Normal matrix. This only needs to be set if face normal lighting is used and/or if face normal backface test is enabled. |
| 4 | TransformTexture | When set causes the incoming texture or the texture generated from the TexGen operation to be multiplied by the Texture matrix. Frequently the texture matrix will be a unit matrix so the transformation can be preferably avoided. |
| 5, 6 | TexGenModeS | This field controls the automatic generation of texture coordinates for the S texture component from the vertex or normal information. The TexGen operations are:<br>0 = None.<br>1 = ObjectLinear.<br>2 = EyeLinear.<br>3 = SphereMap. |
| 7, 8 | TexGenModeT | This field controls the automatic generation of texture coordinates for the T texture component from the vertex or normal information. The TexGen operations are:<br>0 = None.<br>1 = ObjectLinear.<br>2 = EyeLinear.<br>3 = SphereMap. |

| | | |
|---|---|---|
| 9, 10 | TexGenModeR | This field controls the automatic generation of texture coordinates for the R texture component from the vertex or normal information. The TexGen operations are:<br>0 = None.<br>1 = ObjectLinear.<br>2 = EyeLinear.<br>3 = None (SphereMap is illegal). |
| 9, 10 | TexGenModeR | This field controls the automatic generation of texture coordinates for the R texture component from the vertex or normal information. The TexGen operations are:<br>0 = None.<br>1 = ObjectLinear.<br>2 = EyeLinear.<br>3 = None (SphereMap is illegal). |
| 11, 12 | TexGenModeQ | This field controls the automatic generation of texture coordinates for the Q texture component from the vertex or normal information. The TexGen operations are:<br>0 = None.<br>1 = ObjectLinear.<br>2 = EyeLinear.<br>3 = None (SphereMap is illegal). |
| 13 | TexGenS | When this bit is set the S component of the texture coordinate is generated automatically, otherwise it is taken from the current texture S value. This only has an effect when the TexGen operation is ObjectLinear, EyeLinear or SphereMap. |
| 14 | TexGenT | When this bit is set the T component of the texture coordinate is generated automatically, otherwise it is taken from the current texture T value. This only has an effect when the TexGen operation is ObjectLinear, EyeLinear or SphereMap. |
| 15 | TexGenR | When this bit is set the R component of the texture coordinate is generated automatically, otherwise it is taken from the current texture R value. This only has an effect when the TexGen operation is ObjectLinear or EyeLinear. |
| 16 | TexGenQ | When this bit is set the Q component of the texture coordinate is generated automatically, otherwise it is taken from the current texture Q value. This only has an effect when the TexGen operation is ObjectLinear or EyeLinear. |

Writing to the **TransformModeAnd** and **TransformModeOr** registers logically combine the new value with the existing values in the **TransformMode** register rather than replacing its contents.

## TriangleClipAreaThreshold

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| TriangleClipAreaThreshold | 0x37C | 0x0000.1BE0 | Read/Write | Undefined | Float |

This register holds the area below which triangles should not be clipped as it is faster to use screen or window clipping during rasterization . This is enabled in the **GeometryMode** register and automatically disabled if the triangle crossed the near, far or user clipping planes.

The area tested against is twice the desired area in pixels. For example, triangles with a true area of 5 pixels or less should not be clipped then this register will have the value of 10 written into it.

It is best to disable this if any of the following conditions arise:

- The viewport is smaller than the window and user scissoring isn't enabled. The rasterization level clipping cannot be relied on to do the clipping.
- Select or Feedback modes are enabled. Small triangles which would otherwise have been clipped out will be included in the select or feedback data.

## TriangleMode
## TriangleModeAnd
## TriangleModeOr

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| TriangleMode | 0x299 | 0x0000.14C8 | Read/Write | Undefined | Bitfield |
| TriangleModeAnd | 0x560 | 0x0000.2B00 | Write | Undefined | Bitfield |
| TriangleModeOr | 0x561 | 0x0000.2B08 | Write | Undefined | Bitfield |

The **TriangleMode** register has the following fields:

| Bit No. | Name | Description |
|---|---|---|
| 0 | AntialiasEnable | This field, when set, enables antialiasing of triangles. This is qualified by the AntialiasEnable field in the **Begin** command. |
| 1 | AntialiasingQuality | This field defines the quality of antialiased triangles:<br>0 = 4x4<br>1 = 8x8 |
| 2 | UseTrianglePacketInterface | This field, when set, causes the triangle set up to use the Triangle Packet Interface to send the triangle parameters to GLINT. This is only supported in GLINT MX and provides a higher triangle throughput. |

Writing to the **TriangleModeAnd** and **TriangleModeOr** registers logically combine the new value with the existing values in the **TriangleMode** register rather than replacing its contents.

## Tq4
## Tr4
## Ts1
## Ts2
## Ts4
## Tt2
## Tt4

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| Ts10x31F | 0x0000.18F8 | Read/Write/Trigger | Undefined | Float | |
| Ts20x309 | 0x0000.1848 | Read/Write/Trigger | Undefined | Float | |
| Tt20x308 | 0x0000.1840 | Read/Write | Undefined | Float | |
| Ts40x318 | 0x0000.18C0 | Read/Write/Trigger | Undefined | Float | |
| Tt40x317 | 0x0000.18B8 | Read/Write | Undefined | Float | |
| Tr4 | 0x316 | 0x0000.18B0 | Read/Write | Undefined | Float |
| Tq4 | 0x315 | 0x0000.18A8 | Read/Write | Undefined | Float |

These registers hold the s, t, r and q texture components. The **Ts1, Ts2** or **Ts4** registers must be written last as they write will trigger the texture to be entered into Gamma. All the texture components should be written together and not interleaved with writes to the color (C*), face normal (FN*), normal (N*) or vertex (V*) registers.

Writing to **Ts1** will ignore any supplied values for the t, r and q components and set them to 0.0, 0.0 and 1.0 respectively.

Writing to **Ts2** will ignore any supplied values for the r and q components and set them to 0.0 and 1.0 respectively.

The t component can be written using either of the registers (**Tt2** or **Tt4**) however the tags are grouped with the **Ts2** or **Ts4** registers so the natural one to use depends on what other texture components are being sent as well.

A texture is optionally transformed by the **TextureMatrix**.

See also: **TransformMode**.

## UserClip0X…UserClip0W
## UserClip1X…UserClip1W
## UserClip2X…UserClip2W
## UserClip3X…UserClip3W
## UserClip4X…UserClip4W
## UserClip5X…UserClip5W

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| UserClip0X | 0x380 | 0x0000.1C00 | Read/Write | Undefined | Float |
| UserClip0Y | 0x381 | 0x0000.1C08 | Read/Write | Undefined | Float |
| UserClip0Z | 0x382 | 0x0000.1C10 | Read/Write | Undefined | Float |
| UserClip0W | 0x383 | 0x0000.1C18 | Read/Write | Undefined | Float |
| UserClip1X | 0x384 | 0x0000.1C20 | Read/Write | Undefined | Float |
| UserClip1Y | 0x385 | 0x0000.1C28 | Read/Write | Undefined | Float |
| UserClip1Z | 0x386 | 0x0000.1C30 | Read/Write | Undefined | Float |
| UserClip1W | 0x387 | 0x0000.1C38 | Read/Write | Undefined | Float |
| UserClip2X | 0x388 | 0x0000.1C40 | Read/Write | Undefined | Float |
| UserClip2Y | 0x389 | 0x0000.1C48 | Read/Write | Undefined | Float |
| UserClip2Z | 0x38A | 0x0000.1C50 | Read/Write | Undefined | Float |
| UserClip2W | 0x38B | 0x0000.1C58 | Read/Write | Undefined | Float |
| UserClip3X | 0x38C | 0x0000.1C60 | Read/Write | Undefined | Float |
| UserClip3Y | 0x38D | 0x0000.1C68 | Read/Write | Undefined | Float |
| UserClip3Z | 0x38E | 0x0000.1C70 | Read/Write | Undefined | Float |
| UserClip3W | 0x38F | 0x0000.1C78 | Read/Write | Undefined | Float |
| UserClip4X | 0x390 | 0x0000.1C80 | Read/Write | Undefined | Float |
| UserClip4Y | 0x391 | 0x0000.1C88 | Read/Write | Undefined | Float |
| UserClip4Z | 0x392 | 0x0000.1C90 | Read/Write | Undefined | Float |
| UserClip4W | 0x393 | 0x0000.1C98 | Read/Write | Undefined | Float |
| UserClip5X | 0x394 | 0x0000.1CA0 | Read/Write | Undefined | Float |
| UserClip5Y | 0x395 | 0x0000.1CA8 | Read/Write | Undefined | Float |
| UserClip5Z | 0x396 | 0x0000.1CB0 | Read/Write | Undefined | Float |
| UserClip5W | 0x397 | 0x0000.1CB8 | Read/Write | Undefined | Float |

These registers hold the plane equation coefficients for the 6 user clipping planes. The planes are enabled by the **GeometryMode** register.

## V0FixedA
## V0FixedB
## V0FixedF
## V0FixedG
## V0FixedKd
## V0FixedKs
## V0FixedQ
## V0FixedR
## V0FixedS
## V0FixedT
## V0FixedX
## V0FixedY
## V0FixedZ

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| V0FixedA | 0x208 | 0x0000.1040 | Read/Write | Undefined | Fixed Point |
| V0FixedB | 0x207 | 0x0000.1038 | Read/Write | Undefined | Fixed Point |
| V0FixedF | 0x209 | 0x0000.1048 | Read/Write | Undefined | Fixed Point |
| V0FixedG | 0x206 | 0x0000.1030 | Read/Write | Undefined | Fixed Point |
| V0FixedKd | 0x204 | 0x0000.1020 | Read/Write | Undefined | Fixed Point |
| V0FixedKs | 0x203 | 0x0000.1018 | Read/Write | Undefined | Fixed Point |
| V0FixedQ | 0x202 | 0x0000.1010 | Read/Write | Undefined | Fixed Point |
| V0FixedR | 0x205 | 0x0000.1028 | Read/Write | Undefined | Fixed Point |
| V0FixedS | 0x200 | 0x0000.1000 | Read/Write | Undefined | Fixed Point |
| V0FixedT | 0x201 | 0x0000.1008 | Read/Write | Undefined | Fixed Point |
| V0FixedX | 0x20A | 0x0000.1050 | Read/Write | Undefined | Fixed Point |
| V0FixedY | 0x20B | 0x0000.1058 | Read/Write | Undefined | Fixed Point |
| V0FixedZ | 0x20C | 0x0000.1060 | Read/Write | Undefined | Fixed Point |

The input format for each parameter is shown below. The *us* is unsigned and *s* is 2's complement. The clamping range, if clamping is enabled, is also shown.

| Category | Parameter | Fixed Point Format | Clamping Range |
|---|---|---|---|
| Texture | s | 2.30 s[1] | -1.0…1.0[2] |
| | t | 2.30 s | -1.0…1.0 |
| | q | 2.30 s | -1.0…1.0 |
| | Ks | 2.22 us | 0.0…2.0 |
| | Kd | 2.22 us | 0.0…2.0 |
| Color | red | 1.30 us | 0.0…1.0 |
| | green | 1.30 us | 0.0…1.0 |
| | blue | 1.30 us | 0.0…1.0 |
| | alpha | 1.30 us | 0.0…1.0 |
| Fog | f | 10.22 s | -512.0…512.0 |
| Coordinate | x | 16.16 s | -32K…+32K[3][4] |
| | y | 16.16 s | -32K…+32K |
| | z | 1.30 us | 0.0…1.0 |

These values read back as floating point and share the same storage as the **V0Float*** registers.

These are legacy commands and should only be used if Gamma is just being used as a faster GLINT Delta and none of the additional Gamma functionality is being used.

[1]This is the range when Normalize is not used. When Normalize is enabled the fixed point format can be anything, providing it is the same for the s, t and q parameters. The numbers will be interpreted as if they had 2.30 format for the purpose of conversion to floating point. If the fixed point format (2.30) is different from what the user had in mind then the input values are just pre-scaled by a fixed amount (i.e. the difference in binary point positions) prior to conversion.

[2]This is the range when Normalize is not used. When Normalize is enabled the range is extended to $2^{\pm 32}$ approximately. This also applies to the t and q values as well.

[3]The normal range here is limited by the size of the screen.

[4]K = 1024.

## V0FloatA
## V0FloatB
## V0FloatF
## V0FloatG
## V0FloatKd
## V0FloatKs
## V0FloatQ
## V0FloatR
## V0FloatS
## V0FloatT
## V0FloatX
## V0FloatY
## V0FloatZ

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| V0FloatA | 0x238 | 0x0000.11C0 | Read/Write | Undefined | Float |
| V0FloatB | 0x237 | 0x0000.11B8 | Read/Write | Undefined | Float |
| V0FloatF | 0x239 | 0x0000.11C8 | Read/Write | Undefined | Float |
| V0FloatG | 0x236 | 0x0000.11B0 | Read/Write | Undefined | Float |
| V0FloatKd | 0x234 | 0x0000.11A0 | Read/Write | Undefined | Float |
| V0FloatKs | 0x233 | 0x0000.1198 | Read/Write | Undefined | Float |
| V0FloatQ | 0x232 | 0x0000.1190 | Read/Write | Undefined | Float |
| V0FloatR | 0x235 | 0x0000.11A8 | Read/Write | Undefined | Float |
| V0FloatS | 0x230 | 0x0000.1180 | Read/Write | Undefined | Float |
| V0FloatT | 0x231 | 0x0000.1188 | Read/Write | Undefined | Float |
| V0FloatX | 0x23A | 0x0000.11D0 | Read/Write | Undefined | Float |
| V0FloatY | 0x23B | 0x0000.11D8 | Read/Write | Undefined | Float |
| V0FloatZ | 0x23C | 0x0000.11E0 | Read/Write | Undefined | Float |

The clamping range, if enabled, for each parameter is shown below.

| Category | Parameter | Clamping Range |
|---|---|---|
| Texture | s | -1.0…1.0[1] |
| | t | -1.0…1.0 |
| | q | -1.0…1.0 |
| | Ks | 0.0…2.0 |
| | Kd | 0.0…2.0 |
| Color | red | 0.0…1.0 |
| | green | 0.0…1.0 |
| | blue | 0.0…1.0 |
| | alpha | 0.0…1.0 |
| Fog | f | -512.0…512.0 |
| Coordinate | x | -32K…+32K[2][3] |
| | y | -32K…+32K |
| | z | 0.0…1.0 |

These registers share the same storage as the **V0Fixed*** registers.

These are legacy commands and should only be used if Gamma is just being used as a faster GLINT Delta and none of the additional Gamma functionality is being used.

[1]This is the range when Normalize is not used. When Normalize is enabled the range is extended to $2^{\pm 32}$ approximately. This also applies to the t and q values as well.

[2]The normal range here is limited by the size of the screen.

[3]K = 1024.

## V1FixedA V1FixedB V1FixedF V1FixedG V1FixedKd V1FixedKs V1FixedQ V1FixedR V1FixedS V1FixedT V1FixedX V1FixedY V1FixedZ

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| V1FixedA | 0x218 | 0x0000.10C0 | Read/Write | Undefined | Fixed Point |
| V1FixedB | 0x217 | 0x0000.10B8 | Read/Write | Undefined | Fixed Point |
| V1FixedF | 0x219 | 0x0000.10C8 | Read/Write | Undefined | Fixed Point |
| V1FixedG | 0x216 | 0x0000.10B0 | Read/Write | Undefined | Fixed Point |
| V1FixedKd | 0x214 | 0x0000.10A0 | Read/Write | Undefined | Fixed Point |
| V1FixedKs | 0x213 | 0x0000.1098 | Read/Write | Undefined | Fixed Point |
| V1FixedQ | 0x212 | 0x0000.1090 | Read/Write | Undefined | Fixed Point |
| V1FixedR | 0x215 | 0x0000.10A8 | Read/Write | Undefined | Fixed Point |
| V1FixedS | 0x210 | 0x0000.1080 | Read/Write | Undefined | Fixed Point |
| V1FixedT | 0x211 | 0x0000.1088 | Read/Write | Undefined | Fixed Point |
| V1FixedX | 0x21A | 0x0000.10D0 | Read/Write | Undefined | Fixed Point |
| V1FixedY | 0x21B | 0x0000.10D8 | Read/Write | Undefined | Fixed Point |
| V1FixedZ | 0x21C | 0x0000.10E0 | Read/Write | Undefined | Fixed Point |

The input format for each parameter is shown below. The *us* is unsigned and *s* is 2's complement. The clamping range, if clamping is enabled, is also shown.

| Category | Parameter | Fixed Point Format | Clamping Range |
|---|---|---|---|
| Texture | s | 2.30 s[1] | -1.0…1.0[2] |
| | t | 2.30 s | -1.0…1.0 |
| | q | 2.30 s | -1.0…1.0 |
| | Ks | 2.22 us | 0.0…2.0 |
| | Kd | 2.22 us | 0.0…2.0 |
| Color | red | 1.30 us | 0.0…1.0 |
| | green | 1.30 us | 0.0…1.0 |
| | blue | 1.30 us | 0.0…1.0 |
| | alpha | 1.30 us | 0.0…1.0 |
| Fog | f | 10.22 s | -512.0…512.0 |
| Coordinate | x | 16.16 s | -32K…+32K[3][4] |
| | y | 16.16 s | -32K…+32K |
| | z | 1.30 us | 0.0…1.0 |

These values read back as floating point and share the same storage as the **V1Float*** registers.

These are legacy commands and should only be used if Gamma is just being used as a faster GLINT Delta and none of the additional Gamma functionality is being used.

[1]This is the range when Normalize is not used. When Normalize is enabled the fixed point format can be anything, providing it is the same for the s, t and q parameters. The numbers will be interpreted as if they had 2.30 format for the purpose of conversion to floating point. If the fixed point format (2.30) is different from what the user had in mind then the input values are just pre-scaled by a fixed amount (i.e. the difference in binary point positions) prior to conversion.

[2]This is the range when Normalize is not used. When Normalize is enabled the range is extended to $2^{\pm 32}$ approximately. This also applies to the t and q values as well.

[3]The normal range here is limited by the size of the screen.

[4]K = 1024.

## V1FloatA
## V1FloatB
## V1FloatF
## V1FloatG
## V1FloatKd
## V1FloatKs
## V1FloatQ
## V1FloatR
## V1FloatS
## V1FloatT
## V1FloatX
## V1FloatY
## V1FloatZ

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| V1FloatA | 0x248 | 0x0000.1240 | Read/Write | Undefined | Float |
| V1FloatB | 0x247 | 0x0000.1238 | Read/Write | Undefined | Float |
| V1FloatF | 0x249 | 0x0000.1248 | Read/Write | Undefined | Float |
| V1FloatG | 0x246 | 0x0000.1230 | Read/Write | Undefined | Float |
| V1FloatKd | 0x244 | 0x0000.1220 | Read/Write | Undefined | Float |
| V1FloatKs | 0x243 | 0x0000.1218 | Read/Write | Undefined | Float |
| V1FloatQ | 0x242 | 0x0000.1210 | Read/Write | Undefined | Float |
| V1FloatR | 0x245 | 0x0000.1228 | Read/Write | Undefined | Float |
| V1FloatS | 0x240 | 0x0000.1200 | Read/Write | Undefined | Float |
| V1FloatT | 0x241 | 0x0000.1208 | Read/Write | Undefined | Float |
| V1FloatX | 0x24A | 0x0000.1250 | Read/Write | Undefined | Float |
| V1FloatY | 0x24B | 0x0000.1258 | Read/Write | Undefined | Float |
| V1FloatZ | 0x24C | 0x0000.1260 | Read/Write | Undefined | Float |

The clamping range, if enabled, for each parameter is shown below.

| Category | Parameter | Clamping Range |
|---|---|---|
| Texture | s | -1.0…1.0[1] |
| | t | -1.0…1.0 |
| | q | -1.0…1.0 |
| | Ks | 0.0…2.0 |
| | Kd | 0.0…2.0 |
| Color | red | 0.0…1.0 |
| | green | 0.0…1.0 |
| | blue | 0.0…1.0 |
| | alpha | 0.0…1.0 |
| Fog | f | -512.0…512.0 |
| Coordinate | x | -32K…+32K[2][3] |
| | y | -32K…+32K |
| | z | 0.0…1.0 |

These registers share the same storage as the **V1Fixed*** registers.

These are legacy commands and should only be used if Gamma is just being used as a faster GLINT Delta and none of the additional Gamma functionality is being used.

[1]This is the range when Normalize is not used. When Normalize is enabled the range is extended to $2^{\pm 32}$ approximately. This also applies to the t and q values as well.

[2]The normal range here is limited by the size of the screen.

[3]K = 1024.

## V2FixedA
## V2FixedB
## V2FixedF
## V2FixedG
## V2FixedKd
## V2FixedKs
## V2FixedQ
## V2FixedR
## V2FixedS
## V2FixedT
## V2FixedX
## V2FixedY
## V2FixedZ

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| V2FixedA | 0x228 | 0x0000.1140 | Read/Write | Undefined | Fixed Point |
| V2FixedB | 0x227 | 0x0000.1138 | Read/Write | Undefined | Fixed Point |
| V2FixedF | 0x229 | 0x0000.1148 | Read/Write | Undefined | Fixed Point |
| V2FixedG | 0x226 | 0x0000.1130 | Read/Write | Undefined | Fixed Point |
| V2FixedKd | 0x224 | 0x0000.1120 | Read/Write | Undefined | Fixed Point |
| V2FixedKs | 0x223 | 0x0000.1118 | Read/Write | Undefined | Fixed Point |
| V2FixedQ | 0x222 | 0x0000.1110 | Read/Write | Undefined | Fixed Point |
| V2FixedR | 0x225 | 0x0000.1128 | Read/Write | Undefined | Fixed Point |
| V2FixedS | 0x220 | 0x0000.1100 | Read/Write | Undefined | Fixed Point |
| V2FixedT | 0x221 | 0x0000.1108 | Read/Write | Undefined | Fixed Point |
| V2FixedX | 0x22A | 0x0000.1150 | Read/Write | Undefined | Fixed Point |
| V2FixedY | 0x22B | 0x0000.1158 | Read/Write | Undefined | Fixed Point |
| V2FixedZ | 0x22C | 0x0000.1160 | Read/Write | Undefined | Fixed Point |

The input format for each parameter is shown below. The *us* is unsigned and *s* is 2's complement. The clamping range, if clamping is enabled, is also shown.

| Category | Parameter | Fixed Point Format | Clamping Range |
|---|---|---|---|
| Texture | s | 2.30 s[1] | -1.0…1.0[2] |
| | t | 2.30 s | -1.0…1.0 |
| | q | 2.30 s | -1.0…1.0 |
| | Ks | 2.22 us | 0.0…2.0 |
| | Kd | 2.22 us | 0.0…2.0 |
| Color | red | 1.30 us | 0.0…1.0 |
| | green | 1.30 us | 0.0…1.0 |
| | blue | 1.30 us | 0.0…1.0 |
| | alpha | 1.30 us | 0.0…1.0 |
| Fog | f | 10.22 s | -512.0…512.0 |
| Coordinate | x | 16.16 s | -32K…+32K[3][4] |
| | y | 16.16 s | -32K…+32K |
| | z | 1.30us | 0.0…1.0 |

These values read back as floating point and share the same storage as the **V2Float*** registers.

These are legacy commands and should only be used if Gamma is just being used as a faster GLINT Delta and none of the additional Gamma functionality is being used.

[1]This is the range when Normalize is not used. When Normalize is enabled the fixed point format can be anything, providing it is the same for the s, t and q parameters. The numbers will be interpreted as if they had 2.30 format for the purpose of conversion to floating point. If the fixed point format (2.30) is different from what the user had in mind then the input values are just pre-scaled by a fixed amount (i.e. the difference in binary point positions) prior to conversion.

[2]This is the range when Normalize is not used. When Normalize is enabled the range is extended to $2^{\pm 32}$ approximately. This also applies to the t and q values as well.

[3]The normal range here is limited by the size of the screen.

[4]K = 1024.

## V2FloatA
## V2FloatB
## V2FloatF
## V2FloatG
## V2FloatKd
## V2FloatKs
## V2FloatQ
## V2FloatR
## V2FloatS
## V2FloatT
## V2FloatX
## V2FloatY
## V2FloatZ

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| V2FloatA | 0x258 | 0x0000.12C0 | Read/Write | Undefined | Float |
| V2FloatB | 0x257 | 0x0000.12B8 | Read/Write | Undefined | Float |
| V2FloatF | 0x259 | 0x0000.12C8 | Read/Write | Undefined | Float |
| V2FloatG | 0x256 | 0x0000.12B0 | Read/Write | Undefined | Float |
| V2FloatKd | 0x254 | 0x0000.12A0 | Read/Write | Undefined | Float |
| V2FloatKs | 0x253 | 0x0000.1298 | Read/Write | Undefined | Float |
| V2FloatQ | 0x252 | 0x0000.1290 | Read/Write | Undefined | Float |
| V2FloatR | 0x255 | 0x0000.12A8 | Read/Write | Undefined | Float |
| V2FloatS | 0x250 | 0x0000.1280 | Read/Write | Undefined | Float |
| V2FloatT | 0x251 | 0x0000.1288 | Read/Write | Undefined | Float |
| V2FloatX | 0x25A | 0x0000.12D0 | Read/Write | Undefined | Float |
| V2FloatY | 0x25B | 0x0000.12D8 | Read/Write | Undefined | Float |
| V2FloatZ | 0x25C | 0x0000.12E0 | Read/Write | Undefined | Float |

The clamping range, if enabled, for each parameter is shown below.

| Category | Parameter | Clamping Range |
|---|---|---|
| Texture | s | -1.0…1.0[1] |
| | t | -1.0…1.0 |
| | q | -1.0…1.0 |
| | Ks | 0.0…2.0 |
| | Kd | 0.0…2.0 |
| Color | red | 0.0…1.0 |
| | green | 0.0…1.0 |
| | blue | 0.0…1.0 |
| | alpha | 0.0…1.0 |
| Fog | f | -512.0…512.0 |
| Coordinate | x | -32K…+32K[2][3] |
| | y | -32K…+32K |
| | z | 0.0…1.0 |

These registers share the same storage as the **V2Fixed*** registers.

These are legacy commands and should only be used if Gamma is just being used as a faster GLINT Delta and none of the additional Gamma functionality is being used.

[1]This is the range when Normalize is not used. When Normalize is enabled the range is extended to $2^{\pm 32}$ approximately. This also applies to the t and q values as well.

[2]The normal range here is limited by the size of the screen.

[3]K = 1024.

## VertexMachineMode

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| VertexMachineMode | 0x2A0 | 0x0000.1500 | Read/Write | Undefined | Bitfield |

For future use.

## ViewPortOffsetX ViewPortOffsetY ViewPortOffsetZ

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| ViewPortOffsetX | 0x373 | 0x0000.1B98 | Read/Write | Undefined | Float |
| ViewPortOffsetY | 0x374 | 0x0000.1BA0 | Read/Write | Undefined | Float |
| ViewPortOffsetZ | 0x375 | 0x0000.1BA8 | Read/Write | Undefined | Float |

These registers hold the offset values to add to the vertex coordinates (in normalized device coordinates) after they have been scaled by the viewport scale values.

## ViewPortScaleX
## ViewPortScaleY
## ViewPortScaleZ

| **Name** | **Tag** | **Offset** | **Access** | **Reset** | **Format** |
|---|---|---|---|---|---|
| ViewPortScaleX | 0x370 | 0x0000.1B80 | Read/Write | Undefined | Float |
| ViewPortScaleY | 0x371 | 0x0000.1B88 | Read/Write | Undefined | Float |
| ViewPortScaleZ | 0x372 | 0x0000.1B90 | Read/Write | Undefined | Float |

These registers hold the scale values to multiply the vertex coordinates (in normalized device coordinates) by.

## Vw
## Vx2
## Vx3
## Vx4
## Vy
## Vz

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| Vw | 0x30A | 0x0000.1850 | Write | Undefined | Float |
| Vx2 | 0x30D | 0x0000.1868 | Write/Trigger | Undefined | Float |
| Vx3 | 0x30E | 0x0000.1870 | Write/Trigger | Undefined | Float |
| Vx4 | 0x30F | 0x0000.1878 | Write/Trigger | Undefined | Float |
| Vy | 0x30C | 0x0000.1860 | Write | Undefined | Float |
| Vz | 0x30B | 0x0000.1858 | Write | Undefined | Float |

These registers hold the x, y and z vertex components. The **Vx2, Vx3** or **Vx4** registers must be written last as they write will trigger the vertex to be entered into Gamma. All the vertex components should be written together and not interleaved with writes to the color (C*), face normal (FN*), normal (N*) or texture (T*) registers.

Writing to **Vx2** will ignore any supplied values for **Vz** and **Vw** and set them to 0.0 and 1.0 respectively.

Writing to **Vx3** will ignore any supplied values for **Vw** and set it to 1.0.

A vertex is optionally transformed by the **ModelViewMatrix** and/or the **ModelViewProjectionMatrix**.

Once the vertex has been entered into Gamma this may be sufficient to cause a primitive(s) to be generated and rendered. This depends on the primitive type defined by the **Begin** command and how many vertices have been sent (since the previous **Begin**). Vertices are ignored if they are sent before a **Begin** command or after an **End** command.

See also: **Begin**, **End**, **TransformMode**.

## WindowAnd
## WindowOr

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| WindowAnd | 0x570 | 0x0000.2B80 | Write | Undefined | Bitfield |
| WindowOr | 0x571 | 0x0000.2B88 | Write | Undefined | Bitfield |

See the GLINT documentation for the field definition of the **Window** and these registers.

Gamma tracks GLINT's **Window** register to allow its functionality to be extended to better support window clipping using GIDs.

Writing to the **WindowAnd** and **WindowOr** registers logically combine the new value with the existing values in the **Window** register rather than replacing its contents.

## XBias
## YBias

| Name | Tag | Offset | Access | Reset | Format |
|---|---|---|---|---|---|
| XBias | 0x290 | 0x0000.1480 | Read/Write | Undefined | Float |
| YBias | 0x291 | 0x0000.1488 | Read/Write | Undefined | Float |

These registers are added to the primitive's vertex coordinates during the set up calculation. They can be used to convert window relative coordinates into screen relative coordinates, or to remove the bias added in during viewport mapping to force all set up calculation to have the same accuracy, independent of the primitive's screen position.

# Index