



Intel[®] OpenSource HD Graphics Programmer's Reference Manual (PRM) Volume 2 Part 1: 3D/Media – 3D Pipeline (Ivy Bridge)

For the 2012 Intel[®] Core[™] Processor Family

May 2012

Revision 1.0

NOTICE:

This document contains information on products in the design phase of development, and Intel reserves the right to add or remove product features at any time, with or without changes to this open source documentation.



Creative Commons License

You are free to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2012, Intel Corporation. All rights reserved.



Contents

1. 3D Pipeline	9
1.1 Introduction	9
1.2 3D Pipeline Overview	9
1.2.1 3D Pipeline Stages	9
1.3 3D Primitives Overview	10
1.4 3D Pipeline State Overview	16
1.4.1 3D State Model	16
1.4.2 3DSTATE_CC_STATE_POINTERS	18
1.4.3 3DSTATE_BLEND_STATE_POINTERS	19
1.4.4 3DSTATE_DEPTH_STENCIL_STATE_POINTERS	20
1.4.5 3DSTATE_BINDING_TABLE_POINTERS	21
1.5 3DSTATE_SAMPLER_STATE_POINTERS	26
1.5.1 1.5.1 3DSTATE_SAMPLER_STATE_POINTERS_VS	26
1.5.2 3DSTATE_SAMPLER_STATE_POINTERS_HS	27
1.5.3 3DSTATE_SAMPLER_STATE_POINTERS_DS	28
1.5.4 3DSTATE_SAMPLER_STATE_POINTERS_GS	29
1.5.5 3DSTATE_SAMPLER_STATE_POINTERS_PS	30
1.6 3DSTATE_VIEWPORT_STATE_POINTERS	31
1.6.1 3DSTATE_VIEWPORT_STATE_POINTERS_CC	31
1.6.2 3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP	32
1.6.3 3DSTATE_SCISSOR_STATE_POINTERS	33
1.7 3DSTATE_URB Commands	34
1.7.1 3DSTATE_URB_VS	34
1.7.2 3DSTATE_URB_HS	36
1.7.3 3DSTATE_URB_DS	37
1.7.4 3DSTATE_URB_GS	39
1.7.5 Gather Constants	40
1.7.6 Dx9 Constant Buffer Generation	40
1.8 Vertex Data Overview	41
1.8.1 Vertex URB Entry (VUE) Formats	41
1.8.2 Vertex Positions	43
1.9 3D Pipeline Stage Overview	45
1.9.1 Generic 3D FF Unit Block Diagram	46
1.9.2 1Common 3D FF Unit Functions	46
1.9.3 Thread Initiation Management	48
1.9.4 Thread Request Generation	48
1.9.5 Thread Output Handling	54
1.9.6 VUE Readback	54
1.9.7 Statistics Gathering	54
1.10 Synchronization of the 3D Pipeline	56
1.10.1 Top-of-Pipe Synchronization	56
1.10.2 End-of-Pipe Synchronization	56
1.10.3 Synchronization Actions	57
1.10.4 PIPE_CONTROL Command	58
1.11 Push Constant URB Allocation	67
2. 3D Pipeline – Vertex Fetch (VF) Stage	71
2.1 Vertex Fetch (VF) Stage Overview	71
2.1.1 Input Assembly	71
2.1.2 Vertex Cache	72
2.1.3 Input Data: Push Model vs. Pull Model	72
2.1.4 Generated IDs	72



2.2	Index Buffer (IB)	73
2.2.1	3DSTATE_INDEX_BUFFER	73
2.2.2	Index Buffer Access	76
2.3	Vertex Buffers (VBs)	76
2.3.1	3DSTATE_VERTEX_BUFFERS	77
2.3.2	VERTEX_BUFFER_STATE Structure	78
2.3.3	VERTEXDATA Buffers – SEQUENTIAL Access	81
2.3.4	VERTEXDATA Buffers – RANDOM Access	82
2.3.5	INSTANCEDATA Buffers	83
2.4	Input Vertex Definition	83
2.4.1	3DSTATE_VERTEX_ELEMENTS	84
2.4.2	VERTEX_ELEMENT_STATE Structure	85
2.4.3	Vertex Element Data Path	89
2.5	3D Primitive Processing	90
2.5.1	3D PRIMITIVE Command	90
2.5.2	Functional Overview	94
2.5.3	CommandInit	94
2.5.4	InstanceLoop	94
2.5.5	VertexLoop	95
2.5.6	VertexIndexGeneration	95
2.5.7	TerminatePrimitive	96
2.5.8	VertexCacheLookup	96
2.5.9	VertexElementLoop	96
2.5.10	SourceElementFetch	97
2.5.11	Format Conversion	97
2.5.12	DestinationFormatSelection	99
2.5.13	PrimitiveInfoGeneration	100
2.5.14	URBWrite	101
2.5.15	OutputBufferedVertex	101
2.6	Dangling Vertex Removal	101
2.7	Other Vertex Fetch Functionality	102
2.7.1	Statistics Gathering	102
3.	3D Pipeline – Vertex Shader (VS) Stage	104
3.1	VS Stage Overview	104
3.1.1	Vertex Caching	104
3.2	VS Stage Input	106
3.2.1	State	106
3.2.2	Input Vertices	116
3.3	SIMD4x2 VS Thread Request Generation	116
3.3.1	Thread Payload	117
3.4	SIMD4x2 VS Thread Execution	119
3.4.1	Vertex Output	119
3.4.2	Thread Termination	119
3.5	Primitive Output	119
3.6	Other VS Functions	119
3.6.1	Statistics Gathering	119
4.	3D Pipeline – Hull Shader (HS) Stage	121
4.1	HS Stage Overview	121
4.2	HS Stage Input	121
4.2.1	State	121
4.3	3DSTATE_CONSTANT_HS	124
4.4	3DSTATE_HS	127
4.5	Patch Object Staging	133
4.6	HS Thread Payload	133



4.6.1	SINGLE_PATCH Layout (SINGLE-PATCH Mode)	133
4.7	HS Thread Execution	136
4.7.1	Dispatch Mask	137
4.8	ICP Dereferencing	137
4.9	Patch URB Entry (Patch Record) Output	137
4.9.1	Patch Header	137
4.9.2	DOMAIN_POINT Structure	139
4.10	Statistics Gathering	140
4.10.1	HS Invocations	140
5.	3D Pipeline – Tessellation Engine (TE)	141
5.1	3DSTATE_TE	141
5.2	Domain Types and Output Topologies	144
5.3	QUAD Domain Tessellation	144
5.3.1	TRI Domain Tessellation	146
5.4	ISOLINE Domain Tessellation	147
5.5	Patch Culling	147
5.6	Tessellation Factor Limits	147
5.7	Partitioning	148
6.	3D Pipeline – Domain Shader (DS) Stage	149
6.1	3DSTATE_DS	149
6.1.1	3DSTATE_PUSH_CONSTANT_ALLOC_DS	154
6.1.2	3DSTATE_CONSTANT_DS	156
6.2	Thread Payload	159
6.3	DS Thread Execution	163
6.4	Statistics Gathering	164
7.	3D Pipeline – Geometry Shader (GS) Stage	165
7.1	GS Stage Overview	165
7.2	GS Stage Input	165
7.2.1	State	165
7.3	Object Staging	178
7.4	GS Thread Request Generation	178
7.4.1	Object Vertex Ordering	178
7.4.2	GS Thread Payload High-Level Layout	182
7.4.3	GS Thread Payload SIMD 4x2	183
7.5	GS Thread Execution	188
7.5.1	GS Thread Output	189
7.5.2	Stream Output	190
7.5.3	Thread Termination	191
7.6	Primitive Output	191
7.7	Other Functionality	191
7.7.1	Statistics Gathering	191
8.	3D Pipeline - Stream Output Logic (SOL) Stage	193
8.1	Input Buffering	193
8.2	Stream Output Buffers	195
8.3	Stream Output Function	196
8.4	3DSTATE_STREAMOUT	196
8.5	3DSTATE_SO_DECL_LIST Command	201
8.5.1	SO_DECL Structure Definition	205
8.6	3DSTATE_SO_BUFFER	207
8.7	Rendering Disable	208
8.8	Statistics	208
9.	3D Pipeline – Clip Stage	209
9.1	3D Pipeline – CLIP Stage Overview	209



9.1.1	Clip Stage – General-Purpose Processing.....	209
9.1.2	Clip Stage – 3D Clipping	209
9.1.3	Fixed Function Clipper.....	210
9.2	Concepts	210
9.2.1	The Clip Volume	210
9.2.2	User-Specified Clipping	212
9.2.3	Guard Band	212
9.2.4	Vertex-Based Clip Testing & Considerations	215
9.2.5	3D Clipping	217
9.3	CLIP Stage Input.....	217
9.3.1	State	218
9.4	Object Staging.....	223
9.4.1	Partial Object Removal.....	223
9.4.2	ClipDetermination Function	223
9.4.3	ClipMode.....	226
9.5	Object Pass-Through	227
9.6	Primitive Output.....	228
9.7	Other Functionality	229
9.7.1	Statistics Gathering	229
10.	3D Pipeline - Strips and Fans (SF) Stage.....	230
10.1	Overview.....	230
10.1.1	Inputs from CLIP.....	230
10.1.2	Attribute Setup/Interpolation Process.....	231
10.1.3	Outputs to WM.....	231
10.2	Primitive Assembly	231
10.2.1	Point List Decomposition	234
10.2.2	Line List Decomposition	235
10.2.3	Line Strip Decomposition.....	236
10.2.4	Triangle List Decomposition	237
10.2.5	Triangle Strip Decomposition	238
10.2.6	Triangle Fan Decomposition.....	239
10.2.7	Polygon Decomposition.....	240
10.2.8	Rectangle List Decomposition	240
10.3	Object Setup.....	241
10.3.1	Invalid Position Culling (Pre/Post-Transform)	241
10.3.2	Viewport Transformation	241
10.3.3	Destination Origin Bias	241
10.3.4	Point Rasterization Rule Adjustment	242
10.3.5	Drawing Rectangle Offset Application	243
10.3.6	Point Width Application.....	246
10.3.7	Rectangle Completion	247
10.3.8	Vertex X,Y Clamping and Quantization.....	247
10.3.9	Degenerate Object Culling	248
10.3.10	Triangle Orientation (Face) Culling	248
10.3.11	Scissor Rectangle Clipping.....	249
10.3.12	Line Rasterization.....	250
10.3.13	3DSTATE_SF.....	257
10.3.14	3DSTATE_SBE	263
10.3.15	SF_CLIP_VIEWPORT.....	270
10.3.16	SCISSOR_RECT.....	271
10.4	Attribute Interpolation Setup	272
10.4.1	Attribute Swizzling	272
10.4.2	1Interpolation Modes	273
10.4.3	Point Sprites	273



10.5	Depth Offset.....	274
10.6	Other SF Functions.....	274
10.6.1	Statistics Gathering	274
11.	3D Pipeline – Windower (WM) Stage	275
11.1	Overview.....	275
11.1.1	Inputs from SF to WM.....	275
11.2	Windower Pipelined State.....	276
11.2.1	3DSTATE_WM.....	276
11.2.2	3DSTATE_PS.....	283
11.2.3	3DSTATE_CONSTANT_PS.....	291
11.2.4	3DSTATE_PUSH_CONSTANT_ALLOC_PS.....	292
11.2.5	3DSTATE_SAMPLE_MASK.....	293
11.3	Rasterization.....	294
11.3.1	Drawing Rectangle Clipping	295
11.3.2	Line Rasterization.....	295
11.3.3	Polygon (Triangle and Rectangle) Rasterization.....	300
11.4	Multisampling.....	303
11.4.1	Multisample Modes/State	303
11.4.2	3DSTATE_MULTISAMPLE	304
11.5	Early Depth/Stencil Processing	309
11.5.1	Depth Offset.....	309
11.5.2	Early Depth Test/Stencil Test/Write.....	310
11.5.3	Hierarchical Depth Buffer	311
11.5.4	Separate Stencil Buffer.....	314
11.5.5	Depth/Stencil Buffer State	315
11.6	Barycentric Attribute Interpolation	325
11.7	MCS Buffer for Render Target(s)	325
11.8	Render Target Fast Clear.....	328
11.9	Render Target Resolve.....	328
11.10	Pixel Shader Thread Generation.....	329
11.10.1	Pixel Grouping (Dispatch Size) Control.....	329
11.10.2	Multisampling Effects on Pixel Shader Dispatch.....	332
11.10.3	PS Thread Payload for Normal Dispatch	336
11.11	Other WM Functions.....	349
11.11.1	Statistics Gathering	349
12.	3D Pipeline – Color Calculator (Output Merger).....	351
12.1	Overview.....	351
12.1.1	Alpha Coverage.....	352
12.1.2	Alpha Test.....	352
12.1.3	Depth Coordinate Offset.....	353
12.1.4	Stencil Test.....	353
12.1.5	Depth Test	354
12.1.6	Pre-Blend Color Clamping.....	354
12.1.7	Color Buffer Blending	355
12.1.8	Post-Blend Color Clamping	357
12.1.9	Dithering	358
12.1.10	Logic Ops.....	358
12.1.11	Buffer Update	359
12.2	Pixel Pipeline State Summary	361
12.2.1	COLOR_CALC_STATE.....	361
12.2.2	DEPTH_STENCIL_STATE.....	362
12.2.3	BLEND_STATE	366
12.2.4	CC_VIEWPORT	373
12.3	Other Pixel Pipeline Functions	374



12.3.1 Statistics Gathering 374



1. 3D Pipeline

1.1 Introduction

This section covers the programming details for the 3D fixed functions.

1.2 3D Pipeline Overview

1.2.1 3D Pipeline Stages

The following table lists the various stages of the 3D pipeline and describes their major functions.

Pipeline Stage	Functions Performed
Command Stream (CS)	The Command Stream stage is responsible for managing the 3D pipeline and passing commands down the pipeline. In addition, the CS unit reads "constant data" from memory buffers and places it in the URB. Note that the CS stage is shared between the 3D and Media pipelines.
Vertex Fetch (VF)	The Vertex Fetch stage, in response to 3D Primitive Processing commands, is responsible for reading vertex data from memory, reformatting it, and writing the results into Vertex URB Entries. It then outputs primitives by passing references to the VUEs down the pipeline.
Vertex Shader (VS)	The Vertex Shader stage is responsible for processing (shading) incoming vertices by passing them to VS threads.
Hull Shader (HS)	The Hull Shader is responsible for processing (shading) incoming patch primitives as part of the tessellation process.
Tessellation Engine (TE)	The Tessellation Engine is responsible for using tessellation factors (computed in the HS stage) to tessellate U,V parametric domains into domain point topologies.
Domain Shader (DS)	The Domain Shader stage is responsible for processing (shading) the domain points (generated by the TE stage) into corresponding vertices.
Geometry Shader (GS)	The Geometry Shader stage is responsible for processing incoming objects by passing each object's vertices to a GS thread.
Stream Output Logic (SOL)	The Stream Output Logic is responsible for outputting incoming object vertices into Stream Out Buffers in memory.
Clipper (CLIP)	The Clipper stage performs Clip Tests on incoming objects and clips objects if required. Objects are clipped using fixed-function hardware.
Strip/Fan (SF)	The Strip/Fan stage performs object setup. Object setup uses fixed-function hardware.
Windower/Masker (WM)	The Windower/Masker performs object rasterization and spawns WM thread (aka PS thread) to process (shade) the object pixels.



1.3 3D Primitives Overview

The 3DPRIMITIVE command (defined in the VF Stage chapter) is used to submit 3D primitives to be processed by the 3D pipeline. Typically the processing results in the rendering of pixel data into the render targets, but this is not required.

Terminology Note: There is considerable confusion surrounding the term ‘primitive’, e.g., is a triangle strip a ‘primitive’, or is a triangle within a triangle strip a ‘primitive’? In this spec, we will try to avoid ambiguity by using the term ‘object’ to represent the basic shapes (point, line, triangle), and ‘topology’ to represent input geometry (strips, lists, etc.). Unfortunately, terms like ‘3DPRIMITIVE’ must remain for legacy reasons.

The following table describes the basic primitive topology types supported in the 3D pipeline.

Notes:

- There are several variants of the basic topologies. These have been introduced to allow slight variations in behavior without requiring a state change.
- Number of vertices:
 - **Dangling Vertices:** Topologies have an “expected” number of vertices in order to form complete objects within the topologies (e.g., LINELIST is expected to have an even number of vertices). The actual number of vertices specified in the 3DPRIMITIVE command, and as output from the GS unit, is allowed to deviate from this expected number --- in which case any “dangling” vertices are discarded. The removal of dangling vertices is initially performed in the VF unit. In order to filter out dangling vertices emitted by GS threads, the CLIP unit also performs dangling-vertex removal at its input.

3D Primitive Topology Types

3D Primitive Topology Type (ordered alphabetically)	Description
LINELIST	A list of independent line objects (2 vertices per line). Programming Restrictions: Normal usage expects a multiple of 2 vertices, though incomplete objects are silently ignored.
LINELIST_ADJ	A list of independent line objects with adjacency information (4 vertices per line). Programming Restrictions: Normal usage expects a multiple of 4 vertices, though incomplete objects are silently ignored. Not valid as output from GS thread.
LINELOOP	Similar to a 3DPRIM_LINESTRIP, though the last vertex is connected back to the initial vertex via a line object. The LINELOOP topology is converted to LINESTRIP topology at the beginning of the 3D pipeline. Programming Restrictions: Normal usage expects at least 2 vertices, though incomplete objects are silently ignored. (The 2-vertex case is required by OGL).



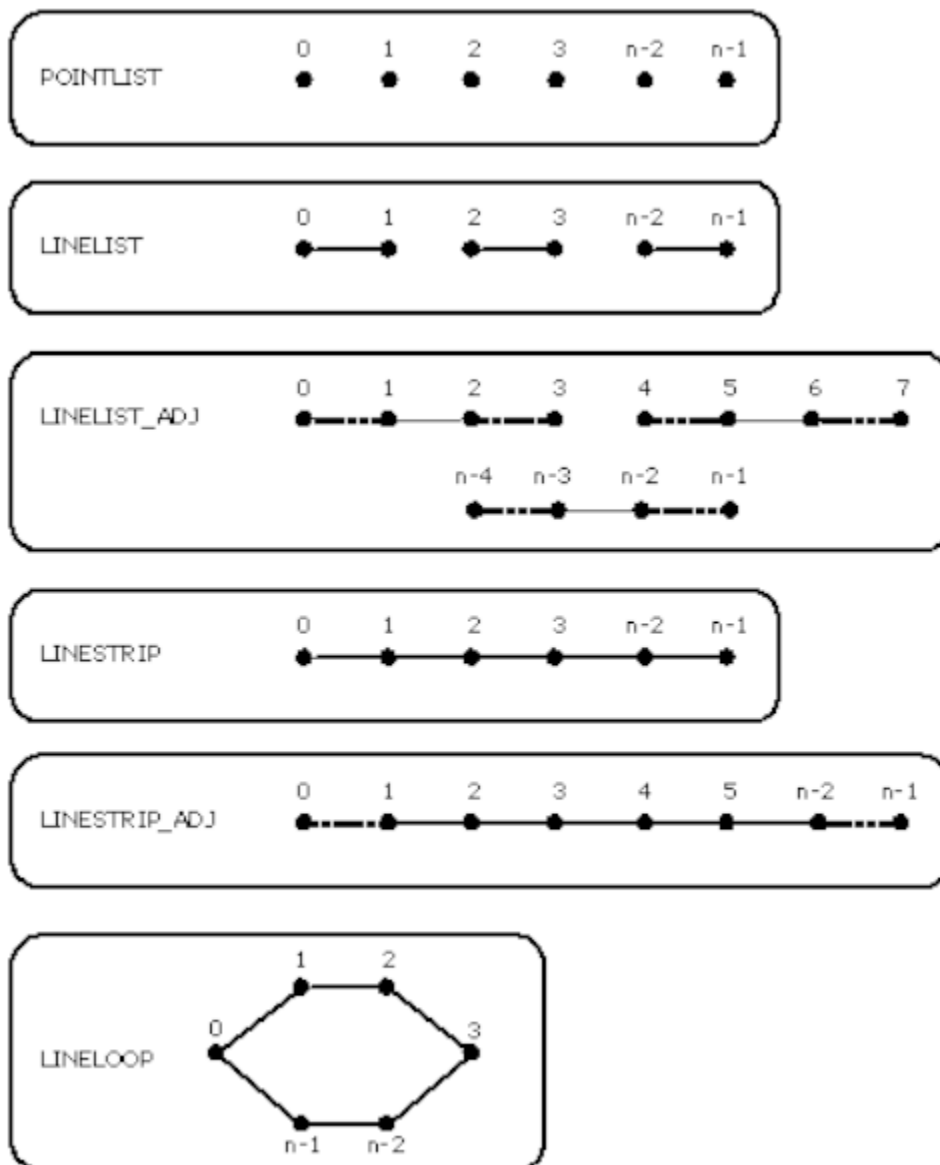
3D Primitive Topology Type (ordered alphabetically)	Description
	Not valid after the GS stage (i.e., must be converted by a GS thread to some other primitive type).
LINESTRIP	<p>A list of vertices connected such that, after the first vertex, each additional vertex is associated with the previous vertex to define a connected line object.</p> <p>Programming Restrictions:</p> <p>Normal usage expects at least 2 vertices, though incomplete objects are silently ignored.</p>
LINESTRIP_ADJ	<p>A list of vertices connected such that, after the first vertex, each additional vertex is associated with the previous vertex to define connected line object. The first and last segments are adjacent—only vertices.</p> <p>Programming Restrictions:</p> <p>Normal usage expects at least 4 vertices, though incomplete objects are silently ignored.</p> <p>Not valid as output from GS thread.</p>
LINESTRIP_BF	<p>Similar to LINESTRIP, except treated as “backfacing” during rasterization (stencil test).</p> <p>This can be used to support “line” polygon fill mode when two-sided stencil is enabled.</p>
LINESTRIP_CONT	<p>Similar to LINESTRIP, except LineStipple (if enabled) is continued (vs. reset) at the start of the primitive topology.</p> <p>This can be used to support line stipple when the API-provided primitive is split across multiple topologies.</p>
LINESTRIP_CONT_BF	Combination of LINESTRIP_BF and LINESTRIP_CONT variations.
POINTLIST	A list of point objects (1 vertex per point).
POINTLIST_BF	<p>Similar to POINTLIST, except treated as “backfacing” during rasterization (stencil test).</p> <p>This can be used to support “point” polygon fill mode when two-sided stencil is enabled.</p>
POLYGON	<p>Similar to TRIFAN, though the first vertex always provides the “flat-shaded” values (vs. this being programmable through state).</p> <p>Programming Restrictions:</p> <p>Normal usage expects at least 3 vertices, though incomplete objects are silently ignored.</p>
QUADLIST	<p>A list of independent quad objects (4 vertices per quad). The QUADLIST topology is converted to POLYGON topology at the beginning of the 3D pipeline.</p> <p>Programming Restrictions:</p> <p>Normal usage expects a multiple of 4 vertices, though incomplete</p>



3D Primitive Topology Type (ordered alphabetically)	Description
	objects are silently ignored.
QUADSTRIP	<p>A list of vertices connected such that, after the first two vertices, each additional pair of vertices are associated with the previous two vertices to define a connected quad object.</p> <p>Programming Restrictions:</p> <p>Normal usage expects an even number (4 or greater) of vertices, though incomplete objects are silently ignored.</p> <p>: To work around IVB bug #3665983 the driver must detect the use of a QUADSTRIP input topology along with the use of primitive ID in the pixel shader, and correspondingly shift right by 1 the primitive ID in the pixel shader.</p>
RECTLIST	<p>A list of independent rectangles, where only 3 vertices are provided per rectangle object, with the fourth vertex implied by the definition of a rectangle. V0=LowerRight, V1=LowerLeft, V2=UpperLeft. Implied V3 = V0-V1+V2.</p> <p>Programming Restrictions:</p> <p>Normal usage expects a multiple of 3 vertices, though incomplete objects are silently ignored.</p> <p>The RECTLIST primitive is supported specifically for 2D operations (e.g., BLTs and “stretch” BLTs) and not as a general 3D primitive. Due to this, a number of restrictions apply to the use of RECTLIST:</p> <p>Must utilize “screen space” coordinates (VPOS_SCREENSPACE) when the primitive reaches the CLIP stage. The W component of position must be 1.0 for all vertices. The 3 vertices of each object should specify a screen-aligned rectangle (after the implied vertex is computed).</p> <p>Clipping: Must not require clipping or rely on the CLIP unit’s ClipTest logic to determine if clipping is required. Either the CLIP unit should be DISABLED, or the CLIP unit’s Clip Mode should be set to a value other than CLIPMODE_NORMAL.</p> <p>Viewport Mapping must be DISABLED (as is typical with the use of screen-space coordinates).</p>
TRIFAN	<p>Triangle objects arranged in a fan (or polygon). The initial vertex is maintained as a common vertex. After the second vertex, each additional vertex is associated with the previous vertex and the common vertex to define a connected triangle object .</p> <p>Programming Restrictions:</p> <p>Normal usage expects at least 3 vertices, though incomplete objects are silently ignored.</p>
TRIFAN_NOSTIPPLE	<p>Similar to TRIFAN, but polygon stipple is not applied (even if enabled).</p> <p>This can be used to support “point” polygon fill mode, under the combination of the following conditions:</p> <p>(a) when the frontfacing and backfacing polygon fill modes are</p>

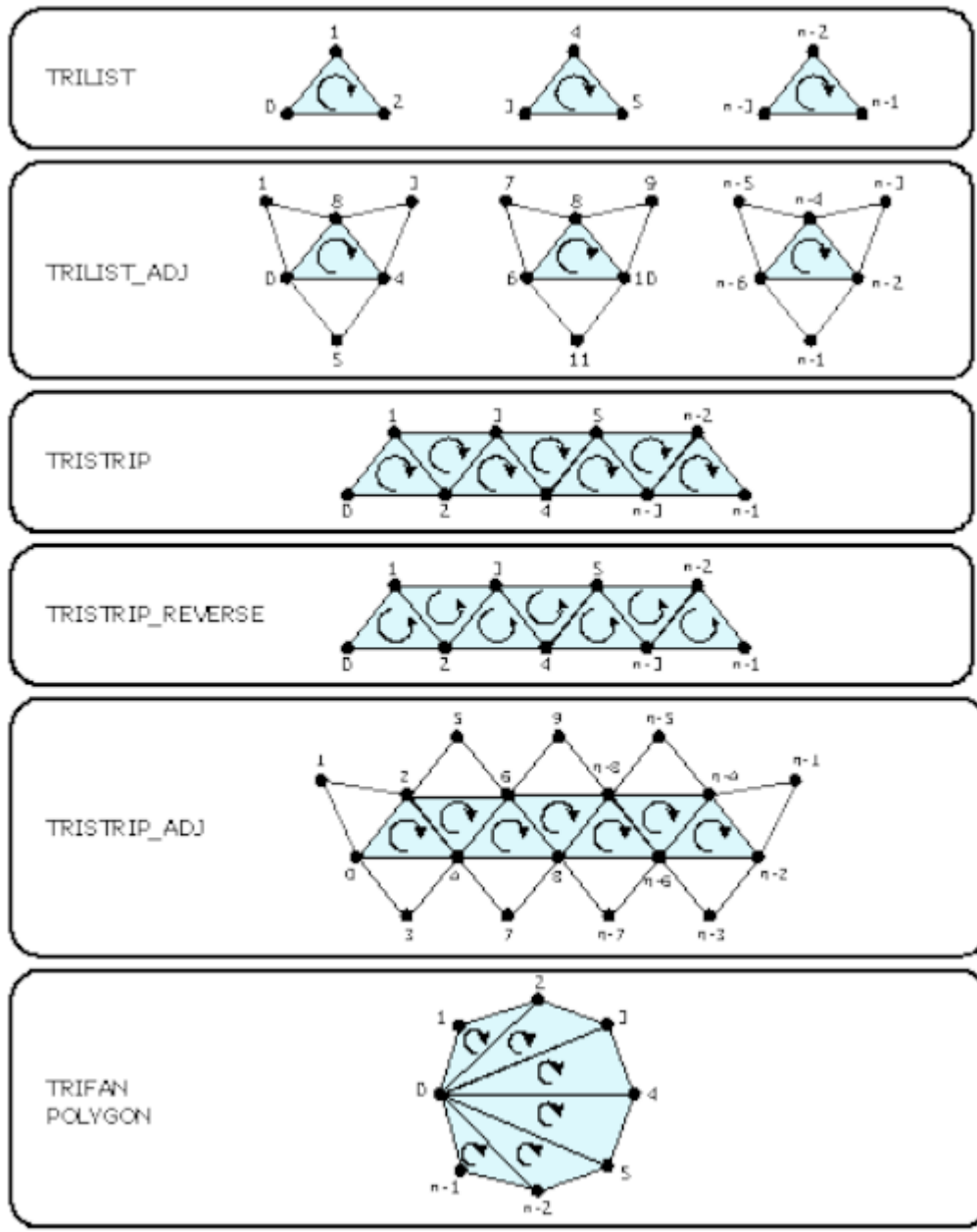
3D Primitive Topology Type (ordered alphabetically)	Description
	<p>different (so the final fill mode is not known to the driver),</p> <p>(b) one of the fill modes is “point” and the other is “solid”,</p> <p>(c) point mode is being emulated by converting the point into a triangle,</p> <p>(d) polygon stipple is enabled. In this case, polygon stipple should not be applied to the points-emulated-as-triangles.</p>
TRILIST	<p>A list of independent triangle objects (3 vertices per triangle).</p> <p>Programming Restrictions:</p> <p>Normal usage expects a multiple of 3 vertices, though incomplete objects are silently ignored.</p>
TRILIST_ADJ	<p>A list of independent triangle objects with adjacency information (6 vertices per triangle).</p> <p>Programming Restrictions:</p> <p>Normal usage expects a multiple of 6 vertices, though incomplete objects are silently ignored.</p> <p>Not valid as output from GS thread.</p>
TRISTRIP	<p>A list of vertices connected such that, after the first two vertices, each additional vertex is associated with the last two vertices to define a connected triangle object.</p> <p>Programming Restrictions:</p> <p>Normal usage expects at least 3 vertices, though incomplete objects are silently ignored.</p>
TRISTRIP_ADJ	<p>A list of vertices where the even-numbered (including 0th) vertices are connected such that, after the first two vertex pairs, each additional even-numbered vertex is associated with the last two even-numbered vertices to define a connected triangle object. The odd-numbered vertices are adjacent-only vertices.</p> <p>Programming Restrictions:</p> <p>Normal usage expects at least 6 vertices, though incomplete objects are silently ignored.</p> <p>Not valid as output from GS thread.</p>
TRISTRIP_REVERSE	<p>Similar to TRISTRIP, though the sense of orientation (winding order) is reversed – this allows SW to break long trisrips into smaller pieces and still maintain correct face orientations.</p>
PATCHLIST_n	<p>List of n-vertex “patch” objects. These topologies cannot be rendered directly – the tessellation units must be utilized to convert them into points, lines or triangles in order to produce rasterization results. (VS, GS and StreamOutput operations can also be performed).</p>

The following diagrams illustrate the basic 3D primitive topologies. (Variants are not shown if they have the same definition with respect to the information provided in the diagrams).

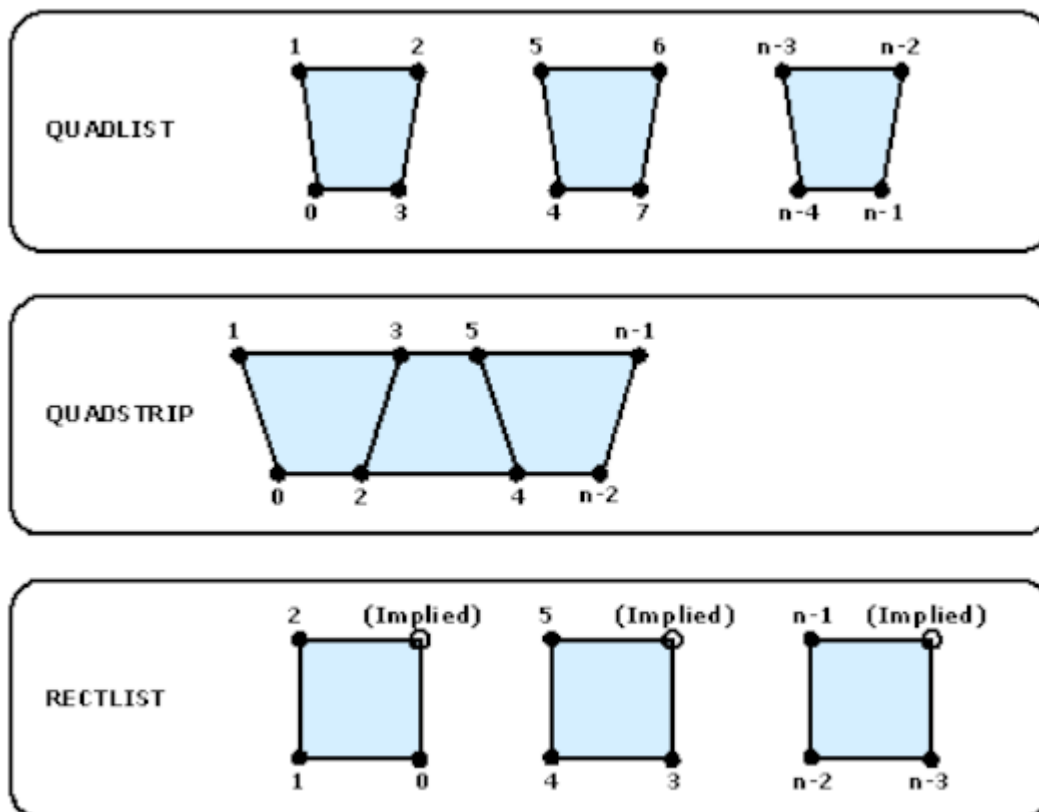


B6815-01

A note on the arrows you see below: These arrows are intended to show the vertex ordering of triangles that are to be considered having “clockwise” winding order in screen space. Effectively, the arrows show the order in which vertices are used in the cross-product (area, determinant) computation. Note that for TRISTRIP, this requires that either the order of odd-numbered triangles be reversed in the cross-product or the sign of the result of the normally-ordered cross-product be flipped (these are identical operations).



B6816-01

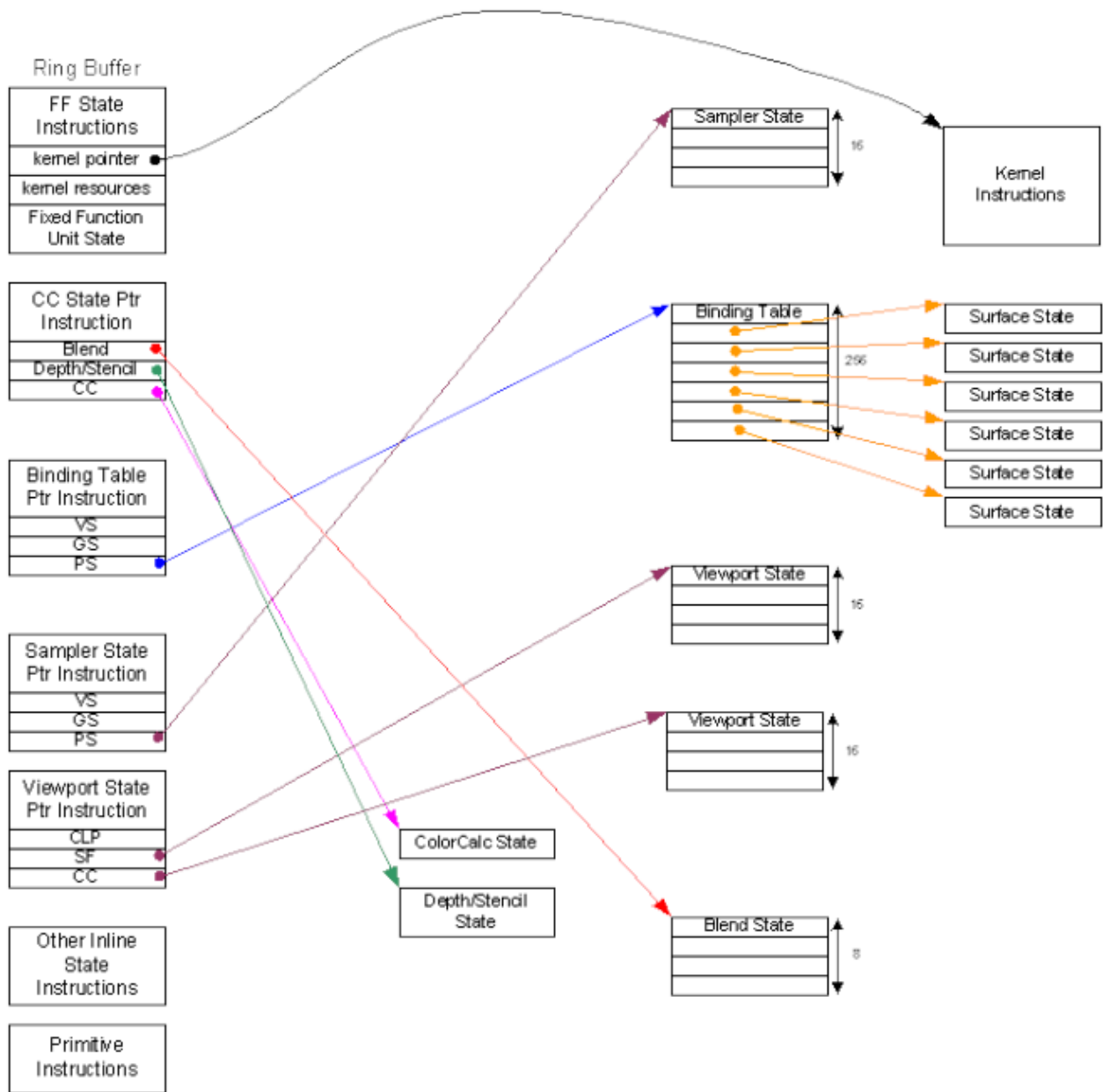


B6818-01

1.4 3D Pipeline State Overview

1.4.1 3D State Model

The locations of the sampler state and viewport state pointers have been moved from the state descriptors to the ring buffer as compared to . In addition, the state for the fixed function pipeline has been moved from indirect state descriptors to inline commands. The color calculator state has been repartitioned.





1.4.2 3DSTATE_CC_STATE_POINTERS

3DSTATE_CC_STATE_POINTERS		
Length Bias: 2		
The 3DSTATE_CC_STATE_POINTERS command is used to set up the pointers to the color calculator state.		
DWord	Bit	Description
0	31:29	Command Type
		Default Value: 3h GFXPIPE
		Format: OpCode
	28:27	Command SubType
		Default Value: 3h GFXPIPE_3D
		Format: OpCode
	26:24	3D Command Opcode
		Default Value: 0h 3DSTATE_PIPELINED
		Format: OpCode
	23:16	3D Command Sub Opcode
Default Value: 0Eh 3DSTATE_CC_STATE_POINTERS		
Format: OpCode		
15:8	Reserved	
	Project: All	
	Format: MBZ	
7:0	DWord Length	
	Default Value: 0h DWORD_COUNT_n	
	Project: All	
	Format: =n	
1	31:6	Pointer to COLOR_CALC_STATE
		Project: All
	Format: DynamicStateOffset[31:6]COLOR_CALC_STATE	
	Specifies the 64-byte aligned offset of the COLOR_CALC_STATE. This offset is relative to the Dynamic State Base Address .	
	5:1	Reserved
Project: All		
Format: MBZ		
0	Reserved	
	Format: MBO	



1.4.3 3DSTATE_BLEND_STATE_POINTERS

3DSTATE_BLEND_STATE_POINTERS			
Length Bias:		2	
The 3DSTATE_BLEND_STATE_POINTERS command is used to set up the pointers to the color calculator state.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value: 3h GFXPIPE	
		Format: OpCode	
	28:27	Command SubType	
		Default Value: 3h GFXPIPE_3D	
		Format: OpCode	
	26:24	3D Command Opcode	
		Default Value: 0h 3DSTATE_PIPELINED	
		Format: OpCode	
	23:16	3D Command Sub Opcode	
	Default Value: 24h 3DSTATE_BLEND_STATE_POINTERS		
	Format: OpCode		
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	0h DWORD_COUNT_n	
	Project:	All	
	Format:	=n	
1	31:6	Pointer to BLEND_STATE	
		Project:	All
		Format:	DynamicStateOffset[31:6]BLEND_STATE*8
	Specifies the 64-byte aligned offset of the BLEND_STATE. This offset is relative to the Dynamic State Base Address .		
	5:1	Reserved	
		Project:	All
		Format:	MBZ
	0	Reserved	
		Format:	MB0



1.4.4 3DSTATE_DEPTH_STENCIL_STATE_POINTERS

3DSTATE_DEPTH_STENCIL_STATE_POINTERS			
Length Bias:		2	
Set up the pointer to the Depth Stencil state.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		25h 3DSTATE_DEPTH_STENCIL_STATE_POINTERS	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	0h DWORD_COUNT_n	
	Project:	All	
	Format:	=n	
1	31:6	Pointer to DEPTH_STENCIL_STATE	
		Project:	All
		Format:	DynamicStateOffset[31:6]DEPTH_STENCIL_STATE
	Specifies the 64-byte aligned offset of the DEPTH_STENCIL_STATE. This offset is relative to the Dynamic State Base Address .		
5:1	Reserved		
	Project:	All	
	Format:	MBZ	
0	Reserved		
	Format:	MB0	



1.4.5 3DSTATE_BINDING_TABLE_POINTERS

1.4.5.1 3DSTATE_BINDING_TABLE_POINTERS_VS

		3DSTATE_BINDING_TABLE_POINTERS_VS	
Length Bias:		2	
The 3DSTATE_BINDING_TABLE_POINTERS_VS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		26h 3DSTATE_BINDING_TABLE_POINTERS_VS	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	0h DWORD_COUNT_n	
	Project:	All	
	Format:	=n	
1	31:16	Reserved	
		Project:	All
		Format:	MBZ
	15:5	Pointer to VS Binding Table	
		Format:	SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256
		Specifies the 32-byte aligned address offset of the VS function's BINDING_TABLE_STATE. This offset is relative to the Surface State Base Address .	
	4:0	Reserved	
		Project:	All
		Format:	MBZ



1.4.5.2 3DSTATE_BINDING_TABLE_POINTERS_HS

3DSTATE_BINDING_TABLE_POINTERS_HS			
Length Bias:		2	
The 3DSTATE_BINDING_TABLE_POINTERS_HS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
		Default Value:	27h 3DSTATE_BINDING_TABLE_POINTERS_HS
		Format:	OpCode
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	0h DWORD_COUNT_n	
	Project:	All	
	Format:	=n	
1	31:16	Reserved	
		Project:	All
		Format:	MBZ
	15:5	Pointer to HS Binding Table	
		Format:	SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256
		Specifies the 32-byte aligned address offset of the HS function's BINDING_TABLE_STATE. This offset is relative to the Surface State Base Address .	
	4:0	Reserved	
		Project:	All
		Format:	MBZ



1.4.5.3 3DSTATE_BINDING_TABLE_POINTERS_DS

3DSTATE_BINDING_TABLE_POINTERS_DS		
Length Bias: 2		
The 3DSTATE_BINDING_TABLE_POINTERS_DS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.		
DWord	Bit	Description
0	31:29	Command Type
		Default Value: 3h GFXPIPE
		Format: OpCode
	28:27	Command SubType
		Default Value: 3h GFXPIPE_3D
		Format: OpCode
	26:24	3D Command Opcode
		Default Value: 0h 3DSTATE_PIPELINED
		Format: OpCode
	23:16	3D Command Sub Opcode
Default Value: 28h 3DSTATE_BINDING_TABLE_POINTERS_DS		
Format: OpCode		
15:8	Reserved	
	Project: All	
	Format: MBZ	
7:0	DWord Length	
	Default Value: 0h DWORD_COUNT_n	
	Project: All	
	Format: =n	
1	31:16	Reserved
		Project: All
		Format: MBZ
	15:5	Pointer to DS Binding Table
		Format: SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256 Specifies the 32-byte aligned address offset of the DS function's BINDING_TABLE_STATE. This offset is relative to the Surface State Base Address .
4:0	Reserved	
	Project: All	
	Format: MBZ	



1.4.5.4 3DSTATE_BINDING_TABLE_POINTERS_GS

3DSTATE_BINDING_TABLE_POINTERS_GS		
Length Bias: 2		
The 3DSTATE_BINDING_TABLE_POINTERS_GS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.		
DWord	Bit	Description
0	31:29	Command Type
		Default Value: 3h GFXPIPE
		Format: OpCode
	28:27	Command SubType
		Default Value: 3h GFXPIPE_3D
		Format: OpCode
	26:24	3D Command Opcode
		Default Value: 0h 3DSTATE_PIPELINED
		Format: OpCode
	23:16	3D Command Sub Opcode
		Default Value: 29h 3DSTATE_BINDING_TABLE_POINTERS_GS
		Format: OpCode
15:8	Reserved	
	Project: All	
	Format: MBZ	
7:0	DWord Length	
	Default Value: 0h DWORD_COUNT_n	
	Project: All	
	Format: =n	
1	31:16	Reserved
		Project: All
		Format: MBZ
	15:5	Pointer to GS Binding Table
		Format: SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256
		Specifies the 32-byte aligned address offset of the GS function's BINDING_TABLE_STATE. This offset is relative to the Surface State Base Address .
	4:0	Reserved
		Project: All
		Format: MBZ



1.4.5.5 3DSTATE_BINDING_TABLE_POINTERS_PS

3DSTATE_BINDING_TABLE_POINTERS_PS			
Length Bias:		2	
The 3DSTATE_BINDING_TABLE_POINTERS_PS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		2Ah 3DSTATE_BINDING_TABLE_POINTERS_PS	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	0h DWORD_COUNT_n	
	Project:	All	
	Format:	=n	
1	31:16	Reserved	
		Project:	All
		Format:	MBZ
	15:5	Pointer to PS Binding Table	
Format:	SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256		
		Specifies the 32-byte aligned address offset of the PS function's BINDING_TABLE_STATE. This offset is relative to the Surface State Base Address .	
4:0	Reserved		
	Project:	All	
	Format:	MBZ	



1.5 3DSTATE_SAMPLER_STATE_POINTERS

1.5.1 3DSTATE_SAMPLER_STATE_POINTERS_VS

3DSTATE_SAMPLER_STATE_POINTERS_VS			
Length Bias:		2	
The 3DSTATE_SAMPLER_STATE_POINTERS_VS command is used to define the location of VS SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		2Bh 3DSTATE_SAMPLER_STATE_POINTERS_VS	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	0h DWORD_COUNT_n	
	Format:	=n	
1	31:5	Pointer to VS Sampler State	
		Project:	All
		Format:	DynamicStateOffset[31:5]SAMPLER_STATE*16
	Specifies the 32-byte aligned address offset of the VS function's SAMPLER_STATE table. This offset is relative to the Dynamic State Base Address.		
	4:0	Reserved	
Project:		All	
Format:		MBZ	



1.5.2 3DSTATE_SAMPLER_STATE_POINTERS_HS

3DSTATE_SAMPLER_STATE_POINTERS_HS			
Length Bias:		2	
The 3DSTATE_SAMPLER_STATE_POINTERS_HS command is used to define the location of HS SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		2Ch 3DSTATE_SAMPLER_STATE_POINTERS_HS	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	0h DWORD_COUNT_n	
	Format:	=n	
1	31:5	Pointer to HS Sampler State	
		Project:	All
		Format:	DynamicStateOffset[31:5]SAMPLER_STATE*16
	Specifies the 32-byte aligned address offset of the HS function's SAMPLER_STATE table. This offset is relative to the Dynamic State Base Address.		
	4:0	Reserved	
Project:		All	
Format:		MBZ	



1.5.3 3DSTATE_SAMPLER_STATE_POINTERS_DS

3DSTATE_SAMPLER_STATE_POINTERS_DS			
Length Bias:		2	
The 3DSTATE_SAMPLER_STATE_POINTERS_DS command is used to define the location of DS SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		2Dh 3DSTATE_SAMPLER_STATE_POINTERS_DS	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	0h DWORD_COUNT_n	
	Format:	=n	
1	31:5	Pointer to DS Sampler State	
		Project:	All
		Format:	DynamicStateOffset[31:5]SAMPLER_STATE*16
	Specifies the 32-byte aligned address offset of the DS function's SAMPLER_STATE table. This offset is relative to the Dynamic State Base Address.		
	4:0	Reserved	
Project:		All	
Format:		MBZ	



1.5.4 3DSTATE_SAMPLER_STATE_POINTERS_GS

3DSTATE_SAMPLER_STATE_POINTERS_GS			
Length Bias:		2	
The 3DSTATE_SAMPLER_STATE_POINTERS_GS command is used to define the location of GS SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		2Eh 3DSTATE_SAMPLER_STATE_POINTERS_GS	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	0h DWORD_COUNT_n	
	Format:	=n	
1	31:5	Pointer to GS Sampler State	
		Project:	All
		Format:	DynamicStateOffset[31:5]SAMPLER_STATE*16
	Specifies the 32-byte aligned address offset of the GS function's SAMPLER_STATE table. This offset is relative to the Dynamic State Base Address.		
	4:0	Reserved	
Project:		All	
Format:		MBZ	



1.5.5 3DSTATE_SAMPLER_STATE_POINTERS_PS

3DSTATE_SAMPLER_STATE_POINTERS_PS			
Length Bias:		2	
The 3DSTATE_SAMPLER_STATE_POINTERS_PS command is used to define the location of PS SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		2Fh 3DSTATE_SAMPLER_STATE_POINTERS_PS	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	0h DWORD_COUNT_n	
	Format:	=n	
1	31:5	Pointer to PS Sampler State	
		Project:	All
		Format:	DynamicStateOffset[31:5]SAMPLER_STATE*16
	Specifies the 32-byte aligned address offset of the PS function's SAMPLER_STATE table. This offset is relative to the Dynamic State Base Address.		
4:0	Reserved		
	Project:	All	
	Format:	MBZ	



1.6 3DSTATE_VIEWPORT_STATE_POINTERS

1.6.1 3DSTATE_VIEWPORT_STATE_POINTERS_CC

3DSTATE_VIEWPORT_STATE_POINTERS_CC			
Length Bias:		2	
The 3DSTATE_VIEWPORT_STATE_POINTERS_CC command is used to define the location of fixed functions' viewport state table.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		23h 3DSTATE_VIEWPORT_STATE_POINTERS	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	0h DWORD_COUNT_n	
	Format:	=n	
1	31:5	Pointer to CC_VIEWPORT	
		Project:	All
		Format:	DynamicStateOffset[31:5]CC_VIEWPORT*16
	Specifies the 32-byte aligned address offset of the CC_VIEWPORT state. This offset is relative to the Dynamic State Base Address.		
	4:0	Reserved	
Project:		All	
Format:		MBZ	



1.6.2 3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP

3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP		
Length Bias: 2		
The 3DSTATE_VIEWPORT_STATE_POINTERS_CLIP command is used to define the location of fixed functions' viewport state table.		
DWord	Bit	Description
0	31:29	Command Type
		Default Value: 3h GFXPIPE
		Format: OpCode
	28:27	Command SubType
		Default Value: 3h GFXPIPE_3D
		Format: OpCode
	26:24	3D Command Opcode
		Default Value: 0h 3DSTATE_PIPELINED
		Format: OpCode
	23:16	3D Command Sub Opcode
Default Value: 21h 3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP		
Format: OpCode		
15:8	Reserved	
	Project: All	
	Format: MBZ	
7:0	DWord Length	
	Default Value: 0h DWORD_COUNT_n	
	Format: =n	
1	31:6	Pointer to SF_CLIP_VIEWPORT
		Project: All
		Format: DynamicStateOffset[31:6]SF_CLIP_VIEWPORT*16
	Specifies the 64-byte aligned address offset of the SF_CLIP_VIEWPORT state. This offset is relative to the Dynamic State Base Address.	
5:0	Reserved	
	Project: All	
	Format: MBZ	



1.6.3 3DSTATE_SCISSOR_STATE_POINTERS

3DSTATE_SCISSOR_STATE_POINTERS			
Length Bias:		2	
The 3DSTATE_SCISSOR_STATE_POINTERS command is used to define the location of the indirect SCISSOR_RECT state.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		0Fh 3DSTATE_SCISSOR_STATE_POINTERS	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	0h DWORD_COUNT_n	
	Format:	=n	
1	31:5	Pointer to SCISSOR_RECT	
		Project:	All
		Format:	DynamicStateOffset[31:5]SCISSOR_RECT*16
Specifies the 32-byte aligned address offset of the SCISSOR_RECT state. This offset is relative to the Dynamic State Base Address .			
4:0	Reserved		
	Project:	All	
	Format:	MBZ	



1.7 3DSTATE_URB Commands

1.7.1 3DSTATE_URB_VS

3DSTATE_URB_VS				
Length Bias:			2	
Description			Project	
VS URB Entry Allocation Size equal to 4(5 512-bit URB rows) may cause performance to decrease due to banking in the URB. Element sizes of 16 to 20 should be programmed with six 512-bit URB rows.				
This command may not overlap with the push constants in the URB defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.				
Programming Notes				
3DSTATE_URB_HS, 3DSTATE_URB_DS, and 3DSTATE_URB_GS must also be programmed in order for the programming of this state to be valid.				
DWord	Bit	Description		
0	31:29	Command Type		
		Default Value:	3h GFXPIPE	
		Format:	OpCode	
	28:27	Command SubType		
		Default Value:	3h GFXPIPE_3D	
		Format:	OpCode	
	26:24	3D Command Opcode		
		Default Value:	0h	
		Format:	OpCode	
	23:16	3D Command Sub Opcode		
Default Value:		30h 3DSTATE_URB_VS		
Format:		OpCode		
15:8	Reserved			
	Project:	All		
	Format:	MBZ		
7:0	DWord Length			
	Default Value:	0h DWORD_COUNT_n		
	Project:	All		
	Format:	=n		
1	31	Reserved		
		Format:	MBZ	
	30	Reserved		
29:25	VS URB Starting Address			
	Format:	U5		
	Offset from the start of the URB memory where VS starts its allocation, specified in multiples of 8 KB.			
		Value	Name	
			Project	



3DSTATE_URB_VS			
	[0,31]		
	[0,15]		
24:16	VS URB Entry Allocation Size		
	Project:	All	
	Format:	U9-1 count of 512-bit units	
	Specifies the length of each URB entry owned by VS. This field is always used (even if VS Function Enable is DISABLED).		
	Programming Notes		
Programming Restriction: As the VS URB entry serves as both the per-vertex input and output of the VS shader, the VS URB Allocation Size must be sized to the maximum of the vertex input and output structures.			
15:0	VS Number of URB Entries		
	Project:	All	
	Format:	U16	
	Specifies the number of URB entries that are used by VS. This field is always used (even if VS Function Enable is DISABLED).		
	Value	Name	Project
	[32,704]		
	[32,512]		
Programming Notes			
Programming Restriction: VS Number of URB Entries must be divisible by 8 if the VS URB Entry Allocation Size is less than 9 512-bit URB entries. "2:0" = reserved "000b"			



1.7.2 3DSTATE_URB_HS

3DSTATE_URB_HS			
Length Bias:		2	
This command may not overlap with the push constants in the URB defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.			
Programming Notes			
3DSTATE_URB_VS, 3DSTATE_URB_DS, and 3DSTATE_URB_GS must also be programmed in order for the programming of this state to be valid.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value: 3h GFXPIPE Format: OpCode	
	28:27	Command SubType	
		Default Value: 3h GFXPIPE_3D Format: OpCode	
	26:24	3D Command Opcode	
		Default Value: 0h 3DSTATE_PIPELINED Format: OpCode	
	23:16	3D Command Sub Opcode	
		Default Value: 31h 3DSTATE_URB_HS Format: OpCode	
	15:8	Reserved	
		Project: All Format: MBZ	
7:0	DWord Length		
	Default Value: 0h DWORD_COUNT_n		
	Project: All Format: =n		
1	31	Reserved	
		Format: MBZ	
	30	Reserved	
		Format: MBZ	
	29:25	HS URB Starting Address	
		Format: U5	
		Offset from the start of the URB memory where HS starts its allocation, specified in multiples of 8 KB.	
		Value	Name
		[0,31]	
	[0,15]		
24:16	HS URB Entry Allocation Size		
	Project: All Format: U9-1 Count of 512-bit units		



3DSTATE_URB_HS		
		Specifies the length of each URB entry owned by HS. This field is always used (even if HS Function Enable is DISABLED).
15:0	HS Number of URB Entries	
	Project:	All
	Specifies the number of URB entries that are used by HS. This field is always used (even if HS Function Enable is DISABLED).	
	Programming Restriction:HS Number of URB Entries must be divisible by 8 if the HS URB Entry Allocation Size is less than 9 512-bit URB entries.“2:0” = reserved “000”	
	Value	Name
	[0,64]	
	[0,32]	

1.7.3 3DSTATE_URB_DS

3DSTATE_URB_DS		
Length Bias:		2
This command may not overlap with the push constants in the URB defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.		
Programming Notes		
3DSTATE_URB_VS, 3DSTATE_URB_HS, and 3DSTATE_URB_GS must also be programmed in order for the programming of this state to be valid.		
DWord	Bit	Description
0	31:29	Command Type
		Default Value: 3h GFXPIPE Format: OpCode
28:27		Command SubType
		Default Value: 3h GFXPIPE_3D Format: OpCode
26:24		3D Command Opcode
		Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
23:16		3D Command Sub Opcode
		Default Value: 32h 3DSTATE_URB_DS Format: OpCode
15:8		Reserved
		Project: All Format: MBZ
7:0		DWord Length
		Default Value: 0h DWORD_COUNT_n Project: All



3DSTATE_URB_DS			
		Format: =n	
1	31	Reserved	
		Format: MBZ	
	30	Reserved	
		Format: MBZ	
	29:25	DS URB Starting Address	
		Format: U5	
		Offset from the start of the URB memory where DS starts its allocation, specified in multiples of 8 KB.	
		Value	Name
		[0,31]	
	[0,15]		
	24:16	DS URB Entry Allocation Size	
		Project: All	
		Format: U9-1 Count of 512-bit units	
		Specifies the length of each URB entry owned by DS. This field is always used (even if DS Function Enable is DISABLED).	
	Value	Name	
	[0,9]		
15:0	DS Number of URB Entries		
	Project: All		
	Description	Project	
	Specifies the number of URB entries that are used by DS. This field is always used (even if DS Function Enable is DISABLED).		
	If Domain Shader Thread Dispatch is Enabled then the minimum number handles that must be allocated is 138 URB entries.		
	"2:0" = reserved "000"		
	Value	Name	Project
	[0,448]		
[0,288]			
Programming Notes			
DS Number of URB Entries must be divisible by 8 if the DS URB Entry Allocation Size is less than 9 512-bit URB entries.If Domain Shader Thread Dispatch is Enabled then the minimum number of handles that must be allocated is 10 URB entries.			



1.7.4 3DSTATE_URB_GS

3DSTATE_URB_GS				
Length Bias:		2		
This command may not overlap with the push constants in the URB defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.				
Programming Notes				
3DSTATE_URB_VS, 3DSTATE_URB_HS, and 3DSTATE_URB_DS must also be programmed in order for the programming of this state to be valid.				
DWord	Bit	Description		
0	31:29	Command Type		
		Default Value:	3h GFXPIPE	
		Format:	OpCode	
	28:27	Command SubType		
		Default Value:	3h GFXPIPE_3D	
		Format:	OpCode	
	26:24	3D Command Opcode		
		Default Value:	0h 3DSTATE_PIPELINED	
		Format:	OpCode	
	23:16	3D Command Sub Opcode		
Default Value:		33h 3DSTATE_URB_GS		
Format:		OpCode		
15:8	Reserved			
	Project:	All		
	Format:	MBZ		
7:0	DWord Length	Default Value:	0h DWORD_COUNT_n	
		Project:	All	
		Format:	=n	
1	31	Reserved		
		Format:	MBZ	
	30	Reserved		
		Format:	MBZ	
	29:25	GS URB Starting Address		
		Format:	U5	
		Offset from the start of the URB memory where GS starts its allocation, specified in multiples of 8 KB.		
		Value	Name	Project
		[0,31]		
	[0,15]			
24:16	GS URB Entry Allocation Size			
	Project:	All		
	Format:	U9-1 512-bit units		



3DSTATE_URB_GS		
		Specifies the length of each URB entry owned by GS. This field is always used (even if GS Function Enable is DISABLED).
15:0	GS Number of URB Entries	
	Project:	All
		Specifies the number of URB entries that are used by GS. This field is always used (even if GS Function Enable is DISABLED).
		Programming Restriction: GS Number of URB Entries must be divisible by 8 if the GS URB Entry Allocation Size is less than 9 512-bit URB entries.
		"2:0" = reserved "000"
	Value	Name
	[0,320]	
	[0,192]	

1.7.5 Gather Constants

In Dx10 the app can provide up to 16 constant buffers. The compiler does some optimizations of constant usage and determines which elements of which constants should be packed in which push constant register for optimum shader performance. While this gathering and packing of constant elements into push constant registers optimizes the shader, it cause the driver additional work at draw call time, since the driver must do the gather and packing at draw time. A new cmd 3D_STATE_GATHER_CONSTANT_* is added to offload the gather and packing functions from the driver. There are 5 FF which support push constants (VS, GS, DS, HS, PS) and they all have corresponding gather cmds. The compiler generates a gather table which instructs what elements of what buffers should be pack into the gather buffer. The gather table indexes the BT to get the surface state which points to the constant buffer. The resource streamer fills gather buffer when it executes a 3D_STATE_GATHER_CONSTANT_* cmd. Once the gather buffer has been filled, the Cmd streamer will execute the 3D_STATE_CONSTANT_* to load the push constant into the URB.

Note: The gather push constants can only be used if the HW generated binding tables are also used.

1.7.6 Dx9 Constant Buffer Generation

The Dx9 constant model is a set of register that the App can incrementally update. The HW requires a constant buffer which lives until the last shader using that buffer retires. To offload the driver the 3DSTATE_DX9_CONSTANT*_* cmds are added. These commands allow the on-die constant register to be maintained. When all the edits to the constant register have been completed, the 3DSTATE_DX9_GENERATE_ACTIVE_* cmd is used to write out a constant buffer to the Dx9 Constant buffer pool. The Dx9 constant buffers are fixed 8KB in size, w/ a large portion of the 2nd 4KB unused.



1.8 Vertex Data Overview

The 3D pipeline FF stages (past VF) receive input 3D primitives as a stream of vertex information packets. (These packets are not directly visible to software). Much of the data associated with a vertex is passed indirectly via a VUE handle. The information provided in vertex packets includes:

- The **URB Handle** of the VUE: This is used by the FF unit to refer to the VUE and perform any required operations on it (e.g., cause it to be read into the thread payload, dereference it, etc.).
- **Primitive Topology Information**: This information is used to identify/delineate primitive topologies in the 3D pipeline. Initially, the VF unit supplies this information, which then passes through the VS stage unchanged. GS and CLIP threads must supply this information with each vertex they produce (via the URB_WRITE message). If a FF unit directly outputs vertices (that were not generated by a thread they spawned), that FF unit is responsible for providing this information.
 - **PrimType**: The type of topology, as defined by the corresponding field of the 3DPRIMITIVE command.
 - **StartPrim**: TRUE only for the first vertex of a topology.
 - **EndPrim**: TRUE only for the last vertex of a topology.
 - The FF unit which owns the VUE
 - Sequence numbers which uniquely identify (with some limits) the VUE output by the owning FF unit. (This data can be used to trap on a specific vertex)
- (Possibly, depending on FF unit) Data read back from the **Vertex Header** of the VUE.

1.8.1 Vertex URB Entry (VUE) Formats

In general, vertex data is stored in Vertex URB Entries (VUEs) in the URB, processed by CLIP threads, and only referenced by the pipeline stages indirectly via VUE handles. Therefore (for the most part) the contents/format of the vertex data is not exposed to 3D pipeline hardware – the FF units are typically only aware of the handles and sizes of VUEs.

VUEs are written in two ways:

- At the top of the 3D Geometry pipeline, the VF's InputAssembly function creates VUEs and initializes them from data extracted from Vertex Buffers as well as internally-generated data.
- VS, GS, and CLIP threads can compute, format and write new VUEs as thread output.

There are only two points in the 3D FF pipeline where the FF units are exposed to the VUE data. Otherwise the VUE remains opaque to the 3D pipeline hardware.

- Just prior to the CLIP stage, all VUEs are read-back:
 - : Optional readback of ClipDistance values (up to 8 floats in an aligned 256-bit URB row)
- Just after the CLIP stage, on clip-generated VUEs are read-back:
 - Readback of the Vertex Header (first 256 bits of the VUE)

Software must ensure that any VUEs subject to readback by the 3D pipeline start with a valid Vertex Header. This extends to all VUEs with the following exceptions listed below:



- If the VS function is enabled, the VF-written VUEs are not required to have Vertex Headers, as the VS-incoming vertices are guaranteed to be consumed by the VS (i.e., the VS thread is responsible for overwriting the input vertex data).
- If the GS FF is enabled, neither VF-written VUEs nor VS thread-generated VUEs are required to have Vertex Headers, as the GS will consume all incoming vertices.
- (There is a pathological case where the CLIP state can be programmed to guarantee that all CLIP-incoming vertices are consumed – regardless of the data read back prior to the CLIP stage – and therefore only the CLIP thread-generated vertices would require Vertex Headers).

The following table defines the Vertex Header. The Position fields are described in further detail below.

VUE Vertex Header ()

DWord	Bit	Description
D0	31:0	Reserved: MBZ
D1	31:0	<p>Render Target Array Index (RTAIndex). This value is (eventually) used to index into a specific element of an “array” Render Target. It is read back by the GS unit (for all exiting vertices) and the Clip unit (for all clip-generated vertices), subsequently routed into the PS thread payload, and eventually included in the RTWrite DataPort message header for use by the DataPort shared function.</p> <p>Software is responsible for ensuring this field is zero whenever a programmable index value is not required. When a programmable index value is required (e.g.) software must ensure that the correct 11-bit value is written to this field. Specifically, the kernels must perform a reange check of computed index values against [0,2047], and output zero if that range is exceeded. Note that the unmodified “renderTargetArrayIndex” must be maintained in the VUE outside of the Vertex Header.</p> <p>Software can force an RTAIndex of 0 to be used (effectively ignoring the setting of this DWord) by use of the ForceZeroRTAIndex bit (3DSTATE_CLIP). Otherwise the read-back value will be used to select an RTArray element, after being clamped to the RTArray surface’s [MinimumArrayElement, Depth] range (SURFACE_STATE).</p> <p>Format: 0-based U32 index value</p>
D2	31:0	<p>Viewport Index. This value is used to select one of a possible 16 sets of viewport (VP) state parameters in the Clip unit’s VertexClipTest function and in the SF unit’s ViewportMapping and Scissor functions.</p> <p>The GS unit (even if disabled) will read back this value for all vertices exiting the GS stage and entering the Clip stage. When enabled, the GS unit will range-check the value against [0,Maximum VPIndex] (see GS_STATE, CLIP_STATE). After this range-check the values are sent down the pipeline and used in the Clip unit’s VertexClipTest function. For vertices passing through the Clip stage, these values will also be sent to the SF unit for use in ViewportMapping and Scissor functions.</p> <p>The Clip unit (if enabled) will read back this value only for vertices generated by CLIP threads. The Clip unit will perform a range clamp similar to the GS unit.</p> <p>Software can force a value of 0 to be used by programming Maximum VPIndex to 0.</p> <p>Format: 0-based U32 index value</p>
D3	31:0	<p>Point Width. This field specifies the width of POINT objects in screen-space pixels. It is used only for vertices within POINTLIST and POINTLIST_BF primitive topologies, and is ignored for vertices associated with other primitive topologies.</p> <p>This field is read back by both the GS and Clip units.</p> <p>Format: FLOAT32</p>
D4	31:0	<p>Vertex Position X Coordinate. This field contains the X component of the vertex’s 4D space position.</p>



DWord	Bit	Description
		Format: FLOAT32
D5	31:0	Vertex Position Y Coordinate. This field contains the Y component of the vertex's 4D space position Format: FLOAT32
D6	31:0	Vertex Position Z Coordinate. This field contains the Z component of the vertex's NDC space position Format: FLOAT32
D7	31:0	Vertex Position W Coordinate. This field contains the Z component of the vertex's 4D space position Format: FLOAT32
D8	31:0	ClipDistance 0 Value (optional). If the UserClipDistance Clip Test Enable Bitmask bit (3DSTATE_CLIP) is set, this value will be read from the URB in the Clip stage. If the value is found to be less than 0 or a NaN, the vertex's UCF<0> bit will set in the Clip unit's VertexClipTest function. If the UserClipDistance Clip Test Enable Bitmask bit is clear, this value will not be read back, and the vertex's UCF<0> bit will be zero by definition. Format: FLOAT32
D9	31:0	ClipDistance 1 Value (optional). See above
D10	31:0	ClipDistance 2 Value (optional). See above
D11	31:0	ClipDistance 3 Value (optional). See above
D12	31:0	ClipDistance 4 Value (optional). See above
D13	31:0	ClipDistance 5 Value (optional). See above
D14	31:0	ClipDistance 6 Value (optional). See above
D15	31:0	ClipDistance 7 Value (optional). See above
	31:0	(Remainder of Vertex Elements). The absolute maximum size limit on this data is specified via a maximum limit on the amount of data that can be read from a VUE (including the Vertex Header) (Vertex Entry URB Read Length has a maximum value of 63 256-bit units). Therefore the Remainder of Vertex Elements has an absolute maximum size of 62 256-bit units. Of course the actual allocated size of the VUE can and will limit the amount of data in a VUE.

1.8.2 Vertex Positions

(For the sake of brevity, the following discussion will use the term map as a shorthand for “compute screen space coordinate via perspective divide followed by viewport transform”.)

The “Position” fields of the Vertex Header are the only vertex position coordinates exposed to the 3D Pipeline. The CLIP and SF units are the only FF units which perform operations using these positions. The VUE will likely contain other position attributes for the vertex outside of the Vertex Header, though this information is not directly exposed to the FF units. For example, the Clip Space position will likely be



required in the VUE (outside of the Vertex Header) in order to perform correct and robust 3D Clipping in the CLIP thread.

In the CLIP unit, the read-back Position fields are interpreted as being in one of two coordinate systems, depending on the **CLIP_STATE.VertexPositionSpace** bit. The CLIP unit will modify its VertexClipTest function depending on the coordinate space of the incoming vertices.

- **VPOS_CLIPSPACE (Homogeneous 4D Clip-space coordinates, pre-perspective division):** The Clip Space position is defined in a homogeneous 4D coordinate space (pre-perspective divide), where the visible “view volume” is defined by the APIs. The API’s VS or GS shader program will include geometric transforms in the computation of this clip space position such that the resulting coordinate is positioned properly in relation to the view volume (i.e., it will include a “view transform” in this computation path). When this coordinate system is selected, the 3D FF pipeline will perform a perspective projection (division of x,y,z by w), perform clip-test on the resulting NDC (Normalized Device Coordinates), and eventually perform viewport mapping (in the SF unit) to yield screen-space (pixel) coordinates.
- **VPOS_SCREENSPACE (Screen Space position):** Under certain circumstances, the position in the Vertex Header will contain the screen-space (pixel) coordinates (post viewport mapping).

The SF unit does not have a state bit defining the coordinate space of the incoming vertex positions. Software must use the Viewport Mapping function of the SF unit in order to ensure that screen-space coordinates are available after that function. If screen space coordinates are passed into SF, then software will likely turn off the Viewport Mapping function.

The following subsections briefly describe the three relevant coordinate spaces.

1.8.2.1 Clip Space Position

The *clip-space* position of a vertex is defined in a homogeneous 4D coordinate space where, after perspective projection (division by W), the visible “view volume” is some canonical (3D) cuboid. Typically the X/Y extents of this cuboid are [-1,+1], while the Z extents are either [-1,+1] or [0,+1]. The API’s VS or GS shader program will include geometric transforms in the computation of this clip space position such that the resulting coordinate is positioned properly in relation to the view volume (i.e., it will include a “view transform” in this computation path).

Note that, under typical perspective projections, the clip-space W coordinate is equal to the view-space Z coordinate.

A vertex’s clip-space coordinates must be maintained in the VUE up to 3D clipping, as this clipping is performed in clip space.

- In , vertex clip-space positions must be included in the Vertex Header, so that they can be read-back (prior to Clipping) and then subjected to perspective projection (in hardware) and subsequent use by the FF pipeline.

1.8.2.2 NDC Space Position

A perspective divide operation performed on a clip-space position yields a [X,Y,Z,RHW] NDC (Normalized Device Coordinates) space position. Here “normalized” means that visible geometry is located within the [-1,+1] or [0,+1] extent view volume cuboid (see clip-space above).

- The NDC X,Y,Z coordinates are the clip-space X,Y,Z coordinates (respectively) divided by the clip-space W coordinate (or, more correctly, the clip-space X,Y,Z coordinates are multiplied by the reciprocal of the clip space W coordinate).
 - Note that the X,Y,Z coordinates may contain INFINITY or NaN values (see below).



- The NDC RHW coordinate is the reciprocal of the clip-space W coordinate and therefore, under normal perspective projections, it is the reciprocal of the view-space Z coordinate. Note that NDC space is really a 3D coordinate space, where this RHW coordinate is retained in order to perform perspective-correct interpolation, etal. Note that, under typical perspective projections.
 - Note that the RHW coordinate make contain an INFINITY or NaN value (see below).

1.8.2.3 Screen-Space Position

Screen-space coordinates are defined as:

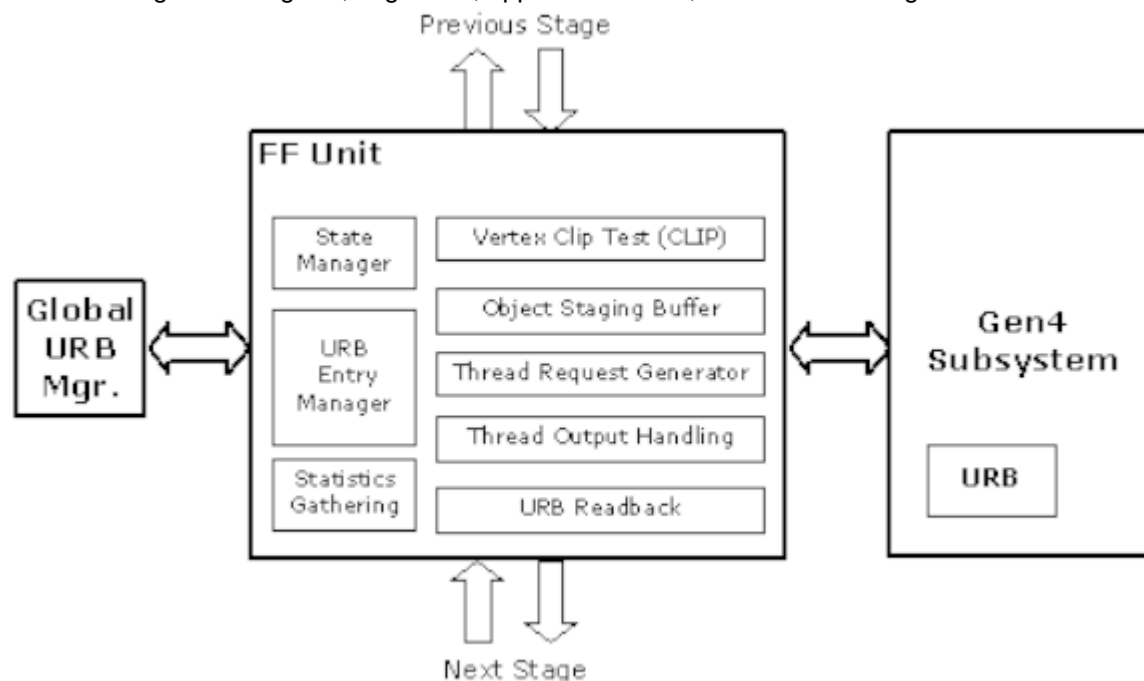
- X,Y coordinates are in absolute screen space (pixel coordinates, upper left origin). See Vertex X,Y Clamping and Quantization in the SF section for a discussion of the limitations/restrictions placed on screenspace X,Y coordinates.
- Z coordinate has been mapped into the range used for DepthTest.
- RHW coordinate is actually the reciprocal of clip-space W coordinate (typically the reciprocal of the view-space Z coordinate).

1.9 3D Pipeline Stage Overview

The fixed-function (FF) stages of the 3D pipeline share some common functionality, specifically related to the creation and management of threads. This chapter is intended to describe the behavior and programming model of these common functions, in an effort to not replicate this information for each pipeline stage. Stage-specific exceptions to the information provided here will be included in the stage-specific chapters to follow.

1.9.1 Generic 3D FF Unit Block Diagram

The following block diagram, in general, applies to the VS, GS and CLIP stages.



B6820-01

1.9.2 1Common 3D FF Unit Functions

A major role of the FF stages is in managing the threads that perform the majority of the processing on the vertex/pixel data. (In general, the amount of non-thread processing performed by the 3DPIPE stages increases towards the end of the pipeline.) In a generic sense, the key functions included are:

- Bypass Mode
- URB Entry Management
- Thread Initiation Management
- Thread Request Data Generation
 - Thread Control Information Generation
 - Thread Payload Header Generation
 - Thread Payload Data Generation
- Thread Output Handling
- URB Entry Readback
- Statistics Gathering



The following table lists the various state variables used to control the common FF functions:

State Variable	Programmed Via	Generic Functions Affected
<stage> Enable		Bypass Mode
Kernel Start Pointer		Thread Request Data Gen.
GRF Register Block Count		Thread Request Data Gen.
Single Program Flow		Thread Request Data Gen.
Thread Priority		Thread Request Data Gen.
Floating Point Mode		Thread Request Data Gen.
Exceptions Enable		Thread Request Data Gen.
Scratch Space Base Pointer		Thread Request Data Gen.
Per Thread Scratch Space		Thread Request Data Gen.
Constant URB Entry Read Length		Payload Data Gen.
Constant URB Entry Read Offset		Payload Data Gen.
Vertex URB Entry Read Length		Payload Data Gen.
Vertex URB Entry Read Offset		Payload Data Gen.
Dispatch GRF Start Register for URB Data		Payload Data Gen.
Maximum Number of Threads		Thread Resource Alloc. Scratch Space Mgt.
<stage> Fence	URB_FENCE_POINTER	URB Entry Mgt.
URB Entry Allocation Size		URB Entry Mgt.
Number of URB Entries		URB Entry Mgt.
Sampler State Pointer	: 3DSTATE_SAMPLER_STATE_POINTERS	Payload Header Gen.
<stage> Binding Table Pointer	3DSTATE_BINDING_TABLE_POINTERS	This gets routed directly to shared functions (transparent to software).
Sampler Count		Thread Request Data Gen.



State Variable	Programmed Via	Generic Functions Affected
Binding Table Entry Count		Thread Request Data Gen.
Statistics Enable		Statistics Gathering

1.9.3 Thread Initiation Management

Those FF stages that can spawn threads must have buffered the input (URB entries) available to supply a thread, and then ensure that there are sufficient resources (within the domain of the 3D pipeline) to make the thread request.

Once a FF stage determines a thread request can be submitted, (a) all input data required to initiate the thread is generated, (b) this information is submitted to the common thread dispatcher, (c) the thread dispatcher will spawn the thread as soon as an EU with sufficient GRF resources becomes available, and finally (d) the thread will start execution. With respect to concurrent threads, steps (c) and (d) can proceed out of order (i.e., a threads are not necessarily dispatched in the order that the thread requests are submitted to the thread dispatcher).

1.9.3.1 Thread Input Buffering

Each FF stage varies with regard to thread input requirements, and so this will not be discussed in this chapter other than the overview information provided in the following table:

FF Stage	Thread Input Requirements
CS	N/A (does not spawn threads)
VF	N/A (does not spawn threads)
VS	Normally, two vertices are buffered before a VS thread is spawned to shade the pair in parallel. Under some circumstances (e.g., a flush, state change, etc.) a single vertex will be shaded.
GS	All the vertices associated with an object must be buffered before a GS thread can be initiated to process the object.
WM	Threads spawned as required by the rasterization algorithm.

1.9.3.2 Thread Resource Allocation

In general, the considerations listed in the preceding section are relevant, with the following exceptions:

- CLIP, SF: Threads are not spawned.

1.9.4 Thread Request Generation

Once a FF unit determines that a thread can be requested, it must gather all the information required to submit the thread request to the Thread Dispatcher. This information is divided into several categories, as listed below and subsequently described in detail.

- **Thread Payload Header:** This is the first portion of the thread payload passed in the GRF, starting at GRF R0. This is information passed directly from the FF unit. It precedes the Thread Payload Input URB Data.



- **Thread Payload Input URB Data:** This is the second portion of the thread payload. It is read from the URB using entry handles supplied by the FF unit.

1.9.4.1 Thread Control Information

The following table describes the various state variables that a FF unit uses to provide information to the Thread Dispatcher and which affect the thread execution environment. Note that this information is not directly passed to the thread in the thread payload (though some fields may be subsequently accessed by the thread via architectural registers).

State Variables Included in Thread Control Information

State Variable	Usage	FFs
Kernel Start Pointer	This field, together with the General State Pointer , specifies the starting location (1 st core instruction) of the kernel program run by threads spawned by this FF unit. It is specified as a 64-byte-granular offset from the General State Pointer .	All FFs spawning threads
GRF Register Block Count	Specifies, in 16-register blocks, how many GRF registers are required to run the kernel. The Thread Dispatcher will only seek candidate EUs that have a sufficient number of GRF register blocks available. Upon selecting a target EU, the Thread Dispatcher will generate a logical-to-physical GRF mapping and provide this to the target EU.	All FFs spawning threads
Single Program Flow (SPF)	Specifies whether the kernel program has a single program flow (SIMD _n x _m with $m = 1$) or multiple program flows (SIMD _n x _m with $m > 1$). See CR0 description in <i>ISA Execution Environment</i> .	All FFs spawning threads
Thread Priority	The Thread Dispatcher will give priority to those thread requests with Thread Priority of HIGH_PRIORITY over those marked as LOW_PRIORITY. Within these two classes of thread requests, the Thread Dispatcher applies a priority order (e.g., round-robin --- though this algorithm is considered a device implementation-dependent detail).	All FFs spawning threads
Floating Point Mode	This determines the initial value of the Floating Point Mode bit of the EU's CR0 architectural register that controls floating point behavior in the EU core. (See ISA.)	All FFs spawning threads
Exceptions Enable	This bitmask controls the exception handing logic in the EU. (See ISA.)	All FFs spawning threads
Sampler Count	This is a <u>hint</u> which specifies how many indirect SAMPLER_STATE structures should be prefetched concurrent with thread initiation. It is recommended that software program this field to equal the number of samplers, though there may be some minor performance impact if this number gets large. This value should not exceed the number of samplers accessed by the thread as there would be no performance advantage. Note that the data prefetch is treated as any other memory fetch (with respect to page faults, etc.).	All stages supporting sampling (VS, GS, WM)
Binding Table Entry Count	This is a <u>hint</u> which specifies how many indirect BINDING_TABLE_STATE structures should be prefetched concurrent with thread initiation. (The notes included in Sampler Count (above) also apply to this field).	All FFs spawning threads

1.9.4.2 Thread Payload Generation

FF units are responsible for generating a thread *payload* – the data pre-loaded into the target EU's GRF registers (starting at R0) that serves as the primary direct input to a thread's kernel. The general format of



these payloads follow a similar structure, though the exact payload size/content/layout is unique to each stage. This subsection describes the common aspects – refer to the specific stage’s chapters for details on any differences.

The payload data is divided into two main sections: the *payload header* followed by the *payload URB data*. The payload header contains information passed directly from the FF unit, while the payload URB data is obtained from URB locations specified by the FF unit.

NOTE: The first 256 bits of the thread payload (the initial contents of R0, aka “the R0 header”) is specially formatted to closely match (and in some cases exactly match) the first 256 bits of thread-generated *messages* (i.e., the message header) accepted by shared functions. In fact, the send instruction supports having a copy of a GR’s contents (such as R0) used as the message header. Software must take this intention into account (i.e., “don’t muck with R0 unless you know what you’re doing”). This is especially important given the fact that several fields in the R0 header are considered opaque to SW, where use or modification of their contents might lead to UNDEFINED results.

The payload header is further (loosely) divided into a leading *fixed payload header* section and a trailing, variable-sized *extended payload header* section. In general the size, content and layout of both payload header sections are FF-specific, though many of the fixed payload header fields are common amongst the FF stages. The extended header is used by the FF unit to pass additional information specific to that FF unit. The extended header is defined to start after the fixed payload header and end at the offset defined by **Dispatch GRF Start Register for URB Data**. Software can cause use the **Dispatch GRF Start Register for URB Data** field to insert padding into the extended header in order to maintain a fixed offset for the start of the URB data.

1.9.4.2.1 Fixed Payload Header

The payload header is used to pass FF pipeline information required as thread input data. This information is a mixture of SW-provided state information (state table pointers, etc.), primitive information received by the FF unit from the FF pipeline, and parameters generated/computed by the FF unit. most of the fields of the fixed header are common between the FF stages. These non-FF-specific fields are described in *Fixed Payload Header*. Note that a particular stage’s header may not contain all these fields, so they are not “common” in the strictest sense.

Fixed Payload Header Fields (non-FF-specific)

Fixed Payload Header Field (non-FF-specific)	Description	FFs
FF Unit ID	Function ID of the FF unit. This value identifies the FF unit within the subsystem. The FF unit will use this field (when transmitted in a Message Header to the URB Function) to detect messages emanating from its spawned threads.	All FFs spawning threads
Snapshot Flag		All FFs spawning threads
Thread ID		All FFs spawning threads
Scratch Space Pointer	This is the starting location of the thread’s allocated scratch space, specified as an offset from the General State Base Address . Note that scratch space is allocated by the FF unit on a per-thread basis, based on the Scratch Space Base Pointer and Per-Thread Scratch Space Size state variables. FF units will assign a thread an arbitrarily-positioned region within this	All FFs spawning threads



Fixed Payload Header Field (non-FF-specific)	Description	FFs
	space. The scratch space for multiple (API-visible) entities (vertices, pixels) will be interleaved within the thread's scratch space.	
Dispatch ID	<p>This field identifies this thread within the outstanding threads spawned by the FF unit. This field does <u>not</u> uniquely identify the thread over any significant period of time.</p> <p><i>Implementation Note:</i> This field is effectively an "active thread index". It is used on a thread's URB allocation request to identify which thread's handle pool is to source the allocation. It is used upon thread termination to free up the thread's scratch space allocation.</p>	All FFs spawning threads
Binding Table Pointer	<p>This field, together with the Surface State Base Pointer, specifies the starting location of the Binding Table used by threads spawned by the FF unit. It is specified as a 64-byte-granular offset from the Surface State Base Pointer.</p> <p>See <i>Shared Functions</i> for a description of a Binding Table.</p>	All FFs spawning threads
Sampler State Pointer	<p>This field, together with the General State Base Pointer, specifies the starting location of the Sampler State Table used by threads spawned by the FF unit. It is specified as a 64-byte-granular offset from the General State Base Pointer.</p> <p>See <i>Shared Functions</i> for a description of a Sampler State Table.</p>	All FFs spawning threads which sample (VS, GS, WM)
Per Thread Scratch Space	<p>This field specifies the amount of scratch space allocated to each thread spawned by the FF unit.</p> <p>The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space.</p>	All FFs spawning threads
Handle ID <n>	<p>This ID is assigned by the FF unit and links the thread to a specific entry within the FF unit. The FF unit will use this information upon detecting a URB_WRITE message issued by the thread.</p> <p>Threads spawned by the GS, CLIP, and SF units are provided with a single Handle ID / URB Return Handle pair. Threads spawned by the VS are provided with one or two pairs (depending on how many vertices are to be processed). Threads spawned by the WM do not write to URB entries, and therefore this info is not supplied.</p>	VS,GS,CLIP,SF
URB Return Handle <n>	<p>This is an initial destination URB handle passed to the thread. If the thread does output URB entries, this identifies the destination URB entry.</p> <p>Threads spawned by the GS, CLIP, and SF units are provided with a single Handle ID / URB Return Handle pair. Threads</p>	VS,GS,CLIP,SF



Fixed Payload Header Field (non-FF-specific)	Description	FFs
	spawned by the VS are provided with one or two pairs (depending on how many vertices are to be processed). Threads spawned by the WM do not write to URB entries, and therefore this info is not supplied.	
Primitive Topology Type	<p>As part of processing an incoming primitive, a FF unit is often required to spawn a number of threads (e.g., for each individual triangle in a TRIANGLE_STRIP). This field identifies the type of primitive which is being processed by the FF unit, and which has lead to the spawning of the thread.</p> <p>Kernels written to process different types of objects can use this value to direct that processing. E.g., when a CLIP kernel is to provide clipping for all the various primitive types, the kernel would need to examine the Primitive Topology Type to distinguish between point, lines, and triangle clipping requests.</p> <p>NOTE: In general, this field is identical to the Primitive Topology Type associated with the primitive vertices as received by the FF unit. Refer to the individual FF unit chapters for cases where the FF unit modifies the value before passing it to the thread. (E.g., certain units perform toggling of TRIANGLESTRIP and TRIANGLESTRIP_REV).</p>	GS, CLIP, SF, WM

1.9.4.2.2 Extended Payload Header

The extended header is of variable-size, where inclusion of a field is determined by FF unit state programming.

In order to permit the use of common kernels (thus reducing the number of kernels required), the **Dispatch GRF Start Register for URB Data** state variable is supported in all FF stages. This SV is used to place the payload URB data at a specific starting GRF register, irrespective of the size of the extended header. A kernel can therefore reference the payload URB data at fixed GRF locations, while conditionally referencing extended payload header information.

1.9.4.2.3 Payload URB Data

In each thread payload, following the payload header, is some amount of URB-sourced data required as input to the thread. This data is divided into an optional *Constant URB Entry* (CURBE), following either by a Primitive URB Entry (WM) or a number of Vertex URB Entries (VS, GS, CLIP, SF). A FF unit only knows the location of this data in the URB, and is never exposed to the contents. For each URB entry, the FF unit will supply a sequence of handles, read offsets and read lengths to the subsystem. The subsystem will read the appropriate 256-bit locations of the URB, optionally perform swizzling (VS only), and write the results into sequential GRF registers (starting at **Dispatch GRF Start Register for URB Data**).



State Variables Controlling Payload URB Data

State Variable	Usage	FFs
Dispatch GRF Start Register for URB Data	This SV identifies the starting GRF register receiving payload URB data. Software is responsible for ensuring that URB data does not overwrite the Fixed or Extended Header portions of the payload.	FFs spawning threads
Vertex URB Entry Read Offset	This SV specifies the starting offset within VUEs from which vertex data is to be read and supplied in this stage's payloads. It is specified as a 256-bit offset into any and all VUEs passed in the payload. This SV can be used to skip over leading data in VUEs that is not required by the stage's threads (e.g., skipping over the Vertex Header data at the SF stage, as that information is not required for setup calculations). Skipping over irrelevant data can only help to improve performance. Specifying a vertex data source extending beyond the end of a vertex entry is UNDEFINED.	VS, GS, t
Vertex URB Entry Read Length	This SV determines the amount of vertex data (starting at Vertex URB Entry Read Offset) to be read from each VUEs and passed into the payload URB data. It is specified in 256-bit units. A zero value is INVALID (at very least one 256-bit unit must be read). Specifying a vertex data source extending beyond the end of a VUE is UNDEFINED.	

Programming Restrictions: (others may already been mentioned)

- The maximum size payload for any thread is limited by the number of GRF registers available to the thread, as determined by $\min(128, 16 * \text{GRF Register Block Count})$. Software is responsible for ensuring this maximum size is not exceeded, taking into account:
 - The size of the Fixed and Extended Payload Header associated with the FF unit.
 - The **Dispatch GRF Start Register for URB Data** SV.
 - The amount of CURBE data included (via **Constant URB Entry Read Length**)
 - The number of VUEs included (as a function of FF unit, it's state programming, and incoming primitive types)
 - The amount of VUE data included for each vertex (via **Vertex URB Entry Read Length**)
 - (For WM-spawned PS threads) The amount of Primitive URB Entry data.
- For any type of URB Entry reads:
 - Specifying a source region (via Read Offset, Read Length) that goes past the end of the URB Entry allocation is illegal.
 - The allocated size of Vertex/Primitive URB Entries is determined by the **URB Entry Allocation Size** value provided in the pipeline state descriptor of the FF unit owning the VUE/PUE.
 - The allocated size of CURBE entries is determined by the **URB Entry Allocation Size** value provided in the CS_URB_STATE command.



1.9.5 Thread Output Handling

Those FF units spawning threads are responsible for monitoring and responding to certain events generated by their spawned threads. Such events are indirectly detected by these FF units monitoring messages sent from threads to the URB Shared Function. By snooping the Message Bus Sideband and Header information, a FF can detect when a particular spawned thread sends a message to the URB function. A subset of this information is then captured and acted upon. Refer to the *URB* chapter for more details (including a table of valid/invalid combinations of the **Complete**, **Used**, **Allocate**, and **EOT** bits)

The following subsections describe functions that FF units perform as part of Thread Output Handling.

1.9.5.1 VUE Allocation (GS)

The following description is applicable only to the GS stage.

The threads are not passed an initial handle. Instead, they request a first handle (if any) via the URB shared function's FF_SYNC message (see Shared Functions). If additional handles are required, the URB_WRITE allocate mechanism (mentioned above) is used.

1.9.5.2 Thread Termination

All threads must explicitly terminate by executing a SEND instruction with the EOT bit set. (See EU chapters). When a thread spawned by a 3D FF unit terminates, the spawning FF unit detects this termination as a part of Thread Management. This allows the FF units to manage the number of concurrent threads it has spawned and also manage the resources (e.g., scratch space) allocated to those threads.

1.9.6 VUE Readback

Starting with the CLIP stage, the 3D pipeline requires vertex information in addition to the VUE handle. For example, the CLIP unit's VertexClipTest function needs the vertex position, as does the SF unit's functions. This information is obtained by the 3D pipeline reading a portion of each vertex's VUE data directly from the URB. This readback (effectively) occurs immediately before the CLIP VertexClipTest function, and immediately after a CLIP thread completes the output of a destination VUE.

The Vertex Header (first 256 bits) of the VUE data is read back. (See the previous *VUE Formats* subsection (above) for details on the content and format of the Vertex Header.) : Additional Clip/Cull data (located immediately past the Vertex Header) may be read prior to clipping.

This readback occurs automatically and is not under software control. The only software implication is that the Vertex Header must be valid at the readback points, and therefore must have been previously loaded or written by a thread.

1.9.7 Statistics Gathering

The table below describes how supports the required API statistics counters.

DX Statistic	HW Support
IAVertices = # of vertices IA generated. May or may not include (a) vertices in partial primitives, (b) unused adjacent-only vertices. Not affected by vertex caching.	VF maintains IA_VERTICES_COUNT . Will include unused adjacent-only vertices. Will not include vertices in partial primitives.
IAPrimitives = # of primitives (objects) IA generated. May or	VF maintains IA_PRIMITIVES_COUNT .



DX Statistic	HW Support
may not include partial primitives.	Will not include partial primitives. Will not count patch topologies that do not match what the HS or GS expects as input, if enabled (i.e., mismatching patch topologies are discarded by VF).
VSInvocations = # of times VS is executed. May be affected by vertex caching. May or may not include (a) shared vertices in non-indexed strips, (b) vertices in partial primitives, (c) unused adjacent-only vertices.	VS maintains VS_INVOCATION_COUNT . Impacted by vertex caching. Will not include vertices in partial primitives. <u>Will</u> include unused adjacent-only vertices. Will not include shared vertices in non-indexed strips, unless pre-empted. Increments even if VS Function Enable is DISABLED.
HSInvocations = # of patches executed by HS.	HS maintains HS_INVOCATION_COUNT . This gets incremented by 1 for each patch whenever HS is enabled.
DSInvocations = # of times DS is executed to shade a domain point. Allows HW to shade identical domain points multiple times, with the exception of point outputs where only unique domain points can be generated.	DS maintains DS_INVOCATION_COUNT . This is incremented for each domain point passed to a DS thread.
GSInvocations = # of times GS is executed. Obviously does not include partial primitives. May be incremented when StreamOut enabled, even if NULL_GS.	GS maintains GS_INVOCATION_COUNT , incrementing it by GSInvocations Increment Value for each dispatched instance. Will not be incremented if NULL_GS.
GSPrimitives = # of primitives GS generated. Does not include primitives passing through a disabled GS stage. May or may not include partial primitives output by GS.	GS maintains GS_PRIMITIVE_COUNT . GS unit will increment this as it parses the GS thread output. Will <u>not</u> include partial primitives output by GS threads.
NumPrimitivesWritten[<stream#>] = # of complete primitives written to the stream's SO buffer, subject to buffer overflow.	SOL maintains SO_NUM_PRIMS_WRITTEN[0-3] .
PrimitiveStorageNeeded[<stream#>] = # of complete primitives which would have been written to the stream's SO buffer ignoring any overflow.	SOL maintains SO_PRIM_STORAGE_NEEDED[0-3] .
ClInvocations = # of primitives <u>entering</u> rasterization (which starts with the clipper) and isn't affected by any actual clipping. Does not increment when rasterization is disabled (e.g., when StreamOut is the last enabled stage). May or may not include partial primitives.	CL OSB maintains CL_INVOCATION_COUNT . Will not include partial primitives. Note that the SOL (regardless of SO enabled) will discard primitives if rendering is disabled, so these primitives will not reach the CL unit.
CPrimitives = # of primitives output from clipper. I.e., doesn't increment if TrivReject or dropped due to NaNs, increments by 1 if TrivAccept, or increments by number of primitives generated if MustClip. Does not increment when rasterization is disabled. May or may not include partial primitives. Accommodates infinite or no guardband.	SF OSB maintains CL_PRIMITIVES_COUNT . Will not include partial primitives.



DX Statistic	HW Support
PSInvocations = # of times PS is executed, including unlit “helper pixels” within a subspan that need to go through the PS shader to provide 2x2 gradients. Accommodates early depth/stencil. Does not increment if NULL PS. Multisampling: counts pixels shaded if PERPIXEL or samples shaded if PERSAMPLE.	WIZ maintains PS_INVOCATION_COUNT .
Occlusion = # of “visible” multisamples which passed both depth and stencil testing. Doesn’t include PS-discarded pixels or oMask/AlphaToCoverage-killed samples. Both (a) a disabled test (depth or stencil) and (b) no bound RT or Depth/Stencil buffer conditions count as always passing.	WIZ & PBE maintain PS_DEPTH_COUNT .

1.10 Synchronization of the 3D Pipeline

Two types of synchronizations are supported for the 3D pipe: top of the pipe and end of the pipe. Top of the pipe synchronization really enforces the read-only cache invalidation. This synchronization guarantees that primitives rendered after such synchronization event fetches the latest read-only data from memory. End of the pipe synchronization enforces that the read and/or read-write buffers do not have outstanding hardware accesses. These are used to implement read and write fences as well as to write out certain statistics deterministically with respect to progress of primitives through the pipeline (and without requiring the pipeline to be flushed.) The PIPE_CONTROL command (see details below) is used to perform all of above synchronizations.

1.10.1 Top-of-Pipe Synchronization

Top-of-pipe synchronization refers to SW actions to prepare HW for new state-binding at the beginning of the rendering sequence in a given context. HW may have residual states cached in the state-caches and read-only surfaces in various caches. With new rendering sequence, read-only surfaces may go through change in the binding. Hence read-only invalidation is required before such new rendering sequence. Read-only cache invalidation is top-of-pipe synchronization. Upon parsing this specific pipe-control command, HW invalidates all caches in GT domain that have read-only surfaces but does not guarantee invalidation beyond GT caches (i.e. LLC). Further, HW does not guarantee that all prior accesses to those read-only surfaces have completed. Therefore SW must guarantee that there are no pending accesses to those read-only surfaces before initializing the top-of-pipe synchronization. PIPE_CONTROL command described below allows for invalidating individual read-only stream type. It is recommended that driver invalidates only the required caches on the need basis so that cache warm-up overhead can be reduced.

1.10.2 End-of-Pipe Synchronization

The driver can use end-of-pipe synchronization to know that rendering is complete (although not necessarily in memory) so that it can de-allocate in-memory rendering state, read-only surfaces, instructions, and constant buffers. An end-of-pipe synchronization point is also sufficient to guarantee that all pending depth tests have completed so that the visible pixel count is complete prior to storing it to memory. End-of-pipe completion is sufficient (although not necessary) to guarantee that read events are complete (a “read fence” completion). Read events are still pending if work in the pipeline requires any type of read except a render target read (blend) to complete.

Write synchronization is a special case of end-of-pipe synchronization that requires that the render cache and/or depth related caches are flushed to memory, where the data will become globally visible. This type



of synchronization is required prior to SW (CPU) actually reading the result data from memory, or initiating an operation that will use as a read surface (such as a texture surface) a previous render target and/or depth/stencil buffer.

1.10.3 Synchronization Actions

In order for the driver to act based on a synchronization point (usually the whole point), the reaching of the synchronization point must be communicated to the driver. This section describes the actions that may be taken upon completion of a synchronization point which can achieve this communication.

1.10.3.1 Writing a Value to Memory

The most common action to perform upon reaching a synchronization point is to write a value out to memory. An immediate value (included with the synchronization command) may be written. In lieu of an immediate value, the 64-bit value of the PS_DEPTH_COUNT (visible pixel count) or TIMESTAMP register may be written out to memory. The captured value will be the value at the moment all primitives parsed prior to the synchronization commands have been completely rendered, and optionally after all said primitives have been pushed to memory. It is not required that a value be written to memory by the synchronization command.

Visible pixel or TIMESTAMP information is only useful as a delta between 2 values, because these counters are free-running and are not to be reset except at initialization. To obtain the delta, two PIPE_CONTROL commands should be initiated with the command sequence to be measured between them. The resulting pair of values in memory can then be subtracted to obtain a meaningful statistic about the command sequence.

1.10.3.2 PS_DEPTH_COUNT

If the selected operation is to write the visible pixel count (PS_DEPTH_COUNT register), the synchronization command should include the **Depth Stall Enable** parameter. There is more than one point at which the global visible pixel count can be affected by the pipeline; once the synchronization command reaches the first point at which the count can be affected, any primitives following it are stalled at that point in the pipeline. This prevents the subsequent primitives from affecting the visible pixel count until all primitives preceding the synchronization point reach the end of the pipeline, the visible pixel count is accurate and the synchronization is completed. This stall has a minor effect on performance and should only be used in order to obtain accurate “visible pixel” counts for a sequence of primitives.

The PS_DEPTH_COUNT count can be used to implement an (API/DDI) “Occlusion Query” function.

1.10.3.3 Generating an Interrupt

The synchronization command may indicate that a “Sync Completion” interrupt is to be generated (if enabled by the MI Interrupt Control Registers – see *Memory Interface Registers*) once the rendering of all prior primitives is complete. Again, the completion of rendering can be considered to be when the internal render cache has been updated, or when the cache contents are visible in memory, as selected by the command options.

1.10.3.4 Invalidating of Caches

If software wishes to use the notification that a synchronization point has been reached in order to reuse referenced structures (surfaces, state, or instructions), it is not sufficient just to make sure rendering is complete. If additional primitives are initiated after new data is laid over the top of old in memory following a synchronization point, it is possible that stale cached data will be referenced for the subsequent



rendering operation. In order to avoid this, the PIPE_CONTROL command must be used. (See *PIPE_CONTROL Command* description below).

1.10.4 PIPE_CONTROL Command

The PIPE_CONTROL command is used to effect the synchronization described above. Parsing of a PIPE_CONTROL command stalls 3D pipe only if the stall enable bit is set. Commands after PIPE_CONTROL will continue to be parsed and processed in the 3D pipeline. This may include additional PIPE_CONTROL commands. The implementation does enforce a practical upper limit (8) on the number of PIPE_CONTROL commands that may be outstanding at once. Parsing of a PIPE_CONTROL command that causes this limit to be reached will stall the parsing of new commands until the first of the outstanding PIPE_CONTROL commands reaches the end of the pipe and retires.

Note that although PIPE_CONTROL is intended for use with the 3D pipe, it is legal to issue PIPE_CONTROL when the Media pipe is selected. In this case PIPE_CONTROL will stall at the top of the pipe until the Media FFs finish processing commands parsed before PIPE_CONTROL. Post-synchronization operations, flushing of caches and interrupts will then occur if enabled via PIPE_CONTROL parameters. Due to this stalling behavior, only one PIPE_CONTROL command can be outstanding at a time on the Media pipe.

For the invalidate operation of the pipe control, the following pointers are affected. The invalidate operation affects the restore of these packets. If the pipe control invalidate operation is completed before the context save, the indirect pointers will not be restored from memory.

1. Pipeline State Pointer
2. Media State Pointer
3. Constant Buffer Packet

It is up to software to program the appropriate read-only cache invalidation such as the sampler and constant read caches or the instruction and state caches. Once notification is observed, new data may then be loaded (potentially “on top of” the old data) without fear of stale cache data being referenced for subsequent rendering.

If software wishes to access the rendered data in memory (for analysis by the application or to copy it to a new location to use as a texture, for examples), it must also ensure that the write cache (render cache) is flushed after the synchronization point is reached so that memory will be updated. This can be accomplished by setting the **Write Cache Flush Enable** bit. Note that the **Depth Stall Enable** bit must be clear in order for the flush of the render cache to occur. **Depth Stall Enable** is intended only for accurate reporting of the PS_DEPTH counter; the render cache cannot be flushed nor can the read caches be invalidated (except for the instruction/state cache) in conjunction with this operation.

1.10.4.1 PIPE_CONTROL

Hardware can support up to 8 pending PIPE_CONTROL flushes

2 Store Data Commands (such as MI_STORE_DATA_IMM or MI_STORE_DATA_INDEX)
PIPE_CONTROL w/ stall (20) and TLB inv bit (18) set

Ring/Batch Contents - ILLEGAL
3DPRIMITIVE
np-state
pipelined (bit 20 = '0') PIPE_CONTROL



Ring/Batch Contents - ILLEGAL
np-state
3DPRIMITIVE

Ring/Batch Contents - LEGAL
3DPRIMITIVE
np-state
3DPRIMITIVE
pipelined (bit 20 = '0') PIPE_CONTROL
np-state
3DPRIMITIVE

- Pipe_control with CS-stall bit set must be issued before a pipe-control command that has the State Cache Invalidate bit set.

Caches Invalidated/Flushed by PIPE_CONTROL Bit Settings

The table below explains all the different flush/invalidation scenarios.

Write cache flush	Notification Enabled	non-VF RO Cache Invalidate	VF RO Cache Invalidate	Marker Sent	pipeline marker enable	Completion Requested	Top of pipe invalidate pulse from CS
0	0	0	0	N/A	N/A	N/A	N/A
0	0	0	1	Yes	No	N/A	No
0	0	1	0	No	N/A	N/A	Yes
0	0	1	1	Yes	No	No	Yes
X	1	0	X	Yes	Yes	Yes	No
X	1	1	X	Yes	Yes	Yes	Yes
1	X	0	X	Yes	Yes	Yes	No
1	X	1	X	Yes	Yes	Yes	Yes

PIPE_CONTROL	
Length Bias: 2	
The PIPE_CONTROL command is used to effect the synchronization described above.	
DWord	Bit Description
0	31:29 Command Type
	Default Value: 3h GFXPIPE
	Format: OpCode
	28:27 Command SubType
	Default Value: 3h GFXPIPE_3D
	Format: OpCode
	26:24 3D Command Opcode
	Default Value: 2h PIPE_CONTROL
	Format: OpCode
	23:16 3D Command Sub Opcode
	Default Value: 0h PIPE_CONTROL
	Format: OpCode



PIPE_CONTROL				
	15:8	Reserved		
		Project:	All	
		Format:	MBZ	
	7:0	DWord Length		
		Default Value:	3h DWORD_COUNT_n	
		Format:	=n	
1	31:28	Reserved		
		Project:	All	
			Format:	MBZ
	27	Reserved		
		Format:	MBZ	
	26	Reserved		
		Project:	All	
			Format:	MBZ
	25	Reserved		
		Format:	MBZ	
	24	Destination Address Type		
		Defines address space of Destination Address		
		Value	Name	Description
		0h	PPGTT	Use PPGTT address space for DW write
		1h	GGTT	Use GGTT address space for DW write
Programming Notes				
Ignored if "No Write" is selected in Operation.				
23	LRI Post-Sync Operation			
	Value	Name	Description	
	0h	No LRI Operation	No LRI operation occurs as a result of this instruction. The Post-Sync Operation field is valid and may be used to specify an operation.	
	1h	MMIO Write Immediate Data	Write the DWord contained in Immediate Data Low (DW3) to the MMIO offset specified in the Address field.	
	Programming Notes			
	This bit causes a post sync operation with an LRI (Load Register Immediate) operation. If this bit is set then the Post-Sync Operation field must be cleared.			
21	Store Data Index			
	Project:	All		
	Format:	U1		
	Ring Buffer Mode Scheduling: This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is actually an index into the global hardware status page. This bit only applies to the Global HW status page. If this field is 1, the Destination Address Type in this command must be set			



PIPE_CONTROL				
		<p>to 1 (GGTT).</p> <p>Execlist Mode Scheduling: This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is index into the global hardware status page when destination address type in the command is set to 1 (GGTT). The store data address is index into the per-process hardware status page when destination address type in the command is set to 0 (PPGTT).</p>		
20	CS Stall			
	Project:	All		
	Format:	U1		
	<p>If ENABLED, the sync operation will not occur until all previous flush operations pending a completion of those previous flushes will complete, including the flush produced from this command. This enables the command to act similar to the legacy MI_FLUSH command.</p>			
	Programming Notes		Project	
<p>One of the following must also be set:</p> <p>Render Target Cache Flush Enable ([12] of DW1)</p> <p>Depth Cache Flush Enable ([0] of DW1)</p> <p>Stall at Pixel Scoreboard ([1] of DW1)</p> <p>Depth Stall ([13] of DW1)</p> <p>Post-Sync Operation ([13] of DW1)</p>				
19	Global Snapshot Count Reset			
	Project:	All		
	Format:	U1		
	Value	Name	Description	Project
	0h	Don't Reset	Do not reset the snapshot counts or Statistics Counters.	All
1h	Reset	Reset the snapshot count for all the units and reset the Statistics Counters except as noted above.	All	
Programming Notes		Project		
<p>TIMESTAMP is not reset by PIPE_CONTROL with this bit set.</p> <p>When Post Sync Operation is set to "Write PS Depth Count" along with Global Snapshot Count Reset, PS Depth Count is Reported first before resetting the value.</p>				
18	TLB Invalidate			
	Project:	All		
	Format:	U1		
<p>If ENABLED, all TLBs will be invalidated once the flush operation is complete. Note that if the flush TLB invalidation mode is clear, a TLB invalidate will occur irrespective of this bit setting</p>				



PIPE_CONTROL			
		If ENABLED, PIPE_CONTROL command will flush the in flight data written out by render engine to Global Observation point on flush done. Also Requires stall bit ([20] of DW1) set.	
		Programming Notes	Project
		If ENABLED, all TLBs will be invalidated once the flush operation is complete. Note that if the flush TLB invalidation mode is clear, a TLB invalidate will occur irrespective of this bit setting.	
17	Reserved		
	Format:	MBZ	
16	Generic Media State Clear		
	Format:	Disable	
	If set, all generic media state context information will not be included with the next context save, assuming no new state is initiated after the flush. If clear, the generic media state context save state will not be affected. An MI_FLUSH with this bit set should be issued once all the Media Objects that will be processed by a given persistent root thread have been issued or when an MI_SET_CONTEXT switching from a generic media context to a 3D context completes. When using MI_SET_CONTEXT, once state is programmed, it will be saved and restarted as part of any context each time that context is saved/restored until an MI_FLUSH with this bit set is issued in that context.		
15:14	Post-Sync Operation		
	Project:	All	
		Description	Project
	This field specifies an optional action to be taken upon completion of the synchronization operation.		
	This field must be cleared if the LRI Post-Sync Operation bit is set.		
	Value	Name	Description
	0h	No Write	No write occurs as a result of this instruction. This can be used to implement a “trap” operation, etc.
	1h	Write Immediate Data	Write the QWord containing Immediate Data Low, High DWs to the Destination Address
	2h	Write PS Depth Count	Write the 64-bit PS_DEPTH_COUNT register to the Destination Address
	3h	Write Timestamp	Write the 64-bit TIMESTAMP register to the Destination Address
		Programming Notes	
	If executed in non-secure batch buffer, the address given will be in a PPGTT address space. If in a secure ring or batch, address given will be in GGTT space		
13	Depth Stall Enable		
	Project:	All	
	Format:	Enable	
	This bit should be set when obtaining a “visible pixel” count to preclude the possible inclusion in the PS_DEPTH_COUNT value written to memory of some fraction of pixels from objects initiated after the PIPE_CONTROL command.		



PIPE_CONTROL				
	Value	Name	Description	Project
	0h	Disable	3D pipeline will not stall subsequent primitives at the Depth Test stage.	All
	1h	Enable	3D pipeline will stall any subsequent primitives at the Depth Test stage until the Sync and Post-Sync operations complete.	All
Programming Notes				
This bit should be DISABLED for operations other than writing PS_DEPTH_COUNT.				
This bit will have no effect (besides preventing write cache flush) if set in a PIPE_CONTROL command issued to the Media pipe.				
12	Render Target Cache Flush Enable			
	Project:		All	
	Format:		Enable	
Setting this bit will force Render Cache to be flushed to memory prior to this synchronization point completing. This bit should be set for all write fence sync operations to assure that results from operations initiated prior to this command are visible in memory once software observes this synchronization.				
	Value	Name	Description	Project
	0h	Disable Flush	Render Target Cache is NOT flushed.	All
	1h	Enable Flush	Render Target Cache is flushed.	All
Programming Notes				
This bit should be DISABLED for End-of-pipe (Read) fences, PS_DEPTH_COUNT or TIMESTAMP queries.				
This bit must not be set when Depth Stall Enable bit is set in this packet.				
11	Instruction Cache Invalidate Enable			
	Project:		All	
	Format:		Enable	
Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the L1 and L2 at the top of the pipe i.e. at the parsing time.				
10	Texture Cache Invalidation Enable			
	Project:		All	
	Format:		Enable	
Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the texture caches at the top of the pipe i.e. at the parsing time.				
9	Indirect State Pointers Disable			
	Project:		All	
	Format:		Enable	
Description				Project
At the completion of the post-sync operation associated with this pipe control packet, the indirect state pointers in the hardware are considered invalid; the indirect pointers are not saved in the context. If any new indirect state commands are executed in the command stream while the pipe control is pending, the new indirect state commands are preserved.				
Using Invalidate State Pointer (ISP) only inhibits context restoring of Push Constant				

PIPE_CONTROL				
		(3DSTATE_CONSTANT_*) commands. Push Constant commands are only considered as Indirect State Pointers. Once ISP is issued in a context, SW must initialize by programming push constant commands for all the shaders (at least to zero length) before attempting any rendering operation for the same context.		
8	Notify Enable			
	Project:	All		
	Format:	Enable		
	If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in Memory Interface Registers for details.			
7	PIPE_CONTROL Flush Enable			
	Format:	Enable		
	If ENABLED, the PIPE_CONTROL command will wait until all previous writes of immediate data from post sync circles are complete before executing the next command.			
5	DC Flush Enable			
	Format:	Enable		
	Setting this bit enables flushing of the L3\$ portions that caches DC writes.			
4	VF(address based) Cache Invalidation Enable			
	Project:	All		
	Format:	Enable		
	Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of VF address based cache at the top of the pipe i.e. at the parsing time.			
3	Constant Cache Invalidation Enable			
	Project:	All		
	Format:	Enable		
	Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the constant cache at the top of the pipe i.e. at the parsing time.			
2	State Cache Invalidation Enable			
	Project:	All		
	Format:	Enable		
	Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the L1 and L2 state caches at the top of the pipe i.e. at the parsing time.			
1	Stall At Pixel Scoreboard			
	Project:	All		
	Format:	Enable		
	Defines the behavior of PIPE_CONTROL command at the pixel scoreboard.			
	Value	Name	Description	Project
	0h	Disable	Stall at the pixel scoreboard is disabled.	All
	1h	Enable	Stall at the pixel scoreboard is enabled.	All



PIPE_CONTROL			
		Programming Notes	
		This bit should be DISABLED for End-of-pipe (Read) fences, PS_DEPTH_COUNT or TIMESTAMP queries. This bit is ignored if Depth Stall Enable is set. Further the render cache is not flushed even if Write Cache Flush Enable bit is set.	
	0	Depth Cache Flush Enable	
		Project:	All
		Format:	Enable
		Setting this bit enables flushing (i.e. writing back the dirty lines to memory and invalidating the tags) of depth related caches. This bit applies to HiZ cache, Stencil cache and depth cache.	
		Value	Name
		Description	Project
		0h	Flush Disabled
		Depth relates caches (HiZ, Stencil and Depth) are NOT flushed.	All
		1h	Flush Enabled
		Depth relates caches (HiZ, Stencil and Depth) are flushed.	All
		Programming Notes	
		Ideally depth caches need to be flushed only when depth is required to be coherent in memory for later use as a texture, source or honoring CPU lock. This bit should be DISABLED for End-of-pipe (Read) fences, PS_DEPTH_COUNT or TIMESTAMP queries.	
		This bit must not be set when Depth Stall Enable bit is set in this packet.	
2	31:2	Address	
		Format:	GraphicsAddress[31:2]U32
		If Post Sync Operation is set to 1h (LRI Post-Sync Operation must be clear): Bits 31:3 specify the QW address of where the Immediate Data following this DW in the packet to be stored. Bit 2 MBZ Ignored if "No Write" is the selected in Post-Sync Operation If LRI Post-Sync Operation is set: Bits 31:2 specify the MMIO offset destination for the data in the Immediate Data Low (DW3) field. Only DW writes are valid.	
	1:0	Reserved	
		Project:	All
		Format:	MBZ
3	31:0	Immediate Data	
		Project:	All
		Format:	U32
		This field specifies the Lower DWord value to be written to the targeted location. Only valid when Post-Sync Operation is 1h (Write Immediate Data) or LRI Post-Sync Operation is set. Ignored if Post-Sync Operation is "No write", "Write PS_DEPTH_COUNT" or "Write TIMESTAMP".	
4	31:0	Immediate Data	
		Project:	All
		Format:	U32
		This field specifies the Upper DWord value to be written to the targeted location. Only valid when Post-Sync Operation is 1h (Write Immediate Data) Ignored if Post-Sync Operation is "No write", "Write PS_DEPTH_COUNT", "Write TIMESTAMP" or "LRI Post Sync Operation".	

1.10.4.2 Programming Restrictions for PIPE_CONTROL

PIPE_CONTROL arguments can be split up into three categories:

- Post-sync operations



- Flush Types
- Stall

Post-sync operation is only indirectly affected by the flush type category via the stall bit. The stall category depends on the both flush type and post-sync operation arguments. A PIPE_CONTROL with no arguments set is **Invalid**.

1.10.4.2.1 Post-Sync Operation

These arguments relate to events that occur after the marker initiated by the PIPE_CONTROL command is completed. The table below shows the restrictions:

Arguments	Bits	Restrictions
LRI Post Sync Operation	23	Post Sync Operation ([15:14] of DW1) must be set to 0x0.
Global Snapshot Count Reset	19	Requires stall bit ([20] of DW1) set.
Generic Media State Clear	16	Requires stall bit ([20] of DW1) set.
Indirect State Pointers Disable	9	Requires stall bit ([20] of DW1) set.
Store Data Index	21	Post-Sync Operation ([15:14] of DW1) must be set to something other than '0'.
Sync GFDT	17	Post-Sync Operation ([15:14] of DW1) must be set to something other than '0' or 0x2520[13] must be set.
TLB inv	18	Also Requires stall bit ([20] of DW1) set.
Post Sync Op	15:14	No Restriction. LRI Post Sync Operation ([23] of DW1) must be set to '0'.
Notify En	8	No Restriction.

1.10.4.2.2 Flush Types

These are arguments related to the type of read only invalidation or write cache flushing is being requested. Note that there is only intra-dependency. That is, it is not affected by the post-sync operation or the stall bit. The table below shows the restrictions.

Arguments	Bit	Restrictions
Depth Stall	13	The following bits must be clear <ul style="list-style-type: none"> • Render Target Cache Flush Enable ([12] of DW1) • Depth Cache Flush Enable ([0] of DW1)
Render Target Cache Flush	12	Depth Stall must be clear ([13] of DW1)
Depth Cache Flush	0	Depth Stall must be clear ([13] of DW1)
Stall Pixel Scoreboard	1	No Restriction
Inst invalidate.	11	No Restriction
Tex invalidate.	10	No Restriction
VF invalidate	4	No Restriction
Constant invalidate	3	No Restriction
State Invalidate	2	No Restriction



1.10.4.2.3 \ Stall

If the stall bit is set, the command streamer waits until the pipe is completely flushed.

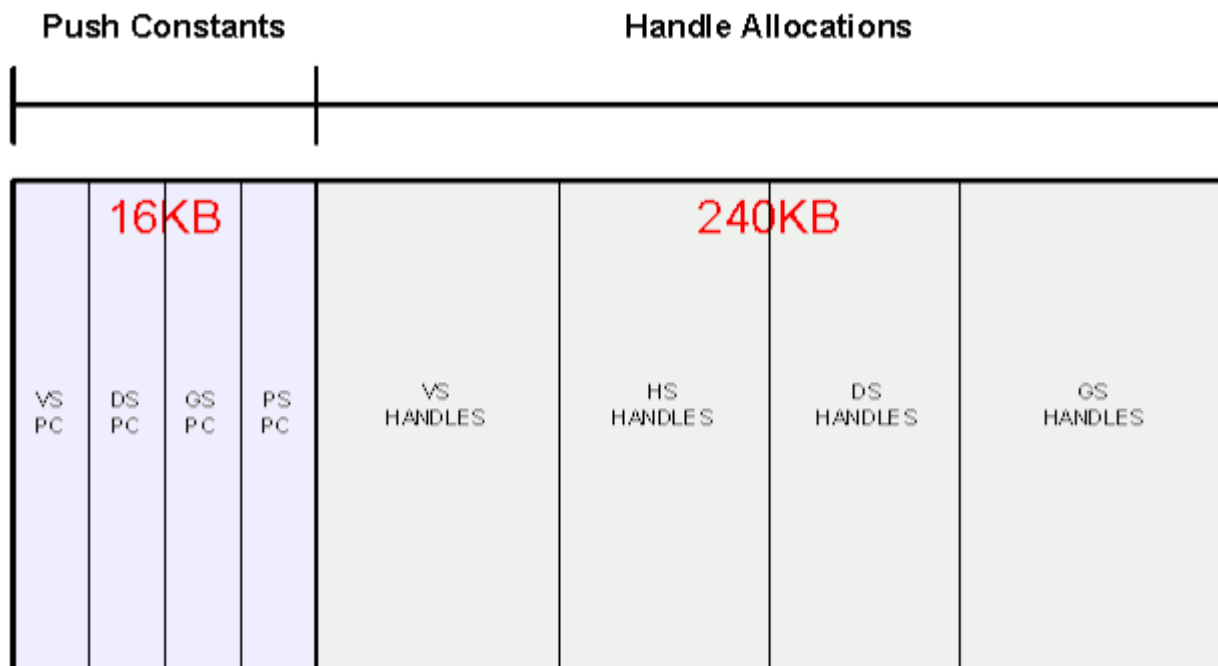
Arguments	Bit	Restrictions
Stall Bit	20	[All Stepping][All SKUs]: One of the following must also be set <ul style="list-style-type: none"> • Render Target Cache Flush Enable ([12] of DW1) • Depth Cache Flush Enable ([0] of DW1) • Stall at Pixel Scoreboard ([1] of DW1) • Depth Stall ([13] of DW1) • Post-Sync Operation ([13] of DW1) • Notify Enable ([8] of DW1)

1.11 Push Constant URB Allocation

The push constants are stored into the URB which is part of the L3\$. Software is required to program the hardware to allocate space in the URB for each shader push constant. The software is limited to the bottom address of the URB and must ensure that none of the shaders have overlapping handles. Below is a diagram that represents a possible programming of the URB with Push Constants:

The sizes of the regions in the diagram will change to 16KB and 80KB, respectively

URB Allocation



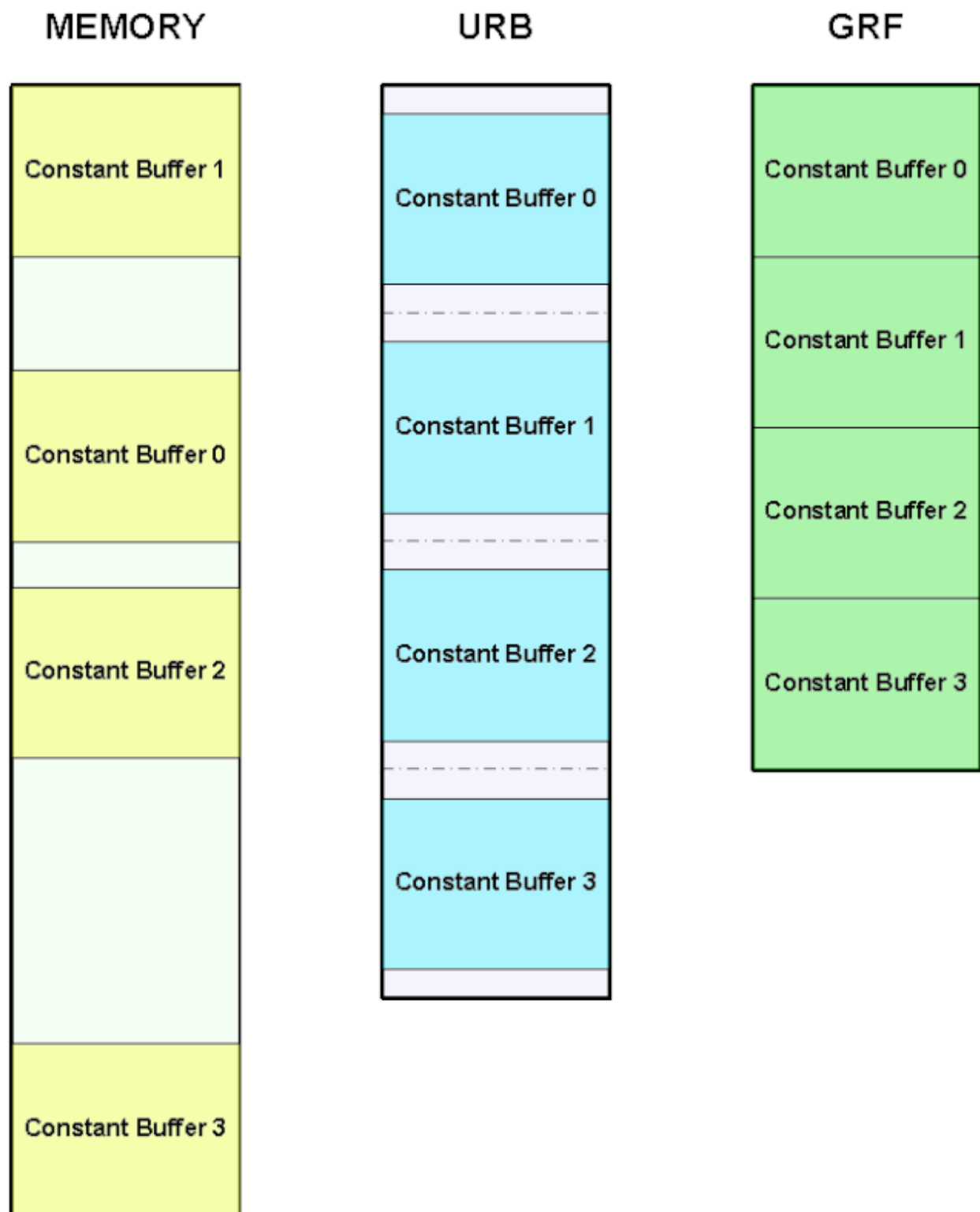


In the above scheme we are allocating 16KB of push constants and 240KB of URB space. The handle allocation is shown in the order of the FF pipeline but with the current hardware and state, the software can program these to be any order and may size them to zero. Software may also use some if not all of the 16KB above as handle allocations as long as none of the push constants or handle allocations overlap. The only limitations are the sizes based off the table below and the restrictions in granularity which are specified in the command descriptions of the URB state and the push constant allocation state for each fixed function.

Below is a table that specifies the maximum size of each buffer:

Max Constant Buffer	URB size
16KB	256KB
16KB	128KB

Below is a diagram that represents how the hardware may move and store one CONSTANT_BUFFER command for a fixed function shader:



The bubbles in the URB are caused by the constant buffer in memory starting on a half cacheline and being an even number in length. If the constant buffer starts on an odd cacheline and has an odd number length then there will only be a bubble at the beginning of the buffer in the URB. If the constant buffer in



memory starts on a cache line boundary and has an odd number length then the bubble will only be at the end of the constant buffer in the URB. Once the constant buffer is written to the GRF space then all the bubbles will be removed.

Software must guarantee that there is enough space in the push constant buffer in the URB to hold one constant buffer from memory. This includes any buffering to write the 512b aligned requests from memory into the URB. Because the L3\$ only supports writes from memory in 512b chunks, the URB may have some bubbles between each constant buffer fetch.



2. 3D Pipeline – Vertex Fetch (VF) Stage

2.1 Vertex Fetch (VF) Stage Overview

The VF stage performs one major function: executing 3DPRIMITIVE commands. This is handled by the VF's InputAssembly function. The InputAssembly process is closely matched to the Input Assembly function. Minor enhancements have been included to better support OpenGL.

The following subsections describe some high-level concepts associated with the VF stage.

2.1.1 Input Assembly

The VF's InputAssembly function includes (for each vertex generated):

- Generation of VertexIndex and InstanceIndex for each vertex, possibly via use of an Index Buffer.
- Lookup of the VertexIndex in the Vertex Cache (if enabled)
- If a cache miss is detected:
 - Use of computed indices to fetch data from memory-resident vertex buffers
 - Format conversion of the fetched vertex data
 - Assembly of the format conversion results (and possibly some internally generated data) to form the complete “input” (raw) vertex
 - Storing the input vertex data in a Vertex URB Entry (VUE) in the URB
 - Output of the VUE handle of the input vertex to the VS stage
- If a cache hit is detected, the VUE handle from the Vertex Cache is passed to the VS stage (marked as a cache hit to prevent any VS processing).

2.1.1.1 Vertex Assembly

The VF utilizes a number of VERTEX_ELEMENT state structures to define the contents and format of the vertex data to be stored in Vertex URB Entries (VUEs) in the URB. See below for a detailed description of the command used to define these structures (3DSTATE_VERTEX_ELEMENTS).

Each active VERTEX_ELEMENT structure defines up to 4 contiguous DWords of VUE data, where each DWord is considered a “component” of the vertex element. The starting destination DWord offset of the vertex element in the VUE is specified, and the VERTEX_ELEMENT structures must be defined with monotonically increasing VUE offsets. For each component, the source of the component is specified. The source may be a constant (0, 0x1, or 1.0f), a generated ID (VertexID, InstanceID or PrimitiveID), or a component of a structure in memory (e.g., the Y component of an XYZW position in memory). In the case of a memory source, the Vertex Buffer sourcing the data, and the location and format of the source data with that VB are specified.

The VF's Vertex Assembly process can be envisioned as the VF unit stepping through the VERTEX_ELEMENT structures in order, fetching and format-converting the source information (if memory resident), and storing the results in the destination VUE.



2.1.2 Vertex Cache

The VF stage communicates with the VS stage in order to implement a Vertex Cache function in the 3D pipeline. The Vertex Cache is strictly a performance-enhancing feature and has no impact on 3D pipeline results (other than a few statistics counters).

The Vertex Cache contains the VUE handles of VS-output (shaded) vertices if the VS function is enabled, and the VUE handles of VF-output (raw) vertices if the VS function is disabled. (Note that the actual vertex data is held in the URB, and only the handles of the vertices are stored in the cache). In either case, the contents of the cache (VUE handles) are tagged with the VertexIndex value used to fetch the input vertex data. The rationale for using the VertexIndex as the tag is that (assuming no other state or parameters change) a vertex with the same VertexIndex as a previous vertex will have the same input data, and therefore the same result from the VF+VS function.

Note that any change to the state controlling the InputAssembly function (e.g., vertex buffer definition), or any change to the state controlling the VS function (if enabled) (e.g., VS kernel), will result in the Vertex Cache being invalidated. In addition, any non-trivial use of instancing (i.e., more than one instance per 3DPRIMITIVE command and the inclusion of instance data in the input vertex) will effectively invalidate the cache between instances, as the InstanceIndex is not included in the cache tag. See Vertex Caching in *Vertex Shader* for more information on the Vertex Cache (e.g., when it is implicitly disabled, etc.)

2.1.3 Input Data: Push Model vs. Pull Model

Given the programmability of the pipeline, and the ability of shaders to input (load/sample) data from memory buffers in an arbitrary fashion, the decision arises in whether to push instance/vertex data into the front of the pipeline or defer the data access (pull) to the shaders that require it.

There are tradeoffs involved in deciding between these models. For vertex data, it is probably always better to push the data into the pipeline, as the VF hardware attempts to cover the latency of the data fetch. The decision is less clear for instance data, as pushing instance data leads to larger Vertex URB entries which will be holding redundant data (as the instance data for vertices of an object are by definition the same). Regardless, the GEN 3D pipeline supports both models.

2.1.4 Generated IDs

[Note that the generated IDs are considered separate from any offset computations performed by the VF unit, and are therefore described separately here.]

The VF generates InstanceID, VertexID, and PrimitiveID values as part of the InputAssembly process.

VertexID and InstanceID are only allowed to be inserted into the input vertex data as it is gathered and written into the URB as a VUE.

The definition/use of PrimitiveID is more complicated than the other auto-generated IDs. PrimitiveID is associated with an “object”, not a particular vertex. It is only available to the GS (: and HS) as a special non-vertex input, and the PS as a constant-interpolated attribute. It is not seen by the VS (or DS) at all. The PrimitiveID therefore is kept separate from the vertex data. Take for example a TRILIST primitive topology: It should be possible to share vertices between triangles in the list (i.e., reuse the VS output of a vertex), even though each triangle has a different PrimitiveID associated with it.



2.1.4.1 Generated IDs

The InstanceID, VertexID, and PrimitiveID values associated with each vertex can be stored in the vertex's VUE, via use of the **Component *n* Control** fields in the VERTEX_ELEMENT structure. This makes the values available to the VS thread.

While the PrimitiveID can still be stored in the VUE (see above), there should be no API-specific reason to do so. The 32-bit PrimitiveIDs associated with objects are passed down the FF pipeline and made available to GS and Setup threads as payload header data. A side effect of this feature is that the vertex cache can operate even when PrimitiveIDs are being used.

2.2 Index Buffer (IB)

The 3DSTATE_INDEX_BUFFER command is used to define an *Index Buffer* (IB) used in subsequent 3DPRIMITIVE commands.

The RANDOM access mode of the 3DPRIMITIVE command involves the use of a memory-resident IB. The IB, defined via the 3DSTATE_INDEX_BUFFER command described below, contains a 1D array of 8, 16 or 32-bit index values. These index values will be fetched by the InputAssembly function, and subsequently used to compute locations in VERTEXDATA buffers from which the actual vertex data is to be fetched. (This is opposed to the SEQUENTIAL access mode where the vertex data is simply fetched sequentially from the buffers).

Software is responsible for ensuring that accesses outside the IB do not occur. This is possible as software can compute the range of IB values referenced by a 3DPRIMITIVE command (knowing the **StartVertexLocation**, **InstanceCount**, and **VerticesPerInstance** values) and can then compare this range to the IB extent.

2.2.1 3DSTATE_INDEX_BUFFER

3DSTATE_INDEX_BUFFER			
Source:	RenderCS		
Length Bias:	2		
<p>This command is used to specify the current IB state used by the VF function. At most one IB is defined and active at any given time.</p> <p>NOTES: The IB must be specified before any RANDOM 3D_PRIMITIVE commands are issued. It is possible to have vertex elements source completely from generated ID values and therefore not require any Index Buffer accesses. In this case, VF function will simply ignore the Index Buffer state.</p>			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode



3DSTATE_INDEX_BUFFER																
23:16	3D Command Sub Opcode															
	<table border="1"> <tr> <td>Default Value:</td> <td>0Ah 3DSTATE_INDEX_BUFFER</td> </tr> <tr> <td>Format:</td> <td>OpCode</td> </tr> </table>	Default Value:	0Ah 3DSTATE_INDEX_BUFFER	Format:	OpCode											
Default Value:	0Ah 3DSTATE_INDEX_BUFFER															
Format:	OpCode															
15:12	Index Buffer Object Control State															
	<table border="1"> <tr> <td>Format:</td> <td>MEMORY_OBJECT_CONTROL_STATE</td> </tr> </table> <p>Specifies the memory object control state for this index buffer.</p>	Format:	MEMORY_OBJECT_CONTROL_STATE													
Format:	MEMORY_OBJECT_CONTROL_STATE															
11	Reserved															
	<table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Project:	All	Format:	MBZ											
Project:	All															
Format:	MBZ															
10	Cut Index Enable															
	<table border="1"> <tr> <td>Format:</td> <td>Enable</td> </tr> </table> <p>If ENABLED, the largest index value (0xFF,0xFFFF,0xFFFFFFFF, depending on Index Format) is interpreted as the “cut” index. (See description of this elsewhere in this section). (Expected OpenGL driver usage)This field can only be enabled for certain primitive topology types. Refer to the table later in this section for details.</p>	Format:	Enable													
Format:	Enable															
9:8	Index Format															
	<table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U2 enumerated type</td> </tr> </table> <p>This field specifies the data format of the index buffer. All index values are UNSIGNED.</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> <th>Project</th> </tr> </thead> <tbody> <tr> <td>0h</td> <td>INDEX_BYTE</td> <td>All</td> </tr> <tr> <td>1h</td> <td>INDEX_WORD</td> <td>All</td> </tr> <tr> <td>2h</td> <td>INDEX_DWORD</td> <td>All</td> </tr> </tbody> </table>	Project:	All	Format:	U2 enumerated type	Value	Name	Project	0h	INDEX_BYTE	All	1h	INDEX_WORD	All	2h	INDEX_DWORD
Project:	All															
Format:	U2 enumerated type															
Value	Name	Project														
0h	INDEX_BYTE	All														
1h	INDEX_WORD	All														
2h	INDEX_DWORD	All														
7:0	DWord Length															
	<table border="1"> <tr> <td>Default Value:</td> <td>1h Excludes DWord (0,1)</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>=n Total Length – 2</td> </tr> </table>	Default Value:	1h Excludes DWord (0,1)	Project:	All	Format:	=n Total Length – 2									
Default Value:	1h Excludes DWord (0,1)															
Project:	All															
Format:	=n Total Length – 2															
1	31:0	Buffer Starting Address														
		<table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>GraphicsAddress[31:0]Index_Buffer_Entry</td> </tr> </table> <p>This field contains the size-aligned (as specified by Index Format) Graphics Address of the first element of interest within the index buffer. Software must program this value with the combination (sum) of the base address of the memory resource and the byte offset from the base address to the starting structure within the buffer.</p> <p style="text-align: center;">Programming Notes</p> <p>Index Buffers can only be allocated in linear (not tiled) graphics memory</p>	Project:	All	Format:	GraphicsAddress[31:0]Index_Buffer_Entry										
Project:	All															
Format:	GraphicsAddress[31:0]Index_Buffer_Entry															
2	31:0	Buffer Ending Address														
		<table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>GraphicsAddress[31:0]</td> </tr> </table> <p>If non-zero, this field contains the address of the last valid byte in the index buffer. Any index buffer reads past this address returns an index value of 0 (as if the index buffer was zero-extended).</p>	Project:	All	Format:	GraphicsAddress[31:0]										
Project:	All															
Format:	GraphicsAddress[31:0]															



3DSTATE_INDEX_BUFFER		
	Software must guarantee that the buffer ends on an index boundary (e.g., for an INDEX_DWORD buffer, Bits [1:0] == 11b).	
Errata	Description	Project
	Software needs to disable the index buffer by setting Index Buffer Start address AFTER Index Buffer End address for draws where the starting index location is greater than the index buffer size.	

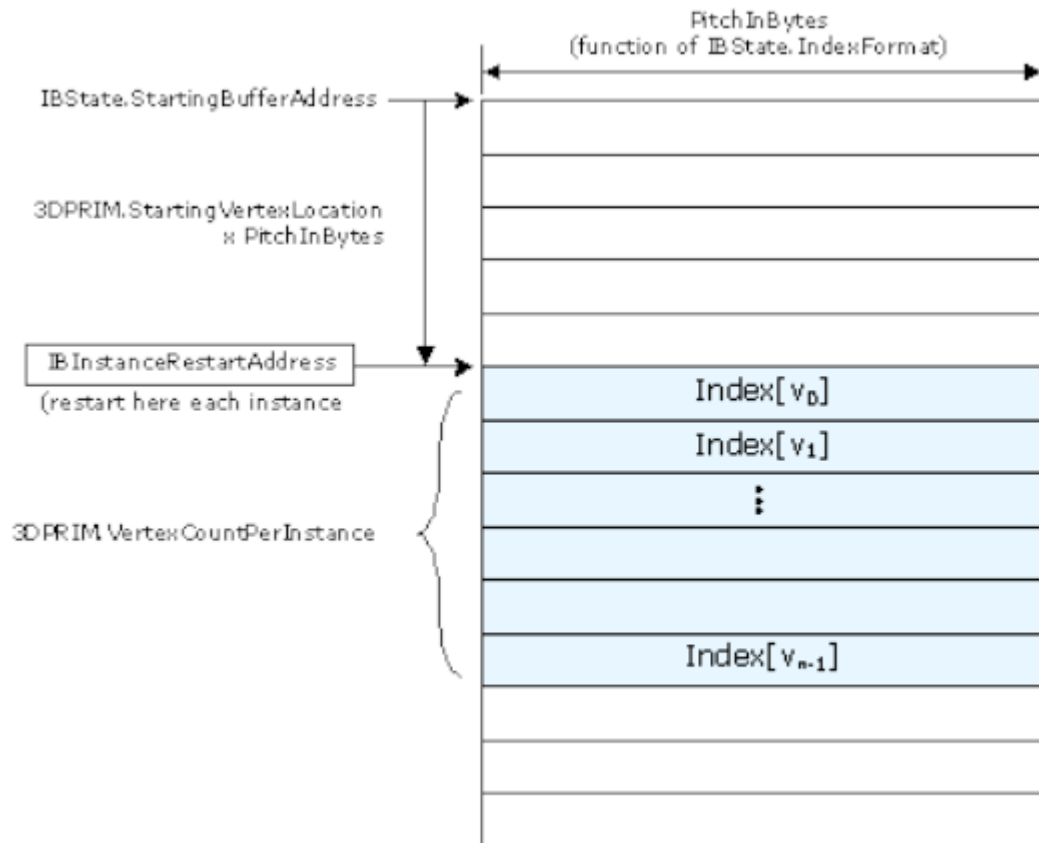
The following table lists which primitive topology types support the presence of Cut Indices.

When 3DSTATE_INDEX_BUFFER has **Cut Index Enable** set, it is UNDEFINED to issue a 3DPRIMITIVE with a primitive topology type not supporting a Cut Index (even if no cut indices are actually present in the index buffer).

Definition	Cut Index?
3DPRIM_POINTLIST	Y
3DPRIM_LINELIST	Y
3DPRIM_LINESTRIP	Y
3DPRIM_TRILIST	Y
3DPRIM_TRISTRIP	Y
3DPRIM_TRIFAN	N
3DPRIM_QUADLIST	N
3DPRIM_QUADSTRIP	N
3DPRIM_LINELIST_ADJ	Y
3DPRIM_LINESTRIP_ADJ	Y
3DPRIM_TRILIST_ADJ	Y
3DPRIM_TRISTRIP_ADJ	Y
3DPRIM_TRISTRIP_REVERSE	Y
3DPRIM_POLYGON	N
3DPRIM_RECTLIST	N
3DPRIM_LINELOOP	N
3DPRIM_POINTLIST_BF	Y
3DPRIM_LINESTRIP_CONT	Y
3DPRIM_LINESTRIP_BF	Y
3DPRIM_LINESTRIP_CONT_BF	Y
3DPRIM_TRIFAN_NOSTIPPLE	N
3DPRIM_PATCHLIST_n	Y

2.2.2 Index Buffer Access

The following figure illustrates how the Index Buffer is accessed.



B.6825-01

2.3 Vertex Buffers (VBs)

The `3DSTATE_VERTEX_BUFFERS` and `3DSTATE_INSTANCE_STEP_RATE` commands are used to define *Vertex Buffers* (VBs) used in subsequent `3DPRIMITIVE` commands.

Most input vertex data is sourced from memory-resident VBs. A VB is a 1D array of structures, where the size of the structure as defined by the VB's **BufferPitch**. VBs are accessed either as *VERTEXDATA buffers* or *INSTANCEDATA buffers*, as defined by the VB's **BufferAccessType**. The VB's access type will determine whether the VF-computed `VertexIndex` or `InstanceIndex` is used to access data in the VB.

Given that the `RANDOM` access mode of the `3DPRIMITIVE` command utilizes an IB (possibly provided by an application) to compute VB index values, VB definitions contain a **MaxIndex** value used to detect accesses beyond the end of the VBs. Any access outside the extent of a VB returns 0.



2.3.1 3DSTATE_VERTEX_BUFFERS

3DSTATE_VERTEX_BUFFERS			
Source:	RenderCS		
Length Bias:	2		
<p>This command is used to specify VB state used by the VF function.</p> <p>This command can specify from 1 to 33 VBs.</p> <p>The VertexBufferID field within a VERTEX_BUFFER_STATE structure indicates the specific VB. If a VB definition is not included in this command, its associated state is left unchanged and is available for use if previously defined.</p>			
Programming Notes			
<p>It is possible to have individual vertex elements sourced completely from generated ID values and therefore not require any vertex buffer accesses for that vertex element. In this case, VF function will simply ignore the VB state associated with that vertex element. If all enabled vertex elements have this characteristic, no VBs are required to process 3DPRIMITIVE commands. For example, this might arise when the user wants to perform all data lookups in the first shader, so only generated index values need to be passed down to it. In this extreme case, SW would not need to program any VB state, and therefore not need to issue any 3DSTATE_VERTEX_BUFFERS commands.</p> <p>For any 3DSTATE_VERTEX_BUFFERS command, at least one VERTEX_BUFFER_STATE structure must be included.</p> <p>VERTEX_BUFFER_STATE structures are 4 DWords for both VERTEXDATA buffers and INSTANCEDATA buffers. Inclusion of partial VERTEX_BUFFER_STATE structures is UNDEFINED.</p> <p>The order in which VBs are defined within this command can be arbitrary, though a vertex buffer must be defined only once in any given command (otherwise operation is UNDEFINED).</p>			
DWord	Bit	Description	
0	31:29	Instruction Type	
		Default Value:	03h GFXPIPE
		Format:	Opcode
	28:27	Instruction Sub-Type	
		Default Value:	3h 3D
		Format:	Opcode
	26:24	Instruction Opcode	
		Default Value:	0h 3DSTATE_VERTEX_BUFFERS
		Format:	Opcode
	23:16	Instruction Sub-Opcode	
Default Value:		08h 3DSTATE_VERTEX_BUFFERS	
Format:		Opcode	
15:8	Reserved		
7:0	DWord Count		
	Default Value:	3 DWORD_COUNT_n	
	Format:	=n	
	n = 4b-1 (where b = # of buffer states included)		
1..n	127:0	Vertex Buffer State [n]	
		Format: VERTEX_BUFFER_STATE	



2.3.2 VERTEX_BUFFER_STATE Structure

VERTEX_BUFFER_STATE				
Source:	RenderCS			
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000			
<p>This structure is used in 3DSTATE_VERTEX_BUFFERS to set the state associated with a VB. The VF function will use this state to determine how/where to extract vertex element data for all vertex elements associated with the VB.</p> <p>The VERTEX_BUFFER_STATE structure is 4 DWords for both INSTANCEDATA and VERTEXDATA buffers. A VB is defined as a 1D array of vertex data structures, accessed via a computed index value. The VF function therefore needs to know the starting address of the first structure (index 0) and size of the vertex data structure.</p>				
Programming Notes			Project	
Vertex element accesses which straddle or go past the VB's End Address will return 0's for all elements.				
DWord	Bit	Description		
0	31:26	Vertex Buffer Index		
		Format:	U6 Index	
		This field contains an index value which selects the VB state being defined.		
		Value	Name	
		[0,32]		
		25:21	Reserved	
			Project:	All
			Format:	MBZ
		20	Buffer Access Type	
			This field determines how vertex element data is extracted from this VB. This control applies to all vertex elements associated with this VB.	
Value	Name		Description	Project
00b	VERTEXDATA		For SEQUENTIAL vertex access, each vertex of an instance is sourced from sequential structures within the VB. For RANDOM vertex access, each vertex of an instance is looked up (separately) via a computed index value	All
01b	INSTANCEDATA		Each vertex of an instance is sourced with the same (instance) data. Subsequent instances may be sourced with the same or different data, depending on Instance Data Step Rate.	All
19:16	Vertex Buffer Memory Object Control State			
	Project:	All		
	Format:	MEMORY_OBJECT_CONTROL_STATE		
	Specifies the memory object control state for this vertex buffer.			
15	Reserved			
	Project:	All		
	Format:	MBZ		
14	Address Modify Enable			
	If set, the Buffer Starting Address and End Address fields are used to update the state of this buffer. If			



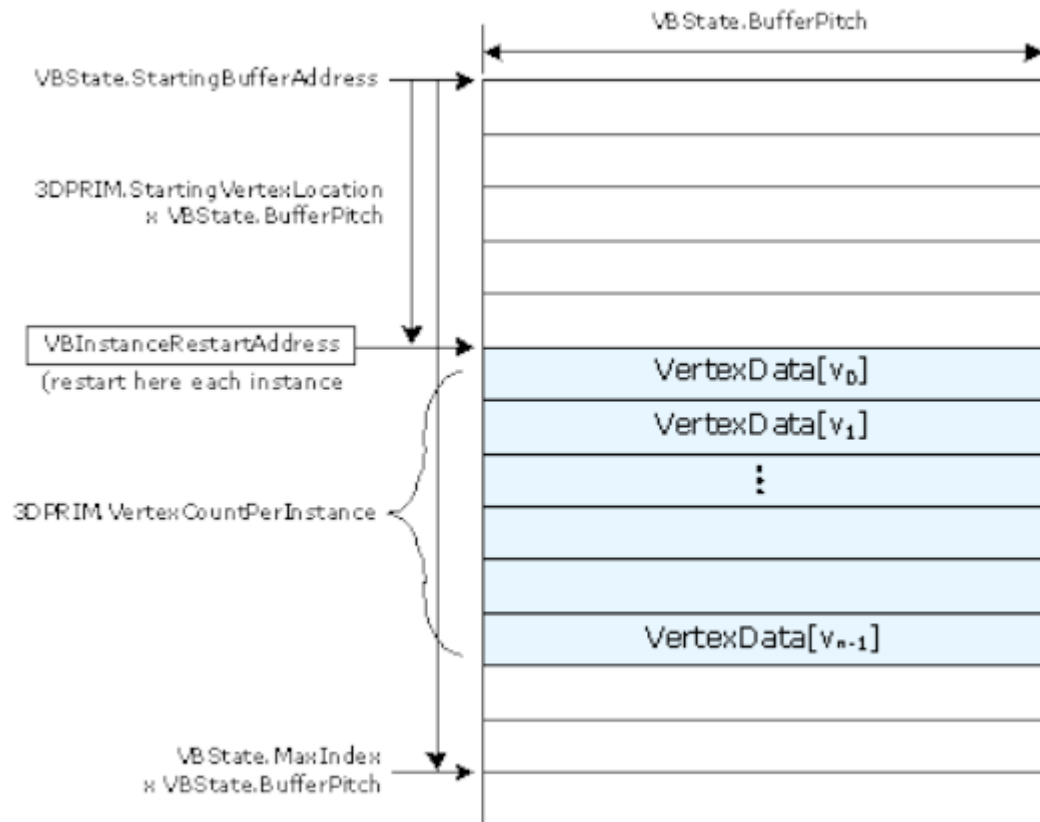
VERTEX_BUFFER_STATE			
		clear, those fields are ignored and the previously-programmed values are maintained.	
13	Null Vertex Buffer		
	Format:	Enable	
	This field enabled causes any fetch for vertex data to return 0.		
12	Vertex Fetch Invalidate		
	Default Value:	0h	
	Invalidate the Vertex overfetch cache when this bit is set. For multiple vertex buffer state structures in one packet, this bit may be set only once in the entire packet.		
11:0	Buffer Pitch		
	Format:	U12 Count of bytes	
	This field specifies the pitch in bytes of the structures accessed within the VB. This information is required in order to access elements in the VB via a structure index.		
	Value	Name	Description
	[0,2048]		Bytes
	Programming Notes		
	Different VERTEX_BUFFER_STATE structures can refer to the same memory region using different Buffer Pitch values.		
	See note on 64-bit float alignment in Buffer Starting Address.		
1	31:0	Buffer Starting Address	
		Format:	GraphicsAddress[31:0]
		Description	Project
		This field contains the byte-aligned Graphics Address of the first element of interest within the VB. Software must program this value with the combination (sum) of the base address of the memory resource and the byte offset from the base address to the starting structure within the buffer.	
		If the Address ModifyEnable bit is clear, this field is ignored and the previous value of Buffer Starting Address for this buffer is maintained.	
		Programming Notes	
		64-bit floating point values must be 64-bit aligned in memory, or UNPREDICTABLE data will be fetched. When accessing an element containing 64-bit floating point values, the Buffer Starting Address and Source Element Offset values must add to a 64-bit aligned address, and BufferPitch must be a multiple of 64-bits.	
		VBs can only be allocated in linear (not tiled) graphics memory.	
		As computed index values are, by definition, interpreted as unsigned values, there is no issue with	



VERTEX_BUFFER_STATE															
	<p>accesses to locations before (lower address value) the start of the buffer. However, these wrapped indices are subject to Max Index checking (see below).</p>														
2	<table border="1"> <tr> <td>31:0</td> <td>End Address</td> </tr> <tr> <td>Format:</td> <td>GraphicsAddress[31:0]U32</td> </tr> <tr> <td colspan="2" style="text-align: center;">Description</td> </tr> <tr> <td colspan="2"> <p>This field defines the address of the last valid byte in this particular VB. Access of a vertex element which either straddles or is beyond this address will return 0's for any data read.</p> <p>If the Address ModifyEnable bit is clear, this field is ignored and the previous value of End Address for this buffer is maintained.</p> </td> </tr> <tr> <td colspan="2" style="text-align: center;">Value</td> </tr> <tr> <td>[0,FFFFFFFFh]</td> <td></td> </tr> <tr> <td>0h</td> <td>[Default]</td> </tr> </table>	31:0	End Address	Format:	GraphicsAddress[31:0]U32	Description		<p>This field defines the address of the last valid byte in this particular VB. Access of a vertex element which either straddles or is beyond this address will return 0's for any data read.</p> <p>If the Address ModifyEnable bit is clear, this field is ignored and the previous value of End Address for this buffer is maintained.</p>		Value		[0,FFFFFFFFh]		0h	[Default]
31:0	End Address														
Format:	GraphicsAddress[31:0]U32														
Description															
<p>This field defines the address of the last valid byte in this particular VB. Access of a vertex element which either straddles or is beyond this address will return 0's for any data read.</p> <p>If the Address ModifyEnable bit is clear, this field is ignored and the previous value of End Address for this buffer is maintained.</p>															
Value															
[0,FFFFFFFFh]															
0h	[Default]														
3	<table border="1"> <tr> <td>31:0</td> <td>Instance Data Step Rate</td> </tr> <tr> <td>Format:</td> <td>U32</td> </tr> <tr> <td colspan="2"> <p>This field only applies to INSTANCEDATA buffers – it is ignored (but still present) for VERTEXDATA buffers.</p> <p>This field determines the rate at which instance data for this particular INSTANCEDATA vertex buffer is changed in sequential instances. Only after the number of instances specified by this field is generated is new (sequential) instance data provided. This process continues for each group of instances defined in the draw command. For example, a value of 1 in this field causes new instance data to be supplied with each sequential (instance) group of vertices. A value of 2 causes every other instance group of vertices to be provided with new instance data. The special value of 0 causes all vertices of all instances generated by the draw command to be provided with the same instance data. (The same effect can be achieved by setting this field to its maximum value.)</p> </td> </tr> </table>	31:0	Instance Data Step Rate	Format:	U32	<p>This field only applies to INSTANCEDATA buffers – it is ignored (but still present) for VERTEXDATA buffers.</p> <p>This field determines the rate at which instance data for this particular INSTANCEDATA vertex buffer is changed in sequential instances. Only after the number of instances specified by this field is generated is new (sequential) instance data provided. This process continues for each group of instances defined in the draw command. For example, a value of 1 in this field causes new instance data to be supplied with each sequential (instance) group of vertices. A value of 2 causes every other instance group of vertices to be provided with new instance data. The special value of 0 causes all vertices of all instances generated by the draw command to be provided with the same instance data. (The same effect can be achieved by setting this field to its maximum value.)</p>									
31:0	Instance Data Step Rate														
Format:	U32														
<p>This field only applies to INSTANCEDATA buffers – it is ignored (but still present) for VERTEXDATA buffers.</p> <p>This field determines the rate at which instance data for this particular INSTANCEDATA vertex buffer is changed in sequential instances. Only after the number of instances specified by this field is generated is new (sequential) instance data provided. This process continues for each group of instances defined in the draw command. For example, a value of 1 in this field causes new instance data to be supplied with each sequential (instance) group of vertices. A value of 2 causes every other instance group of vertices to be provided with new instance data. The special value of 0 causes all vertices of all instances generated by the draw command to be provided with the same instance data. (The same effect can be achieved by setting this field to its maximum value.)</p>															

2.3.3 VERTEXDATA Buffers – SEQUENTIAL Access

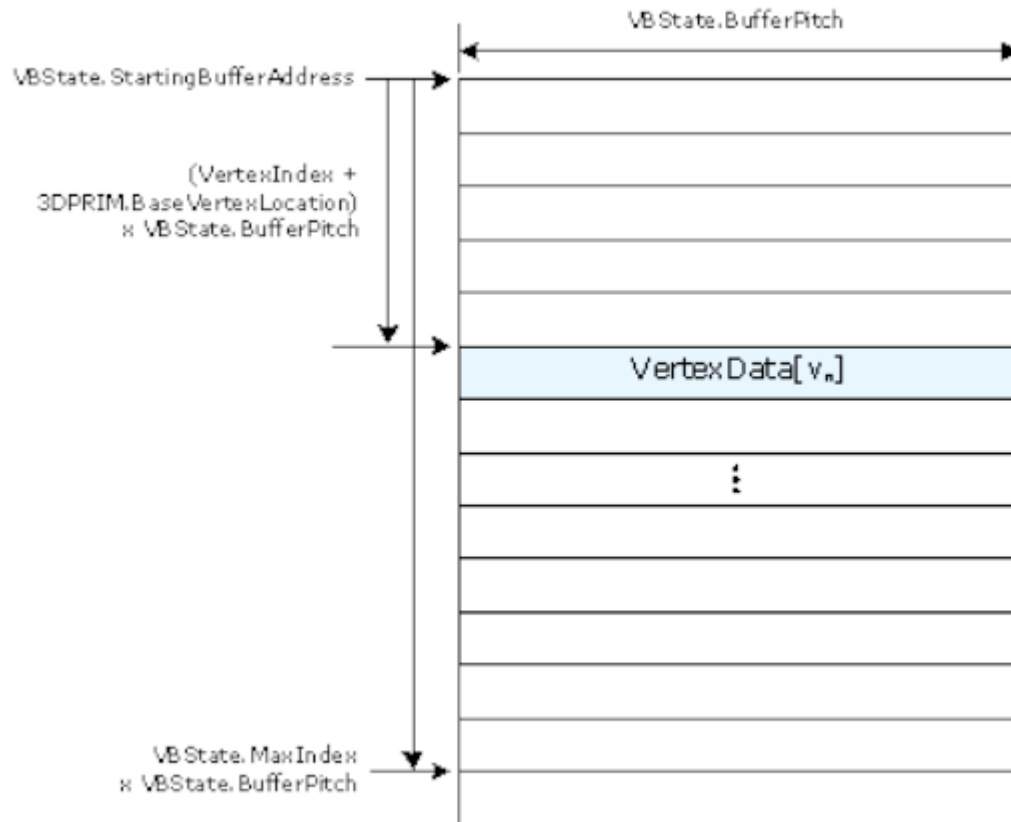
Instead of “VBState.StartingBufferAddress + VBState.MaxIndex x VBState.BufferPitch”, the address of the byte immediately beyond the last valid byte of the buffer is determined by “VBState.EndAddress+1”.



B.6826-01

2.3.4 VERTEXDATA Buffers – RANDOM Access

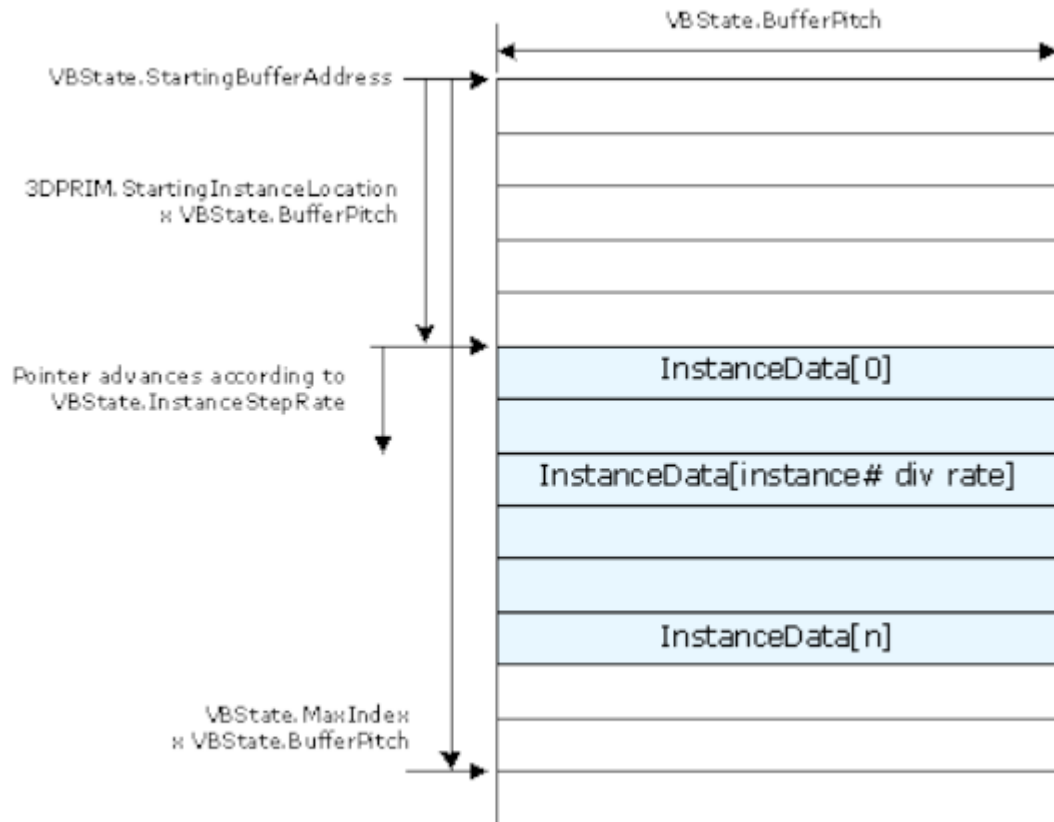
Instead of “VBState.StartingBufferAddress + VBState.MaxIndex x VBState.BufferPitch”, the address of the byte immediately beyond the last valid byte of the buffer is determined by “VBState.EndAddress+1”.



B 6827-01

2.3.5 INSTANCEDATA Buffers

Instead of “VBState.StartingBufferAddress + VBState.MaxIndex x VBState.BufferPitch”, the address of the byte immediately beyond the last valid byte of the buffer is determined by “VBState.EndAddress+1”.



B6839-01

2.4 Input Vertex Definition

The 3DSTATE_VERTEX_ELEMENTS command is used to define the source and format of input vertex data and the format of how it is stored in the destination VUE as part of 3DPRIMITIVE processing in the VF unit.

Refer to *3DPRIMITIVE Processing* below for the general flow of how input vertices are input and stored during processing of the 3DPRIMITIVE command.



2.4.1 3DSTATE_VERTEX_ELEMENTS

3DSTATE_VERTEX_ELEMENTS				
Source:	RenderCS			
Length Bias:	2			
This is a variable-length command used to specify the active vertex elements (up to 34) Each VERTEX_ELEMENT_STATE structure contains a Valid bit which determines which elements are used.				
Programming Notes			Project	
At least one VERTEX_ELEMENT_STATE structure must be included.				
Inclusion of partial VERTEX_ELEMENT_STATE structures is UNDEFINED.				
SW must ensure that at least one vertex element is defined prior to issuing a 3DPRIMITIVE command, or operation is UNDEFINED.				
There are no 'holes' allowed in the destination vertex: NOSTORE components must be overwritten by subsequent components unless they are the trailing DWords of the vertex. Software must explicitly chose some value (probably 0) to be written into DWords that would otherwise be 'holes'.				
Within a VERTEX_ELEMENT_STATE structure, if a Component Control field is set to something other than VFCOMP_STORE_SRC, no higher-numbered Component Control fields may be set to VFCOMP_STORE_SRC. In other words, only trailing components can be set to something other than VFCOMP_STORE_SRC.				
(See additional restrictions listed in the command fields and VERTEX_ELEMENT_STATE description).				
Element[0] must be valid.				
All elements must be valid from Element[0] to the last valid element. (i.e. if Element[2] is valid then Element[1] and Element[0] must also be valid)				
The pitch between elements packed in the URB will always be 128 bits.				
DWord	Bit	Description		
0	31:29	Instruction Type		
		Default Value:	03h GFXPIPE	
		Format:	Opcode	
	28:27	Instruction Sub-Type		
		Default Value:	3h 3D	
		Format:	Opcode	
	26:24	Instruction Opcode		
		Default Value:	0h 3DSTATE_VERTEX_ELEMENTS	
		Format:	Opcode	
	23:16	Instruction Sub-Opcode		
Default Value:		09h 3DSTATE_VERTEX_ELEMENTS		
Format:		Opcode		
15:8	Reserved			
	7:0	DWord Count		
Format:		=n		
Vertex Element Count = (DWord Count + 1) / 2				
Value		Name	Description	Project
1		DWORD_COUNT_n [Default]	excludes DWords 0,1	
[0,66]	Range	1-34 Elements		
1..n	63:0	Element [n]		
		Format:	VERTEX_ELEMENT_STATE	



2.4.2 VERTEX_ELEMENT_STATE Structure

VERTEX_ELEMENT_STATE					
Project:	All				
Source:	RenderCS				
Default Value:	0x00000000, 0x00000000				
<p>This structure is used in 3DSTATE_VERTEX_ELEMENTS to set the state associated with a vertex element. A vertex element is defined as an entity supplying from 1 to 4 DWord vertex components to be stored in the vertex URB entry. The number of supported vertex elements is: 34</p> <p>The VF function will use this state, and possibly the state of the associated vertex buffer, to fetch/generate the source vertex element data, perform any required format conversions, padding with zeros, and store the resulting destination vertex element data into the vertex URB entry.</p>					
DWord	Bit	Description			
0	31:26	Vertex Buffer Index			
		Format:	U6		
		This field specifies which vertex buffer the element is sourced from.			
		Value	Name		
		[0,32]	Up to 33 VBs are supported		
		Programming Notes			
		It is possible for a vertex element to include only internally-generated data (VertexID, etc.), in which case the associated vertex buffer state is ignored.			
		25		Valid	
				Format:	Boolean
				Value	Name
1h	TRUE				
0h	FALSE				
24:16		Source Element Format			
		Project:	All		
		Format:	SURFACE_FORMAT		
		Range: Valid encodings are those marked as “Y” in the “Vertex Buffer” column of the table of Surface Format encodings in the Sampler chapter.			
		Format: The encoding of this field is identical the Surface Format field of the SURFACE_STATE structure, as described in the Sampler chapter.			
		This field specifies the format in which the memory-resident source data for this particular vertex element is stored in the memory buffer. This only applies to elements stored with VFCOMP_STORE_SRC component control. (All other component types have an explicit format).			
		15		Edge Flag Enable	
				Format:	Enable
				Description	
				When ENABLED, the source element is interpreted as an EdgeFlag for the vertex. If the source element is zero, the EdgeFlag will be set to FALSE. If the source element is non-zero,	
Project					



VERTEX_ELEMENT_STATE	
	<p>the EdgeFlag will be set to TRUE. The EdgeFlag bit will travel down the fixed function pipeline along with the vertex handle, etc. and not be stored in the vertex data like the other vertex elements. Refer to the fixed function descriptions for how this EdgeFlag affects rendering.</p> <p>Edge flags are supported for the following primitive topology types only, otherwise EdgeFlagEnable must not be ENABLED.</p> <ul style="list-style-type: none"> • 3DPRIM_TRILIST* • 3DPRIM_TRISTRIP* • 3DPRIM_TRIFAN* • 3DPRIM_POLYGON <p>If this bit is DISABLED for all valid VERTEX_ELEMENTS, the vertex will be assigned a default EdgeFlag of TRUE.</p>
	EdgeFlagEnable must not be ENABLED for 3DPRIM_TRISTRIP* and 3DPRIM_TRIFAN*
	Edge flags are supported for all primitive topology types.
	Programming Notes
	<ul style="list-style-type: none"> • This bit must only be ENABLED on the last valid VERTEX_ELEMENT structure. • When set, Component 0 Control must be set to VFCOMP_STORE_SRC, and Component 1-3 Control must be set to VFCOMP_NOSTORE.
	Edge Flags are not supported for QUADLIST primitives. Software may elect to convert QUADLIST primitives to some set of corresponding edge-flag-supported primitive types (e.g., POLYGONS) prior to submission to the 3D pipeline.
14:11	Reserved
	Project: All
	Format: MBZ
10:0	Source Element Offset (in bytes)
	Project: All
	Format: U11 byte offset
	Byte offset of the source vertex element data in the structures comprising the vertex buffer.
	Value
	Name
	[0,2047]
	Programming Notes
	See note on 64-bit float alignment in Buffer Starting Address.
1	31 Reserved
	Project: All
	Format: MBZ
	30:28 Component 0 Control
	Project: All
	Format: 3D_VertexComponentControl
	Refer to the 3D_VertexComponentControl table below
	27 Reserved



VERTEX_ELEMENT_STATE		
	Project:	All
	Format:	MBZ
26:24	Component 1 Control	
	Format:	3D_VertexComponentControl
	Refer to the 3D_VertexComponentControl table below	
23	Reserved	
	Project:	All
	Format:	MBZ
22:20	Component 2 Control	
	Format:	3D_VertexComponentControl
	Refer to the 3D_VertexComponentControl table below	
19	Reserved	
	Project:	All
	Format:	MBZ
18:16	Component 3 Control	
	Format:	3D_VertexComponentControl
	Refer to the 3D_VertexComponentControl table below	
15:8	Reserved	
	Project:	All
	Format:	MBZ
7:0	Reserved	
	Format:	MBZ

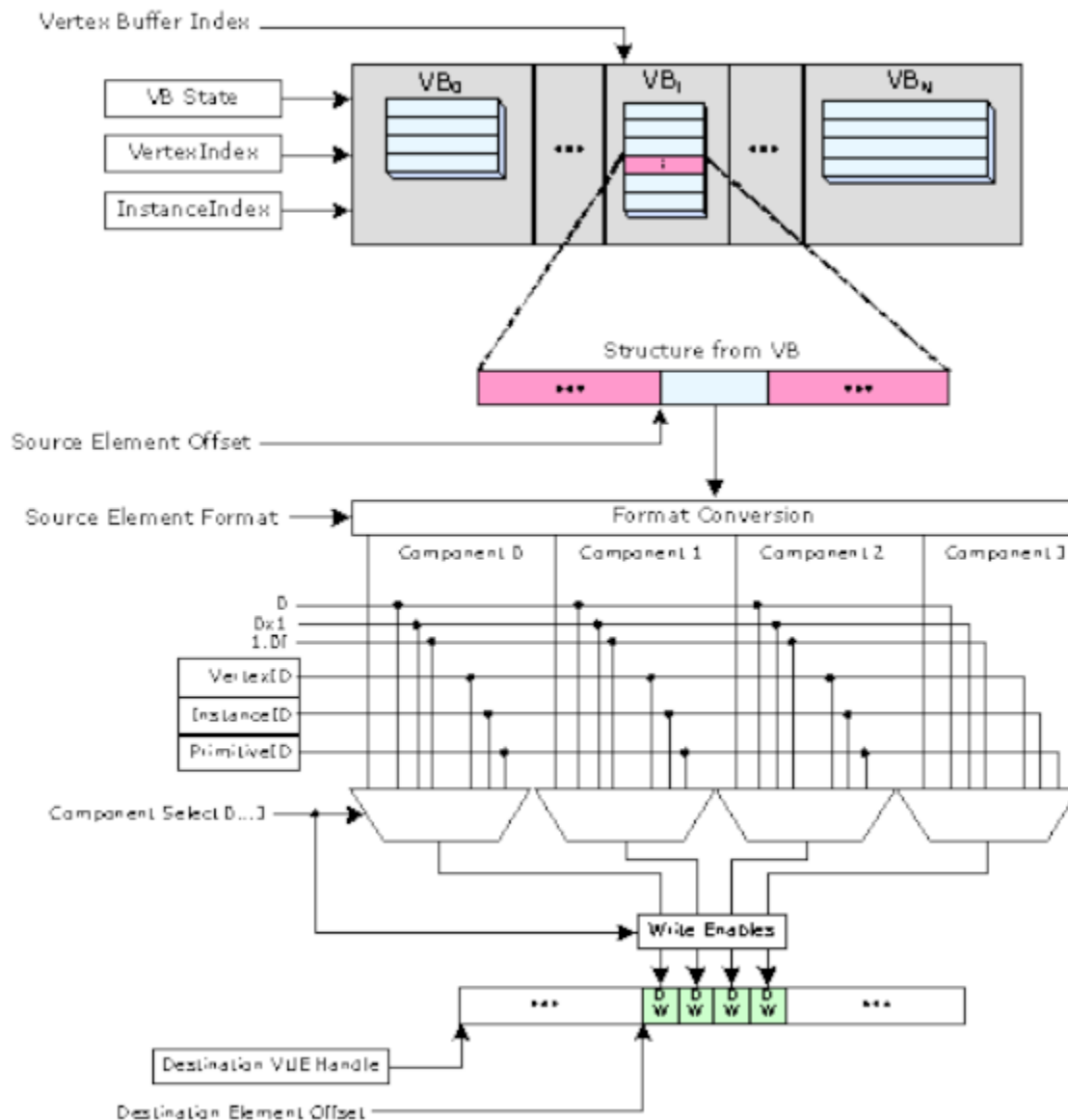
3D_VertexComponentControl			
Project:	All		
Source:	RenderCS		
Size (in bits):	3		
Value	Name	Description	Project
0	VFCOMP_NOSTORE	Don't store this component. (Not valid for Component 0, but can be used for Component 1-3). Once this setting is used for a component, all higher-numbered components (if any) MUST also use this setting. (I.e., no holes within any particular vertex element). Also, there are no 'holes' allowed in the destination vertex: NOSTORE components must be overwritten by subsequent components unless they are the trailing DWords of the vertex. Software must explicitly chose some value (probably 0) to be written into DWords that would otherwise be 'holes'.	All
1	VFCOMP_STORE_SRC	Store corresponding component from format-converted source element. Storing a component that is not included in the Source	All



3D_VertexComponentControl			
		Element Format results in an UNPREDICTABLE value being stored. Software should used the STORE_0 or STORE_1 encoding to supply default components. Within a VERTEX_ELEMENT_STATE structure, if a Component Control field is set to something other than VFCOMP_STORE_SRC, no higher-numbered Component Control fields may be set to VFCOMP_STORE_SRC. In other words, only trailing components can be set to something other than VFCOMP_STORE_SRC.	
2	VFCOMP_STORE_0	Store 0 (interpreted as 0.0f if accessed as a float value)	All
3	VFCOMP_STORE_1_FP	Store 1.0f	All
4	VFCOMP_STORE_1_INT	Store 0x1	All
5	VFCOMP_STORE_VID	Store Vertex ID (as U32)	
6	VFCOMP_STORE_IID	Store Instance ID (as U32)	
7	VFCOMP_STORE_PID	Store Primitive ID (as U32) Software should no longer need to use this encoding as PrimitiveID is passed down the FF pipeline – see explanation above.	All

2.4.3 Vertex Element Data Path

The following diagram shows the path by which a vertex element within the destination VUE is generated and how the fields of the VERTEX_ELEMENT_STATE structure is used to control the generation.



B.6940-01



2.5 3D Primitive Processing

2.5.1 3D PRIMITIVE Command

3DPRIMITIVE	
Source:	RenderCS
Length Bias:	2
<p>The 3DPRIMITIVE command is used to submit 3D primitives to be processed by the 3D pipeline. Typically the processing results in rendering pixel data into the render targets, but this is not required.</p> <p>The parameters passed in this command are forwarded to the Vertex Fetch function. The Vertex Fetch function will use this information to generate vertex data structures and store them in the URB. These vertices are then passed down the 3D pipeline.</p>	
Programming Notes	
<p>If the threads spawned by this command are required to observe memory writes performed by threads spawned from a previous command, software must precede this command with a command that performs a (preferably pipelined) memory flush (e.g., 3D_PIPECONTROL).</p> <p>Workloads enabling topology filter using MI_TOPOLOGY_FILTER must always program PIPECONTROL command with only Post-Sync Operation (Write Immediate data) prior to every 3DPRIMITIVE command programmed.</p>	
Project	
DWord	Bit
Description	
0	31:29
Command Type	
Default Value: 3h GFXPIPE	
Format: OpCode	
	28:27
Command SubType	
Default Value: 3h GFXPIPE_3D	
Format: OpCode	
	26:24
3D Command Opcode	
Default Value: 3h 3DPRIMITIVE	
Format: OpCode	
	23:16
3D Command Sub Opcode	
Default Value: 0h 3DPRIMITIVE	
Format: OpCode	
	15:11
Reserved	
Project: All	
Format: MBZ	
10	
Indirect Parameter Enable	
Project: All	
Format: U1	
<p>If set, the values in DW 2-5 are ignored and replaced by the current values of the corresponding 3DPRIM_xxx MMIO registers:</p> <ul style="list-style-type: none"> • 3DPRIM_VERTEX_COUNT (instead of DW2: VertexCountPerInstance) • 3DPRIM_START_VERTEX (instead of DW3: StartVertexLocation) • 3DPRIM_INSTANCE_COUNT (instead of DW4: InstanceCount) • 3DPRIM_START_INSTANCE (instead of DW5: StartInstanceLocation) 	



3DPRIMITIVE							
	<ul style="list-style-type: none"> 3DPRIM_BASE_VERTEX (instead of DW6: BaseVertexLocation) <p>Indirect Parameter Enable and End Offset Enable must not be ENABLED at the same time, or behavior is UNDEFINED.</p>						
9	<p>Reserved</p> <table border="1"> <tr> <td>Project:</td> <td></td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Project:		Format:	MBZ		
Project:							
Format:	MBZ						
8	<p>Predicate Enable</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> </table> <p>If set, this command is executed (or not) depending on the current value of the MI Predicate internal state bit. This command is ignored only if PredicateEnable is set and the Predicate state bit is 0.</p>	Project:	All	Format:	Enable		
Project:	All						
Format:	Enable						
7:0	<p>DWord Length</p> <table border="1"> <tr> <td>Default Value:</td> <td>5h Excludes DWord (0,1)</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>=n Total Length - 2</td> </tr> </table>	Default Value:	5h Excludes DWord (0,1)	Project:	All	Format:	=n Total Length - 2
Default Value:	5h Excludes DWord (0,1)						
Project:	All						
Format:	=n Total Length - 2						
1	<p>31:10 Reserved</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Project:	All	Format:	MBZ		
Project:	All						
Format:	MBZ						
9	<p>End Offset Enable</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> </table> <p>If set, the Vertex Count Per Instance field is IGNORED, and the VB0ENDOFFSET register is used to indirectly specify the vertex count by defining the amount of valid data in VB0. The following restrictions apply:</p> <ul style="list-style-type: none"> VB0 must be enabled for use VertexAccessType = SEQUENTIAL Instance Count = 1 Start Vertex Location = 0 Start Instance Location = 0 Base Vertex Location = 0 <p>Vertices are output until EndOffset is reached or exceeded in VB0. If EndOffset is reached or exceeded within the data associated with a vertex, that vertex is considered incomplete and will not be output. Partial objects will be discarded (as is normally done).</p> <p>If clear, End Offset is ignored.</p> <p>Indirect Parameter Enable and End Offset Enable must not be ENABLED at the same time, or behavior is UNDEFINED.</p>	Project:	All	Format:	Enable		
Project:	All						
Format:	Enable						
8	<p>Vertex Access Type</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>VertexAccessType</td> </tr> </table>	Project:	All	Format:	VertexAccessType		
Project:	All						
Format:	VertexAccessType						



3DPRIMITIVE			
		This field specifies how data held in vertex buffers marked as VERTEXDATA is accessed by Vertex Fetch.	
		Value	Name
		Description	Project
	0h	SEQUENTIAL	VERTEXDATA buffers are accessed sequentially. Require if End Offset Enable is ENABLED.
	1h	RANDOM	VERTEXDATA buffers are accessed randomly via an index obtained from the Index Buffer.
7:6	Reserved		
	Project:	All	
	Format:	MBZ	
5:0	Primitive Topology Type		
	Project:	All	
	Format:	3D_PrimTopoType See table below for encoding, see 3D Overview for diagrams and general comments	
	This field specifies the topology type of 3D primitive generated by this command. Note that a single primitive topology (list/strip/fan/etc.) can contain a number of basic objects (lines, triangles, etc.).		
2	31:0	Vertex Count Per Instance	
	Project:	All	
	Format:	U32 Count of vertices	
	Format:	GraphicsAddress[31:0]U32*1	
	This field specifies how many vertices are to be generated for each instance of the primitive topology. If End Offset Enable is clear: Format = U32 count of vertices Range = [0, 2^32-1] (upper limit probably constrained by VB size) Ignored if End Offset Enable or Indirect Parameter Enable is ENABLED.		
	Programming Notes		
	<ul style="list-style-type: none"> This per-instance value should specify a valid number of vertices for the primitive topology type. E.g., for 3DPRIM_TRILIST_ADJ, this field should specify a multiple of 6 vertices. However, in cases where too few or too many vertices are provided, the unused vertices will be silently discarded by the pipeline. A 0 value in this field effectively makes the command a 'no-operation'. 		
3	31:0	Start Vertex Location	
	Project:	All	
	Format:	U32 structure index	
	This field specifies the "starting vertex" for each instance. This allows skipping over part of the vertices in a buffer if, for example, a previous 3DPRIMITIVE command had already drawn the primitives associated with the earlier entries. For SEQUENTIAL access, this field specifies, for each instance, a starting structure index into the vertex buffers For RANDOM access, this field specifies, for each instance, a starting index into the Index Buffer.		
	Programming Notes		
	<ul style="list-style-type: none"> Access of any data outside of the valid extent of a vertex or index buffer will return the value 0 (i.e., appears as if the data stored at the invalid location was 0). 		



3DPRIMITIVE		
		<ul style="list-style-type: none"> Must be set to 0 if End Offset Enable is ENABLED. Ignored if Indirect Parameter Enable is ENABLED
4	31:0	Instance Count
		Project: All
		Format: U32 Count of instances
		Description
		This field specifies the number of instances by which the primitive topology is to be regenerated. A value of 0 indicates “no instances” (no-op operation). A value of 1 effectively specifies “non-instanced” operation, though vertex buffers will still be used to provide instance data, if so programmed. Ignored if Indirect Parameter Enable is ENABLED. Must be set to 1 if End Offset Enable is ENABLED.
		Value
		Name
		[0,FFFFFFFFh]
5	31:0	Start Instance Location
		Project: All
		Format: U32 structure index
		Description
		This field specifies the “starting instance” for the command as an initial structure index into INSTANCEDATA buffers. Subsequent instances will access sequential instance data structures, as controlled by the Instance Data Step Rate.
		Programming Notes
		<ul style="list-style-type: none"> Access of any data outside of the valid extent of a vertex or index buffer will return the value 0 (i.e., appears as if the data stored at the invalid location was 0). Must be set to 0 if End Offset Enable is ENABLED. Ignored if Indirect Parameter Enable is ENABLED.
6	31:0	Base Vertex Location
		Project: All
		Format: S31 index structure bias
		This field specifies a signed bias to be added to values read from the index buffer. This allows the same index buffer values to access different vertex data for different commands. This field applies only to RANDOM access mode. This field is ignored for SEQUENTIAL access mode, where there Start Vertex Location can be used to specify different regions in the vertex buffers.
		Programming Notes

3DPRIMITIVE	
	<ul style="list-style-type: none"> • Access of any data outside of the valid extent of a vertex or index buffer will return the value 0 (i.e., appears as if the data stored at the invalid location was 0). • Must be set to 0 if End Offset Enable is ENABLED. • Ignored if Indirect Parameter Enable is ENABLED.

2.5.2 Functional Overview

The following pseudocode summarizes the general flow of 3D Primitive Processing.

```

CommandInit
  InstanceLoop{
    VertexLoop{
      VertexIndexGeneration
      if (CutFlag)
        TerminatePrimitive
    else
      OutputBufferedVertex
      VertexCacheLookup
      if (miss) {
        VertexElementLoop {
          SourceElementFetch
          FormatConversion
          DestinationComponentSelection
          PrimitiveInfoGeneration
          URBWrite
        }
      }
    }
  }
  TerminatePrimitive
}

```

2.5.3 CommandInit

The InstanceID value is initialized to 0.

2.5.4 InstanceLoop

The InstanceLoop is the outmost loop, iterating through each instance of primitives. There is no special “non-instanced” mode – at a minimum there is one instance of primitives.

For SEQUENTIAL accessing, the VertexID value is initialized to 0 at the start of each instance. (For RANDOM accessing, there is no initial value for VertexID, as it is derived from the fetched IB value).

The PrimitiveID is also initialized to 0 at the start of each instance. StartPrim is initialized to TRUE.



The VertexLoop (see below) is then executed to iterate through the instance vertices and output vertices to the pipeline as required.

The end of each iteration of InstanceLoop includes an implied “cut” operation.

The InstanceID value is incremented at the end of each InstanceLoop. Note that each instance will produce the same vertex outputs with the exception of any data dependent on InstanceID (i.e., “instance data”).

2.5.5 VertexLoop

The VertexLoop iterates VertexNumber through the VertexCountPerInstance vertices for the instance.

For each iteration, a number of processing steps are performed (see below) to generate the information that comprises a vertex. Note that, due to CutProcessing, each iteration does not necessarily output a vertex to the pipeline. When a vertex is to be output, the following information is generated for that vertex:

- PrimitiveType associated with the vertex. This is simply a copy of the PrimitiveTopologyType field of the 3DPRIMITIVE
- VUE handle at which the vertex data is stored
 - For a Vertex Cache hit, the VUE handle is marked with a VCHit boolean, so that the VS unit will not attempt to process (shade) that vertex.
 - Otherwise, the VertexLoop will generate and store the input vertex data into the VUE referenced by this handle.
- The PrimitiveID associated with the vertex. See PrimitiveInfoGeneration.
- PrimStart and PrimEnd booleans associated with the vertex. See PrimitiveInfoGeneration.

(Note that a single vertex of buffering is required in order to associate PrimEnd with a vertex, as this information may not be known until the next iteration through the VertexLoop (see *OutputPrimitiveDelimiter*).

VertexNumber value is incremented by 1 at the end of the loop.

2.5.6 VertexIndexGeneration

A VertexIndex value needs to be derived for each vertex. With the exception of the “cut” index, this index value is used as the vertex cache tag and will be used as a structure index into all VERTEXDATA VBs.

For SEQUENTIAL accessing, the VertexID and VertexIndex value is derived as shown below:

```
VertexIndex = StartVertexLocation + VertexNumber
VertexID = VertexNumber
```

For RANDOM access, the VertexID and VertexIndex is derived from an IBValue read from the IB, as shown below:

```
IBIndex = StartVertexLocation + VertexNumber
VertexID = IB[IBIndex]
if (CutIndexEnable && VertexID == CutIndex)
    CutFlag = 1
else
    VertexIndex = VertexID + BaseVertexLocation
```



```
        CutFlag = 0
    endif
```

2.5.7 TerminatePrimitive

For RANDOM accessing, and when enabled via **Cut Index Enable**, a fetched IBValue of 'all ones' (0xFF, 0xFFFF, or 0xFFFFFFFF depending on **Index Format**) is interpreted as a 'cut value' and signals the termination of the current primitive and the possible start of the next primitive. This allows the application to specify an instance as a sequence of variable-sized strip primitives (though the cut value applies to any primitive type).

Also, there is an implied primitive termination at the end of each InstanceLoop (and so strip primitives cannot span multiple instances).

In either case, the currently-buffered vertex (if any) is marked with EndPrim and then flushed out to the pipeline.

The next-output vertex (if any) will be marked with StartPrim.

Whenever a primitive delimiter is encountered, the PIDCounterS and PIDCounterR counters are reset to 0. These counters control the incrementing (in PrimitiveInfoGeneration, below) of PrimitiveID within each primitive topology of an instance.

```
    if (PIDCounterS != 0) // There is a buffered vertex
        if (primType == TRISTRIP_ADJ)
            if (PIDCounterS==6 || PIDCounterR==1)
                PrimitiveID++
            endif
        endif
        PrimEnd = TRUE
        OutputBufferedVertex
    endif
    PrimEnd = FALSE
    PrimStart = TRUE
```

2.5.8 VertexCacheLookup

The VertexIndex value is used as the tag value for the VertexCache (see *Vertex Cache* above). If the Vertex Cache is enabled and the VertexIndex value hits in the cache, the VUE handle is read from the cache and inserted into the vertex stream. It is marked with a VCHit boolean to suppress processing (shading) in the VS unit.

Otherwise, for Vertex Cache misses, a VUE handle is obtained to provide storage for the generated vertex data. VertexLoop processing then proceeds to iterate through the VEs to generate the destination VUE data.

2.5.9 VertexElementLoop

The VertexElementLoop generates and stores vertex data in the destination VUE one VE at a time.



2.5.10 SourceElementFetch

The following assumes the VE requires data from a VB, which is the typical case. In the case that the VE is completely comprised of constant and/or auto-generated IDs, the SourceElementFetch and FormatConversion steps are skipped.

The structure index within the VE's selected VB is computed as follows:

```

if (VB is a VERTEXDATA VB)
    VBIndex = VertexIndex
else // INSTANCEDATA VB
    VBIndex = StartInstanceLocation
    if (VB.InstanceDataStepRate > 0)
        VBIndex += InstanceID/VB.InstanceDataStepRate
    endif
endif

```

If VBIndex is invalid (i.e., negative or past **Max Index**), the data returned from the VB fetch is defined to be zero. Otherwise, the address of the source data required for the VE is then computed and the data is read from the VB. The amount of data read from the VB is determined by the **Source Element Format**.

```

if ( (VBIndex<0) || (VBIndex>VB.MaxIndex) )
    srcData = 0
else
    pSrcData = VB.BufferStartingAddress + (VBIndex * VB.BufferPitch) +
    VE.SourceElementOffset
    srcData = MemoryRead( pSrcData, VE.SourceElementFormat )
endif

```

2.5.11 Format Conversion

Once the VE source data has been fetched, it is subjected to format conversion. The output of format conversion is up to 4 32-bit components, each either integer or floating-point (as specified by the **Source Element Format**). See *Sampler* for conversion algorithms.

The following table lists the valid **Source Element Format** selections, along with the format and availability of the converted components (if a component is listed as “-“, it cannot be used as source of a VUE component). **Note:** This table is a subset of the list of supported surface formats defined in the *Sampler* chapter. Please refer to that table as the “master list”. This table is here only to identify the components available (per format) and their format.

Source Element Formats supported in VF Unit

Source Element Surface Format Name	Converted Component Format	Converted Component			
		0	1	2	3
R32G32B32A32_FLOAT	FLOAT	R	G	B	A
R32G32B32A32_SINT	SINT	R	G	B	A
R32G32B32A32_UINT	UINT	R	G	B	A



Source Element	Converted Component				
Surface Format Name	Format	0	1	2	3
R32G32B32A32_UNORM	FLOAT	R	G	B	A
R32G32B32A32_SNORM	FLOAT	R	G	B	A
R64G64_FLOAT	FLOAT	R	G	-	-
R32G32B32A32_SSCALED	FLOAT	R	G	B	A
R32G32B32A32_USCALED	FLOAT	R	G	B	A
R32G32B32_FLOAT	FLOAT	R	G	B	-
R32G32B32_SINT	SINT	R	G	B	-
R32G32B32_UINT	UINT	R	G	B	-
R32G32B32_UNORM	FLOAT	R	G	B	-
R32G32B32_SNORM	FLOAT	R	G	B	-
R32G32B32_SSCALED	FLOAT	R	G	B	-
R32G32B32_USCALED	FLOAT	R	G	B	-
R16G16B16A16_UNORM	FLOAT	R	G	B	A
R16G16B16A16_SNORM	FLOAT	R	G	B	A
R16G16B16A16_SINT	SINT	R	G	B	A
R16G16B16A16_UINT	UINT	R	G	B	A
R16G16B16A16_FLOAT	FLOAT	R	G	B	A
R32G32_FLOAT	FLOAT	R	G	-	-
R32G32_SINT	SINT	R	G	-	-
R32G32_UINT	UINT	R	G	-	-
R32G32_UNORM	FLOAT	R	G	-	-
R32G32_SNORM	FLOAT	R	G	-	-
R64_FLOAT	FLOAT	R	-	-	-
R16G16B16A16_SSCALED	FLOAT	R	G	B	A
R16G16B16A16_USCALED	FLOAT	R	G	B	A
R32G32_SSCALED	FLOAT	R	G	-	-
R32G32_USCALED	FLOAT	R	G	-	-
B8G8R8A8_UNORM	FLOAT	B	G	R	A
R10G10B10A2_UNORM	FLOAT	R	G	B	A
R10G10B10A2_UINT	UINT	R	G	B	A
R10G10B10_SNORM_A2_UNORM	FLOAT	R	G	B	A
R8G8B8A8_UNORM	FLOAT	R	G	B	A
R8G8B8A8_SNORM	FLOAT	R	G	B	A
R8G8B8A8_SINT	SINT	R	G	B	A
R8G8B8A8_UINT	UINT	R	G	B	A
R16G16_UNORM	FLOAT	R	G	-	-
R16G16_SNORM	FLOAT	R	G	-	-
R16G16_SINT	SINT	R	G	-	-
R16G16_UINT	UINT	R	G	-	-
R16G16_FLOAT	FLOAT	R	G	-	-
R11G11B10_FLOAT	FLOAT	R	G	B	-
R32_SINT	SINT	R	-	-	-
R32_UINT	UINT	R	-	-	-
R32_FLOAT	FLOAT	R	-	-	-
R32_UNORM	FLOAT	R	-	-	-
R32_SNORM	FLOAT	R	-	-	-
R10G10B10X2_USCALED	FLOAT	R	G	B	-
R8G8B8A8_SSCALED	FLOAT	R	G	B	A
R8G8B8A8_USCALED	FLOAT	R	G	B	A



Source Element	Converted Component				
Surface Format Name	Format	0	1	2	3
R16G16_SSCALED	FLOAT	R	G	-	-
R16G16_USCALED	FLOAT	R	G	-	-
R32_SSCALED	FLOAT	R	-	-	-
R32_USCALED	FLOAT	R	-	-	-
R8G8_UNORM	FLOAT	R	G	-	-
R8G8_SNORM	FLOAT	R	G	-	-
R8G8_SINT	SINT	R	G	-	-
R8G8_UINT	UINT	R	G	-	-
R16_UNORM	FLOAT	R	-	-	-
R16_SNORM	FLOAT	R	-	-	-
R16_SINT	SINT	R	-	-	-
R16_UINT	UINT	R	-	-	-
R16_FLOAT	FLOAT	R	-	-	-
R8G8_SSCALED	FLOAT	R	G	-	-
R8G8_USCALED	FLOAT	R	G	-	-
R16_SSCALED	FLOAT	R	-	-	-
R16_USCALED	FLOAT	R	-	-	-
R8_UNORM	FLOAT	R	-	-	-
R8_SNORM	FLOAT	R	-	-	-
R8_SINT	SINT	R	-	-	-
R8_UINT	UINT	R	-	-	-
R8_SSCALED	FLOAT	R	-	-	-
R8_USCALED	FLOAT	R	-	-	-
R8G8B8_UNORM	FLOAT	R	G	B	-
R8G8B8_SNORM	FLOAT	R	G	B	-
R8G8B8_SSCALED	FLOAT	R	G	B	-
R8G8B8_USCALED	FLOAT	R	G	B	-
R64G64B64A64_FLOAT	FLOAT	R	G	B	A
R64G64B64_FLOAT	FLOAT	R	G	B	A
R16G16B16_FLOAT	FLOAT	R	G	B	-
R16G16B16_UNORM	FLOAT	R	G	B	-
R16G16B16_SNORM	FLOAT	R	G	B	-
R16G16B16_SSCALED	FLOAT	R	G	B	-
R16G16B16_USCALED	FLOAT	R	G	B	-

2.5.12 DestinationFormatSelection

The **Component Select 0..3** bits are then used to select, on a per-component basis, which destination components will be written and with which value. The supported selections are the converted source component, VertexID, InstanceID, PrimitiveID, the constants 0 or 1.0f, or nothing (VFCOMP_NO_STORE). If a converted component is listed as '-' (not available) in *FormatConversion*, it must not be selected (via VFCOMP_STORE_SRC), or an UNPREDICTABLE value will be stored in the destination component.

The selection process sequences from component 0 to 3. Once a **Component Select** of VFCOMP_NO_STORE is encountered, all higher-numbered **Component Select** settings must also be programmed as VFCOMP_NO_STORE. It is therefore not permitted to have 'holes' in the destination VE.



2.5.13 PrimitiveInfoGeneration

A PrimitiveID value and PrimStart boolean need to be associated with the vertex.

If the vertex is either the first vertex of an instance or the first vertex following a 'cut index', the vertex is marked with PrimStart.

PrimitiveID gets incremented such that subsequent per-object processing (i.e., in the GS or SF/MM) will see an incrementing value associated with each sequential object within an instance. The PrimitiveID associated with the provoking, non-adjacent vertex of an object is applied to the object.

The following pseudocode describe the logic used in the VertexLoop to compute the PrimitiveID value associated with the vertex. Recall that PrimitiveID is reset to 0 at the start of each InstanceLoop.

```

if (PIDCounterS < S[primType])
    PIDCounterS++
else
    if (PIDCounterR < R[primType])
        PIDCounterR++
    else
        PrimitiveID++
        PIDCounterR = 0
    endif
endif
endif

```

Two counters are employed to control the incrementing of PrimitiveID. The counters are compared against two corresponding parameters associated with the primitive topology type.

The PIDCounterS is used to 'skip over' some number (possibly zero) initial vertices of the primitive topology. This counter gets reset to 0 after each primitive is terminated.

Then the PIDCounterR is used to periodically increment the PrimitiveID, where the incrementing interval (vertex count) is topology-specific.

The following table lists the S[] and R[] values associated with each primitive topology type.

PrimTopologyType	S, R	PrimitiveID Outputs
POINTLIST POINTLIST_BF	1, 0	0,1,2,3, ..
LINELIST	1, 1	0,0,1,1,2,2,3,3, ..
LINELIST_ADJ	1, 3	0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3 ..
LINESTRIP LINESTRIP_BF LINESTRIP_CONT	2, 0	0,0,1,2,3, ..
LINESTRIP_ADJ	3, 0	0,0,1,2,3,.. Note: this breaks the usage model (as the initial vertex is the provoking vertex for the closing line, but it has an invalid PrimitiveID of 0), but is effectively a don't care as PrimitiveID is only required for LINELOOP is an OpenGL-only primitive.) The LINELOOP topology is converted to LINESTRIP topology



PrimTopologyType	S, R	PrimitiveID Outputs
		at the beginning of the 3D pipeline.
TRILIST RECTLIST	1, 2	0,0,0,1,1,1,2,2,2,3,3,3,..
TRILIST_ADJ	1, 5	0,0,0,0,0,0,1,1,1,1,1,1,2,2,2,2,2,..
TRISTRIP TRISTRIP_REV	3, 0	0,0,0,1,2,3, ..
TRISTRIP_ADJ	5, 1	0,0,0,0,0,0,1,1,2,2,3,3, ..
TRIFAN TRIFAN_NOSTIPPLE POLYGON	3, 0	0,0,0,1,2,3, ..
QUADLIST	1, 3	0,0,0,0,1,1,1,1,2,2,2,3,3,3,3, .. Note: The QUADLIST topology is converted to POLYGON topology at the beginning of the 3D pipeline.
QUADSTRIP	3, 1	0,0,0,0,1,1,2,2,3,3, .. Note: The QUADSTRIP topology is converted to POLYGON topology at the beginning of the 3D pipeline.
PATCHLIST_n	1,n-1	PATCHLIST_1: 0,1,2,3, .. PATCHLIST_2: 0,0,1,1,2,2,3,3 .. and so on.

2.5.14 URBWrite

The selected destination components are written into the destination VUE starting at **Destination Offset Select**. See the description of 3DPRIMITIVE for restrictions on this field.

2.5.15 OutputBufferedVertex

In order to accommodate 'cut' processing, the VF unit buffers one output vertex. The generation of a new vertex or the termination of a primitive causes the buffered vertex to be output to the pipeline.

2.6 Dangling Vertex Removal

The last functional stage of processing of the 3DPRIMITIVE command is the removal of "dangling" vertices. This includes the discarding of primitive topologies without enough vertices for a single object (e.g., a TRISTRIP with only two vertices), as well as the discarding of trailing vertices that do not form a complete primitive (e.g., the last two vertices of a 5-vertex TRILIST).

This function is best described as a filter operating on the vertex stream emitted from the processing of the 3DPRIMITIVE. The filter inputs the PrimType, PrimStart and PrimEnd values associated with the generated vertices. The filter only outputs primitive topologies without dangling vertices. This requires the filter to (a) be able to buffer some number of vertices, and (b) be able to remove dangling vertices from the pipeline and dereference the associated VUE handles.



2.7 Other Vertex Fetch Functionality

2.7.1 Statistics Gathering

3DSTATE_VF_STATISTICS									
Source:	RenderCS								
Length Bias:	1								
The VF stage tracks two pipeline statistics, the number of vertices fetched and the number of objects generated. VF will increment the appropriate counter for each when statistics gathering is enabled by issuing the 3DSTATE_VF_STATISTICS command with the [Statistics Enable] bit set.									
DWord	Bit	Description							
0	31:29	Instruction Type							
		Default Value:	03h GFXPIPE						
		Format:	Opcode						
	28:27	Instruction Sub-Type							
		Format:	Opcode						
			<table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> <th>Project</th> </tr> </thead> <tbody> <tr> <td>1h</td> <td>Pipelined, Single DWord [Default]</td> <td></td> </tr> </tbody> </table>	Value	Name	Project	1h	Pipelined, Single DWord [Default]	
	Value	Name	Project						
	1h	Pipelined, Single DWord [Default]							
	26:24	Instruction Opcode							
Default Value:		0h 3DSTATE_VF_STATISTICS							
Format:		Opcode							
GFXPIPE[28:27 = 1h, 26:24 = 0h, 23:16 = 0Bh] (Pipelined, Single DWord)									
23:16	Instruction Sub-Opcode								
	Default Value:	0Bh 3DSTATE_VF_STATISTICS							
	Format:	Opcode							
GFXPIPE[28:27 = 1h, 26:24 = 0h, 23:16 = 0Bh] (Pipelined, Single DWord)									
15:1	Reserved								
	Format:	MBZ							
0	Statistics Enable								
	Format:	Enable							
	If ENABLED, VF will increment the pipeline statistics counters IA_VERTICES_COUNT and IA_PRIMITIVES_COUNT for each vertex fetched and each object output, respectively, for 3DPRIMITIVE commands issued subsequently. If DISABLED, these counters will not be incremented for subsequent 3DPRIMITIVE commands.								

2.7.1.1 Vertices Generated

VF will increment the IA_VERTICES_COUNT Register (see Memory Interface Registers in Volume Ia, GPU) for each vertex it fetches, even if that vertex comes from a cache rather than directly from a vertex buffer in memory. Any “dangling” vertices (fetched vertices that are part of an incomplete object) will not be included.



2.7.1.2 Objects Generated

VF will increment the IA_PRIMITIVES_COUNT Register (see Memory Interface Registers in vol1a System Overview) for each object (point, line, triangle, or quadrilateral) that it forwards down the pipeline.

Note: For LINELOOP, the last (closing) line object is not counted.

3. 3D Pipeline – Vertex Shader (VS) Stage

3.1 VS Stage Overview

The VS stage of the 3D Pipeline is used to perform processing (“shading”) of vertices after being assembled and written to the URB by the VF function. The primary function of the VS stage is to pass vertices that miss in the Vertex Cache to VS threads, and then pass the VS thread-generated vertices down the pipeline. Vertices that hit in the Vertex Cache are passed down the pipeline unmodified.

When the VS stage is disabled, vertices flow through the unit unmodified (i.e., as written by the VF unit).

Refer to the *Common 3D FF Unit Functions* subsection in the *3D Overview* chapter for a general description of a 3D pipeline stage, as much of the VS stage operation and control falls under these “common” functions; i.e., most stage state variables and VS thread payload parameters are described in *3D Overview*, and although they are listed here for completeness, that chapter provides the detailed description of the associated functions.

Refer to this chapter for an overall description of the VS stage, and any exceptions the VS stage exhibits with respect to common FF unit functions.

3.1.1 Vertex Caching

The 3D Pipeline employs a Vertex Cache that is shared between the VF and VS units. (See *Vertex Fetch* chapter for additional information). The Vertex Cache may be explicitly DISABLED via the **Vertex Cache Disable** bit in VS_STATE. Even when explicitly ENABLED, the VS unit can (by default) *implicitly* disable and invalidate the Vertex Cache when it detects one of the following conditions:

1. Either VertexID or PrimitiveID is selected as part of the vertex data stored in the URB.
2. Sequential indices are used in the 3DPRIMITIVE command (though this is effectively a don’t care as there wouldn’t be any hits anyway).

The implicit disable will persist as long as one of these conditions persist.

The Vertex Cache is implicitly invalidated between 3DPRIMITIVE commands and between instances within a 3DPRIMITIVE command – therefore use of InstanceID in a Vertex Element is not a condition under which the cache is implicitly disabled.

The following table summarizes the modes of operation of the Vertex Cache:

Vertex Cache	VS Function Enable	Mode of Operation
DISABLED (implicitly or explicitly)	DISABLED	<p>Vertex Cache is not used. VF unit will assemble all vertices and write them into the URB entry supplied by the VS unit. VS unit will pass references to these VUEs down the pipeline unmodified.</p> <p>Usage Model: This is an exceptional condition, only required when the VF-generated vertices contain InstanceID or</p>



Vertex Cache	VS Function Enable	Mode of Operation
		<p><i>PrimitiveID and more than one instance is produced. Otherwise the Vertex Cache should be enabled.</i></p>
<p>DISABLED <i>(implicitly or explicitly)</i></p>	<p>ENABLED</p>	<p><i>Vertex Cache is not used. VF unit will assemble all vertices and write them into the URB entry supplied by the VS unit. VS unit will spawn VS threads to process all vertices, overwriting the input data with the results. The VS unit pass references to these VUEs down the pipeline.</i></p> <p><i>Usage Model: This mode is only used when the VS function is required, but either (a) the input vertex contains InstanceID or PrimitiveID and more than one instance is generated or (b) the VS kernel produces a side effect (e.g., writes to a memory buffer) which requires every vertex to be processed by a VS thread.</i></p>
<p>ENABLED</p>	<p>DISABLED</p>	<p><i>Vertex Cache is used to provide reuse of VF-generated vertices. The VF unit will check the cache and only process (assemble/write) vertices that miss in the cache. In either case, the VS unit will pass references to vertices (that hit or miss) down the pipeline without spawning any VS threads.</i></p> <p><u>Usage Model: Normal operation when the VS function is <i>not</i> required. Note that there may be situations which require the VS function to be used even when not explicitly required by the API. E.g., perspective divide may be required for clip testing.</u></p>
<p>ENABLED</p>	<p>ENABLED</p>	<p><i>Vertex Cache is used to provide reuse of VS-processed vertices. The VF unit will check the cache and only process (assemble/write) vertices that miss in the cache. The VS unit will only process (shade) the vertices that missed in the cache. The VS unit sends references to hit or missed vertices down the pipeline in the correct order.</i></p> <p><u>Usage Model: Normal operation when the VS function is required and use of the Vertex Cache is permissible.</u></p>



3.2 VS Stage Input

As a stage of the 3D pipeline, the VS stage receives inputs from the previous (VF) stage. Refer to *3D Overview* for an overview of the various types of input to a 3D Pipeline stage. The remainder of this subsection describes the inputs specific to the VS stage.

3.2.1 State

A PIPE_CONTROL with Post-Sync Operation set to 1h and a depth stall needs to be sent just prior to any 3DSTATE_VS, 3DSTATE_URB_VS, 3DSTATE_CONSTANT_VS, 3DSTATE_BINDING_TABLE_POINTER_VS, 3DSTATE_SAMPLER_STATE_POINTER_VS command. Only one PIPE_CONTROL needs to be sent before any combination of VS associated 3DSTATE.

3.2.1.1 URB_FENCE

Refer to *3D Overview* for a description of how the VS stage processes this command.

3.2.1.2 3DSTATE_VS

3DSTATE_VS			
Source:		RenderCS	
Length Bias:		2	
Description			Project
The state used by VS is defined with this inline state packet.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		10h 3DSTATE_VS	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	4h Excludes DWord (0,1)	
	Project:	All	
	Format:	=n Total Length - 2	
1	31:6	Kernel Start Pointer	
		Project:	All



3DSTATE_VS			
		Format:	InstructionBaseOffset[31:6]Kernel
		This field specifies the starting location (1st core instruction) of the kernel program run by threads spawned by this FF unit. It is specified as a 64-byte-granular offset from the Instruction Base Address. This field is ignored if VS Function Enable is DISABLED.	
	5:0	Reserved	
		Project:	All
		Format:	MBZ
2	31	Single Vertex Dispatch	
		Project:	All
		Format:	U1 Enumerated type
		This field can be used to force single vertex SIMD4x2 VS threads.	
		Value	Name
			Description
			Project
		0h	Multiple
			Dual vertex SIMD4x2 thread dispatches are allowed.
			All
		1h	Single
			Single vertex SIMD4x2 thread dispatches are forced.
			All
	30	Vector Mask Enable (VME)	
		Project:	All
		When SPF=0, VME specifies which mask to use to initialize the initial channel enables. When SPF=1, VME specifies which mask to use to generate execution channel enables.	
		Value	Name
			Description
			Project
		0h	Dmask
			Channels are enabled based on the dispatch mask
			All
		1h	Vmask
			Channels are enabled based on the vector mask
			All
	29:27	Sampler Count	
		Project:	All
		Specifies how many samplers (in multiples of 4) the vertex shader 0 kernel uses. Used only for prefetching the associated sampler state entries. This field is ignored if VS Function Enable is DISABLED.	
		Value	Name
			Description
			Project
		0h	No Samplers
			no samplers used
			All
		1h	1-4 Samplers
			between 1 and 4 samplers used
			All
		2h	5-8 Samplers
			between 5 and 8 samplers used
			All
		3h	9-12 Samplers
			between 9 and 12 samplers used
			All
		4h	13-16 Samplers
			between 13 and 16 samplers used
			All
	26	Reserved	
		Project:	All
		Format:	MBZ
	25:18	Binding Table Entry Count	
		Project:	All
		Format:	U8
		Specifies how many binding table entries the kernel uses. Used only for prefetching of the binding table entries and associated surface state.	
		Note: For kernels using a large number of binding table entries, it may be wise to set this field to zero to avoid prefetching too many entries and thrashing the state cache.	
		This field is ignored if VS Function Enable is DISABLED.	
		Value	Name
		[0,255]	



3DSTATE_VS			
17	Reserved		
	Project:		
	Format:		MBZ
16	Floating Point Mode		
	Project:	All	
	Format:	U1 enumerated type	
Specifies the initial floating point mode used by the dispatched thread. This field is ignored if VS Function Enable is DISABLED.			
	Value	Name	Description
	0h	IEEE-754	Use IEEE-754 Rules
	1h	Alternate	Use alternate rules
15:14	Reserved		
	Project:		All
	Format:		MBZ
13	Illegal Opcode Exception Enable		
	Project:		All
	Format:		Enable
This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment. This field is ignored if VS Function Enable is DISABLED.			
12	Reserved		
	Format:		MBZ
11:8	Reserved		
	Format:		MBZ
7	Software Exception Enable		
	Format:		Enable
This bit gets loaded into EU CR0.1[13] (note the bit # difference). See Exceptions and ISA Execution Environment. This field is ignored if VS Function Enable is DISABLED.			
6:0	Reserved		
	Format:		MBZ
3	31:10	Scratch Space Base Offset	
		Project:	All
	Format:	GeneralStateOffset[31:10]ScratchSpace	
Specifies the starting location of the scratch space area allocated to this FF unit as a 1K-byte aligned offset from the General State Base Address. If required, each thread spawned by this FF unit will be allocated some portion of this space, as specified by Per-Thread Scratch Space. The computed offset of the thread-specific portion will be passed in the thread payload as Scratch Space Offset. The thread is expected to utilize "stateless" DataPort read/write requests to access scratch space, where the DataPort will cause the General State Base Address to be added to the offset passed in the request header.			
This field is ignored if VS Function Enable is DISABLED.			
9:4	Reserved		
	Project:		All



3DSTATE_VS		
	Format:	MBZ
3:0	Per-Thread Scratch Space	
	Project:	All
	Format:	U4 power of 2 Bytes over 1K Bytes
	<p>Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit. The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space. This field is ignored if VS Function Enable is DISABLED.</p>	
	Value	Name
	[0,11]	indicating [1K Bytes, 2M Bytes]
	Programming Notes	
	<p>This amount is available to the kernel for information only. It will be passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port will ignore it.</p>	
4	Reserved	
	Project:	All
	Format:	MBZ
	Dispatch GRF Start Register for URB Data	
	Format:	U5
	<p>Specifies the starting GRF register number for the URB portion (Constant + Vertices) of the thread payload. This field is ignored if VS Function Enable is DISABLED.</p>	
	Value	Name
	[0,31]	indicating GRF [R0,R31]
Reserved		
Project:	All	
Format:	MBZ	
Vertex URB Entry Read Length		
Project:	All	
Format:	U6	
<p>Specifies the number of pairs of 128-bit vertex elements to be passed into the payload for each vertex. This field is ignored if VS Function Enable is DISABLED.</p> <p>For SIMD4x2 dispatch, each vertex element requires one GRF of payload data, therefore the number of GRFs with vertex data will be double the value programmed in this field.</p>		
Value	Name	
[1,63]		
Programming Notes		
<p>It is UNDEFINED to set this field to 0 indicating no Vertex URB data to be read and passed to the thread.</p>		
Reserved		
Project:	All	



3DSTATE_VS			
	Format:	MBZ	
9:4	Vertex URB Entry Read Offset		
	Project:	All	
	Format:	U6	
	Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB before being included in the thread payload. This offset applies to all Vertex URB entries passed to the thread. This field is ignored if VS Function Enable is DISABLED.		
	Value	Name	
[0,63]			
3:0	Reserved		
	Project:	All	
	Format:	MBZ	
5	31:25	Maximum Number of Threads	
		Format:	U7-1 representing thread count
	Specifies the maximum number of simultaneous threads allowed to be active. Used to avoid using up the scratch space. Programming the value of the max threads over the number of threads based off number of threads supported in the execution units may improve performance since the architecture allows threads to be buffered between the check for max threads and the actual dispatch into the EU. Programming the max values to a number less than the number of threads supported in the execution units may reduce performance. This field is ignored if VS Function Enable is DISABLED.		
	Value	Name	Project
	[0,127]	indicating thread count of [1,128]	
[0,35]	indicating thread count of [1,36]		
24:23	Reserved		
	Format:	MBZ	
22:11	Reserved		
	Project:	All	
	Format:	MBZ	
10	Statistics Enable		
	Project:	All	
	Format:	Enable	
	Description	Project	
	If ENABLED, this FF unit will engage in statistics gathering. See the Statistics Gathering section later in this chapter. If DISABLED, statistics information associated with this FF stage will be left unchanged. This field is used even if VS Function Enable is DISABLED.		
9:3	Reserved		
	Project:	All	
	Format:	MBZ	
2	Reserved		
	Format:	MBZ	
1	Vertex Cache Disable		



3DSTATE_VS															
	<table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Disable</td> </tr> </table> <p>This bit controls the operation of the Vertex Cache. This field is always used. If the Vertex Cache is DISABLED and the VS Function is ENABLED, the Vertex Cache is not used and all incoming vertices will be passed to VS threads.</p> <p>If the Vertex Cache is ENABLED and the VS Function is ENABLED, incoming vertices that do not hit in the Vertex Cache will be passed to VS threads.</p> <p>If the Vertex Cache is ENABLED and the VS Function is DISABLED, input vertices that miss in the Vertex Cache will be assembled and written to the URB, though pass thru the VS stage unmodified (not shaded).</p> <p>The Vertex Cache is invalidated whenever the Vertex Cache becomes DISABLED, whenever the VS Function Enable toggles, between 3DPRIMITIVE commands and between instances within a 3DPRIMITIVE command.</p>	Project:	All	Format:	Disable										
Project:	All														
Format:	Disable														
0	<p>VS Function Enable</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> </table> <table border="1"> <thead> <tr> <th style="text-align: center;">Description</th> <th style="text-align: center;">Project</th> </tr> </thead> <tbody> <tr> <td>If ENABLED, VS threads may be spawned to process VF-generated vertices before the resulting vertices are passed down the pipeline.</td> <td></td> </tr> <tr> <td>If DISABLED, VF-generated vertices will pass thru the VS function and sent down the pipeline unmodified. The Vertex Cache is still available in this mode, if enabled.</td> <td></td> </tr> <tr> <td>If Statistics Enable is ENABLED, VS_INVOCATION_COUNT will increment by 1 for every vertex that passes through the VS stage, even if VS Function Enable is DISABLED.</td> <td></td> </tr> <tr> <td>This field is always used.</td> <td></td> </tr> </tbody> </table>	Project:	All	Format:	Enable	Description	Project	If ENABLED, VS threads may be spawned to process VF-generated vertices before the resulting vertices are passed down the pipeline.		If DISABLED, VF-generated vertices will pass thru the VS function and sent down the pipeline unmodified. The Vertex Cache is still available in this mode, if enabled.		If Statistics Enable is ENABLED, VS_INVOCATION_COUNT will increment by 1 for every vertex that passes through the VS stage, even if VS Function Enable is DISABLED.		This field is always used.	
Project:	All														
Format:	Enable														
Description	Project														
If ENABLED, VS threads may be spawned to process VF-generated vertices before the resulting vertices are passed down the pipeline.															
If DISABLED, VF-generated vertices will pass thru the VS function and sent down the pipeline unmodified. The Vertex Cache is still available in this mode, if enabled.															
If Statistics Enable is ENABLED, VS_INVOCATION_COUNT will increment by 1 for every vertex that passes through the VS stage, even if VS Function Enable is DISABLED.															
This field is always used.															



3.2.1.3 3DSTATE_CONSTANT_VS

3DSTATE_CONSTANT_VS			
Source:	RenderCS		
Length Bias:	2		
This command sets pointers to the push constants for VS unit. The constant data pointed to by this command is loaded into the VS unit's push constant buffer (PCB).			
Programming Notes		Project	
It is invalid to execute this command more than once between 3D_PRIMITIVE commands.			
Constant buffers must be enabled in order from Constant Buffer 0 to Constant Buffer 3 within this command. For example, it is not allowed to enable Constant Buffer 1 by programming a non-zero value in the VS Constant Buffer 1 Read Length without a non-zero value in VS Constant Buffer 0 Read Length.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		15h 3DSTATE_CONSTANT_VS	
Format:		OpCode	
15:8	Reserved		
	Format:	MBZ	
7:0	DWord Length	Project:	All
		Format:	=n Total Length - 2
	Value	Name	Project
	5h	Excludes DWord (0,1) [Default]	
1..6	191:0	Constant Body	
		Project:	
		Format:	3DSTATE_CONSTANT(Body)
Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS			



DWord		Bit	Description
3DSTATE_CONSTANT(Body)			
Project:		All	
Source:		RenderCS	
Default Value:		0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
0	31:16	Constant Buffer 1 Read Length	
		Project:	All
		Format:	U16 read length
		This field specifies the length of the constant data to be loaded from memory in 256-bit units.	
		Programming Notes	
		The sum of all four read length fields must be less than or equal to the size of 64	
		Setting the value of the register to zero will disable buffer 1.	
		If disabled, the Pointer to Constant Buffer 1 must be programmed to zero.	
	15:0	Constant Buffer 0 Read Length	
		Project:	All
		Format:	U16 read length
		This field specifies the length of the constant data to be loaded from memory in 256-bit units.	
		Programming Notes	
		The sum of all four read length fields must be less than or equal to the size of 64	
		Setting the value of the register to zero will disable buffer 0.	
		If disabled, the Pointer to Constant Buffer 0 must be programmed to zero.	
1	31:16	Constant Buffer 3 Read Length	
		Project:	All
		Format:	U16 read length
		This field specifies the length of the constant data to be loaded from memory in 256-bit units.	
		Programming Notes	
		The sum of all four read length fields must be less than or equal to the size of 64	
		Setting the value of the register to zero will disable buffer 3.	
		If disabled, the Pointer to Constant Buffer 3 must be programmed to zero.	
	15:0	Constant Buffer 2 Read Length	
		Project:	All



3DSTATE_CONSTANT(Body)													
	<table border="1"> <tr> <td>Format:</td> <td>U16 read length</td> </tr> </table> <p>This field specifies the length of the constant data to be loaded from memory in 256-bit units.</p> <p style="text-align: center;">Programming Notes</p> <p>The sum of all four read length fields must be less than or equal to the size of 64</p> <p>Setting the value of the register to zero will disable buffer 2.</p> <p>If disabled, the Pointer to Constant Buffer 2 must be programmed to zero.</p>	Format:	U16 read length										
Format:	U16 read length												
2	<table border="1"> <tr> <td>31:5</td> <td>Pointer to Constant Buffer 0</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>GraphicsAddress[31:5]ConstantBuffer</td> </tr> <tr> <td colspan="2">This field points to the location of Constant Buffer 0. The state of INSTPM<CONSTANT_BUFFER Address Offset Disable> determines whether the Dynamic State Base Address is added to this pointer.</td> </tr> <tr> <td colspan="2" style="text-align: center;">Programming Notes</td> </tr> <tr> <td colspan="2">Constant buffers must be allocated in linear (not tiled) graphics memory.</td> </tr> </table>	31:5	Pointer to Constant Buffer 0	Project:	All	Format:	GraphicsAddress[31:5]ConstantBuffer	This field points to the location of Constant Buffer 0. The state of INSTPM<CONSTANT_BUFFER Address Offset Disable> determines whether the Dynamic State Base Address is added to this pointer.		Programming Notes		Constant buffers must be allocated in linear (not tiled) graphics memory.	
	31:5	Pointer to Constant Buffer 0											
Project:	All												
Format:	GraphicsAddress[31:5]ConstantBuffer												
This field points to the location of Constant Buffer 0. The state of INSTPM<CONSTANT_BUFFER Address Offset Disable> determines whether the Dynamic State Base Address is added to this pointer.													
Programming Notes													
Constant buffers must be allocated in linear (not tiled) graphics memory.													
	<table border="1"> <tr> <td>4:0</td> <td>Constant Buffer Object Control State</td> </tr> <tr> <td>Format:</td> <td>MEMORY_OBJECT_CONTROL_STATE</td> </tr> <tr> <td colspan="2">Specifies the memory object control state for all constant buffers defined in this command.</td> </tr> </table>	4:0	Constant Buffer Object Control State	Format:	MEMORY_OBJECT_CONTROL_STATE	Specifies the memory object control state for all constant buffers defined in this command.							
4:0	Constant Buffer Object Control State												
Format:	MEMORY_OBJECT_CONTROL_STATE												
Specifies the memory object control state for all constant buffers defined in this command.													
3	<table border="1"> <tr> <td>31:5</td> <td>Pointer to Constant Buffer 1</td> </tr> <tr> <td>Format:</td> <td>GraphicsAddress[31:5]ConstantBuffer</td> </tr> <tr> <td colspan="2">This field points to the location of Constant Buffer 1.</td> </tr> <tr> <td colspan="2" style="text-align: center;">Programming Notes</td> </tr> <tr> <td colspan="2">Constant buffers must be allocated in linear (not tiled) graphics memory.</td> </tr> </table>	31:5	Pointer to Constant Buffer 1	Format:	GraphicsAddress[31:5]ConstantBuffer	This field points to the location of Constant Buffer 1.		Programming Notes		Constant buffers must be allocated in linear (not tiled) graphics memory.			
	31:5	Pointer to Constant Buffer 1											
Format:	GraphicsAddress[31:5]ConstantBuffer												
This field points to the location of Constant Buffer 1.													
Programming Notes													
Constant buffers must be allocated in linear (not tiled) graphics memory.													
	<table border="1"> <tr> <td>4:0</td> <td>Reserved</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	4:0	Reserved	Project:	All	Format:	MBZ						
4:0	Reserved												
Project:	All												
Format:	MBZ												
4	<table border="1"> <tr> <td>31:5</td> <td>Pointer to Constant Buffer 2</td> </tr> <tr> <td>Format:</td> <td>GraphicsAddress[31:5]ConstantBuffer</td> </tr> <tr> <td colspan="2">This field points to the location of Constant Buffer 2.</td> </tr> <tr> <td colspan="2" style="text-align: center;">Programming Notes</td> </tr> <tr> <td colspan="2">Constant buffers must be allocated in linear (not tiled) graphics memory.</td> </tr> </table>	31:5	Pointer to Constant Buffer 2	Format:	GraphicsAddress[31:5]ConstantBuffer	This field points to the location of Constant Buffer 2.		Programming Notes		Constant buffers must be allocated in linear (not tiled) graphics memory.			
	31:5	Pointer to Constant Buffer 2											
Format:	GraphicsAddress[31:5]ConstantBuffer												
This field points to the location of Constant Buffer 2.													
Programming Notes													
Constant buffers must be allocated in linear (not tiled) graphics memory.													
	<table border="1"> <tr> <td>4:0</td> <td>Reserved</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	4:0	Reserved	Project:	All	Format:	MBZ						
4:0	Reserved												
Project:	All												
Format:	MBZ												
5	<table border="1"> <tr> <td>31:5</td> <td>Pointer to Constant Buffer 3</td> </tr> <tr> <td>Format:</td> <td>GraphicsAddress[31:5]ConstantBuffer</td> </tr> <tr> <td colspan="2">This field points to the location of Constant Buffer 3.</td> </tr> <tr> <td colspan="2" style="text-align: center;">Programming Notes</td> </tr> <tr> <td colspan="2">Constant buffers must be allocated in linear (not tiled) graphics memory.</td> </tr> </table>	31:5	Pointer to Constant Buffer 3	Format:	GraphicsAddress[31:5]ConstantBuffer	This field points to the location of Constant Buffer 3.		Programming Notes		Constant buffers must be allocated in linear (not tiled) graphics memory.			
31:5	Pointer to Constant Buffer 3												
Format:	GraphicsAddress[31:5]ConstantBuffer												
This field points to the location of Constant Buffer 3.													
Programming Notes													
Constant buffers must be allocated in linear (not tiled) graphics memory.													



3DSTATE_CONSTANT(Body)	
4:0	Reserved
	Format: MBZ

3.2.1.4 3DSTATE_PUSH_CONSTANT_ALLOC_VS

3DSTATE_PUSH_CONSTANT_ALLOC_VS	
Source:	RenderCS
Length Bias:	2

This command sets up the URB configuration for VS Push Constant Buffer.

Programming Notes

Programming Restriction:

The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size.

The sum of the constant length programmed in 3DSTATE_CONSTANT_VS must be equal or smaller then the size of the allocated space in the URB including the buffering for half cachelines. See **Push Constant URB Allocation** section for more details.

The 3DSTATE_CONSTANT_VS must be reprogrammed prior to the next 3DPRIMITIVE command after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_VS.

DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	1h GFXPIPE_NONPIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
		Default Value:	12h 3DSTATE_PUSH_CONSTANT_ALLOC_VS
		Format:	OpCode
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	0h Excludes DWord (0,1)	
	Project:	All	
	Format:	=n Total Length - 2	



3DSTATE_PUSH_CONSTANT_ALLOC_VS			
1	31:20	Reserved	
		Format: MBZ	
	19:16	Constant Buffer Offset	
		Format: U4	
		Specifies the offset of the VS constant buffer into the URB.	
		Value	Name
		[0,15]	(0KB - 15KB)
	0h	0KB [Default]	
	15:5	Reserved	
		Format: MBZ	
	4:0	Constant Buffer Size	
		Format: U5	
		Specifies the size of the VS constant buffer. This value will determine the amount of data the command stream can pre-fetch before the buffer is full. Value of zero is only valid when constants are not enabled for VS.	
		Value	Name
[0,15]		(0KB – 15KB) Increments of 1KB	
0h	0KB [Default]		

3.2.2 Input Vertices

Refer to *3D Overview* for a description of the vertex information input to the VS stage.

3.3 SIMD4x2 VS Thread Request Generation

This section describes SIMD4x2 thread request generation, which is the only mode available.

The following discussion assumes the VS Function is ENABLED.

When the Vertex Cache is disabled, the VS unit will pass each pair of incoming vertices to a VS thread. Under certain circumstances (e.g., prior to a state change or pipeline flush) the VS unit will spawn a VS thread to process a single vertex. Note that, in this case, the “unused” vertex slot will be “disabled” via the Execution Mask provided by the VS unit to the subsystem as part of the thread dispatch (See ISA doc). The VS thread will in itself be unaware of the single-vertex case, and therefore a single VS kernel can be used to process one or two vertices. (The performance of single-vertex processing will roughly equal the two-vertex case).

When the Vertex Cache is enabled, the VF unit will detect vertices that hit in the cache and mark these vertices so that they will bypass VS thread processing and be output via a reference to the cached VUE. The VS unit will keep track of these cache-hit vertices as it proceeds to process cache-miss vertices. The VS unit guarantees that vertices will exit the unit in the order they are received. This may require the VS unit to issue single-vertex VS threads to process a cache-miss vertex that has yet to be paired up with another cache-miss vertex (if this condition is preventing the VS unit from producing any output).



3.3.1 Thread Payload

The following table describes the payload delivered to VS threads.

VS Thread Payload (SIMD4x2)

DWord	Bit	Description
R0.7	31	
	30:0	Reserved
R0.6	31:24	Reserved
	23:0	<p>Thread ID: This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time.</p> <p>Format: Reserved for HW Implementation Use.</p>
R0.5	31:10	<p>Scratch Space Offset: Specifies the of the scratch space allocated to the thread, specified as a 1KB-granular offset from the General State Base Address. See Scratch Space Base Offset description in VS_STATE.</p> <p>(See <i>3D Pipeline</i> for further description on scratch space allocation).</p> <p>Format = GeneralStateOffset[31:10]</p>
	9:0	Reserved
	8:0	<p>FFTID: This ID is assigned by the FF unit and used to identify the thread within the set of outstanding threads spawned by the FF unit.</p> <p>Format: Reserved for HW Implementation Use.</p> <p>Format:</p> <p>U7</p> <p>Range:</p> <p>0-127</p>
R0.4	31:5	<p>Binding Table Pointer. Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the Surface State Base Address.</p> <p>Format = SurfaceStateOffset[31:5]</p>
	4:0	Reserved
R0.3	31:5	<p>Sampler State Pointer. Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the General State Base Address or Dynamic State Base Address.</p> <p>Format = DynamicStateOffset[31:5]</p>
	4	Reserved
	3:0	<p>Per Thread Scratch Space: Specifies the amount of scratch space allowed to be</p>



DWord	Bit	Description
		<p>used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space).</p> <p>(See <i>3D Pipeline</i> for further description).</p> <p>Format = U4 power of two (in excess of 10)</p> <p>Range = [0,11] indicating [1K Bytes, 2M Bytes]</p>
R0.2	31:0	Reserved : delivered as zeros (reserved for message header fields)
R0.1	31:16	Reserved
	15:0	<p>URB Return Handle 1: This is the 64B-aligned URB offset where the EU's upper channels (DWords 7:4) results are to be stored.</p> <p>If only one vertex is to be processed (shaded) by the thread, this field will effectively be ignored (no results are stored for these channels, as controlled by the thread's Channel Mask).</p> <p>(See <i>Generic FF Unit</i> for further description).</p> <p>Format: U12 64B-aligned offset</p>
R0.0	31:16	Reserved
	15:0	<p>URB Return Handle 0: This is the 64B-aligned URB offset where the EU's lower channels (DWords 3:0) results are to be stored.</p> <p>(See <i>Generic FF Unit</i> for further description).</p> <p>Format: U12 64B-aligned offset</p>
[Varies] optional	255:0	<p>Constant Data (optional) :</p> <p>Some amount of constant data (possible none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_VS command (taking the buffer enables into account).</p> <p>The Constant Data arrives in a non-interleaved format.</p>
Varies	255:0	<p>Vertex Data : Data from (possibly) one or (more typically) two Vertex URB Entries is passed to the thread in the thread payload. The Vertex URB Entry Read Offset and Vertex URB Entry Read Length state variables define the regions of the URB entries that are read from the URB and passed in the thread payload. These SVs can be used to provide a subset of the URB data as required by SW.</p> <p>The vertex data is laid out in the thread header in an interleaved format. The lower DWords (0-3) of these GRF registers always contain data from a Vertex URB Entry. The upper DWords (4-7) may contain data from another Vertex URB Entry. This allows two vertices to be processed (shaded) in parallel SIMD8 fashion. The VS kernel is not aware of the validity of the upper vertex.</p>



3.4 SIMD4x2 VS Thread Execution

This section describes SIMD4x2 thread execution.

A VS kernel (with one exception mentioned below) assumes it is to operate on two vertices in parallel. Input data is either passed directly in the thread payload (including the input vertex data) or indirectly via pointers passed in the payload.

Refer to *ISA* chapters for specifics on writing kernels that operate in SIMD4x2 fashion.

Refer to 3D Pipeline Stage Overview (*3D Overview*) for information on FF-unit/Thread interactions.

In the (unlikely) event that the VS kernel needs to determine whether it is processing one or two vertices, the kernel can compare the **URB Return Handle 0** and **URB Return Handle 1** fields of the thread payload. These fields will be different if two vertices are being processed, and identical if one vertex is being processed. An example of when this test may be required is if the kernel outputs some vertex-dependent results into a memory buffer – without the test the single vertex case might incorrectly output two sets of results. Note that this is not the case for writing the URB destinations, as the Execution Mask will prevent the write of an undefined output.

Note: Prior to sending an End Of Thread, the kernel must dispatch a write commit cycle, if there were any previous writes to memory that had caused no dependency checks.

3.4.1 Vertex Output

VS threads must always write the destination URB handles passed in the payload. VS threads are not permitted to request additional destination handles. Refer to 3D Pipeline Stage Overview (*3D Overview*) for details on how destination vertices are written and any required contents/formats.

3.4.2 Thread Termination

VS threads must signal thread termination, in all likelihood on the last message output to the URB shared function. Refer to the *ISA* doc for details on End-Of-Thread indication.

3.5 Primitive Output

The VS unit will produce an output vertex reference for every input vertex reference received from the VF unit, in the order received. The VS unit simply copies the PrimitiveType, StartPrim, and EndPrim information associated with input vertices to the output vertices, and does not use this information in any way. Neither does the VS unit perform any readback of URB data.

3.6 Other VS Functions

3.6.1 Statistics Gathering

The VS stage tracks a single pipeline statistic, the number of times a vertex shader is executed. A vertex shader is executed for each vertex that is fetched on behalf of a 3DPRIMITIVE command, unless the shaded results for that vertex are already available in the vertex cache. If the **Statistics Enable** bit in VS_STATE is set, the VS_INVOCATION_COUNT Register (see Memory Interface Registers in Volume Ia, *GPU*) will be incremented for *each vertex* that is dispatched to a VS thread. This counter will often



need to be incremented by 2 for each thread invoked since 2 vertices are dispatched to one VS thread in the general case.

When **VS Function Enable** is DISABLED and **Statistics Enable** is ENABLED, VS_INVOCATION_COUNT will increment by one for every vertex that passes through the VS stage, even though no VS threads are spawned.



4. 3D Pipeline – Hull Shader (HS) Stage

4.1 HS Stage Overview

The Hull Shader (HS) stage of the pipeline is used to process patchlist (PATCHLIST_ *n*) topologies in support of higher-order surface (HOS) tessellation. If the HS stage is enabled, each incoming patch object is processed by a possible series of HS threads. The combined output of these threads is a Patch URB Entry (“patch record”) written to the URB. This patch record is used by subsequent stages (TE, DS) to complete the HOS tessellation operations.

For SW Tessellation mode, the HS thread can also write tessellated domain point topologies to memory. The domain point count and starting memory address of the domain points is passed via the Patch Header in the patch record.

The vertices associated with patchlist primitives are also referred to as “Input Control Points” (ICPs) to contrast them with any “Output Control Points” the HS threads may write to the patch record. (The definition and use of OCPs are outside the scope of this document).

The HS stage also performs statistics counting. Incomplete topologies will not reach the HS stage.

The HS, TE and DS stages must be enabled and disabled together. When these stages are disabled, all topologies (including patchlist topologies) will be simply pass through to the GS stage. When these stages are enabled, only patchlist topologies should be issued to the pipeline, otherwise behavior is UNDEFINED.

4.2 HS Stage Input

4.2.1 State

4.2.1.1 3DSTATE_CONSTANT_HS

3DSTATE_CONSTANT_HS		
Source:		RenderCS
Length Bias:		2
This command sets pointers to the push constants for the HS unit. The constant data pointed to by this command is loaded into the HS unit’s push constant buffer (PCB).		
Programming Notes		Project
It is invalid to execute this command more than once between 3D_PRIMITIVE commands.		
Constant buffers must be enabled in order from Constant Buffer 0 to Constant Buffer 3 within this command. For example, It is not allowed to enable Constant Buffer 1 by programming a non-zero value in the HS Constant Buffer 1 Read Length without a non-zero value in HS Constant Buffer 0 Read Length.		
DWord	Bit	Description
0	31:29	Command Type
		Default Value: 3h GFXPIPE
	Format: OpCode	
	28:27	Command SubType



3DSTATE_CONSTANT_HS								
		Default Value: 3h Format: OpCode						
26:24		3D Command Opcode Default Value: 0h 3DSTATE_PIPELINED Format: OpCode						
23:16		3D Command Sub Opcode Default Value: 19h 3DSTATE_CONSTANT_HS Format: OpCode						
15:8		Reserved Format: MBZ						
7:0		DWord Length Project: All Format: =n Total Length - 2						
		<table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> <th>Project</th> </tr> </thead> <tbody> <tr> <td>5h</td> <td>Excludes DWord (0,1) [Default]</td> <td></td> </tr> </tbody> </table>	Value	Name	Project	5h	Excludes DWord (0,1) [Default]	
Value	Name	Project						
5h	Excludes DWord (0,1) [Default]							
1..6	191:0	Constant Body Format: 3DSTATE_CONSTANT(Body) Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS						



4.2.1.2 3DSTATE_PUSH_CONSTANT_ALLOC_HS

DWord		Bit	Description
3DSTATE_PUSH_CONSTANT_ALLOC_HS			
Project:		All	
Source:		RenderCS	
Length Bias:		2	
This command sets up the URB configuration for HS Push Constant Buffer.			
Programming Notes			
Programming Restriction:			
<p>The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size.</p> <p>The sum of the constant length programmed in 3DSTATE_CONSTANT_HS must be equal or smaller then the size of the allocated space in the URB including the buffering for half cachelines. See Push Constant URB Allocation section for more details.</p> <p>The 3DSTATE_CONSTANT_HS must be reprogrammed prior to the next 3DPRIMITIVE command after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_HS.</p>			
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	1h GFXPIPE_NONPIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
		Default Value:	13h 3DSTATE_PUSH_CONSTANT_ALLOC_HS
		Format:	OpCode
	15:8	Reserved	
		Project:	All
		Format:	MBZ
	7:0	DWord Length	
		Default Value:	0h Excludes DWord (0,1)
		Project:	All
		Format:	=n Total Length - 2
1	31:20	Reserved	
		Format:	MBZ
	19:16	Constant Buffer Offset	



3DSTATE_PUSH_CONSTANT_ALLOC_HS							
	Format: U5 Specifies the offset of the HS constant buffer into the URB.						
	<table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>[0,15]</td> <td>(0KB - 15KB)</td> </tr> <tr> <td>0h</td> <td>0KB [Default]</td> </tr> </tbody> </table>	Value	Name	[0,15]	(0KB - 15KB)	0h	0KB [Default]
Value	Name						
[0,15]	(0KB - 15KB)						
0h	0KB [Default]						
15:5	Reserved						
	Format: MBZ						
4:0	Constant Buffer Size						
	Format: U5						
	Specifies the size of the HS constant buffer. This value will determine the amount of data the command stream can pre-fetch before the buffer is full. Value of zero is only valid when constants are not enabled for HS.						
	<table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>[0,15]</td> <td>(0KB – 15KB) Increments of 1KB</td> </tr> <tr> <td>0h</td> <td>0KB [Default]</td> </tr> </tbody> </table>	Value	Name	[0,15]	(0KB – 15KB) Increments of 1KB	0h	0KB [Default]
Value	Name						
[0,15]	(0KB – 15KB) Increments of 1KB						
0h	0KB [Default]						

4.3 3DSTATE_CONSTANT_HS

3DSTATE_CONSTANT_HS	
Source:	RenderCS
Length Bias:	2
This command sets pointers to the push constants for the HS unit. The constant data pointed to by this command is loaded into the HS unit's push constant buffer (PCB).	
Programming Notes	
It is invalid to execute this command more than once between 3D_PRIMITIVE commands.	
Constant buffers must be enabled in order from Constant Buffer 0 to Constant Buffer 3 within this command. For example, It is not allowed to enable Constant Buffer 1 by programming a non-zero value in the HS Constant Buffer 1 Read Length without a non-zero value in HS Constant Buffer 0 Read Length.	
DWord	Bit
0	31:29
Command Type	
Default Value: 3h GFXPIPE	
Format: OpCode	
	28:27
Command SubType	
Default Value: 3h	
Format: OpCode	
	26:24
3D Command Opcode	
Default Value: 0h 3DSTATE_PIPELINED	
Format: OpCode	
	23:16
3D Command Sub Opcode	
Default Value: 19h 3DSTATE_CONSTANT_HS	
Format: OpCode	



3DSTATE_CONSTANT_HS								
	15:8	Reserved						
		Format: MBZ						
	7:0	DWord Length						
		Project: All						
		Format: =n Total Length - 2						
		<table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> <th>Project</th> </tr> </thead> <tbody> <tr> <td>5h</td> <td>Excludes DWord (0,1) [Default]</td> <td></td> </tr> </tbody> </table>	Value	Name	Project	5h	Excludes DWord (0,1) [Default]	
Value	Name	Project						
5h	Excludes DWord (0,1) [Default]							
1..6	191:0	Constant Body Format: 3DSTATE_CONSTANT(Body) Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS						

3DSTATE_CONSTANT(Body)		
Project: All		
Source: RenderCS		
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000		
DWord	Bit	Description
0	31:16	Constant Buffer 1 Read Length
		Project: All Format: U16 read length This field specifies the length of the constant data to be loaded from memory in 256-bit units. Programming Notes The sum of all four read length fields must be less than or equal to the size of 64 Setting the value of the register to zero will disable buffer 1. If disabled, the Pointer to Constant Buffer 1 must be programmed to zero.
	15:0	Constant Buffer 0 Read Length
		Project: All Format: U16 read length This field specifies the length of the constant data to be loaded from memory in 256-bit units. Programming Notes



3DSTATE_CONSTANT(Body)					
	<p>The sum of all four read length fields must be less than or equal to the size of 64</p> <p>Setting the value of the register to zero will disable buffer 0.</p> <p>If disabled, the Pointer to Constant Buffer 0 must be programmed to zero.</p>				
1	31:16 Constant Buffer 3 Read Length <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U16 read length</td> </tr> </table> <p>This field specifies the length of the constant data to be loaded from memory in 256-bit units.</p> <p style="text-align: center;">Programming Notes</p> <p>The sum of all four read length fields must be less than or equal to the size of 64</p> <p>Setting the value of the register to zero will disable buffer 3.</p> <p>If disabled, the Pointer to Constant Buffer 3 must be programmed to zero.</p>	Project:	All	Format:	U16 read length
	Project:	All			
Format:	U16 read length				
15:0 Constant Buffer 2 Read Length <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U16 read length</td> </tr> </table> <p>This field specifies the length of the constant data to be loaded from memory in 256-bit units.</p> <p style="text-align: center;">Programming Notes</p> <p>The sum of all four read length fields must be less than or equal to the size of 64</p> <p>Setting the value of the register to zero will disable buffer 2.</p> <p>If disabled, the Pointer to Constant Buffer 2 must be programmed to zero.</p>	Project:	All	Format:	U16 read length	
Project:	All				
Format:	U16 read length				
2	31:5 Pointer to Constant Buffer 0 <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>GraphicsAddress[31:5]ConstantBuffer</td> </tr> </table> <p>This field points to the location of Constant Buffer 0. The state of INSTPM<CONSTANT_BUFFER Address Offset Disable> determines whether the Dynamic State Base Address is added to this pointer.</p> <p style="text-align: center;">Programming Notes</p> <p>Constant buffers must be allocated in linear (not tiled) graphics memory.</p>	Project:	All	Format:	GraphicsAddress[31:5]ConstantBuffer
	Project:	All			
Format:	GraphicsAddress[31:5]ConstantBuffer				
4:0 Constant Buffer Object Control State <table border="1"> <tr> <td>Format:</td> <td>MEMORY_OBJECT_CONTROL_STATE</td> </tr> </table> <p>Specifies the memory object control state for all constant buffers defined in this command.</p>	Format:	MEMORY_OBJECT_CONTROL_STATE			
Format:	MEMORY_OBJECT_CONTROL_STATE				



3DSTATE_CONSTANT(Body)		
3	31:5	Pointer to Constant Buffer 1
		Format: GraphicsAddress[31:5]ConstantBuffer
		This field points to the location of Constant Buffer 1.
		Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	Reserved
		Project: All Format: MBZ
4	31:5	Pointer to Constant Buffer 2
		Format: GraphicsAddress[31:5]ConstantBuffer
		This field points to the location of Constant Buffer 2.
		Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	Reserved
		Project: All Format: MBZ
5	31:5	Pointer to Constant Buffer 3
		Format: GraphicsAddress[31:5]ConstantBuffer
		This field points to the location of Constant Buffer 3.
		Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	Reserved
		Format: MBZ

4.4 3DSTATE_HS

The state used by HS is defined with the following 3DSTATE_HS inline state packet.

3DSTATE_HS		
Source:	RenderCS	
Length Bias:	2	
Controls the HS stage hardware.		
DWord	Bit	Description
0	31:29	Command Type
		Default Value: 3h GFXPIPE
		Format: OpCode
28:27	28:27	Command SubType
		Default Value: 3h GFXPIPE_3D
		Format: OpCode
26:24	26:24	3D Command Opcode
		Default Value: 0h 3DSTATE_PIPELINED



3DSTATE_HS				
	Format:	OpCode		
23:16	3D Command Sub Opcode			
	Default Value:	1Bh 3DSTATE_HS		
	Format:	OpCode		
15:8	Reserved			
	Project:	All		
	Format:	MBZ		
7:0	DWord Length			
	Format:	=n		
	Value	Name	Project	
	5	Excludes DWord (0,1) [Default]		
1	31:30	Reserved		
		Project:	All	
		Format:	MBZ	
	29:27	Sampler Count		
		Project:	All	
		Format:	U3	
		Specifies how many samplers (in multiples of 4) the HS kernels use. Used only for prefetching the associated sampler state entries.		
		Value	Name	Description
		0h	No Samplers	no samplers used
		1h	1-4 Samplers	between 1 and 4 samplers used
		2h	5-8 Samplers	between 5 and 8 samplers used
		3h	9-12 Samplers	between 9 and 12 samplers used
		4h	13-16 Samplers	between 13 and 16 samplers used
		5h-7h	Reserved	Reserved
		26	Reserved	
		Project:	All	
		Format:	MBZ	
25:18	Binding Table Entry Count			
	Project:	All		
	Format:	U8		
	When HW Generated Binding Table is disabled: Specifies how many binding table entries the kernel uses. Used only for prefetching of the binding table entries and associated surface state. Note: For kernels using a large number of binding table entries, it may be wise to set this field to zero to avoid prefetching too many entries and thrashing the state cache.			
17	Reserved			
	Format:	MBZ		
16	Floating Point Mode			
	Project:	All		
	Specifies the initial floating point mode used by the dispatched thread.			
	Value	Name	Description	
			Project	



3DSTATE_HS			
	0h	IEEE-754	Use IEEE-754 Rules
	1h	alternate	Use alternate rules
15:14	Reserved		
	Project:		All
	Format:		MBZ
13	Illegal Opcode Exception Enable		
	Project:		All
	Format:		Enable
	This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment.		
12	Reserved		
	Format:		MBZ
11:8	Reserved		
	Project:		All
	Format:		MBZ
7	Software Exception Enable		
	Format:		Enable
	This bit gets loaded into EU CR0.1[13] (note the bit # difference). See Exceptions and ISA Execution Environment.		
6:0	Maximum Number of Threads		
	Format:		U7-1 thread count
	Specifies the maximum number of simultaneous threads allowed to be active. Used to avoid using up the scratch space. Programming the value of the max threads over the number of threads based off number of threads supported in the execution units may improve performance since the architecture allows threads to be buffered between the check for max threads and the actual dispatch into the EU. Programming the max values to a number less than the number of threads supported in the execution units may reduce performance. Limit is based on max number of HS URB handles.		
	Value	Name	Project
	[0,127]	indicating a thread count of [1,128]	
	[0,35]	indicating a thread count of [1,36]	
	Programming Notes		
	A URB_FENCE command must be issued subsequent to any change to the value in this field and before any subsequent pipeline processing (e.g., via 3DPRIMITIVE or CONSTANT_BUFFER). See Graphics Processing Engine (Command Ordering Rules)		
2	31	HS Enable	
	Project:		All
	Format:		Enable
	Specifies whether the HS function is enabled or disabled (pass-through). If ENABLED MI_TOPOLOGY_FILTER must be used to silently discard any topologies that the HS kernel is not expecting. E.g., if the HS kernel is expecting PATCHLIST_32 topologies, MI_TOPOLOGY_FILTER		



3DSTATE_HS			
		must be set to PATCHLIST_32 so only those topologies can reach the enabled HS.	
		Programming Notes	
		The tessellation stages (HS, TE and DS) must be enabled/disabled as a group. I.e., draw commands can only be issued if all three stages are enabled or all three stages are disabled, otherwise the behavior is UNDEFINED.	
	30	Reserved	
		Project:	All
		Format:	MBZ
	29	HS Statistics Enable	
		Project:	All
		Format:	Enable
		This bit controls whether HS-unit-specific statistics register(s) will increment (for each patch).	
		Value	Name
		Description	Project
		0h	Disable
		1h	Enable
	28:18	Reserved	
		Project:	All
		Format:	MBZ
	16:8	Reserved	
		Format:	MBZ
	7:4	Reserved	
		Project:	All
		Format:	MBZ
	3:0	Instance Count	
		Project:	All
		Format:	U4-1
		This field determines the number of threads (minus one) spawned per input patch. If the HS kernel uses a barrier function, software must restrict the Instance Count to the number of threads that can be simultaneously active within a half-slice. Factors which must be considered includes scratch memory availability.	
		Value	Name
		Description	
		[0,15]	representing [1,16] instances
3	31:6	Kernel Start Pointer	
		Project:	All
		Format:	InstructionBaseOffset[31:6]Kernel
		This field specifies the starting location (1st GEN core instruction) of the kernel program run by threads spawned by this FF unit. It is specified as a 64-byte-granular offset from the Instruction Base Address.	
	5:0	Reserved	
		Project:	All
		Format:	MBZ
4	31:10	Scratch Space Base Pointer	
		Format:	GeneralStateOffset[31:10]
		Format:	GraphicsAddress[31:0]
		Specifies the location of the scratch space area allocated to this FF unit, specified as a 1KB-granular offset from the General State Base Address. If required, each thread spawned by this FF unit will be	



3DSTATE_HS				
		allocated some portion of this space, as specified by Per-Thread Scratch Space.		
9:4	Reserved			
	Format:		MBZ	
	Per-Thread Scratch Space			
	Format:		U4 power of 2 Bytes over 1K Bytes	
Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit. The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space.				
	Value	Name		
	[0,11]	indicating [1K Bytes, 2M Bytes]		
5	Reserved			
	Format:		MBZ	
	Single Program Flow (SPF)			
	Specifies the initial condition of the kernel program as either a single program flow (SIMDn _{xm} with m = 1) or as multiple program flows (SIMDn _{xm} with m > 1). See CR0 description in ISA Execution Environment.			
	Value	Name	Description	Project
	0h	Reserved		All
	1h	Enable	Single Program Flow enabled	All
	Vector Mask Enable			
	Format:		U1 FormatDesc: Enumerated Type	
	When SPF=0, VME specifies which mask to use to initialize the initial channel enables. When SPF=1, VME specifies which mask to use to generate execution channel enables.			
Value	Name	Description	Project	
0h	Dmask	Channels are enabled based on the dispatch mask	All	
1h	Vmask	Channels are enabled based on the vector mask	All	
25	Reserved			
	Format:		MBZ	
24	Include Vertex Handles			
	Format:		Boolean	
	If set, all the input Vertex URB handles are included in payloads. This field is ignored if HS Function Enable is DISABLED. Programming Restriction: This field must be set if value if Vertex URB Entry Read Length is cleared to zero.			
23:19	Dispatch GRF Start Register for URB Data			



3DSTATE_HS											
	<table border="1"> <tr> <td>Format:</td> <td>U5</td> </tr> <tr> <td colspan="2">Specifies the starting GRF register number for the URB portion (Constant + Vertices) of the thread payload. This field is ignored if HS Function Enable is DISABLED.</td> </tr> <tr> <td style="text-align: center;">Value</td> <td style="text-align: center;">Name</td> </tr> <tr> <td>[0,31]</td> <td>indicating GRF [R0,R31]</td> </tr> </table>	Format:	U5	Specifies the starting GRF register number for the URB portion (Constant + Vertices) of the thread payload. This field is ignored if HS Function Enable is DISABLED.		Value	Name	[0,31]	indicating GRF [R0,R31]		
Format:	U5										
Specifies the starting GRF register number for the URB portion (Constant + Vertices) of the thread payload. This field is ignored if HS Function Enable is DISABLED.											
Value	Name										
[0,31]	indicating GRF [R0,R31]										
18:17	<table border="1"> <tr> <td colspan="2">Reserved</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Reserved		Format:	MBZ						
Reserved											
Format:	MBZ										
16:11	<table border="1"> <tr> <td colspan="2">Vertex URB Entry Read Length</td> </tr> <tr> <td>Format:</td> <td>U6</td> </tr> <tr> <td colspan="2">Specifies the amount of URB data read and passed in the thread payload for each Vertex URB entry, in 256-bit register increments. This field is ignored if HS Function Enable is DISABLED. Programming Restriction: This field must be a non-zero value if Include Vertex Handles is cleared to zero.</td> </tr> <tr> <td style="text-align: center;">Value</td> <td style="text-align: center;">Name</td> </tr> <tr> <td>[0,63]</td> <td></td> </tr> </table>	Vertex URB Entry Read Length		Format:	U6	Specifies the amount of URB data read and passed in the thread payload for each Vertex URB entry, in 256-bit register increments. This field is ignored if HS Function Enable is DISABLED. Programming Restriction: This field must be a non-zero value if Include Vertex Handles is cleared to zero.		Value	Name	[0,63]	
Vertex URB Entry Read Length											
Format:	U6										
Specifies the amount of URB data read and passed in the thread payload for each Vertex URB entry, in 256-bit register increments. This field is ignored if HS Function Enable is DISABLED. Programming Restriction: This field must be a non-zero value if Include Vertex Handles is cleared to zero.											
Value	Name										
[0,63]											
10	<table border="1"> <tr> <td colspan="2">Reserved</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Reserved		Format:	MBZ						
Reserved											
Format:	MBZ										
9:4	<table border="1"> <tr> <td colspan="2">Vertex URB Entry Read Offset</td> </tr> <tr> <td>Format:</td> <td>U6</td> </tr> <tr> <td colspan="2">Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB before being included in the thread payload. This offset applies to all Vertex URB entries passed to the thread. This field is ignored if HS Function Enable is DISABLED.</td> </tr> <tr> <td style="text-align: center;">Value</td> <td style="text-align: center;">Name</td> </tr> <tr> <td>[0,63]</td> <td></td> </tr> </table>	Vertex URB Entry Read Offset		Format:	U6	Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB before being included in the thread payload. This offset applies to all Vertex URB entries passed to the thread. This field is ignored if HS Function Enable is DISABLED.		Value	Name	[0,63]	
Vertex URB Entry Read Offset											
Format:	U6										
Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB before being included in the thread payload. This offset applies to all Vertex URB entries passed to the thread. This field is ignored if HS Function Enable is DISABLED.											
Value	Name										
[0,63]											
3:0	<table border="1"> <tr> <td colspan="2">Reserved</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Reserved		Format:	MBZ						
Reserved											
Format:	MBZ										
6	<table border="1"> <tr> <td colspan="2">Reserved</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Reserved		Project:	All	Format:	MBZ				
	Reserved										
Project:	All										
Format:	MBZ										
12	<table border="1"> <tr> <td colspan="2">Reserved</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Reserved		Format:	MBZ						
Reserved											
Format:	MBZ										
11:0	<table border="1"> <tr> <td colspan="2">Semaphore Handle</td> </tr> <tr> <td>Format:</td> <td>URBOffset[17:6]</td> </tr> <tr> <td colspan="2">This is the URB offset pointing to the first of the GS semaphore DWords in the URB. The size of the region is 32 DWs(16 – 512b URB entries). Software is responsible for allocating combined GS and/or HS semaphore Dwords in a single contiguous region of the URB. Software must also make sure the 3D pipeline is IDLE prior to allocating or deallocating the region. The semaphores can be located in an unused area within a FF unit's URB fenced region or an unused area within the Push Constant region.</td> </tr> </table>	Semaphore Handle		Format:	URBOffset[17:6]	This is the URB offset pointing to the first of the GS semaphore DWords in the URB. The size of the region is 32 DWs(16 – 512b URB entries). Software is responsible for allocating combined GS and/or HS semaphore Dwords in a single contiguous region of the URB. Software must also make sure the 3D pipeline is IDLE prior to allocating or deallocating the region. The semaphores can be located in an unused area within a FF unit's URB fenced region or an unused area within the Push Constant region.					
Semaphore Handle											
Format:	URBOffset[17:6]										
This is the URB offset pointing to the first of the GS semaphore DWords in the URB. The size of the region is 32 DWs(16 – 512b URB entries). Software is responsible for allocating combined GS and/or HS semaphore Dwords in a single contiguous region of the URB. Software must also make sure the 3D pipeline is IDLE prior to allocating or deallocating the region. The semaphores can be located in an unused area within a FF unit's URB fenced region or an unused area within the Push Constant region.											



4.5 Patch Object Staging

The HS unit accepts patchlist topologies as a stream of incoming vertices. Depending on the number of vertices per patch object (as specified by the PATCHLIST_*n* topology), the HS thread will assemble each complete patch object and pass it (its vertices, PrimitiveID, etc.) to HS thread(s) as described below.

4.6 HS Thread Payload

4.6.1 SINGLE_PATCH Layout (SINGLE-PATCH Mode)

The following tables show the layout of the payload delivered to HS threads. Refer to 3D Pipeline Stage Overview (*3D Pipeline*) for details on those fields that are common amongst the various pipeline stages.

Patch object vertex (ICP) data can be passed by value (data pushed in the payload) and/or by reference (URB handle pushed in the payload).

SINGLE_PATCH HS Thread Payload

GRF DWord	Bit	Description
R0:7	31:0	Reserved
R0.6	31	Dereference Thread This bit is defined to send back the Handle ID back to HS to dereference the input handles for this thread.
	30:24	Reserved
R0.5	23:0	Thread ID. This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
	31:10	Scratch Space Pointer. Specifies the location of the scratch space allocated to this thread, specified as a 1KB-aligned offset from the General State Base Address . Format = GeneralStateOffset[31:10]
	9:0	Reserved
R0.4	8:0	FFTID. This ID is assigned by the fixed function unit and is relative identifier for the thread. It is used to free up resources used by the thread upon thread completion. Format: Reserved for Implementation Use Format: U7 Range: 0-127
	31:5	Binding Table Pointer: Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the Surface State Base Address . Format = SurfaceStateOffset[31:5]



GRF DWord	Bit	Description
	4:0	Reserved
R0.3	31:5	<p>Sampler State Pointer. Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the General State Base Address or Dynamic State Base Address.</p> <p>Format = DynamicStateOffset[31:5]</p>
	4	Reserved
	3:0	<p>Per Thread Scratch Space. Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space).</p> <p>Programming Notes:</p> <p>This amount is available to the kernel for information only. It will be passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port will ignore it.</p> <p>Format = U4 power of two (in excess of 10)</p> <p>Range = [0,11] indicating [1K Bytes, 2M Bytes]</p>
R0.2	31:24	<p>Semaphore Index. This is a Dword index to be used in URB_ATOMIC commands if the thread is using data pulled from input handles. This information is only required for pull-model vertex inputs and InstanceCount>1.</p> <p>Format = U8</p>
	23	Reserved
	22:16	<p>Instance Number. A patch-relative instance number between 0 and InstanceCount-1.</p> <p>Format = U7</p>
	15:12	<p>Barrier Index. This index is to be used in any BarrierMsgs sent by this thread to the Gateway.</p> <p>Format = U4</p>
	11:0	<p>Semaphore Handle: This is the URB handle pointing to the first HS semaphore DWord in the URB. Software is responsible for statically allocating the semaphore Dwords in the URB. Refer to Semaphore Handle field in 3DSTATE_HS for size of semaphore allocation.</p> <p>Format: U12 64B-aligned URB Offset</p>
R0.1	31:0	<p>Primitive ID. This field contains the Primitive ID associated with the patch.</p> <p>Format: U32</p>
R0.0	31:16	Reserved
	15:0	<p>Patch Data Record URB Return Handle.</p> <p>Format:</p> <p>U12 64B-aligned URB Offset</p>
R1 is only included for dispatches that have Include Vertex Handles enabled.		
R1.7	31:16	ICP 7 Handle ID



GRF DWord	Bit	Description
	15:0	ICP 7 Handle Format:
R1.6	31:16	ICP 6 Handle ID
	15:0	ICP 6 Handle
R1.5	31:16	ICP 5 Handle ID
	15:0	ICP 5 Handle
R1.4	31:16	ICP 4 Handle ID
	15:0	ICP 4 Handle
R1.3	31:16	ICP 3 Handle ID
	15:0	ICP 3 Handle
R1.2	31:16	ICP 2 Handle ID
	15:0	ICP 2 Handle
R1.1	31:16	ICP 1 Handle ID
	15:0	ICP 1 Handle
R1.0	31:16	ICP 0 Handle ID
	15:0	ICP 0 Handle
R2 is only included for dispatches that have Include Vertex Handles enabled and when ICP Count >7		
R2.7	31:16	ICP 15 Handle ID
	15:0	ICP 15 Handle
R2.6	31:16	ICP 14 Handle ID
	15:0	ICP 14 Handle
R2.5	31:16	ICP 13 Handle ID
	15:0	ICP 13 Handle
R2.4	31:16	ICP 12 Handle ID
	15:0	ICP 12 Handle
R2.3	31:16	ICP 11 Handle ID
	15:0	ICP 11 Handle
R2.2	31:16	ICP 10 Handle ID
	15:0	ICP 10 Handle
R2.1	31:16	ICP 9 Handle ID
	15:0	ICP 9 Handle
R2.0	31:16	ICP 8 Handle ID
	15:0	ICP 8 Handle
R3 is only included for dispatches that have Include Vertex Handles enabled and when ICP Count >15		
R3.7	31:16	ICP 23 Handle ID
	15:0	ICP 23 Handle
R3.6	31:16	ICP 22 Handle ID
	15:0	ICP 22 Handle
R3.5	31:16	ICP 21 Handle ID
	15:0	ICP 21 Handle
R3.4	31:16	ICP 20 Handle ID
	15:0	ICP 20 Handle
R3.3	31:16	ICP 19 Handle ID
	15:0	ICP 19 Handle
R3.2	31:16	ICP 18 Handle ID
	15:0	ICP 18 Handle
R3.1	31:16	ICP 17 Handle ID
	15:0	ICP 17 Handle



GRF DWord	Bit	Description
R3.0	31:16	ICP 16 Handle ID
	15:0	ICP 16 Handle
R4 is only included for dispatches that have Include Vertex Handles enabled and when ICP Count >23		
R4.7	31:16	ICP 31 Handle ID
	15:0	ICP 31 Handle
R4.6	31:16	ICP 30 Handle ID
	15:0	ICP 30 Handle
R4.5	31:16	ICP 29 Handle ID
	15:0	ICP 29 Handle
R4.4	31:16	ICP 28 Handle ID
	15:0	ICP 28 Handle
R4.3	31:16	ICP 27 Handle ID
	15:0	ICP 27 Handle
R4.2	31:16	ICP 26 Handle ID
	15:0	ICP 26 Handle
R4.1	31:16	ICP 25 Handle ID
	15:0	ICP 25 Handle
R4.0	31:16	ICP 24 Handle ID
	15:0	ICP 24 Handle
[Varies] optional	255:0	Constant Data (optional): Some amount of constant data (possible none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_HS command (taking the buffer enables into account).
[Varies] optional	255:0	ICP Vertex Data (optional): There can be up to 32 vertices supplied, each with a size defined by the Vertex URB Entry Read Length state. Vertex 0 DWord 0 is located at Rn.0, Vertex 0 DWord 1 is located at Rn.1, etc. Vertex 1 DWord 0 immediately follows the last DWord of Vertex 0, and so on.

4.7 HS Thread Execution

Input to HS threads is comprised of:

- Input Control Points (incoming patch vertices), pushed into the payload and/or passed indirectly via URB handles.
- Push Constants (common to all threads)
- Patch Data handle
- Resources available via binding table entries (accessed through shared functions)
- Miscellaneous payload fields (Instance Number, etc.)

Typically the only output of the HS threads is the Patch URB Entry (patch record). All thread instances for an input patch are passed the same patch record handle. As the (possibly concurrent) threads can both read and write the patch record, it is up to the kernels to ensure deterministic results. One approach would be to use the thread's Instance Number as an index for URB write destinations.



4.7.1 Dispatch Mask

HS threads will be dispatched with the dispatch mask set to 0xFFFF. It is the responsibility of the kernel to modify the execution mask as required (e.g., if operating in SIMD4x2 mode but only the lower half is active, as would happen in one thread if the threads were computing an odd number of OCPs via SIMD4x2 operation).

4.8 ICP Dereferencing

If ICPs are only pushed in HS payloads (i.e., the **Include Vertex Handles** state bit is clear), the ICP handles will automatically be released after the last instance for the patch is dispatched.

If **Include Vertex Handles** is set (the HS thread(s) will be reading ICP data in from the URB, it is the responsibility of the HS thread instances to explicitly dereference all the ICP handles via use of the **Complete** bit in URB_READ_XXX commands.

- If only one instance is used, that instance can dereference the ICP handles as soon as they are no longer needed, by setting **Complete** in the last URB_READ from that handle. Otherwise all (or the remaining) ICP handles need to be explicitly dereferenced via (possibly null-response-length) URB_READ commands prior to thread EOT.
- If more than one instance is spawned, the last-terminating instance is responsible for dereferencing all the ICP handles before it terminates. Instances can detect that they are the last-terminating thread via use of the semaphore allocated to the patch (via the **Semaphore Handle** and **Semaphore Index** payload fields). A URB_ATOMIC_INC operation (URB_ATOMIC command) can be performed on this semaphore by each instance prior to terminating. Only the last-terminating thread will observe the value (InstanceCount – 1) as a return value. After dereferencing all the ICPs, the last-terminating thread must also reset the semaphore to 0 via the URB_ATOMIC_MOV operation.

4.9 Patch URB Entry (Patch Record) Output

For each patch, the HS thread(s) generate a single patch record, starting with a fixed 32B Patch Header. When the final thread instance terminates, the patch record handle is passed down the pipeline to the Tessellation Engine (TE).

4.9.1 Patch Header

The first 8 DWords of the patch record is defined as a “Patch Header”. The Patch Header is written by an HS thread and read by the TE stage. It normally contains up to six **Tessellation Factors** (TFs) that determine how finely the TE stage needs to tessellate a domain (if at all). In SW Tessellation mode, the header contains **Domain Point Count** and **Domain Point Buffer Starting Address** fields which identify the domain points generated by an HS thread. The following diagram shows the fixed layouts of the Patch Header, depending on DomainType and SW Tessellation Mode.



Patch Header (QUAD Domain)

DW ord	Bits	Description
7	31:0	UEQ0 Tessellation Factor Format: FLOAT32
6	31:0	VEQ0 Tessellation Factor Format: FLOAT32
5	31:0	UEQ1 Tessellation Factor Format: FLOAT32
4	31:0	VEQ1 Tessellation Factor Format: FLOAT32
3	31:0	Inside U Tessellation Factor Format: FLOAT32
2	31:0	Inside V Tessellation Factor Format: FLOAT32
1-0	31:0	Reserved : MBZ

Patch Header (TRI Domain)

DW ord	Bits	Description
7	31:0	UEQ0 Tessellation Factor Format: FLOAT32
6	31:0	VEQ0 Tessellation Factor Format: FLOAT32
5	31:0	WEQ0 Tessellation Factor Format: FLOAT32
4	31:0	Inside Tessellation Factor Format: FLOAT32
3-0	31:0	Reserved : MBZ

Patch Header (ISOLINE Domain)

DW ord	Bits	Description
--------	------	-------------



DW ord	Bits	Description
7	31:0	Line Detail Tessellation Factor Format: FLOAT32
6	31:0	Line Density Tessellation Factor Format: FLOAT32
5-0	31:0	Reserved : MBZ

Patch Header (SW Tessellation Mode)

DW ord	Bits	Description
7	31:0	Domain Point Count Specifies the number of DOMAIN_POINT structures in the domain point list in memory. If 0, there are no domain points defined, the patch will considered “culled”, and the TE stage will discard the patch. Otherwise the TS stage will send this number of domain points down the pipeline. Format: U32
6	31:6	Domain Point Buffer Starting Address (DPBSA) This field specifies the starting memory offset from SW Tessellation Base Address (set by the SWTESS_BASE_ADDRESS command) at which the HS thread has written a list of DOMAIN_POINT structures. This field is ignored if Domain Point Count is 0. Format: 64B-aligned offset from SW Tessellation Base Address
	5:0	Reserved : MBZ
5-0	31:0	Reserved: MBZ

4.9.2 DOMAIN_POINT Structure

In SW Tessellation Mode (i.e., when the TE State is SW_TESS), the TE stage will read a sequence of DOMAIN_POINT structures from memory, starting at the Domain Point Buffer Starting Address field of the patch header. (The DPBSA is treated as an offset from the SW Tessellation Base Address as set by the SWTESS_BASE_ADDRESS command).

DOMAIN_POINT Memory Structure (SW Tessellation)

DW ord	Bit	Description
0	31	PrimStart Set on the first domain point of the topology (e.g., first vertex in a TRISTRIP).
	30	PrimEnd Set on the last domain point of the topology (e.g., last vertex in a TRISTRIP). Programming note: Software must ensure that incomplete primitives are not output, or



DW ord	Bit	Description
		behavior is UNDEFINED.
	29	<p>PatchEnd</p> <p>Set on the last domain point for the <u>patch</u>. By definition, PrimEnd must also be set.</p> <p>Programming Note: Software must ensure that the Domain Point Count coincides with the domain point marked with PatchEnd.</p>
	28:24	<p>PrimType</p> <p>This is the primitive topology type.</p> <p>Format: See 3DPRIMITIVE for encodings</p> <p>Valid values: POINTLIST, LINESTRIP, LINELIST, TRISTRIP, TRISTRIP_REV, TRILIST, TRIFAN.</p>
	23:19	Reserved
	18:17	<p>DS Tag [16:15]</p> <p>This field provides bits [16:15] of the DS Tag value for this domain point. See DS Tag [14:0].</p> <p>Format: U2</p>
	16:0	<p>U Coordinate</p> <p>Format: U1.16</p>
1	31:17	<p>DS Tag [14:0]</p> <p>This field provides bits [14:0] of the DS Tag value for this domain point.</p> <p>In order to utilize the DS cache, the 17-bit DS Tag must be unique for the associated U,V coordinate. If software cannot guarantee this, the DS cache must be disabled when in SW Tessellation mode.</p> <p>Format: U15</p>
	16:0	<p>V Coordinate</p> <p>Format: U1.16</p>

4.10 Statistics Gathering

4.10.1 HS Invocations

The HS unit controls the HS_INVOCATIONS counter, which counts the number of patches processed by the HS stage.



5. 3D Pipeline – Tessellation Engine (TE)

When enabled, the Tessellation Engine (TE) stage performs fixed-function domain tessellation (decomposition into smaller objects) of incoming patches, as referenced by an HS-generated input PDR handle and as controlled by TE state and Tessellation Factors (TFs) read from the Patch URB Entry (patch record). The TE stage is entirely fixed-function and does not spawn threads.

The TE stage can also operate in SW Tessellation mode, where it simply reads “pre-tessellated” domain point topologies from memory and passes them down the pipeline.

The fixed-function tessellation algorithm is considered an implementation detail and is therefore beyond the scope of this document. That detail includes both the order of output topologies as well as the order of vertices (domain points) within the output topologies. Only a high-level overview is provided to describe how the (few) state variables can be used to control aspects of tessellation behavior. The implementation will generate deterministic results (given the same exact inputs it will produce exactly the same outputs).

Several domain types (QUAD, TRI, and ISOLINE) are supported. Depending on the domain type, the TE stage outputs the required point/line/triangle topologies including a domain point per vertex. These topologies will be output to the DS stage, where the domain points will be converted to 3D object vertices, resulting in 3D objects as typically input to the 3D pipeline when HOS tessellation is not used.

The HS, TE, and DS stages must be enabled and disabled together. When these stages are disabled, all topologies (including patchlist topologies) simply pass through to the GS stage. When these stages are enabled, only patchlist topologies should be issued to the pipeline, else behavior is UNDEFINED. The MI_TOPOLOGY_FILTER command can be used to ensure this happens, i.e., it can be used to have the Command Stream ignore 3DPRIMITIVE commands that do not match a specific topology type.

5.1 3DSTATE_TE

3DSTATE_TE		
Source:	RenderCS	
Length Bias:	2	
The state used by TE is defined with this inline state packet.		
DWord	Bit	Description
0	31:29	Command Type
		Default Value: 3h GFXPIPE
		Format: OpCode
	28:27	Command SubType
	Default Value: 3h GFXPIPE_3D	
	Format: OpCode	
26:24	3D Command Opcode	
	Default Value: 0h 3DSTATE_PIPELINED	
	Format: OpCode	
23:16	3D Command Sub Opcode	
	Default Value: 1Ch 3DSTATE_TE	
	Format: OpCode	
15:8	Reserved	



3DSTATE_TE			
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	2h Excludes DWord (0,1)	
	Project:	All	
	Format:	=n Total Length - 2	
1	31:14	Reserved	
	Project:	All	
	Format:	MBZ	
13:12	Partitioning		
	Project:	All	
	Format:	U2	
	This field specifies how edges are partitioned based on tessellation factor.		
	Value	Name	Description
	0h	INTEGER	Outside/inside edges are divided into an integer number of equal-sized segments.
	1h	ODD_FRACTIONAL	Outside/inside edges are divided into an odd number of possibly-unequal-sized segments.
	2h	EVEN_FRACTIONAL	Outside/inside edges are divided into an even number of possibly-unequal-sized segments.
11:10	Reserved		
	Project:	All	
	Format:	MBZ	
9:8	Output Topology		
	Project:	All	
	Format:	U2	
	This field specifies which primitive types are to be output.		
	Value	Name	Description
	0h	POINT	Points are output (as POINTLIST topologies)
	1h	LINE	Lines are output (as LINESSTRIP topologies). Only valid if ISOLINE domain is selected.
	2h	TRI_CW	Clockwise-ordered triangles are output (either as TRISTRIP, TRISTRIP_REV or TRILIST topologies). Not valid if ISOLINE domain is selected.
	3h	TRI_CCW	Count-clockwise-ordered triangles are output (either as TRISTRIP, TRISTRIP_REV or TRILIST topologies). Not valid if ISOLINE domain is selected.
7:6	Reserved		
	Project:	All	
	Format:	MBZ	
5:4	Domain		
	Project:	All	
	Format:	U2	
	This field specifies which type of domain is to be tessellated.		
	Value	Name	Description
			Project



3DSTATE_TE			
	0h	QUAD	2D (U,V) domain is tessellated
	1h	TRI	Triangular (U,V,W) domain is tessellated
	2h	ISOLINE	2D (U,V) domain is tessellated.
3	Reserved		
	Project:	All	
	Format:	MBZ	
2:1	TE Mode		
	Project:	All	
	Format:	U2	
	When TE Enable is ENABLED, this field specifies the overall operation of the TE stage. This field is ignored if TE Enable is DISABLED.		
	Value Name	Description	Project
	0h	HW_TESS Normal HW Tessellation Mode. The TessFactors are read from the patch URB entry, and are used to perform fixed-function hardware tessellation of the specified domain.	All
	1h	SW_TESS Software Tessellation Mode. The TE unit will pass down HS-thread-generated tessellated domain points instead of generating them itself from TessFactors. The TE unit will read the Domain Point Count and Domain Point Buffer Starting Address fields from the patch header, and if the count is 0 it will consider the patch culled and discard it. Otherwise the address is used to start fetching DOMAIN_POINT structures from memory and passing them down the pipeline to DS.	All
0	TE Enable		
	Project:	All	
	Format:	Enable	
	If ENABLED, the TE stage will perform tessellation processing on incoming patch primitives. The TE Mode field determines how this tessellation operation proceeds. If DISABLED, the TE goes into pass-through mode. All other state fields are ignored.		
	Programming Notes		
	The tessellation stages (HS, TE and DS) must be enabled/disabled as a group. I.e., draw commands can only be issued if all three stages are enabled or all three stages are disabled, otherwise the behavior is UNDEFINED.		
2	31:0	Max TessFactor Odd	
	Project:	All	
	Format:	FLOAT32	
	This field specifies the maximum TessFactor for ODD_FRACTIONAL partitioning when in HW_TESS mode. Note that ISOLINE's LineDensity TF is always subjected to INTEGER partitioning regardless of the Partitioning state. For normal operation (as per API spec) software should set this value to 63.0.		
3	31:0	Max TessFactor Not Odd	
	Project:	All	
	Format:	FLOAT32	
	This field specifies the maximum TessFactor for EVEN_FRACTIONAL or INTEGER partitioning when in HW_TESS mode. Note that ISOLINE's LineDensity TF is always subjected to INTEGER partitioning regardless of the Partitioning state. For normal operation (as per API spec) software should set this value to 64.0.		

5.2 Domain Types and Output Topologies

The major (if only) task of the TE stage is to tessellate a 2D (u,v) domain region—as selected by the Domain state—into some number of 2D object topologies. (If the patch is culled, that number may be zero). The options for Domain state are as follows:

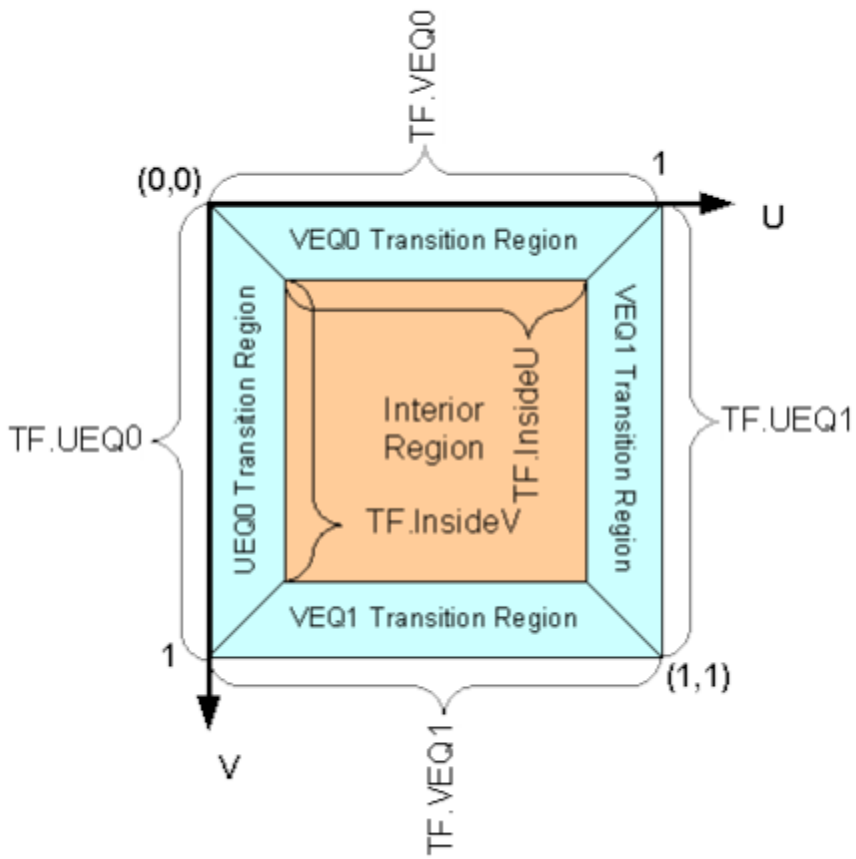
- **QUAD:** A square 2D region within a u,v Cartesian (rectangular) space. The region extends from the origin to $u=1$ and $v=1$. Within the region, tessellation domain locations are determined. The possible output topologies include points, clockwise triangles, and counter-clockwise triangles.
- **TRI:** A triangular 2D region with u,v,w barycentric (areal) coordinates. The three edges correspond to $u=0$, $v=0$, and $w=0$ boundaries. In barycentric coordinates, $w = 1 - u - v$, therefore points within the region are fully defined as 2D (u,v) coordinates. Within the region, tessellation domain locations are determined. The possible output topologies include points, clockwise triangles, and counter-clockwise triangles.
- **ISOLINE:** A series of points within a QUAD domain, where the points lie on lines parallel to the u axis and extending from [0,1) in the v direction. Either the segmented lines (linestrips) or individual point topologies can be output.

5.3 QUAD Domain Tessellation

The four “outside” TFs (TF.UEQ0, TF.VEQ0, TF.UEQ1, TF.VEQ1) are used to specify the level of tessellation along the four corresponding edges of the 2D quad domain. The two “inside” TFs (TF.InsideU, TF.InsideV) are used to determine the level of tessellation within a 2D “interior” region. Typically the interior region appears as a “regularly-tessellated 2D grid”, however under certain conditions the interior region may collapse in which case only the outside TFs are relevant.

In general, a transition region exists between each edge of the interior region and the corresponding outside edge. The topologies generated for these regions effectively “stitch together” locations along the outside and inside edges, as each of these edges can contain a different number of tessellated segments. In the case where all TFs in a given direction (e.g., TF.VEQ0, TF.InsideU, and TF.VEQ1) are the same value, it appears as if the regularly-tessellated interior region extends all the way to the outside edges. If this condition simultaneously exists for both u and v directions, the entire domain will appear to be tessellated into a regular grid, with no noticeable transition regions.

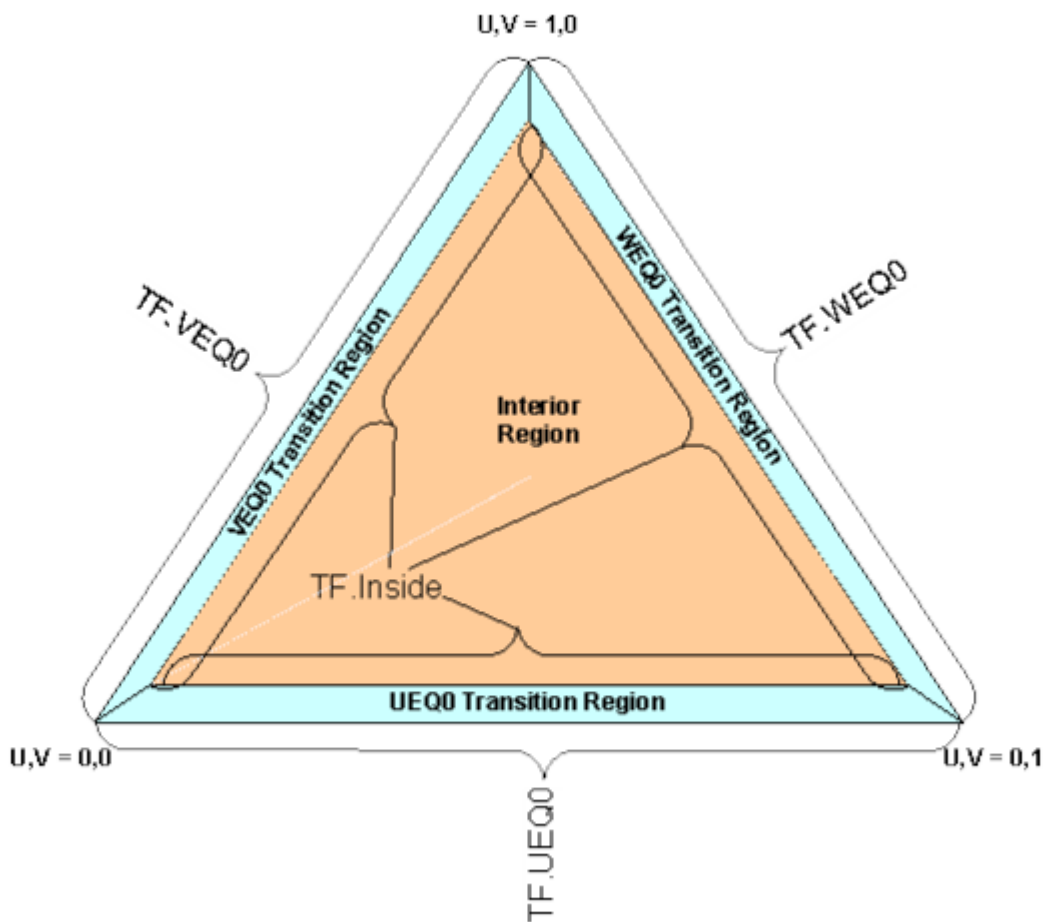
QUAD Domain



5.3.1 TRI Domain Tessellation

Tessellation of the TRI domain is similar to the QUAD domain, except only three outside edges/TFs are used, and the tessellation of the interior region is controlled by a single TF.

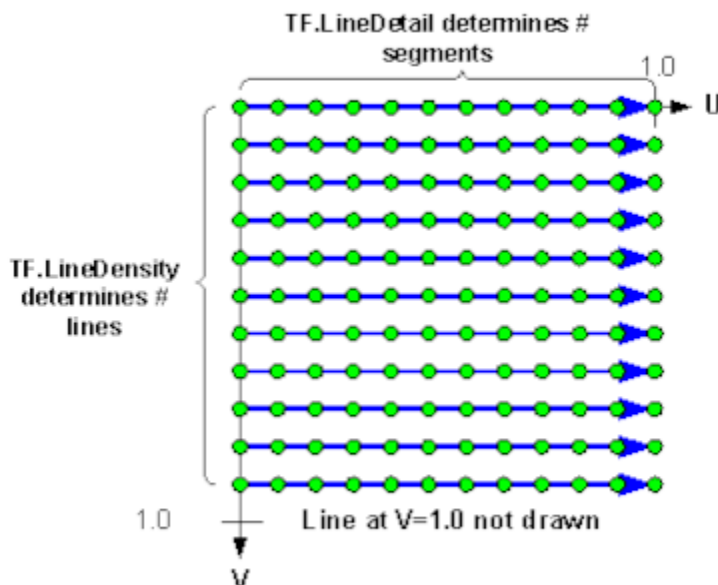
TRI Domain



5.4 ISOLINE Domain Tessellation

Tessellation of the ISOLINE domain is different but much simpler than QUAD and TRI domains. The TF.LineDetail TF controls how finely the U direction is tessellated, while the TF.LineDensity TF controls how finely the V direction is tessellated. When LINE output topology is selected, a series of segmented lines parallel to the U axis (constant V) are output. When POINT output topology is selected, only the line segment endpoints are output (as point objects). In either case there is no topology output for the V=1 edge, which avoids overlapping lines for adjacent patches.

ISOLINE Domain



5.5 Patch Culling

Normally, if any “outside” TF is ≤ 0.0 or NaN, the entire patch is culled at the TE stage.

Inside TFs are not used to cull patches.

In SW Tessellation mode, a Domain Point Count of 0 indicates that a patch is to be culled.

5.6 Tessellation Factor Limits

MinTessFactor and MaxTessFactor state variables are used to perform a floating-point range clamp on the TessFactors.

See TE_STATE for programming details.



5.7 Partitioning

The Partitioning state controls how the TFs are used to divide their corresponding edges.

- **INTEGER:** The edge will be divided into an integral number of equal segments (given some fixed-point tolerance).

After clamping, the TF is rounded up to an integer value. The edge will be divided into that many equal segments.

- **EVEN_FRACTIONAL:** The edge will be divided into an *even* number of possibly-unequal segments. The total number of segments is determined by rounding up the post-clamped TF to an even number.

More specifically, the edge is divided exactly in half. Like the endpoints of the edge, the midpoint of the edge is by definition a tessellation point. Each half will contain some number of equal segments and possibly one smaller segment. The size of the smaller segment is determined by the position of the TF value within the range defined by the TF rounded down and up to even numbers. The closer the TF is to the smaller value, the smaller the segment size is. When the TF reaches the smaller even value, the smaller segment disappears. The closer the TF gets to the larger even value, the closer the smaller segment size approaches the size of the other segments. When the TF reaches the larger even value, all segments will be equal. The position of the smaller segment along the half edge varies as a function of the TF value.

- **ODD_FRACTIONAL:** The edge will be divided into an *odd* number of possibly-unequal segments. The tessellation scheme is very similar to **EVEN_FRACTIONAL** partitioning, except the edge midpoint is not included as a tessellation point. This, and the fact that the tessellation points are mirrored about the edge midpoint, causes an “odd” segment (which may or may not be the “smaller” segment) to straddle the edge midpoint, therefore resulting in the number of segments for the edge always being odd.

6. 3D Pipeline – Domain Shader (DS) Stage

The DS stage is very similar to the VS stage in that it is responsible for dispatching EU threads to shade vertices and maintaining a cache (with reference counts) of the shaded vertex outputs of these threads. Major differences are as follows:

- The DS receives topologies with “domain points” instead of vertices. The only data specific to a domain point are its U,V coordinates. These coordinates (plus a default or computed W coordinate) are passed directly in the DS thread payload. There is no other vertex-specific “input vertex data”
- The concatenation of the domain point U,V coordinates (vs. a vertex index) is used as the cache tag.
- The cache is invalidated between patches.

The DS stage accepts state information via the inline 3DSTATE_DS command.

6.1 3DSTATE_DS

3DSTATE_DS			
Source:		RenderCS	
Length Bias:		2	
The state used by DS is defined with this inline state packet			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		1Dh 3DSTATE_DS	
Format:		OpCode	
15:8	Reserved		
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	4h Excludes DWord (0,1)	
	Format:	=n Total Length - 2	
1	31:6	Kernel Start Pointer	
		Project:	All



3DSTATE_DS					
		Format:	InstructionBaseOffset[31:6]Kernel		
	This field specifies the starting location of the kernel program run by threads spawned by this FF unit. It is specified as a 64-byte-granular offset from the Instruction Base Address. This field is ignored if DS Function Enable is DISABLED.				
	5:0	Reserved			
		Project:	All		
		Format:	MBZ		
2	31	Single Domain Point Dispatch			
		Format:	U1 Enumerated Type		
		This field can be used to force single domain point SIMD4x2 DS threads.			
		Value	Name	Description	
		0h	Multiple	Dual domain point SIMD4x2 thread dispatches are allowed.	
		1h	Single	Single domain point SIMD4x2 thread dispatches are forced.	
	30	Vector Mask Enable (VME)			
		Format:	U1 Enumerated Type		
		When SPF=0, VME specifies which mask to use to initialize the initial channel enables. When SPF=1, VME specifies which mask to use to generate execution channel enables.			
		Value	Name	Description	Project
		0h	Dmask	Channels are enabled based on the dispatch mask	
		1h	Vmask	Channels are enabled based on the vector mask	
29:27	Sampler Count				
	Format:	U3			
	Specifies how many samplers (in multiples of 4) the kernel uses. Used only for prefetching the associated sampler state entries. This field is ignored if DS Function Enable is DISABLED.				
	Value	Name	Description	Project	
		0h	No Samplers	no samplers used	All
		1h	1-4 Samplers	between 1 and 4 samplers used	All
		2h	5-8 Samplers	between 5 and 8 samplers used	All
		3h	9-12 Samplers	between 9 and 12 samplers used	All
		4h	13-16 Samplers	between 13 and 16 samplers used	All
	26	Reserved			
Format:		MBZ			
25:18	Binding Table Entry Count				
	Format:	U8			
	When HW Generated Binding Table is disabled: Specifies how many binding table entries the kernel uses. Used only for prefetching of the binding table entries and associated surface state.				



3DSTATE_DS													
		<p>Note:For kernels using a large number of binding table entries, it may be wise to set this field to zero to avoid prefetching too many entries and thrashing the state cache. This field is ignored if DS Function Enable is DISABLED.</p> <table border="1" style="width: 100%;"> <thead> <tr> <th style="width: 50%; text-align: center;">Value</th> <th style="width: 50%; text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">[0,255]</td> <td></td> </tr> </tbody> </table>	Value	Name	[0,255]								
Value	Name												
[0,255]													
17	Reserved	<table border="1" style="width: 100%;"> <tr> <td style="width: 60%;">Format:</td> <td>MBZ</td> </tr> </table>	Format:	MBZ									
Format:	MBZ												
16	Floating Point Mode	<table border="1" style="width: 100%;"> <tr> <td style="width: 30%;">Format:</td> <td>U1 Enumerated Type</td> </tr> </table> <p>Specifies the initial floating point mode used by the dispatched thread.This field is ignored if DS Function Enable is DISABLED.</p> <table border="1" style="width: 100%;"> <thead> <tr> <th style="width: 30%; text-align: center;">Value</th> <th style="width: 30%; text-align: center;">Name</th> <th style="width: 40%; text-align: center;">Description</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0h</td> <td>IEEE-754</td> <td>Use IEEE-754 Rules</td> </tr> <tr> <td style="text-align: center;">1h</td> <td>Alternate</td> <td>Use alternate rules</td> </tr> </tbody> </table>	Format:	U1 Enumerated Type	Value	Name	Description	0h	IEEE-754	Use IEEE-754 Rules	1h	Alternate	Use alternate rules
Format:	U1 Enumerated Type												
Value	Name	Description											
0h	IEEE-754	Use IEEE-754 Rules											
1h	Alternate	Use alternate rules											
15	Reserved	<table border="1" style="width: 100%;"> <tr> <td style="width: 60%;">Format:</td> <td>MBZ</td> </tr> </table>	Format:	MBZ									
Format:	MBZ												
14	Reserved	<table border="1" style="width: 100%;"> <tr> <td style="width: 60%;">Format:</td> <td>MBZ</td> </tr> </table>	Format:	MBZ									
Format:	MBZ												
13	Illegal Opcode Exception Enable	<table border="1" style="width: 100%;"> <tr> <td style="width: 50%;">Format:</td> <td>Enable</td> </tr> </table> <p>This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment.This field is ignored if DS Function Enable is DISABLED.</p>	Format:	Enable									
Format:	Enable												
12:8	Reserved	<table border="1" style="width: 100%;"> <tr> <td style="width: 60%;">Format:</td> <td>MBZ</td> </tr> </table>	Format:	MBZ									
Format:	MBZ												
7	Software Exception Enable	<table border="1" style="width: 100%;"> <tr> <td style="width: 50%;">Format:</td> <td>Enable</td> </tr> </table> <p>This bit gets loaded into EU CR0.1[13] (note the bit # difference). See Exceptions and ISA Execution Environment. This field is ignored if DS Function Enable is DISABLED.</p>	Format:	Enable									
Format:	Enable												
6:0	Reserved	<table border="1" style="width: 100%;"> <tr> <td style="width: 60%;">Format:</td> <td>MBZ</td> </tr> </table>	Format:	MBZ									
Format:	MBZ												
3	31:10	<p>Scratch Space Base Offset</p> <table border="1" style="width: 100%;"> <tr> <td style="width: 20%;">Format:</td> <td>GeneralStateOffset[31:10]ScratchSpace</td> </tr> </table> <p>Specifies the starting location of the scratch space area allocated to this FF unit as a 1K-byte aligned offset from the General State Base Address. If required, each thread spawned by this FF unit will be</p>	Format:	GeneralStateOffset[31:10]ScratchSpace									
Format:	GeneralStateOffset[31:10]ScratchSpace												



3DSTATE_DS																											
	<p>allocated some portion of this space, as specified by Per-Thread Scratch Space. The computed offset of the thread-specific portion will be passed in the thread payload as Scratch Space Offset. The thread is expected to utilize “stateless” DataPort read/write requests to access scratch space, where the DataPort will cause the General State Base Address to be added to the offset passed in the request header. This field is ignored if DS Function Enable is DISABLED.</p>																										
9:4	<p>Reserved</p> <table border="1" style="width: 100%;"> <tr> <td style="width: 50%;"></td> <td style="width: 50%;"></td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>			Format:	MBZ																						
Format:	MBZ																										
3:0	<p>Per-Thread Scratch Space</p> <table border="1" style="width: 100%;"> <tr> <td style="width: 50%;"></td> <td style="width: 50%;"></td> </tr> <tr> <td>Format:</td> <td>U4 power of 2 Bytes over 1K Bytes</td> </tr> </table> <p>Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit. The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space. This field is ignored if DS Function Enable is DISABLED.</p> <table border="1" style="width: 100%;"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td>[0,11]</td> <td>indicating [1K Bytes, 2M Bytes]</td> </tr> </tbody> </table> <p style="text-align: center;">Programming Notes</p> <p>This amount is available to the kernel for information only. It will be passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port will ignore it.</p>			Format:	U4 power of 2 Bytes over 1K Bytes	Value	Name	[0,11]	indicating [1K Bytes, 2M Bytes]																		
Format:	U4 power of 2 Bytes over 1K Bytes																										
Value	Name																										
[0,11]	indicating [1K Bytes, 2M Bytes]																										
4	<p>31:25 Reserved</p> <table border="1" style="width: 100%;"> <tr> <td style="width: 50%;"></td> <td style="width: 50%;"></td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table> <p>24:20 Dispatch GRF Start Register for URB Data</p> <table border="1" style="width: 100%;"> <tr> <td style="width: 50%;"></td> <td style="width: 50%;"></td> </tr> <tr> <td>Format:</td> <td>U5</td> </tr> </table> <p>Specifies the starting GRF register number for the URB portion (Constant + Vertices) of the thread payload. This field is ignored if DS Function Enable is DISABLED.</p> <table border="1" style="width: 100%;"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td>[0,31]</td> <td>indicating GRF [R0,R31]</td> </tr> </tbody> </table> <p>19:18 Reserved</p> <table border="1" style="width: 100%;"> <tr> <td style="width: 50%;"></td> <td style="width: 50%;"></td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table> <p>17:11 Patch URB Entry Read Length</p> <table border="1" style="width: 100%;"> <tr> <td style="width: 50%;"></td> <td style="width: 50%;"></td> </tr> <tr> <td>Format:</td> <td>U7</td> </tr> </table> <p>Specifies how much data (in 256-bit units) is to be read from the Patch URB entry and passed in the DS thread payload. This field is ignored if DS Function Enable is DISABLED.</p> <table border="1" style="width: 100%;"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td>[0, 64]</td> <td></td> </tr> </tbody> </table> <p>10 Reserved</p> <table border="1" style="width: 100%;"> <tr> <td style="width: 50%;"></td> <td style="width: 50%;"></td> </tr> </table>			Format:	MBZ			Format:	U5	Value	Name	[0,31]	indicating GRF [R0,R31]			Format:	MBZ			Format:	U7	Value	Name	[0, 64]			
Format:	MBZ																										
Format:	U5																										
Value	Name																										
[0,31]	indicating GRF [R0,R31]																										
Format:	MBZ																										
Format:	U7																										
Value	Name																										
[0, 64]																											



3DSTATE_DS			
		Format:	MBZ
9:4	Patch URB Entry Read Offset		
	Specifies the offset (in 256-bit units) at which Patch URB data is to be read from the URB before being included in the thread payload. This field is ignored if DS Function Enable is DISABLED.		
	Value	Name	
	[0, 63]		
3:0	Reserved		
	Format: MBZ		
5	31:25	Maximum Number of Threads	
		Format:	U7-1 Thread Count
	Specifies the maximum number of simultaneous DS threads allowed to be active. Used to avoid using up the scratch space. Programming the value of the max threads over the number of threads based off number of threads supported in the execution units may improve performance since the architecture allows threads to be buffered between the check for max threads and the actual dispatch into the EU. Programming the max values to a number less than the number of threads supported in the execution units may reduce performance. This field is ignored if DS Function Enable is DISABLED.		
	Value	Name	Description
	[0,127]		indicating thread count of [1,128]
	[0,35]		indicating thread count of [1,36]
	24:21	Reserved	
		Format:	MBZ
	20:11	Reserved	
10	Statistics Enable		
		Format:	Enable
	If ENABLED, this FF unit will engage in statistics gathering. If DISABLED, statistics information associated with this FF stage will be left unchanged. This field is ignored if DS Function Enable is DISABLED.		
	9:3	Reserved	
2	Compute W Coordinate Enable		
		Format:	Enable
	If ENABLED, the DS unit will (for each domain point) compute $W = 1 - (U + V)$ and pass the result as a floating point value in the DS thread payload. If DISABLED, 0.0 will be passed. This field must only be ENABLED for the tessellation of TRI domains, where UVW coordinates are required. This field must be DISABLED for other domains (as they only require UV coordinates) otherwise the computed W coordinate is UNDEFINED. This field is ignored if DS Function Enable is DISABLED.		



3DSTATE_DS	
1	<p>DS Cache Disable</p> <p>Project: _____</p> <p>Format: _____ Disable</p> <p>This bit controls the operation of the DS Cache. This field is ignored if DS Function Enable is DISABLED.</p> <p>If the DS Cache is DISABLED and the DS Function is ENABLED, the DS Cache is not used and all incoming domain points will be passed to DS threads.</p> <p>If the DS Cache is ENABLED and the DS Function is ENABLED, incoming domain points that do not hit in the DS Cache will be passed to DS threads. The DS Cache is invalidated whenever the DS Cache becomes DISABLED, whenever the DS Function Enable toggles, and between patches.</p>
0	<p>DS Function Enable</p> <p>Format: _____ Enable</p> <p>If ENABLED, DS threads will be spawned to process incoming domain points which miss in the DS cache.</p> <p>If DISABLED, the DS stage goes into pass-through mode and performs no specific processing. This field is always used.</p> <p style="text-align: center;">Programming Notes</p> <p>The tessellation stages (HS, TE and DS) must be enabled/disabled as a group. I.e., draw commands can only be issued if all three stages are enabled or all three stages are disabled, otherwise the behavior is UNDEFINED.</p>

6.1.1 3DSTATE_PUSH_CONSTANT_ALLOC_DS

3DSTATE_PUSH_CONSTANT_ALLOC_DS	
Source:	RenderCS
Length Bias:	2
This command sets up the URB configuration for DS Push Constant Buffer.	
Programming Notes	
<p>Programming Restriction:</p> <p>The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size.</p> <p>The sum of the constant length programmed in 3DSTATE_CONSTANT_DS must be equal or smaller then the size of the allocated space in the URB including the buffering for half cachelines. See Push Constant URB Allocation section for more details.</p> <p>The 3DSTATE_CONSTANT_DS must be reprogrammed prior to the next 3DPRIMITIVE command after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_DS.</p>	
DWord	Bit
0	31:29
Command Type	



3DSTATE_PUSH_CONSTANT_ALLOC_DS			
	Default Value:	3h GFXPIPE	
	Format:	OpCode	
28:27	Command SubType		
	Default Value:	3h GFXPIPE_3D	
	Format:	OpCode	
26:24	3D Command Opcode		
	Default Value:	1h GFXPIPE_NONPIPELINED	
	Format:	OpCode	
23:16	3D Command Sub Opcode		
	Default Value:	14h 3DSTATE_PUSH_CONSTANT_ALLOC_DS	
	Format:	OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	0h Excludes DWord (0,1)	
	Project:	All	
	Format:	=n Total Length - 2	
1	31:20	Reserved	
		Format:	MBZ
	19:16	Constant Buffer Offset	
		Format:	U4
		Specifies the offset of the DS constant buffer into the URB.	
		Value	Name
		[0,15]	(0KB - 15KB)
		0h	0KB [Default]
	15:5	Reserved	
		Format:	MBZ
4:0	Constant Buffer Size		
		Format:	U5
		Specifies the size of the DS constant buffer. This value will determine the amount of data the command stream can pre-fetch before the buffer is full. Value of zero is only valid when constants are not enabled for DS.	
		Value	Name
		[0,15]	(0KB – 15KB) Increments of 1KB
		0h	0KB [Default]



6.1.2 3DSTATE_CONSTANT_DS

3DSTATE_CONSTANT_DS			
Source:		RenderCS	
Length Bias:		2	
This command sets pointers to the push constants for the DS unit. The constant data pointed to by this command is loaded into the DS unit's push constant buffer (PCB).			
Programming Notes			Project
It is invalid to execute this command more than once between 3D_PRIMITIVE commands.			
Constant buffers must be enabled in order from Constant Buffer 0 to Constant Buffer 3 within this command. For example, It is not allowed to enable Constant Buffer 1 by programming a non-zero value in the DS Constant Buffer 1 Read Length without a non-zero value in DS Constant Buffer 0 Read Length.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		1Ah 3DSTATE_CONSTANT_DS	
Format:		OpCode	
15:8	Reserved		
	Format:	MBZ	
7:0	DWord Length		
	Project:	All	
	Format:	=n Total Length - 2	
	Value	Name	Project
5h	Excludes DWord (0,1) [Default]		
1..6	191:0	Constant Body	
		Format:	3DSTATE_CONSTANT(Body)
Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS			



DWord		Bit	Description
3DSTATE_CONSTANT(Body)			
Project:		All	
Source:		RenderCS	
Default Value:		0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
0	31:16	Constant Buffer 1 Read Length	
		Project:	All
		Format:	U16 read length
		This field specifies the length of the constant data to be loaded from memory in 256-bit units.	
		Programming Notes	
		The sum of all four read length fields must be less than or equal to the size of 64	
		Setting the value of the register to zero will disable buffer 1.	
		If disabled, the Pointer to Constant Buffer 1 must be programmed to zero.	
	15:0	Constant Buffer 0 Read Length	
		Project:	All
		Format:	U16 read length
		This field specifies the length of the constant data to be loaded from memory in 256-bit units.	
		Programming Notes	
		The sum of all four read length fields must be less than or equal to the size of 64	
		Setting the value of the register to zero will disable buffer 0.	
		If disabled, the Pointer to Constant Buffer 0 must be programmed to zero.	
1	31:16	Constant Buffer 3 Read Length	
		Project:	All
		Format:	U16 read length
		This field specifies the length of the constant data to be loaded from memory in 256-bit units.	
		Programming Notes	
		The sum of all four read length fields must be less than or equal to the size of 64	
		Setting the value of the register to zero will disable buffer 3.	
		If disabled, the Pointer to Constant Buffer 3 must be programmed to zero.	
	15:0	Constant Buffer 2 Read Length	
		Project:	All



3DSTATE_CONSTANT(Body)							
	<table border="1"> <tr> <td>Format:</td> <td>U16 read length</td> </tr> </table> <p>This field specifies the length of the constant data to be loaded from memory in 256-bit units.</p> <p style="text-align: center;">Programming Notes</p> <p>The sum of all four read length fields must be less than or equal to the size of 64</p> <p>Setting the value of the register to zero will disable buffer 2.</p> <p>If disabled, the Pointer to Constant Buffer 2 must be programmed to zero.</p>	Format:	U16 read length				
Format:	U16 read length						
2	<table border="1"> <tr> <td>31:5</td> <td>Pointer to Constant Buffer 0</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>GraphicsAddress[31:5]ConstantBuffer</td> </tr> </table> <p>This field points to the location of Constant Buffer 0. The state of INSTPM<CONSTANT_BUFFER Address Offset Disable> determines whether the Dynamic State Base Address is added to this pointer.</p> <p style="text-align: center;">Programming Notes</p> <p>Constant buffers must be allocated in linear (not tiled) graphics memory.</p>	31:5	Pointer to Constant Buffer 0	Project:	All	Format:	GraphicsAddress[31:5]ConstantBuffer
	31:5	Pointer to Constant Buffer 0					
Project:	All						
Format:	GraphicsAddress[31:5]ConstantBuffer						
4:0	<table border="1"> <tr> <td colspan="2">Constant Buffer Object Control State</td> </tr> <tr> <td>Format:</td> <td>MEMORY_OBJECT_CONTROL_STATE</td> </tr> </table> <p>Specifies the memory object control state for all constant buffers defined in this command.</p>	Constant Buffer Object Control State		Format:	MEMORY_OBJECT_CONTROL_STATE		
Constant Buffer Object Control State							
Format:	MEMORY_OBJECT_CONTROL_STATE						
3	<table border="1"> <tr> <td>31:5</td> <td>Pointer to Constant Buffer 1</td> </tr> <tr> <td>Format:</td> <td>GraphicsAddress[31:5]ConstantBuffer</td> </tr> </table> <p>This field points to the location of Constant Buffer 1.</p> <p style="text-align: center;">Programming Notes</p> <p>Constant buffers must be allocated in linear (not tiled) graphics memory.</p>	31:5	Pointer to Constant Buffer 1	Format:	GraphicsAddress[31:5]ConstantBuffer		
	31:5	Pointer to Constant Buffer 1					
Format:	GraphicsAddress[31:5]ConstantBuffer						
4:0	<table border="1"> <tr> <td colspan="2">Reserved</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Reserved		Project:	All	Format:	MBZ
Reserved							
Project:	All						
Format:	MBZ						
4	<table border="1"> <tr> <td>31:5</td> <td>Pointer to Constant Buffer 2</td> </tr> <tr> <td>Format:</td> <td>GraphicsAddress[31:5]ConstantBuffer</td> </tr> </table> <p>This field points to the location of Constant Buffer 2.</p> <p style="text-align: center;">Programming Notes</p> <p>Constant buffers must be allocated in linear (not tiled) graphics memory.</p>	31:5	Pointer to Constant Buffer 2	Format:	GraphicsAddress[31:5]ConstantBuffer		
	31:5	Pointer to Constant Buffer 2					
Format:	GraphicsAddress[31:5]ConstantBuffer						
4:0	<table border="1"> <tr> <td colspan="2">Reserved</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Reserved		Project:	All	Format:	MBZ
Reserved							
Project:	All						
Format:	MBZ						
5	<table border="1"> <tr> <td>31:5</td> <td>Pointer to Constant Buffer 3</td> </tr> <tr> <td>Format:</td> <td>GraphicsAddress[31:5]ConstantBuffer</td> </tr> </table> <p>This field points to the location of Constant Buffer 3.</p> <p style="text-align: center;">Programming Notes</p> <p>Constant buffers must be allocated in linear (not tiled) graphics memory.</p>	31:5	Pointer to Constant Buffer 3	Format:	GraphicsAddress[31:5]ConstantBuffer		
31:5	Pointer to Constant Buffer 3						
Format:	GraphicsAddress[31:5]ConstantBuffer						



3DSTATE_CONSTANT(Body)		
4:0	Reserved	
	Format:	MBZ

6.2 Thread Payload

The following tables describe the payload delivered to DS threads.

DS Thread Payload (SIMD4x2)

DWord	Bit	Description
R0.7	31	Snapshot Flag
	30:0	Reserved
R0.6	31:24	Reserved
	23:0	Thread ID: This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	Scratch Space Offset: Specifies the of the scratch space allocated to the thread, specified as a 1KB-granular offset from the General State Base Address . See Scratch Space Base Offset description in VS_STATE. (See <i>3D Pipeline</i> for further description on scratch space allocation). Format = GeneralStateOffset[31:10]
	9:0	Reserved
	8:0	FFTID: This ID is assigned by the FF unit and used to identify the thread within the set of outstanding threads spawned by the FF unit. Format: Reserved for HW Implementation Use. Format: U9
R0.4	31:5	Binding Table Pointer. Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the Surface State Base Address . Format = SurfaceStateOffset[31:5]
	4:0	Reserved
R0.3	31:5	Sampler State Pointer. Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the General State Base Address or Dynamic State Base Address . Format = DynamicStateOffset[31:5]
	4	Reserved
	3:0	Per Thread Scratch Space: Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of



DWord	Bit	Description
		scratch space). Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]
R0.2	31:0	Reserved : delivered as zeros (reserved for message header fields)
R0.1	31:26	Reserved
	25:16	Handle ID 1: This ID is assigned by the FF unit and used to identify the URB Return Handle 1 to the FF unit (as FF-specific index value, not a URB address). If only one vertex is to be processed (shaded) by the thread, this field will effectively be ignored (no results are stored for these channels, as controlled by the thread's Channel Mask). Format = Reserved for HW Implementation Use.
	15:14	Reserved
	13:0	URB Return Handle 1: This is the URB handle where Vertex 1 data (the EU's upper channels (DWords 7:4)) results are to be stored. If only one vertex is to be processed (shaded) by the thread, this field will effectively be ignored (no results are stored for these channels, as controlled by the thread's Channel Mask). Format: U12 handle (512-bit granular); Bit 13:12 Reserved
R0.0	31:26	Reserved
	25:16	Handle ID 0: This ID is assigned by the FF unit and used to identify the URB Return Handle 0 to the FF unit (as FF-specific index value, not a URB address). Format = Reserved for HW Implementation Use.
	15:14	Reserved
	13:0	URB Return Handle 0: This is the URB handle where Vertex 0 data (the EU's lower channels (DWords 3:0)) results are to be stored. Format: U12 handle (512-bit granular); Bit 13:12 Reserved
R1.7	31:0	PrimitiveID: This is the 32-bit PrimitiveID value associated with the patch. It is common to all output vertices resulting from the tessellation of the patch. Format: U32
R1.6	31:0	Domain Point 1 W Coordinate: (See Domain Point 0 W Coordinate) Format: FLOAT32
R1.5	31:0	Domain Point 1 V Coordinate: (See Domain Point 0 V Coordinate) Format: FLOAT32
R1.4	31:0	Domain Point 1 U Coordinate: (See Domain Point 0 U Coordinate) Format: FLOAT32
R1.3	31:14	Reserved



DWord	Bit	Description
	13:0	Patch URB Handle: This is the URB handle of the Patch Record (common to both vertices). Format: U12 handle; Bit 13:12 Reserved
R1.2	31:0	Domain Point 0 W Coordinate: If Compute W Coordinate Enable is set, this field will receive the computed value (1 – U – V) for Domain Point 0. Otherwise it is passed as 0.0. Format: FLOAT32
R1.1	31:0	Domain Point 0 V Coordinate: V coordinate associated with Domain Point 0. Format: FLOAT32
R1.0	31:0	Domain Point 0 U Coordinate: U coordinate associated with Domain Point 0. Format: FLOAT32
Varies [Optional]	255:0	Constant Data (optional) : Some amount of constant data (possible none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_DS command (taking the buffer enables into account). The Constant Data arrives in a non-interleaved format.
Varies [Optional]	255:0	Patch URB Data (optional): Some amount of Patch Data (possible none) can be extracted from the URB and passed to the thread in this location in the payload. The amount of data provided is defined by the Patch URB Entry Read Length state (3DSTATE_DS) The Patch Data arrives in a non-interleaved format.

DS Thread Payload (SIMD8)

DWord	Bit	Description
R0.7	31	Snapshot Flag
	30:0	Reserved
R0.6	31:24	Reserved
	23:0	Thread ID: This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	Scratch Space Offset: Specifies the of the scratch space allocated to the thread, specified as a 1KB-granular offset from the General State Base Address. See Scratch Space Base Offset description in VS_STATE. (See 3D Pipeline for further description on scratch space allocation). Format = GeneralStateOffset[31:10]
	9:0	Reserved
	8:0	FFTID: This ID is assigned by the FF unit and used to identify the thread within the set of



DWord	Bit	Description
		<p>outstanding threads spawned by the FF unit.</p> <p>Format: Reserved for HW Implementation Use.</p> <p>Format:</p> <p>U9</p>
R0.4	31:5	<p>Binding Table Pointer. Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the Surface State Base Address.</p> <p>Format = SurfaceStateOffset[31:5]</p>
	4:0	Reserved
R0.3	31:5	<p>Sampler State Pointer. Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the General State Base Address or Dynamic State Base Address.</p> <p>Format = DynamicStateOffset[31:5]</p>
	4	Reserved
	3:0	<p>Per Thread Scratch Space: Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space).</p> <p>Format = U4 power of two (in excess of 10)</p> <p>Range = [0,11] indicating [1K Bytes, 2M Bytes]</p>
R0.2	31:0	Reserved : delivered as zeros (reserved for message header fields)
R0.1	31:0	<p>PrimitiveID: This is the 32-bit PrimitiveID value associated with the patch. It is common to all output domain points resulting from the tessellation of the patch.</p> <p>Format: U32</p>
R0.0	31:27	Reserved
	26:16	<p>Patch Handle ID: This ID is assigned by the FF unit and used to identify the patch URB entry to the FF unit (as FF-specific index value, not a URB address).</p> <p>Format = Reserved for HW Implementation Use.</p>
	15:0	<p>Patch URB Offset: This is the offset within the URB where the patch data is stored.</p> <p>Format: U14 64B-granular offset into the URB</p>
R1.7	31:0	Domain Point 7 U Coordinate. (See Domain Point 0 U Coordinate)
R1.6	31:0	Domain Point 6 U Coordinate. (See Domain Point 0 U Coordinate)
R1.5	31:0	Domain Point 5 U Coordinate. (See Domain Point 0 U Coordinate)
R1.4	31:0	Domain Point 4 U Coordinate. (See Domain Point 0 U Coordinate)
R1.3	31:0	Domain Point 3 U Coordinate. (See Domain Point 0 U Coordinate)
R1.2	31:0	Domain Point 2 U Coordinate. (See Domain Point 0 U Coordinate)
R1.1	31:0	Domain Point 1 U Coordinate. (See Domain Point 0 U Coordinate)
R1.0	31:0	<p>Domain Point 0 U Coordinate: U coordinate associated with Domain Point 0.</p> <p>Format: FLOAT32</p>
R2.7	31:0	Domain Point 7 V Coordinate. (See Domain Point 0 V Coordinate)
R2.6	31:0	Domain Point 6 V Coordinate. (See Domain Point 0 V Coordinate)
R2.5	31:0	Domain Point 5 V Coordinate. (See Domain Point 0 V Coordinate)



DWord	Bit	Description
R2.4	31:0	Domain Point 4 V Coordinate. (See Domain Point 0 V Coordinate)
R2.3	31:0	Domain Point 3 V Coordinate. (See Domain Point 0 V Coordinate)
R2.2	31:0	Domain Point 2 V Coordinate. (See Domain Point 0 V Coordinate)
R2.1	31:0	Domain Point 1 V Coordinate. (See Domain Point 0 V Coordinate)
R2.0	31:0	Domain Point 0 V Coordinate: V coordinate associated with Domain Point 0. Format: FLOAT32
R3.7	31:0	Domain Point 7 W Coordinate. (See Domain Point 0 W Coordinate)
R3.6	31:0	Domain Point 6 W Coordinate. (See Domain Point 0 W Coordinate)
R3.5	31:0	Domain Point 5 W Coordinate. (See Domain Point 0 W Coordinate)
R3.4	31:0	Domain Point 4 W Coordinate. (See Domain Point 0 W Coordinate)
R3.3	31:0	Domain Point 3 W Coordinate. (See Domain Point 0 W Coordinate)
R3.2	31:0	Domain Point 2 W Coordinate. (See Domain Point 0 W Coordinate)
R3.1	31:0	Domain Point 1 W Coordinate. (See Domain Point 0 W Coordinate)
R3.0	31:0	Domain Point 0 W Coordinate: If Compute W Coordinate Enable is set, this field will receive the computed value (1 – U – V) for Domain Point 0. Otherwise it is passed as 0.0. Format: FLOAT32
R4.7	31:0	Domain Point 7 URB Return Handle (see R4.0)
R4.6	31:0	Domain Point 6 URB Return Handle (see R4.0)
R4.5	31:0	Domain Point 5 URB Return Handle (see R4.0)
R4.4	31:0	Domain Point 4 URB Return Handle (see R4.0)
R4.3	31:0	Domain Point 3 URB Return Handle (see R4.0)
R4.2	31:0	Domain Point 2 URB Return Handle (see R4.0)
R4.1	31:0	Domain Point 1 URB Return Handle (see R4.0)
R4.0	31:16	Reserved
	15:0	Domain Point 0 URB Return Handle: This is the offset within the URB where domain point 0 is to be stored. Format: U14 64B-granular offset into the URB
Varies [Optional]	255:0	Constant Data (optional) : Some amount of constant data (possible none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_DS command (taking the buffer enables into account).
Varies [Optional]	255:0	Patch URB Data (optional): Some amount of Patch Data (possible none) can be extracted from the URB and passed to the thread in this location in the payload. The amount of data provided is defined by the Patch URB Entry Read Length state (3DSTATE_DS)

6.3 DS Thread Execution

A DS kernel assumes it is to operate on two domain points in parallel using the EU's SIMD4x2 execution model . Refer to ISA chapters for specifics on writing kernels that operate in SIMD4x2 fashion.



DS threads must always write the destination URB handles passed in the payload. DS threads are not permitted to request additional destination handles. Refer to 3D Pipeline Stage Overview (*3D Overview*) for details on how destination vertices are written and any required contents/formats.

DS threads must signal thread termination on the last message output to the URB shared function.

6.4 Statistics Gathering

The DS stage maintains the DS_INVOCATIONS statistics counter, which counts the number of incoming domain points, irrespective of cache hit/miss. Note that this is different than VS_INVOCATIONS, which counts shader invocations and therefore doesn't count cache hits.

7. 3D Pipeline – Geometry Shader (GS) Stage

7.1 GS Stage Overview

The GS stage of the 3D Pipeline is used to convert objects within incoming primitives into new primitives through use of a spawned thread. When enabled, the GS unit buffers incoming vertices, assembles the vertices of each individual object within the primitives, and passes these object vertices (along with other data) to the subsystem for processing by a GS thread.

When the GS stage is disabled, vertices flow through the unit unmodified.

Refer to the *Common 3D FF Unit Functions* subsection in the *3D Pipeline* chapter for a general description of a 3D Pipeline stage, as much of the GS stage operation and control falls under these “common” functions; i.e., most stage state variables and GS thread payload parameters are described in *3D Pipeline*, and although they are listed here for completeness, that chapter provides the detailed description of the associated functions.

Refer to this chapter for an overall description of the GS stage, and any exceptions the GS stage exhibits with respect to common FF unit functions.

7.2 GS Stage Input

As a stage of the 3D pipeline, the GS stage receives inputs from the previous (DS) stage. Refer to *3D Pipeline* for an overview of the various types of input to a 3D Pipeline stage. The remainder of this subsection describes the inputs specific to the GS stage.

7.2.1 State

7.2.1.1 3DSTATE_GS

The state used by GS is defined with this inline state packet.

3DSTATE_GS		
Source:	RenderCS	
Length Bias:	2	
Controls the GS stage hardware.		
DWord	Bit	Description
0	31:29	Command Type
		Default Value: 3h GFXPIPE Format: OpCode
	28:27	Command SubType
		Default Value: 3h GFXPIPE_3D Format: OpCode



3DSTATE_GS														
	26:24	3D Command Opcode												
		Default Value: 0h 3DSTATE_PIPELINED Format: OpCode												
	23:16	3D Command Sub Opcode												
		Default Value: 11h 3DSTATE_GS Format: OpCode												
	15:8	Reserved												
		Project: All Format: MBZ												
	7:0	DWord Length												
		Default Value: 5h Excludes DWord (0,1) Format: =n												
	1	31:6	Kernel Start Pointer											
			Project: All Format: InstructionBaseOffset[31:6]Kernel This field specifies the starting location (1st core instruction) of the kernel program run by threads spawned by this FF unit. It is specified as a 64-byte-granular offset from the Instruction Base Address.											
		5:0	Reserved											
			Project: All Format: MBZ											
2	31	Single Program Flow (SPF)												
		Project: All												
		Specifies the initial condition of the kernel program as either a single program flow (SIMDn _{xm} with m = 1) or as multiple program flows (SIMDn _{xm} with m > 1). See CR0 description in ISA Execution Environment.												
		<table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0h</td> <td>Disable</td> <td>Single Program Flow disabled</td> </tr> <tr> <td>1h</td> <td>Enable</td> <td>Single Program Flow enabled</td> </tr> </tbody> </table>	Value	Name	Description	0h	Disable	Single Program Flow disabled	1h	Enable	Single Program Flow enabled			
	Value	Name	Description											
	0h	Disable	Single Program Flow disabled											
	1h	Enable	Single Program Flow enabled											
	30	Vector Mask Enable (VME)	Project: All											
			Format: U1 enumerated type											
		When SPF=0, VME specifies which mask to use to initialize the initial channel enables. When SPF=1, VME specifies which mask to use to generate execution channel enables.												
<table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0h</td> <td>Dmask</td> <td>Channels are enabled based on the dispatch mask</td> </tr> <tr> <td>1h</td> <td>Vmask</td> <td>Channels are enabled based on the vector mask</td> </tr> </tbody> </table>		Value	Name	Description	0h	Dmask	Channels are enabled based on the dispatch mask	1h	Vmask	Channels are enabled based on the vector mask				
Value		Name	Description											
0h		Dmask	Channels are enabled based on the dispatch mask											
1h	Vmask	Channels are enabled based on the vector mask												
29:27	Sampler Count	Project: All												
		Format: U3												
	Specifies how many samplers (in multiples of 4) the geometry shader kernel uses. Used only for prefetching the associated sampler state entries.													
	<table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0h</td> <td>No Samplers</td> <td>no samplers used</td> </tr> <tr> <td>1h</td> <td>1-4 Samplers</td> <td>between 1 and 4 samplers used</td> </tr> <tr> <td>2h</td> <td>5-8 Samplers</td> <td>between 5 and 8 samplers used</td> </tr> </tbody> </table>		Value	Name	Description	0h	No Samplers	no samplers used	1h	1-4 Samplers	between 1 and 4 samplers used	2h	5-8 Samplers	between 5 and 8 samplers used
	Value	Name	Description											
	0h	No Samplers	no samplers used											
1h	1-4 Samplers	between 1 and 4 samplers used												
2h	5-8 Samplers	between 5 and 8 samplers used												



3DSTATE_GS			
	3h	9-12 Samplers	between 9 and 12 samplers used
	4h	13-16 Samplers	between 13 and 16 samplers used
	5h-7h	Reserved	Reserved
26	Reserved		
	Project:		All
	Format:		MBZ
25:18	Binding Table Entry Count		
	Project:		All
	Format:		U8
	When HW Generated Binding Table is disabled: Specifies how many binding table entries the kernel uses. Used only for prefetching of the binding table entries and associated surface state. Note: For kernels using a large number of binding table entries, it may be wise to set this field to zero to avoid prefetching too many entries and thrashing the state cache.		
17	Thread Priority		
	Project:		All
	Specifies the priority of the thread for dispatch		
	Value	Name	Description
	0h	Normal Priority	Normal Priority
	1h	High Priority	High Priority
16	Floating Point Mode		
	Project:		All
	Specifies the initial floating point mode used by the dispatched thread.		
	Value	Name	Description
	0h	IEEE-754	Use IEEE-754 Rules
	1h	alternate	Use alternate rules
15:14	Reserved		
	Project:		All
	Format:		MBZ
13	Illegal Opcode Exception Enable		
	Project:		All
	Format:		Enable
	Double Buffer Armed By:	This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment.	
12	Reserved		
	Format:		MBZ
11	Mask Stack Exception Enable		
	Project:		All
	Format:		Enable
	Double Buffer Armed By:	This bit gets loaded into EU CR0.1[11]. See Exceptions and ISA Execution Environment.	
10:8	Reserved		
	Project:		All
	Format:		MBZ
7	Software Exception Enable		
	Project:		All



3DSTATE_GS																																								
	<table border="1"> <tr> <td>Format:</td> <td>Enable</td> </tr> <tr> <td colspan="2">This bit gets loaded into EU CR0.1[13] (note the bit # difference). See Exceptions and ISA Execution Environment.</td> </tr> </table>	Format:	Enable	This bit gets loaded into EU CR0.1[13] (note the bit # difference). See Exceptions and ISA Execution Environment.																																				
Format:	Enable																																							
This bit gets loaded into EU CR0.1[13] (note the bit # difference). See Exceptions and ISA Execution Environment.																																								
6	<table border="1"> <tr> <td colspan="2">Reserved</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Reserved		Project:	All	Format:	MBZ																																	
Reserved																																								
Project:	All																																							
Format:	MBZ																																							
5:0	<table border="1"> <tr> <td colspan="2">Reserved</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Reserved		Format:	MBZ																																			
Reserved																																								
Format:	MBZ																																							
3	<table border="1"> <tr> <td>31:10</td> <td colspan="2">Scratch Space Base Pointer</td> </tr> <tr> <td>Project:</td> <td colspan="2">All</td> </tr> <tr> <td>Format:</td> <td colspan="2">GeneralStateOffset[31:10]</td> </tr> <tr> <td colspan="3">Specifies the location of the scratch space area allocated to this FF unit, specified as a 1KB-granular offset from the General State Base Address. If required, each thread spawned by this FF unit will be allocated some portion of this space, as specified by Per-Thread Scratch Space.</td> </tr> <tr> <td>9:4</td> <td colspan="2">Reserved</td> </tr> <tr> <td>Project:</td> <td colspan="2">All</td> </tr> <tr> <td>Format:</td> <td colspan="2">MBZ</td> </tr> <tr> <td>3:0</td> <td colspan="2">Per-Thread Scratch Space</td> </tr> <tr> <td>Project:</td> <td colspan="2">All</td> </tr> <tr> <td>Format:</td> <td colspan="2">U4 Power of 2 Bytes over 1K Bytes</td> </tr> <tr> <td colspan="3">Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit. The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space.</td> </tr> <tr> <td colspan="2" style="text-align: center;">Value</td> <td style="text-align: center;">Name</td> </tr> <tr> <td colspan="2">[0,11]</td> <td>indicating [1K Bytes, 2M Bytes]</td> </tr> </table>	31:10	Scratch Space Base Pointer		Project:	All		Format:	GeneralStateOffset[31:10]		Specifies the location of the scratch space area allocated to this FF unit, specified as a 1KB-granular offset from the General State Base Address. If required, each thread spawned by this FF unit will be allocated some portion of this space, as specified by Per-Thread Scratch Space.			9:4	Reserved		Project:	All		Format:	MBZ		3:0	Per-Thread Scratch Space		Project:	All		Format:	U4 Power of 2 Bytes over 1K Bytes		Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit. The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space.			Value		Name	[0,11]		indicating [1K Bytes, 2M Bytes]
31:10	Scratch Space Base Pointer																																							
Project:	All																																							
Format:	GeneralStateOffset[31:10]																																							
Specifies the location of the scratch space area allocated to this FF unit, specified as a 1KB-granular offset from the General State Base Address. If required, each thread spawned by this FF unit will be allocated some portion of this space, as specified by Per-Thread Scratch Space.																																								
9:4	Reserved																																							
Project:	All																																							
Format:	MBZ																																							
3:0	Per-Thread Scratch Space																																							
Project:	All																																							
Format:	U4 Power of 2 Bytes over 1K Bytes																																							
Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit. The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space.																																								
Value		Name																																						
[0,11]		indicating [1K Bytes, 2M Bytes]																																						
4	<table border="1"> <tr> <td>31:29</td> <td colspan="2">Reserved</td> </tr> <tr> <td>Project:</td> <td colspan="2">All</td> </tr> <tr> <td>Format:</td> <td colspan="2">MBZ</td> </tr> <tr> <td>28:23</td> <td colspan="2">Output Vertex Size</td> </tr> <tr> <td>Project:</td> <td colspan="2">All</td> </tr> <tr> <td>Format:</td> <td colspan="2">U6</td> </tr> <tr> <td colspan="3">[0,62] indicating [1,63] 16B units</td> </tr> <tr> <td colspan="3">Specifies the size of each vertex stored in the GS output entry (following any Control Header data) as a number of 128-bit units (minus one).</td> </tr> <tr> <td colspan="3" style="text-align: center;">Programming Notes</td> </tr> <tr> <td colspan="3">Programming Restrictions: The vertex size must be programmed as a multiple of 32B units with the following exception: Rendering is disabled (as per SOL stage state) and the vertex size output by the GS thread is 16B.</td> </tr> <tr> <td colspan="3">If rendering is enabled (as per SOL state) the vertex size must be programmed as a multiple of 32B</td> </tr> </table>	31:29	Reserved		Project:	All		Format:	MBZ		28:23	Output Vertex Size		Project:	All		Format:	U6		[0,62] indicating [1,63] 16B units			Specifies the size of each vertex stored in the GS output entry (following any Control Header data) as a number of 128-bit units (minus one).			Programming Notes			Programming Restrictions: The vertex size must be programmed as a multiple of 32B units with the following exception: Rendering is disabled (as per SOL stage state) and the vertex size output by the GS thread is 16B.			If rendering is enabled (as per SOL state) the vertex size must be programmed as a multiple of 32B								
31:29	Reserved																																							
Project:	All																																							
Format:	MBZ																																							
28:23	Output Vertex Size																																							
Project:	All																																							
Format:	U6																																							
[0,62] indicating [1,63] 16B units																																								
Specifies the size of each vertex stored in the GS output entry (following any Control Header data) as a number of 128-bit units (minus one).																																								
Programming Notes																																								
Programming Restrictions: The vertex size must be programmed as a multiple of 32B units with the following exception: Rendering is disabled (as per SOL stage state) and the vertex size output by the GS thread is 16B.																																								
If rendering is enabled (as per SOL state) the vertex size must be programmed as a multiple of 32B																																								



3DSTATE_GS									
	units. In other words, the only time software can program a vertex size with an odd number of 16B units is when rendering is disabled.								
22:17	<p>Output Topology</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>3DPrimType</td> </tr> </table> <p>This field specifies the topology type (3DPrimType) to be associated with GS-thread output vertices (if any).</p>	Project:	All	Format:	3DPrimType				
Project:	All								
Format:	3DPrimType								
16:11	<p>Vertex URB Entry Read Length</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U6</td> </tr> </table> <p>Specifies the amount of URB data read and passed in the thread payload for each Vertex URB entry, in 256-bit register increments.</p> <table border="1"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td>[0,63]</td> <td></td> </tr> </tbody> </table> <p style="text-align: center;">Programming Notes</p> <p>Programming Restriction: This field must be a non-zero value if Include Vertex Handles is cleared to zero.</p>	Project:	All	Format:	U6	Value	Name	[0,63]	
Project:	All								
Format:	U6								
Value	Name								
[0,63]									
10	<p>Include Vertex Handles</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Boolean</td> </tr> </table> <p>If set, all the input Vertex URB handles are included in the payload. These are referred to as “pull model” URB handles, as the thread will use them to read from the URB.</p> <p style="text-align: center;">Programming Notes</p> <p>This field must be set if Vertex URB Entry Read Length is cleared to zero.</p> <p>When this field is set and GS is enabled, only PATCHLIST topologies may be submitted. I.e., pull-model vertices are only supported for PATCH objects, other object types must completely push all vertex data into the payload.</p>	Project:	All	Format:	Boolean				
Project:	All								
Format:	Boolean								
9:4	<p>Vertex URB Entry Read Offset</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U6</td> </tr> </table> <p>Double Buffer Armed By: Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB before being included in the thread payload. This offset applies to all Vertex URB entries passed to the thread.</p> <table border="1"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td>[0,63]</td> <td></td> </tr> </tbody> </table>	Project:	All	Format:	U6	Value	Name	[0,63]	
Project:	All								
Format:	U6								
Value	Name								
[0,63]									
3:0	<p>Dispatch GRF Start Register for URB Data</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U4</td> </tr> </table> <p>Specifies the starting GRF register number for the URB portion (Constant + Vertices) of the thread payload.</p> <table border="1"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td>[0,15]</td> <td>indicating GRF [R0,R15]</td> </tr> </tbody> </table>	Project:	All	Format:	U4	Value	Name	[0,15]	indicating GRF [R0,R15]
Project:	All								
Format:	U4								
Value	Name								
[0,15]	indicating GRF [R0,R15]								



3DSTATE_GS											
Programming Notes											
<p>If Include Vertex Handles is enabled (pull or hybrid handles case), then For DUAL_OBJECT dispatch mode this field should be: $((2 * \text{numVerticesPerObject}) + 8 - 1) / 8 + 1$ For SINGLE and DUAL_INSTANCE dispatch modes this field should be: $(\text{numVerticesPerObject} + 8 - 1) / 8 + 1$ If Include Primitive ID is set, then add 1 to the value obtained by using the above</p>											
5	31:25	Maximum Number of Threads									
		Format: U7-1 thread count									
		Specifies the maximum number of simultaneous threads allowed to be active. Used to avoid using up the scratch space, or to avoid potential deadlock.									
		<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> <th style="text-align: center;">Project</th> </tr> </thead> <tbody> <tr> <td>[0,127]</td> <td>indicating thread count of [1,128]</td> <td></td> </tr> <tr> <td>[0,35]</td> <td>indicating thread count of [1,36]</td> <td></td> </tr> </tbody> </table>	Value	Name	Project	[0,127]	indicating thread count of [1,128]		[0,35]	indicating thread count of [1,36]	
		Value	Name	Project							
[0,127]	indicating thread count of [1,128]										
[0,35]	indicating thread count of [1,36]										
24		Control Data Format									
		Format: U1									
		This field specifies the format of the control data header (if any).									
		<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> <th style="text-align: center;">Description</th> </tr> </thead> <tbody> <tr> <td>0h</td> <td>GSCTL_CUT</td> <td>The control data header contains cut bits.</td> </tr> <tr> <td>1h</td> <td>GSCTL_SID</td> <td>The control data header contains StreamID bits. . Output Topology must be set to POINTLIST, or behavior is UNDEFINED.</td> </tr> </tbody> </table>	Value	Name	Description	0h	GSCTL_CUT	The control data header contains cut bits.	1h	GSCTL_SID	The control data header contains StreamID bits. . Output Topology must be set to POINTLIST, or behavior is UNDEFINED.
		Value	Name	Description							
0h	GSCTL_CUT	The control data header contains cut bits.									
1h	GSCTL_SID	The control data header contains StreamID bits. . Output Topology must be set to POINTLIST, or behavior is UNDEFINED.									
23:20		Control Data Header Size									
		Project: All									
		Format: U4									
		Specifies the number of 32B units of control data header located at the start of the GS URB entry. The value 0 indicates there is no control data header, and Control Data Format is ignored. Software must ensure that the Control Data Header Size is sufficient to accommodate the maximum number of vertices output by the GS thread. It is UNDEFINED for a GS thread to report more output vertices than can be accommodated in a non-zero-sized header. (If the header size is zero, by definition neither cut nor StreamID bits are defined.)									
		<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td>[0,8]</td> <td>32B units</td> </tr> </tbody> </table>	Value	Name	[0,8]	32B units					
Value	Name										
[0,8]	32B units										
19:15		Instance Control									
		Project: All									
		Format: U5-1 in #instances									
		[0,31] indicating [1,32] instances									
		Specifies the number of instances (minus one) for each input object. To avoid confusion, this document uses the term "InstanceCount" to refer to InstanceControl+1, with a range of [1,32]If									



3DSTATE_GS		
	<p>InstanceCount>1, DUAL_OBJECT mode is invalid. Software will likely want to use DUAL_INSTANCE mode for higher performance, but SINGLE mode is also supported. When InstanceCount=1 (one instance per object) software can decide which dispatch mode to use. DUAL_OBJECT mode would likely be the best choice for performance, followed by SINGLE mode. DUAL_INSTANCE mode is not recommended but is supported.</p>	
14:13	Default StreamID	
	Project:	All
	Format:	U2
	<p>When the GS is enabled, unless the GS output entry contains StreamID bits in the control header, this field specifies the default StreamID associated with any GS-thread output vertices. When the GS is disabled, StreamID will be output as 0.</p>	
12:11	Dispatch Mode	
	Project:	All
	Format:	U2
	<p>This field specifies how the GS unit dispatches multiple instances and/or multiple objects.</p>	
	Value	Name Description
	0h	SINGLE Each thread shades a single instance of one object.
	1h	DUAL_INSTANCE Each thread shades possibly two instances of one object. If the InstanceCount is odd, a trailing dispatch of only one instance will be made for each object received. Not recommended if InstanceCount = 1, assuming a kernel optimized for SINGLE or DUAL_OBJECT dispatch would outperform a kernel compiled for DUAL_INSTANCE but only passed one instance. The GS must be allocated at least two URB handles or behavior is UNDEFINED.
	2h	DUAL_OBJECT Each thread shades one instance of possibly two objects. The GS unit attempt to pair objects together into one dispatch, but under some circumstances only one object may be dispatched (as controlled by the DispatchMask generated by the GS unit). Not valid for objects with more than 16 vertices per object. Not valid if InstanceCount > 1 (more than one instance per object). The GS must be allocated at least two URB handles or behavior is UNDEFINED.
	3h	Reserved
10	GS Statistics Enable	
	Project:	All
	<p>This bit controls whether GS-unit-specific statistics register(s) can be incremented.</p>	
	Value	Name Description
	0h	Disable GS_INVOCATIONS_COUNT and GS_PRIMITIVES_COUNT cannot increment
	1h	Enable GS_INVOCATIONS_COUNT and GS_PRIMITIVES_COUNT can increment
9:5	GSInvocations Increment Value	
	Project:	All
	Format:	U5
	<p>Specifies how much to increment the GS_INVOCATIONS_COUNT for each instance of each object. This control is provided to allow software to process multiple instances (from an API POV) in a single kernel invocation. In SINGLE dispatch mode, the counter will increment by this value for each dispatch (as it's only one instance of one object). In DUAL_INSTANCE mode, the counter will be incremented by the value if only one instance is included in the dispatch (i.e., the last odd instance), otherwise the counter will be incremented by twice this value. In DUAL_OBJECT dispatch mode, the counter will be incremented by the value if only one object is included in the dispatch (i.e., a forced dispatch of one object), otherwise the counter will be incremented by twice this value.</p>	



3DSTATE_GS		
	Value	Name
	[0,31]	indicating an increment of [1,32]
4	Include Primitive ID	
	Project:	All
	Format:	Boolean
	If set, R1 of the payload is written with Primitive ID value(s). If clear, these Primitive ID values are not included in the payload R1.	
3	Hint	
	Project:	All
	Format:	U1
	This state bit is simply passed in GS thread payloads for use by the GS kernel – it has no other impact on hardware operation.	
2	Reorder Enable	
	Format:	Enable
	This bit controls whether the GS unit reorders TRISTRIP/TRISTRIP_REV vertices passed in the GS thread payload. If ENABLED, the GS unit will reorder the vertices for “odd-numbered” triangles originating from TRISTRIP topologies and “even-numbered” triangles originating from TRISTRIP_REV topologies. (Note that the first triangle is considered “triangle 0”, which is even-numbered). With respect to the PrimType passed in the GS thread payload, the GS unit passes TRISTRIP when the vertices are not reordered, and TRISTRIP_REV when the vertices are reordered (regardless of whether a TRISTRIP or TRISTRIP_REV topology was being processed). If DISABLED, TRISTRIP/TRISTRIP_REV vertices are not reordered, and always passed in the order they are received from the pipeline. The GS unit will still toggle PrimType on alternating (as described above) so that the GS thread can perform the reordering internally (or do whatever is necessary to account for the non-reordering of its input).	
1	Discard Adjacency	
	Project:	All
	Format:	Enable
	When set, adjacent vertices will not be passed in the GS payload when objects with adjacency are processed. Instead, only the non-adjacent vertices will be passed in the same fashion as the without-adjacency form of the primitive. Software should set this bit whenever a GS kernel is used that does not expect adjacent vertices. This allows both with-adjacency/without-adjacency variants of the primitive to be submitted to the pipeline (via 3DPRIMITIVE) – the GS unit will silently discard any adjacent vertices and present the GS thread with only the internal object. When clear, adjacent vertices will be passed to the GS thread, as dictated by the incoming primitive type. Software should only clear this bit when a GS kernel is used that does expect adjacent vertices. E.g., if the GS kernel is compiled to expect a TRIANGLE_ADJ object, software must clear this bit. Software should also clear this bit if the GS kernel expects a POINT or PATCHLIST_n object (which don't have with-adjacency variants). This bit is used to provide limited compatibility between submitted primitive types and the object type expected by the GS kernel. The only hardware assistance is to allow the submission of a with-adjacency variant of a primitive when operating with a GS kernel that expects the without-adjacency variant of the object. (E.g., when the GS kernel is compiled to expect a TRIANGLE object, software should set this bit just in case a TRILIST_ADJ is submitted to the pipeline.) Note that the GS unit is otherwise not aware of the object type that is expected by the GS kernel. It is up to software to ensure that the submitted primitive type (in 3DPRIMITIVE) is otherwise compatible with the object type expected by the GS kernel. (E.g., if the GS kernel expects a LINE_ADJ object, only LINELIST_ADJ or LINESTRIP_ADJ should be submitted, otherwise the GS kernel will produce unpredictable results.) Also	



3DSTATE_GS													
	note that it is possible to craft a GS kernel which can accept any object type that's thrown at it by first examining the PrimType passed in the payload and then using this info to correctly interpret the number of vertices passed in the payload.												
0	<p>GS Enable</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> </table> <p>Specifies whether the GS stage is enabled or disabled (pass-through).</p>	Project:	All	Format:	Enable								
Project:	All												
Format:	Enable												
6	<p>31 Reserved</p> <table border="1"> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table> <p>30:13 Reserved</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table> <p>12 Reserved</p> <table border="1"> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table> <p>11:0 Semaphore Handle</p> <table border="1"> <tr> <td>Project:</td> <td></td> </tr> <tr> <td>Format:</td> <td>URBOffset[17:6]</td> </tr> </table> <p>This is the URB offset pointing to the first of the GS semaphore DWords in the URB. The size of the region is 128 DWs(8 – 512b URB entries). Software is responsible for allocating combined GS and/or HS semaphore Dwords in a single contiguous region of the URB. Software must also make sure the 3D pipeline is IDLE prior to allocating or deallocating the region. The semaphores can be located in an unused area within a FF unit's URB fenced region or an unused area within the Push Constant region.</p>	Format:	MBZ	Project:	All	Format:	MBZ	Format:	MBZ	Project:		Format:	URBOffset[17:6]
Format:	MBZ												
Project:	All												
Format:	MBZ												
Format:	MBZ												
Project:													
Format:	URBOffset[17:6]												

7.2.1.2 3DSTATE_CONSTANT_GS

3DSTATE_CONSTANT_GS	
Source:	RenderCS
Length Bias:	2
This command sets pointers to the push constants for the GS unit. The constant data pointed to by this command will be loaded into the GS unit's push constant buffer (PCB).	
Programming Notes	
It is invalid to execute this command more than once between 3D_PRIMITIVE commands.	
Constant buffers must be enabled in order from Constant Buffer 0 to Constant Buffer 3 within this command. For example, it is not allowed to enable Constant Buffer 1 by programming a non-zero value in the GS Constant Buffer 1 Read Length without a non-zero value in GS Constant Buffer 0 Read Length.	
DWord	Bit
Description	
0	31:29
Command Type	
Default Value: 3h GFXPIPE	
Format: OpCode	
	28:27
Command SubType	
Default Value: 3h	



3DSTATE_CONSTANT_GS		
		Format: OpCode
26:24	3D Command Opcode	
	Default Value:	0h 3DSTATE_PIPELINED
	Format:	OpCode
23:16	3D Command Sub Opcode	
	Default Value:	16h 3DSTATE_CONSTANT_GS
	Format:	OpCode
15	Reserved	
	Project:	All
	Format:	MBZ
14:8	Reserved	
	Format:	MBZ
7:0	DWord Length	
	Project:	All
	Format:	=n Total Length - 2
	Value	Name
	5h	Excludes DWord (0,1) [Default]
1..6	191:0	Constant Body
		Format: 3DSTATE_CONSTANT(Body)
		Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS

3DSTATE_CONSTANT(Body)		
Project:	All	
Source:	RenderCS	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	Bit	Description
0	31:16	Constant Buffer 1 Read Length
	Project:	All
	Format:	U16 read length
	This field specifies the length of the constant data to be loaded from memory in 256-bit units.	
	Programming Notes	
	The sum of all four read length fields must be less than or equal to the size of 64	



3DSTATE_CONSTANT(Body)					
	<p>Setting the value of the register to zero will disable buffer 1.</p> <p>If disabled, the Pointer to Constant Buffer 1 must be programmed to zero.</p>				
15:0	<p>Constant Buffer 0 Read Length</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U16 read length</td> </tr> </table> <p>This field specifies the length of the constant data to be loaded from memory in 256-bit units.</p> <p style="text-align: center;">Programming Notes</p> <p>The sum of all four read length fields must be less than or equal to the size of 64</p> <p>Setting the value of the register to zero will disable buffer 0.</p> <p>If disabled, the Pointer to Constant Buffer 0 must be programmed to zero.</p>	Project:	All	Format:	U16 read length
Project:	All				
Format:	U16 read length				
1	<p>31:16 Constant Buffer 3 Read Length</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U16 read length</td> </tr> </table> <p>This field specifies the length of the constant data to be loaded from memory in 256-bit units.</p> <p style="text-align: center;">Programming Notes</p> <p>The sum of all four read length fields must be less than or equal to the size of 64</p> <p>Setting the value of the register to zero will disable buffer 3.</p> <p>If disabled, the Pointer to Constant Buffer 3 must be programmed to zero.</p>	Project:	All	Format:	U16 read length
Project:	All				
Format:	U16 read length				
	<p>15:0 Constant Buffer 2 Read Length</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U16 read length</td> </tr> </table> <p>This field specifies the length of the constant data to be loaded from memory in 256-bit units.</p> <p style="text-align: center;">Programming Notes</p> <p>The sum of all four read length fields must be less than or equal to the size of 64</p> <p>Setting the value of the register to zero will disable buffer 2.</p> <p>If disabled, the Pointer to Constant Buffer 2 must be programmed to zero.</p>	Project:	All	Format:	U16 read length
Project:	All				
Format:	U16 read length				
2	<p>31:5 Pointer to Constant Buffer 0</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> </table>	Project:	All		
Project:	All				



3DSTATE_CONSTANT(Body)		
		Format: GraphicsAddress[31:5]ConstantBuffer This field points to the location of Constant Buffer 0. The state of INSTPM<CONSTANT_BUFFER Address Offset Disable> determines whether the Dynamic State Base Address is added to this pointer. Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.
		4:0 Constant Buffer Object Control State Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command.
		3 31:5 Pointer to Constant Buffer 1 Format: GraphicsAddress[31:5]ConstantBuffer This field points to the location of Constant Buffer 1. Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	Reserved Project: All Format: MBZ
		4 31:5 Pointer to Constant Buffer 2 Format: GraphicsAddress[31:5]ConstantBuffer This field points to the location of Constant Buffer 2. Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.
		Reserved Project: All Format: MBZ
5	31:5	Pointer to Constant Buffer 3 Format: GraphicsAddress[31:5]ConstantBuffer This field points to the location of Constant Buffer 3. Programming Notes Constant buffers must be allocated in linear (not tiled) graphics memory.
		4:0 Reserved
		Format: MBZ



7.2.1.3 3DSTATE_PUSH_CONSTANT_ALLOC_GS

3DSTATE_PUSH_CONSTANT_ALLOC_GS			
Source:	RenderCS		
Length Bias:	2		
This command sets up the URB configuration for GS Push Constant Buffer.			
Programming Notes			
<ul style="list-style-type: none"> The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size. The sum of the constant length programmed in 3DSTATE_CONSTANT_GS must be equal or smaller than the size of the allocated space in the URB including the buffering for half cachelines. The 3DSTATE_CONSTANT_GS must be reprogrammed prior to the next 3DPRIMITIVE command after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_GS. 			
See Push Constant URB Allocation section for more details.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	1h GFXPIPE_NONPIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		15h 3DSTATE_PUSH_CONSTANT_ALLOC_GS	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Format:	=n	
	Total Length - 2		
	Value	Name	Description
	0h	3DSTATE_PUSH_CONSTANT_ALLOC_GS [Default]	Excludes DWord (0,1)
1	31:20	Reserved	
		Format:	MBZ
		Constant Buffer Offset	
	Format:	U5	
	Specifies the offset of the GS constant buffer into the URB.		
	Value	Name	
[0,15]	(0KB - 15KB)		
0h	0KB [Default]		



3DSTATE_PUSH_CONSTANT_ALLOC_GS							
15:5	Reserved Format: MBZ						
4:0	Constant Buffer Size Format: U5 Specifies the size of the GS constant buffer. This value will determine the amount of data the command stream can pre-fetch before the buffer is full. Value of zero is only valid when constants are not enabled for GS.						
	<table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>[0,15]</td> <td>(0KB – 15KB) Increments of 1KB</td> </tr> <tr> <td>0h</td> <td>0KB [Default]</td> </tr> </tbody> </table>	Value	Name	[0,15]	(0KB – 15KB) Increments of 1KB	0h	0KB [Default]
Value	Name						
[0,15]	(0KB – 15KB) Increments of 1KB						
0h	0KB [Default]						

7.3 Object Staging

The GS unit's Object Staging Buffer (OSB) accepts primitive topologies as a stream of incoming vertices, and spawns a thread for each individual object within the topology.

7.4 GS Thread Request Generation

7.4.1 Object Vertex Ordering

The following table defines the number and order of object vertices passed in the Vertex Data portion of the GS thread payload, assuming an input topology with N vertices. The ObjectType passed to the thread is, by default, the incoming PrimTopologyType. Exceptions to this rule (for the TRISTRIP variants) are called out.

The following table also shows which vertex is selected to provide PrimitiveID (bold, underlined vertex number). *In general*, the vertex selected is the last vertex for non-adjacent prims, and the next-to-last vertex for adjacent prims. Note, however, that there are exceptions:

- reorder-enabled TRISTRIP[_REV]
- “odd-numbered” objects in TRISTRIP_ADJ

PrimTopologyType	Order of Vertices in Payload	GS Notes
<PRIMITIVE_TOPOLOGY> (N = # of vertices)	[<object#>] = (<vert#>, ...); [{modified PrimType passed to thread}]	
POINTLIST	[0] = (0); [1] = (1); ...; [N-2] = (N-2);	
POINTLIST_BF	N/A	
LINELIST	[0] = (0,1);	



PrimTopologyType	Order of Vertices in Payload	GS Notes
<PRIMITIVE_TOPOLOGY> (N = # of vertices)	[<object#>] = (<vert#>,...); [{modified PrimType passed to thread}]	
(N is multiple of 2)	[1] = (2, <u>3</u>); ...; [(N/2)-1] = (N-2, <u>N-1</u>)	
LINELIST_ADJ (N is multiple of 4)	[0] = (0,1, <u>2</u> ,3); [1] = (4,5, <u>6</u> ,7); ...; [(N/4)-1] = (N-4,N-3, <u>N-2</u> ,N-1)	
LINESTRIP (N >= 2)	[0] = (0, <u>1</u>); [1] = (1, <u>2</u>); ...; [N-2] = (N-2, <u>N-1</u>)	
LINESTRIP_ADJ (N >= 4)	[0] = (0,1, <u>2</u> ,3); [1] = (1,2, <u>3</u> ,4); ...; [N-4] = (N-4,N-3, <u>N-2</u> ,N-1)	
LINESTRIP_BF	N/A	
LINESTRIP_CONT	Same as LINESTRIP	Handled same as LINESTRIP
LINESTRIP_CONT_BF	Same as LINESTRIP	Handled same as LINESTRIP
LINELOOP (N >= 2)	[0] = (0, <u>1</u>); [1] = (1, <u>2</u>); [N] = (N-1, <u>0</u>);	Not supported after GS.
TRILIST (N is multiple of 3)	[0] = (0,1, <u>2</u>); [1] = (3,4, <u>5</u>); ...; [(N/3)-1] = (N-3,N-2, <u>N-1</u>)	
RECTLIST	Same as TRILIST	Handled same as TRILIST
TRILIST_ADJ (N is multiple of 6)	[0] = (0,1,2,3, <u>4</u> ,5); [1] = (6,7,8,9, <u>10</u> ,11); ...; [(N/6)-1] = (N-6,N-5,N-4,N-3, <u>N-2</u> ,N-1)	
TRISTRIP (<u>Reorder ENABLED</u>) (N >= 3)	[0] = (0,1, <u>2</u>); {TRISTRIP} [1] = (1, <u>3</u> ,2); {TRISTRIP_REV} [k even] = (k,k+1, <u>k+2</u>) {TRISTRIP} [k odd] = (k, <u>k+2</u> ,k+1) {TRISTRIP_REV}	"Odd" triangles have vertices reordered , though identified as TRISTRIP_REV so the thread knows this



PrimTopologyType	Order of Vertices in Payload	GS Notes
<PRIMITIVE_TOPOLOGY> (N = # of vertices)	[<object#>] = (<vert#>,...); [{modified PrimType passed to thread}]	
	[N-3] = (see above)	
TRISTRIP (<u>Reorder DISABLED</u>) (N >= 3)	[0] = (0,1, <u>2</u>) {TRISTRIP} [1] = (1,2, <u>3</u>) {TRISTRIP_REV}; ... [N-3] = (N-3,N-2, <u>N-1</u>) {TRISTRIP or TRISTRIP_REV}	“Odd” triangles <u>do not</u> have vertices reordered, though identified as TRISTRIP_REV so the thread knows this
TRISTRIP_REV (<u>Reorder ENABLED</u>) (N >= 3)	[0] = (0, <u>2</u> ,1) {TRISTRIP_REV}; [1] = (1,2, <u>3</u>) {TRISTRIP}; ...; [k even] = (k, <u>k+2</u> ,k+1) {TRISTRIP_REV} [k odd] = (k,k+1, <u>k+2</u>) {TRISTRIP} [N-3] = (see above)	“Odd” triangles have vertices reordered , though identified as TRISTRIP so the thread knows this
TRISTRIP_REV (<u>Reorder DISABLED</u>) (N >= 3)	[0] = (0,1, <u>2</u>) {TRISTRIP_REV} [1] = (1,2, <u>3</u>) {TRISTRIP}; ...; [N-3] = (N-3,N-2, <u>N-1</u>) {TRISTRIP or TRISTRIP_REV}	“Odd” triangles <u>do not</u> have vertices reordered, though identified as TRISTRIP so the thread knows this
TRISTRIP_ADJ (N even, N >= 6)	N = 6 or 7: [0] = (0,1,2,5, <u>4</u> ,3) N = 8 or 9: [0] = (0,1,2,6, <u>4</u> ,3); [1] = (2,5, <u>6</u> ,7,4,0); ...; N > 10: [0] = (0,1,2,6, <u>4</u> ,3); [1] = (2,5, <u>6</u> ,8,4,0); ...; [k>1, even] = (2k,2k-2, 2k+2, 2k+6, <u>2k+4</u> , 2k+3); [k>2, odd] = (2k, 2k+3, <u>2k+4</u> , 2k+6, 2k+2, 2k-2);...; Trailing object: [(N/2)-3, even] = (N-6,N-8,N-4,N-1, <u>N-2</u> ,N-3);	“Odd” objects have vertices reordered .

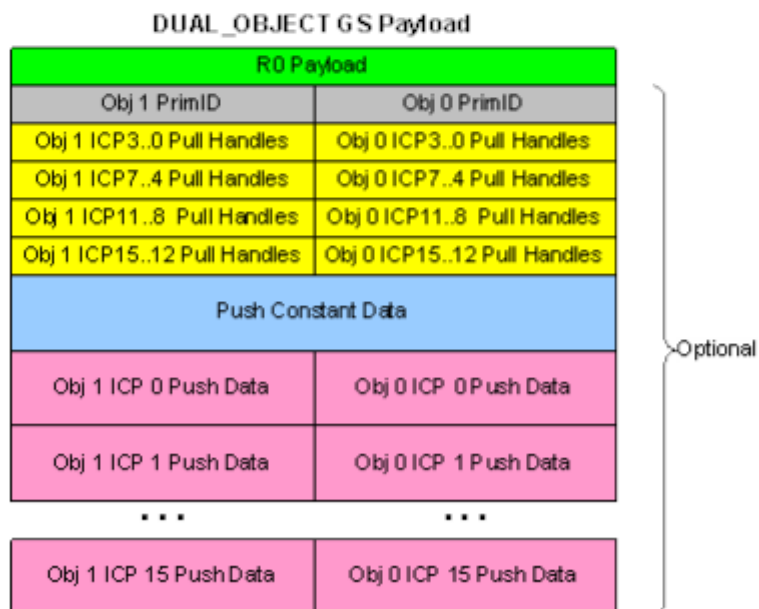
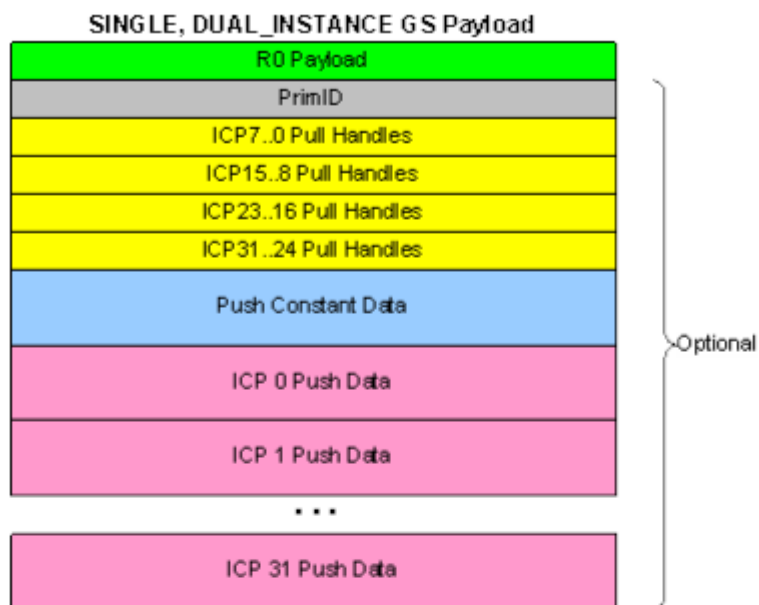


PrimTopologyType	Order of Vertices in Payload	GS Notes
<PRIMITIVE_TOPOLOGY> (N = # of vertices)	[<object#>] = (<vert#>,...); [{modified PrimType passed to thread}]	
	[(N/2)-3, odd] = (N-6,N-3, N-2 ,N-1,N-4,N-8);	
TRIFAN (N > 2)	[0] = (0,1, 2); [1] = (0,2, 3); ...; [N-3] = (0, N-2, N-1);	Only used by OGL
TRIFAN_NOSTIPPLE	Same as TRIFAN	
POLYGON	Same as TRIFAN	
QUADLIST (N is multiple of 4)	[0] = (0,1,2, 3); [1] = (4,5,6, 7); ...; [(N/4)-1] = (N-4,N-3,N-2, N-1);	Not supported after GS.
QUADSTRIP (N is multiple of 2, N >=4)	[0] = (0,1,3,2); [1] = (2,3,5,4); ... ; [(N/2)-2] = (N-4,N-3,N-1,N-2);	Not supported after GS.
PATCHLIST_1 PATCHLIST_2 PATCHLIST_3..32	[0] = (0); [1] = (1); ...; [N-2] = (N-2); [0] = (0, 1); [1] = (2, 3); ...; [(N/2)-1] = (N-2, N-1) similar to above	

7.4.2 GS Thread Payload High-Level Layout

GS Thread Payload High-Level Layout shows the high-level layout of the payload delivered to GS threads.

GS Dispatch Layouts



Subsequent sections provide detailed layouts for different processor generations.



7.4.3 GS Thread Payload SIMD 4x2

The table below shows the layout of the payload delivered to GS threads.

Refer to the [3D Pipeline Stage Overview section in vol2a 3D Pipeline](#) for details on those fields that are common among the various pipeline stages.

GS Thread Payload SIMD 4x2

GRF DWord	Bits	Description
R0.7	31	
	31:0	Reserved.
R0.6	31	Dereference Thread. This bit is defined to send back the Handle ID back to HS to dereference the input handles for this thread.
	30:24	Reserved.
	23:0	Thread ID. This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	Scratch Space Pointer. Specifies the location of the scratch space allocated to this thread, specified as a 1KB-aligned offset from the General State Base Address . Format = GeneralStateOffset[31:10]
	9:0	Reserved
	8:0	FFTID. This ID is assigned by the fixed function unit and is relative identifier for the thread. It is used to free up resources used by the thread upon thread completion. Format: U7 Range: 0-127
R0.4	31:5	Binding Table Pointer: Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the Surface State Base Address . Format = SurfaceStateOffset[31:5]
	4:0	Reserved.
R0.3	31:5	Sampler State Pointer. Specifies the location of the Sampler State Table used by this thread, specified as a 32-byte granular offset from the Dynamic State Base Address . Format = DynamicStateOffset[31:5]
	4	Reserved.
	3:0	Per Thread Scratch Space. Specifies the amount of scratch space allowed for this thread. The value specifies the power that two is raised to (over determine the amount of scratch space). Programming Notes:



GRF	DWord	Bits	Description
			<p>This amount is available to the kernel for information only. It is passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port ignores it.</p> <p>Format = U4 power of two (in excess of 10)</p> <p>Range = [0,11] indicating [1K Bytes, 2M Bytes]</p>
R0.2		31:24	<p>Semaphore Index. This is a DWord index used in URB_ATOMIC commands if the thread is using data pulled from input handles. This information is only required for pull-model vertex inputs and InstanceCount > 1.</p> <p>Format = U8</p>
		23	Reserved.
		22	<p>Hint. This is a copy of the corresponding 3DSTATE_GS bit.</p> <p>Format: U1</p>
		21:16	<p>Primitive Topology Type. This field identifies the Primitive Topology Type associated with the primitive containing this object. It indirectly specifies the number of input vertices included in the thread payload. Note that the GS unit may toggle this value between TRISTRIP and TRISTRIP_REV. If the Discard Adjacency bit is set, the topology type passed in the payload is UNDEFINED.</p> <p>Format: See <i>3D Pipeline</i>.</p>
		15:13	Reserved.
		12:0	<p>Semaphore Handle. This is the URB offset pointing to the first GS semaphore DWord in the URB. Software is responsible for statically allocating the semaphore DWords in the URB. Refer to Semaphore Handle field in 3DSTATE_GS for size of semaphore allocation.</p> <p>Format:</p> <p>U12 64B-aligned URB offset; bit 12 is reserved</p>
R0:1		31:27	<p>GS Instance ID 1. For each input object, the GS unit can spawn multiple threads (instances). This field starts at zero for the first instance of an object and increments for subsequent instances. If “dispatch mode” is DUAL_OBJECT this field is not valid. Format: U5</p>
		26:16	Reserved
		15:0	<p>URB Return Handle 1. This is the URB offset where the EU’s upper channels (DWords 7:4) results are stored. If only one object/instance is processed (shaded) by the thread, this field is effectively ignored (no results are stored for these channels, as controlled by the thread’s Channel Mask).</p> <p>Format: U12 64B-aligned URB offset; bit 12 is reserved</p>
R0.0		31:27	<p>GS Instance ID 0. For each input object, the GS unit can spawn multiple threads (instances). This field starts at zero for the first instance of an object and increments for subsequent instances.</p> <p>If “dispatch mode” is DUAL_OBJECT, this field is not valid.</p> <p>Format: U5</p>



GRF DWord	Bits	Description
	26:16	Reserved.
	15:0	<p>URB Return Handle 0. This is the URB offset where the EU's lower channels (DWords 3:0) results are stored.</p> <p>Format:</p> <p>U12 64B-aligned URB offset; bit 12 is reserved</p>
The following register is included only if Include PrimitiveID is enabled.		
R1.7-R1.5	31:0	Reserved: MBZ.
R1.4	31:0	<p>Primitive ID 1. This field contains the Primitive ID associated with (all instances) of input object 1. Only valid in DUAL_OBJECT mode.</p> <p>Format: U32</p>
R1.3-R1.1	31:0	Reserved: MBZ.
R1.0	31:0	<p>Primitive ID 0. This field contains the Primitive ID associated with (all instances) of input object 0.</p> <p>Format: U32</p>
The following register is included only if SINGLE or DUAL_INSTANCE mode and Include Vertex Handles is enabled.		
Rn.7	31:16	ICP 7 Handle ID
	15:0	ICP 7 Handle
Rn.6	31:16	ICP 6 Handle ID
	15:0	ICP 6 Handle
Rn.5	31:16	ICP 5 Handle ID
	15:0	ICP 5 Handle
Rn.4	31:16	ICP 4 Handle ID
	15:0	ICP 4 Handle
Rn.3	31:16	ICP 3 Handle ID
	15:0	ICP 3 Handle
Rn.2	31:16	ICP 2 Handle ID
	15:0	ICP 2 Handle
Rn.1	31:16	ICP 1 Handle ID
	15:0	ICP 1 Handle
Rn.0	31:16	ICP 0 Handle ID
	15:0	ICP 0 Handle
The following register is included only if SINGLE or DUAL_INSTANCE mode and Include Vertex Handles is enabled and ICP Count > 7.		
Rn+1.7	31:16	ICP 15 Handle ID
	15:0	ICP 15 Handle
Rn+1.6	31:16	ICP 14 Handle ID
	15:0	ICP 14 Handle
Rn+1.5	31:16	ICP 13 Handle ID
	15:0	ICP 13 Handle
Rn+1.4	31:16	ICP 12 Handle ID
	15:0	ICP 12 Handle
Rn+1.3	31:16	ICP 11 Handle ID
	15:0	ICP 11 Handle
Rn+1.2	31:16	ICP 10 Handle ID



GRF DWord	Bits	Description
	15:0	ICP 10 Handle
Rn+1.1	31:16	ICP 9 Handle ID
	15:0	ICP 9 Handle
Rn+1.0	31:16	ICP 8 Handle ID
	15:0	ICP 8 Handle
The following register is included only if SINGLE or DUAL_INSTANCE mode and Include Vertex Handles is enabled and ICP Count > 15.		
Rn+2.7	31:16	ICP 23 Handle ID
	15:0	ICP 23 Handle
Rn+2.6	31:16	ICP 22 Handle ID
	15:0	ICP 22 Handle
Rn+2.5	31:16	ICP 21 Handle ID
	15:0	ICP 21 Handle
Rn+2.4	31:16	ICP 20 Handle ID
	15:0	ICP 20 Handle
Rn+2.3	31:16	ICP 19 Handle ID
	15:0	ICP 19 Handle
Rn+2.2	31:16	ICP 18 Handle ID
	15:0	ICP 18 Handle
Rn+2.1	31:16	ICP 17 Handle ID
	15:0	ICP 17 Handle
Rn+2.0	31:16	ICP 16 Handle ID
	15:0	ICP 16 Handle
The following register is included only if SINGLE or DUAL_INSTANCE mode and Include Vertex Handles is enabled and ICP Count > 23.		
Rn+3.7	31:16	ICP 31 Handle ID
	15:0	ICP 31 Handle
Rn+3.6	31:16	ICP 30 Handle ID
	15:0	ICP 30 Handle
Rn+3.5	31:16	ICP 29 Handle ID
	15:0	ICP 29 Handle
Rn+3.4	31:16	ICP 28 Handle ID
	15:0	ICP 28 Handle
Rn+3.3	31:16	ICP 27 Handle ID
	15:0	ICP 27 Handle
Rn+3.2	31:16	ICP 26 Handle ID
	15:0	ICP 26 Handle
Rn+3.1	31:16	ICP 25 Handle ID
	15:0	ICP 25 Handle
Rn+3.0	31:16	ICP 24 Handle ID
	15:0	ICP 24 Handle
The following register is included only if DUAL_OBJECT mode and Include Vertex Handles is enabled.		
Rn.7	31:16	Object 1 ICP 3 Handle ID
	15:0	Object 1 ICP 3 Handle
Rn.6	31:16	Object 1 ICP 2 Handle ID
	15:0	Object 1 ICP 2 Handle
Rn.5	31:16	Object 1 ICP 1 Handle ID
	15:0	Object 1 ICP 1 Handle
Rn.4	31:16	Object 1 ICP 0 Handle ID



GRF DWord	Bits	Description
	15:0	Object 1 ICP 0 Handle
Rn.3	31:16	Object 0 ICP 3 Handle ID
	15:0	Object 0 ICP 3 Handle
Rn.2	31:16	Object 0 ICP 2 Handle ID
	15:0	Object 0 ICP 2 Handle
Rn.1	31:16	Object 0 ICP 1 Handle ID
	15:0	Object 0 ICP 1 Handle
Rn.0	31:16	Object 0 ICP 0 Handle ID
	15:0	Object 0 ICP 0 Handle
The following register is included only if DUAL_OBJECT mode and Include Vertex Handles is enabled and ICP Count > 3.		
Rn+1.7	31:16	Object 1 ICP 7 Handle ID
	15:0	Object 1 ICP 7 Handle
Rn+1.6	31:16	Object 1 ICP 6 Handle ID
	15:0	Object 1 ICP 6 Handle
Rn+1.5	31:16	Object 1 ICP 5 Handle ID
	15:0	Object 1 ICP 5 Handle
Rn+1.4	31:16	Object 1 ICP 4 Handle ID
	15:0	Object 1 ICP 4 Handle
Rn+1.3	31:16	Object 0 ICP 7 Handle ID
	15:0	Object 0 ICP 7 Handle
Rn+1.2	31:16	Object 0 ICP 6 Handle ID
	15:0	Object 0 ICP 6 Handle
Rn+1.1	31:16	Object 0 ICP 5 Handle ID
	15:0	Object 0 ICP 5 Handle
Rn+1.0	31:16	Object 0 ICP 4 Handle ID
	15:0	Object 0 ICP 4 Handle
The following register is included only if DUAL_OBJECT mode and Include Vertex Handles is enabled and ICP Count > 7.		
Rn+2.7	31:16	Object 1 ICP 11 Handle ID
	15:0	Object 1 ICP 11 Handle
Rn+2.6	31:16	Object 1 ICP 10 Handle ID
	15:0	Object 1 ICP 10 Handle
Rn+2.5	31:16	Object 1 ICP 9 Handle ID
	15:0	Object 1 ICP 9 Handle
Rn+2.4	31:16	Object 1 ICP 8 Handle ID
	15:0	Object 1 ICP 8 Handle
Rn+2.3	31:16	Object 0 ICP 11 Handle ID
	15:0	Object 0 ICP 11 Handle
Rn+2.2	31:16	Object 0 ICP 10 Handle ID
	15:0	Object 0 ICP 10 Handle
Rn+2.1	31:16	Object 0 ICP 9 Handle ID
	15:0	Object 0 ICP 9 Handle
Rn+2.0	31:16	Object 0 ICP 8 Handle ID
	15:0	Object 0 ICP 8 Handle
The following register is included only if DUAL_OBJECT mode and Include Vertex Handles is enabled and ICP Count > 11.		
Rn+3.7	31:16	Object 1 ICP 15 Handle ID
	15:0	Object 1 ICP 15 Handle



GRF DWord	Bits	Description
Rn+3.6	31:16	Object 1 ICP 14 Handle ID
	15:0	Object 1 ICP 14 Handle
Rn+3.5	31:16	Object 1 ICP 13 Handle ID
	15:0	Object 1 ICP 13 Handle
Rn+3.4	31:16	Object 1 ICP 12 Handle ID
	15:0	Object 1 ICP 12 Handle
Rn+3.3	31:16	Object 0 ICP 15 Handle ID
	15:0	Object 0 ICP 15 Handle
Rn+3.2	31:16	Object 0 ICP 14 Handle ID
	15:0	Object 0 ICP 14 Handle
Rn+3.1	31:16	Object 0 ICP 13 Handle ID
	15:0	Object 0 ICP 13 Handle
Rn+3.0	31:16	Object 0 ICP 12 Handle ID
	15:0	Object 0 ICP 12 Handle
Varies (optional)	31:0	<p>Constant Data (optional):</p> <p>Some amount of constant data (possibly none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_GS command (taking the buffer enables into account).</p> <p>The Constant Data arrives in a non-interleaved format.</p>
Varies	31:0	<p>Pushed Vertex Data. There can be up to 32 vertices supplied, each with a size defined by the Vertex URB Entry Read Length state. The amount of data provided for each vertex is defined by the Vertex URB Entry Read Length state.</p> <p>For SINGLE or DUAL_INSTANCE dispatch modes, the pushed data for Vertex 0 immediately follows any pushed constant data. The pushed data for Vertex 1 immediately follows Vertex 0, and so on. There is no upper/lower swizzling of data.</p> <p>For DUAL_OBJECT dispatch mode, the pushed vertex data is split into upper and lower halves with Object 0 input vertices in the lower half, and Object 1 input vertices in the upper half.</p>

7.5 GS Thread Execution

A GS thread is capable of performing arbitrary algorithms given the thread payload (especially vertex) data and associated data structures (binding tables, sampler state, etc.) as input. Output can take the form of vertices output to the FF pipeline (at the GS unit) and/or data written to memory buffers via the DataPort.

The primary usage models for GS threads include (possible combinations of):

- Compiled application-provided “GS shader” programs, specifying an algorithm to convert the vertices of an input object into some output primitives. For example, a GS shader may convert lines of a line strip into polygons representing a corresponding segment of a blade of grass centered on the line. Or it could use adjacency information to detect silhouette edges of triangles and output polygons extruding out from the those edges. Or it could output absolutely nothing, effectively terminating the pipeline at the GS stage.



- Driver-generated instructions used to write pre-clipped vertices into memory buffers (see Stream Output below). This may be required whether or not an app-provided GS shader is enabled.
- Driver-generated instructions used to emulate API functions not supported by specialized hardware. These functions might include (but are not limited to):
 - Conversion of API-defined topologies into topologies that can be rendered (e.g., LINELOOP→LINESTRIP, POLYGON→TRIFAN, QUADs→TRIFAN, etc.)
 - Emulation of “Polygon Fill Mode”, where incoming polygons can be converted to points, lines (wireframe), or solid objects.
 - Emulation of wide/sprite points.
- Things best left to the imagination.

When rendering is required, concurrent GS threads must use the FF_SYNC message (URB shared function) to request an initial VUE handle and synchronize output of VUEs to the pipeline (see **URB** in *Shared Functions*). Only one GS thread can be outputting VUEs to the pipeline at a time. In order to achieve parallelism, GS threads should perform the GS shader algorithm (along with any other required functions) and buffer results (either in the GRF or scratch memory) before issuing the FF_SYNC message. The issuing GS thread will be stalled on the FF_SYNC writeback until it is that thread’s turn to output VUEs. As only one GS thread at a time can output VUEs, the post-FF_SYNC output portion of the kernel should be optimized as much as possible to maximize parallelism.

7.5.1 GS Thread Output

7.5.1.1 GS URB Entry

All outputs of a GS thread will be stored in the single GS thread output URB entry. Cut (1 bit/vertex) or StreamID (2 bits/vertex) bits are packed into an optional 1-8 32B header. The **Control Data Format** and **Control Data Header Size** states are used to specify the size and contents of the header data (if any).



Following the optional header is a variable number of 16B or 32B-aligned/granular vertices:

- When rendering is DISABLED, typically output vertices are 32B-aligned, with the exception of 16B-alignment for vertices $\leq 16B$ in length.
 - The absolute worst case size comes from three DW scalars output per vertex. If these are, say, three “.x” outputs, you need to store each DW in a 128b (16B) element, plus another pad 16B to keep the 32B alignment. So you require $4 \cdot 16B = 64B/\text{vertex}$. You have to have room for $1024 \text{ scalars} / 3 \text{ scalar/vtx} = 341 \text{ vertices}$. $341 \cdot 64B = 21,824B$. Then add 96B to hold 2b/vtx streamID and you get 21,920B entries.
- When rendering is ENABLED, each output vertex is 32B-aligned. Here the vertex header and vertex ‘position’ is required and therefore the minimum size vertex is 32B.



- Here the worst case size isn't as bad as render-disabled, as you have to have a 4DW position output, plus any additional output. So, say you output 5 DW per vertex. You need 64B/vertex (16B vtx header, 16B position, 16B for the 2nd element, and 16B of pad). You have to have room for 1024 scalars / 5 = 204 vertices. 204*64 = 13,056B. Then add 64B to hold 2b/vtx streamID and you get 13,120B entries.

The size of the URB entry should be based on the declared maximum # of output vertices and the declared output vertex size (the union of per-stream vertex structures, if required).

7.5.1.2 GS Output Topologies

The following table lists which primitive topology types are valid for output by a GS thread.

PrimTopologyType	Supported for GS Thread Output?
LINELIST	Yes
LINELIST_ADJ	No
LINESTRIP	Yes
LINESTRIP_ADJ	No
LINESTRIP_BF	Yes
LINESTRIP_CONT	Yes
LINESTRIP_CONT_BF	Yes
LINELOOP	No
POINTLIST	Yes
POINTLIST_BF	Yes
POLYGON	Yes
QUADLIST	No
QUADSTRIP	No
RECTLIST	Yes
TRIFAN	Yes
TRIFAN_NOSTIPPLE	Yes
TRILIST	Yes
TRILIST_ADJ	No
TRISTRIP	Yes
TRISTRIP_ADJ	No
TRISTRIP_REV	Yes
PATCHLIST_xxx	Yes

7.5.1.3 GS Output StreamID

When the **GS Enable** is DISABLED, output vertices will be assigned a StreamID = 0;

When the **GS Enable** is ENABLED, output vertices will be assigned a StreamID = **Default StreamID** under the following conditions:

- **Control Data Format** = 0, or
- **Control Data Format** > 0 and **Control Data Format** = GSCTL_CUT

When the GS is enabled, **Control Data Format** > 0 and **Control Data Format** = GSCTL_SID, output vertices will be assigned a StreamID as programmed in the Control Data output by the thread.

7.5.2 Stream Output

The final contents of Stream Output buffers must follow the strict pipeline ordering of vertices. Given this ordering requirement, it will be necessary to run the GS stage in a single-threaded fashion (**Maximum**



Number of Threads == 1). Otherwise concurrent GS threads might append vertices to the output buffer out of order.

Hardware support for the Stream Output is limited to a special “Streamed Vertex Buffer Write” DataPort message. (Refer to *DataPort* chapter). Through use of this message type, the GS thread can write from 1 to 4 DWords to specified ‘element’ (indexed entry) in a BUFFER surface. The DataPort will inhibit writes past the end of the buffer.

Software will likely need to define separate surface states for each SEB, and separate surface states for each element within the MEB structure. The surfaces are selected via the normal binding table mechanisms.

The need for separate SEB surface states is obvious, as the SEBs are separate buffers in memory. The MEB surface-per-element allows the GS kernel to address the MEB using an structure index. Here each surface would be specified as having the same structure pitch, but with different starting addresses corresponding to the different element offsets within the structure – in effect, defining a set of interleaved surfaces. The GS kernel would output one write message per element.

(Note that software could, if it wished, treat the MEB as a single 1D array of DWords, though it would then have to write the buffer one DWord at a time, performing the address calculations within the GS kernel. This should not be necessary, and is certainly not recommended due to obvious performance and complexity reasons.)

Programming Note: If the GS stage is enabled, software must always allocate at least one GS URB Entry. This is true even if the GS thread never needs to output vertices to the pipeline, e.g., when only performing stream output. This is an artifact of the need to pass the GS thread an initial destination URB handle.

7.5.3 Thread Termination

GS threads must terminate by sending a URB_WRITE_xxx message with the **EOT** and **Complete** bits set. The message header must contain correct values for the **GS Number of Output Handles for Slot 0**, **Handle ID 0**, and **URB Handle 0** fields. If in DUAL_INSTANCE or DUAL_OBJECT mode, the corresponding Object 1 fields must also be correct.

7.6 Primitive Output

(This section refers to output from the GS unit to the pipeline, not output from the GS thread)

The GS unit will output primitives (either passed-through or generated by a GS thread) in the proper order. This includes the buffering of a concurrent GS thread’s output until the preceding GS thread terminates. Note that the requirement to buffer subsequent GS thread output until the preceding GS thread terminates has ramifications on determining the number of VUEs allocated to the GS unit and the number of concurrent GS threads allowed.

7.7 Other Functionality

7.7.1 Statistics Gathering

There are a number of GS/StreamOutput pipeline statistics counters associated with the GS stage and GS threads. This subsection describes these counters and controls depending on device, even in the cases where functions outside of the GS stage (e.g., DataPort) are involved in the statistics gathering.



Refer to the *Statistics Gathering* summary provided earlier in this specification. Refer to the *Memory Interface Registers* chapter for details on these MMIO pipeline statistics counter registers, as well as the chapters corresponding to the other functions involved (e.g., DataPort, URB shared functions).

7.7.1.1 GS Invocations

The `GS_INVOCATIONS` counter is incremented by the **GSInvocations Increment Value** state for every input object, with the exception of `DUAL_OBJECT` dispatch where the counter is incremented by twice that amount. This allows software to (for example) support multiple instances in the GS kernel.

8. 3D Pipeline - Stream Output Logic (SOL) Stage

The Stream Output Logic (SOL) stage receives 3D topologies originating in the VF or GS stage. If enabled, the SOL stage uses programmed state information to copy portions of the vertex data associated with the incoming topologies across one or more Stream Output (SO) Buffers.

8.1 Input Buffering

For the purposes of stream output, the SOL stage breaks incoming topologies into independent objects without adjacency information. In the process, any adjacent-only vertices are ignored. For example, convert TRISTRIP_ADJ into independent 3-vertex triangles. However, if rendering is enabled, incoming topologies are passed to the Clip stage unmodified and therefore the Clip unit must be enabled if there is any possibility of “ADJ” topologies reaching it.

Note that the SOL unit should not see incomplete objects: the VF will remove incomplete input objects, and the GS will remove GS-generated incomplete objects.

The OSB (Object Staging Buffer) reorders the vertices of odd-numbered triangles in TRISTRIP topologies to match API requirements.

Incoming topologies are tagged with a 2-bit StreamID. The StreamID is 0 for topologies originating from the VF stage (i.e., 3DPRIMITIVE_XXX). For topologies output from the GS stage, the StreamID is set by the GS shader. A Stream n Vertex Length is associated with each stream, and defines how much data is read from the URB for vertices in that stream.

The following table specifies how the SOL stage streams out object vertices for each incoming topology type.

PrimTopologyType	Order of Vertices Streamed Out	Any SOL Notes
<PRIMITIVE_TOPOLOGY> (N = # of vertices)	[<object#>] = (<vert#>, ...);	
POINTLIST POINTLIST_BF	[0] = (0); [1] = (1); ...; [N-2] = (N-2);	
LINELIST (N is multiple of 2)	[0] = (0,1); [1] = (2,3); ...; [(N/2)-1] = (N-2,N-1)	
LINELIST_ADJ (N is multiple of 4)	[0] = (1,2); [1] = (5,6); ...; [(N/4)-1] = (N-3,N-2)	
LINESTRIP LINESTRIP_BF LINESTRIP_CONT LINESTRIP_CONT_BF (N >= 2)	[0] = (0,1); [1] = (1,2); ...; [N-2] = (N-2,N-1)	
LINESTRIP_ADJ (N >= 4)	[0] = (1,2); [1] = (2,3); ...; [N-4] = (N-3,N-2)	
LINELOOP	N/A	Not supported after VF.



PrimTopologyType	Order of Vertices Streamed Out	Any SOL Notes
TRILIST (N is multiple of 3)	[0] = (0,1,2); [1] = (3,4,5); ...; [(N/3)-1] = (N-3,N-2,N-1)	
RECTLIST	Same as TRILIST	Handled same as TRILIST.
TRILIST_ADJ (N is multiple of 6)	[0] = (0,2,4); [1] = (6,8,10); ...; [(N/6)-1] = (N-6,N-4,N-2)	
TRISTRIP (N >= 3) REORDER_LEADING	[0] = (0,1,2); [1] = (1,3,2); [k even] = (k,k+1,k+2) [k odd] = (k,k+2,k+1) [N-3] = (see above)	"Odd" triangles have vertices reordered to yield increasing leading vertices starting with v0.
TRISTRIP (N >= 3) REORDER_TRAILING	[0] = (0,1,2); [1] = (2,1,3); [k even] = (k,k+1,k+2) [k odd] = (k+1,k,k+2) [N-3] = (see above)	"Odd" triangles have vertices reordered to yield increasing trailing vertices starting with v2.
TRISTRIP_REV (N >= 3) REORDER_LEADING	[0] = (0,2,1) [1] = (1,2,3);...; [k even] = (k,k+2,k+1) [k odd] = (k,k+1,k+2) [N-3] = (see above)	"Even" triangles have vertices reordered to yield increasing leading vertices starting with v0.
TRISTRIP_REV (N >= 3) REORDER_TRAILING	[0] = (1,0,2) [1] = (1,2,3);...; [k even] = (k+1,k,k+2) [k odd] = (k,k+1,k+2) [N-3] = (see above)	"Even" triangles have vertices reordered to yield increasing trailing vertices starting with v2.
TRISTRIP_ADJ (N even, N >= 6) REORDER_LEADING	N = 6 or 7: [0] = (0,2,4) N = 8 or 9: [0] = (0,2,4); [1] = (2,6,4); ...; N > 10: [0] = (0,2,4); [1] = (2,6,4); ...; [k>1, even] = (2k, 2k+2, 2k+4); [k>2, odd] = (2k, 2k+4, 2k+2);...; Trailing object: [(N/2)-3, even] = (N-6,N-4,N- 2); [(N/2)-3, odd] = (N-6,N-2,N- 4);	"Odd" objects have vertices reordered to yield increasing-by-2 leading vertices starting with v0.
TRISTRIP_ADJ (N even, N >= 6) REORDER_TRAILING	N = 6 or 7: [0] = (0,2,4) N = 8 or 9: [0] = (0,2,4); [1] = (4,2,6); ...; N > 10: [0] = (0,2,4); [1] = (4,2,6); ...; [k>1, even] = (2k, 2k+2,	"Odd" objects have vertices reordered to yield increasing-by-2 trailing vertices starting with v4.



PrimTopologyType	Order of Vertices Streamed Out	Any SOL Notes
	$2k+4$; $[k>2, \text{odd}] = (2k+2, 2k, 2k+4, \dots)$; Trailing object: $[(N/2)-3, \text{even}] = (N-6, N-4, N-2)$; $[(N/2)-3, \text{odd}] = (N-4, N-6, N-2)$; 	
TRIFAN ($N > 2$)	$[0] = (0, 1, 2)$; $[1] = (0, 2, 3); \dots$; $[N-3] = (0, N-2, N-1)$; 	
TRIFAN_NOSTIPPLE	Same as TRIFAN	
POLYGON	Same as TRIFAN	
QUADLIST QUADSTRIP	N/A	Not supported after VF.
: PATCHLIST_1	$[0] = (0)$; $[1] = (1); \dots$; $[N-2] = (N-2)$; 	
: PATCHLIST_2	$[0] = (0, 1)$; $[1] = (2, 3); \dots$; $[(N/2)-1] = (N-2, N-1)$; 	
: PATCHLIST_3..32	similar to above	

8.2 Stream Output Buffers

Up to four SO buffers are supported. The SO buffer parameters (start/end address, etc.) are specified by the 3DSTATE_SO_BUFFER command.

The 3DSTATE_STREAMOUT command specifies an SO Buffer Enable bit for each of the buffers. If a buffer is disabled, its state is ignored and no output will be attempted for that buffer. Any attempt to output to that buffer will immediately signal an overflow condition.

The SOL stage maintains a current Write Offset register value for each SO buffer. These registers can be written via MI_LOAD_REGISTER_MEM or MI_LOAD_REGISTER_IMM commands. The SOL stage will increment the Write Offsets as a part of the SO function. Software can cause a Write Offset register to be written to memory via an MI_STORE_REGISTER_MEM command, though a preceding flush operation may be required to ensure that any previous SO functions have completed.

Project	Surface Format Name	Security
	R32G32B32A32_FLOAT	
	R32G32B32A32_SINT	
	R32G32B32A32_UINT	
	R32G32B32_FLOAT	
	R32G32B32_SINT	
	R32G32B32_UINT	
	R32G32_FLOAT	
	R32G32_SINT	
	R32G32_UINT	
	R32_SINT	
	R32_UINT	
	R32_FLOAT	



8.3 Stream Output Function

As previously mentioned, incoming 3D topologies are targeted at one of the four streams. The SOL stage contains state information specific to each of the four streams.

A stream's list of SO declarations (SO_DECL structures) is used to perform the SO function for objects targeted to that particular stream. The 3DSTATE_SO_DECL_LIST command is used to specify the list of SO_DECL structures for all four streams in parallel. Software is required to scan the SO_DECL lists of streams to determine which SO buffers are targeted. The Stream To Buffer Selects bits in 3DSTATE_SO_DECL_LIST must be programmed accordingly (if the buffer is targeted, the select bit must be set, else it must be cleared).

If a stream has no SO_DECL state defined (NumEntries is 0), incoming objects targeting that stream are effectively ignored. As there is no attempt to perform stream output, overflow detection is neither required nor performed.

Otherwise, an overflow check is performed. First any attempt to output to a disabled buffer is detected. This occurs when the stream has a Stream To Buffer Selects bit set but the corresponding SO Buffer Enable is clear. Assuming all targeted buffers are enabled, an additional check is made to ensure that there is enough room in each targeted buffer to hold the number of vertices which be output to it (for the input object). Here the buffer's current end address is compared to what the write offset would be if the output was performed. The latter value is computed as $(write_offset + vertex_count * buffer_pitch)$. If this value is greater than the end address, an overflow is signalled. This check is performed for each buffer included in Stream To Buffer Selects.

If an overflow is not signaled, the SO function is performed. The SO_DECL list for the targeted stream is traversed independently for each object vertex, and the operation specified by the SO_DECL structure is performed (typically causing data to be appended to an SO buffer). In the process, SO buffer Write Offsets are incremented.

8.4 3DSTATE_STREAMOUT

The 3DSTATE_STREAMOUT command specifies control information for the SOL stage. Included are enables and sizes for input streams and enables for output buffers.

Anytime the SOL unit MMIO registers or non-pipeline state are written, the SOL unit needs to receive a pipeline state update with SOL unit dirty state for information programmed in MMIO/NP to get loaded into the SOL unit.

The SOL unit incorrectly double buffers MMIO/NP registers and only moves them into the design for usage when control topology is received with the SOL unit dirty state.

If the state does not change, need to resend the same state.



Because of corruption, software must flush the whole fixed function pipeline when 3DSTATE_STREAMOUT changes state.

3DSTATE_STREAMOUT		
Source:	RenderCS	
Length Bias:	2	
This command contains pipelined state required by the SOL unit.		
DWord	Bit	Description
0	31:29	Command Type
		Default Value: 3h GFXPIPE
		Format: OpCode
	28:27	Command SubType
		Default Value: 3h GFXPIPE_3D
		Format: OpCode
	26:24	3D Command Opcode
		Default Value: 0h 3DSTATE_PIPELINED
		Format: OpCode
	23:16	3D Command Sub Opcode
	Default Value: 1Eh 3DSTATE_STREAMOUT	
	Format: OpCode	
15:8	Reserved	
	Project:	All
	Format:	MBZ
7:0	DWord Length	
	Default Value:	1h
	Format:	=n
	Total Length – 2	
1	31	SO Function Enable
		Project: All
		Format: U1
	If set, the SO function is enabled. Vertex data will be streamed out to memory (subject to overflow detection) as controlled by the various SO-related state variables. If clear, the SO function is disabled, and therefore no vertex data will be streamed out to memory. However, the Rendering Disable and Render Stream Select fields will still be used to determine which vertices (if any) are forwarded down the pipeline for (possible) rendering.	
	30	Rendering Disable
		Format: U1
	If set, the SO stage will not forward any topologies down the pipeline. If clear, the SO stage will forward topologies associated with Render Stream Select down the pipeline. This bit is used even if SO Function Enable is DISABLED.	
	29	Reserved
		Project: All
		Format: MBZ



3DSTATE_STREAMOUT			
28:27	Render Stream Select		
	Project:	All	
	Format:	U2	
	Description		
This field specifies which stream has been selected to be forwarded down the pipeline for possible rendering. Topologies from other streams will not be passed down the pipeline. If Rendering Disable is set, this field is ignored, as no topologies are sent down the pipeline. This bit is used even if SO Function Enable is DISABLED.			
26	Reorder Mode		
	Project:	All	
	This bit controls how vertices of triangle objects in TRISTRIP[_ADJ] and TRISTRIP_REV are reordered for the purposes of stream-out only (does not impact rendering). See table in Input Buffering.		
	Value	Name	Description
0h	REORDER_LEADING	Reorder the vertices of alternating triangles of a TRISTRIP[_ADJ] such that the leading (first) vertices are in consecutive order starting at v0. A similar reordering is performed on alternating triangles in a TRISTRIP_REV.	All
1h	REORDER_TRAILING	Reorder the vertices of alternating triangles of a TRISTRIP[_ADJ] such that the trailing (last) vertices are in consecutive order starting at v2. A similar reordering is performed on alternating triangles in a TRISTRIP_REV.	All
25	SO Statistics Enable		
	Project:	All	
	Format:	Enable	
	This bit controls whether StreamOutput statistics register(s) can be incremented.		
Value	Name	Description	Project
0h	Disable	SO_NUM_PRIMS_WRITTEN[0..3] and SO_PRIM_STORAGE_NEEDED[0..3] registers cannot increment.	All
1h	Enable	SO_NUM_PRIMS_WRITTEN[0..3] and SO_PRIM_STORAGE_NEEDED[0..3] registers can increment.	All
24:23	Reserved		
	Format:	MBZ	
22:12	Reserved		
	Project:	All	
	Format:	MBZ	
11	SO Buffer Enable [3]		
	Format:	U1 (See SO Buffer Enable [0])	
10	SO Buffer Enable [2]		
	Format:	U1 (See SO Buffer Enable [0])	
9	SO Buffer Enable [1]		



3DSTATE_STREAMOUT		
		Format: U1 (See SO Buffer Enable [0])
8	SO Buffer Enable [0]	Format: U1 If set, stream output to SO Buffer 0 is enabled. If clear, SO Buffer 0 is considered “not bound” and effectively treated as a zero-length buffer for the purposes of SO output and overflow detection. If an enabled stream’s Stream to Buffer Selects includes this buffer it is by definition an overflow condition. That stream will cause no writes to occur, and only SO_PRIM_STORAGE_NEEDED[<stream>] will increment. This bit is ignored if SO Function Enable is DISABLED.
7:0	Reserved	Project: All Format: MBZ
2	31:30	Reserved Project: All Format: MBZ
	29	Stream 3 Vertex Read Offset Project: All Format: U1 count of 256-bit units Specifies amount of data to skip over before reading back Stream 3 vertex data. (See Stream 0 Vertex Read Offset)
	28:24	Stream 3 Vertex Read Length Project: All Format: U5-1 count of 256-bit units (See Stream 0 Vertex Read Length)
	23:22	Reserved Project: All Format: MBZ
	21	Stream 2 Vertex Read Offset Project: All Format: U1 count of 256-bit units Specifies amount of data to skip over before reading back Stream 2 vertex data. (See Stream 0 Vertex Read Offset)
	20:16	Stream 2 Vertex Read Length Project: All Format: U5-1 count of 256-bit units



3DSTATE_STREAMOUT					
15:14	<p>Reserved</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Project:	All	Format:	MBZ
Project:	All				
Format:	MBZ				
13	<p>Stream 1 Vertex Read Offset</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U1 count of 256-bit units</td> </tr> </table> <p>Specifies amount of data to skip over before reading back Stream 1 vertex data. (See Stream 0 Vertex Read Offset)</p>	Project:	All	Format:	U1 count of 256-bit units
Project:	All				
Format:	U1 count of 256-bit units				
12:8	<p>Stream 1 Vertex Read Length</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U5-1 count of 256-bit units</td> </tr> </table> <p>(See Stream 0 Vertex Read Length)</p>	Project:	All	Format:	U5-1 count of 256-bit units
Project:	All				
Format:	U5-1 count of 256-bit units				
7:6	<p>Reserved</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Project:	All	Format:	MBZ
Project:	All				
Format:	MBZ				
5	<p>Stream 0 Vertex Read Offset</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U1 count of 256-bit units</td> </tr> </table> <p>Specifies amount of data to skip over before reading back Stream 0 vertex data. Must be zero if the GS is enabled and the Output Vertex Size field in 3DSTATE_GS is programmed to 0 (i.e., one 16B unit).</p>	Project:	All	Format:	U1 count of 256-bit units
Project:	All				
Format:	U1 count of 256-bit units				
4:0	<p>Stream 0 Vertex Read Length</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U5-1 count of 256-bit units</td> </tr> </table> <p>Specifies amount of vertex data to read back for Stream 0 vertices, starting at the Stream 0 Vertex Read Offset location. Maximum readback is 17 256-bit units (34 128-bit vertex attributes). Read data past the end of the valid vertex data has undefined contents, and therefore shouldn't be used to source stream out data.</p> <p>Must be zero (i.e., read length = 256b) if the GS is enabled and the Output Vertex Size field in 3DSTATE_GS is programmed to 0 (i.e., one 16B unit).</p>	Project:	All	Format:	U5-1 count of 256-bit units
Project:	All				
Format:	U5-1 count of 256-bit units				



8.5 3DSTATE_SO_DECL_LIST Command

The 3DSTATE_SO_DECL_LIST instruction defines a list of Stream Output (SO) declaration entries (SO_DECLs) and associated information for all specific SO streams in parallel.

Errata: All 128 decls for all four streams must be included whenever this command is issued. The “Num Entries [n]” fields still contain the actual numbers of valid decls.

3DSTATE_SO_DECL_LIST			
Source:		RenderCS	
Length Bias:		2	
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	1h 3DSTATE_NONPIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		17h 3DSTATE_SO_DECL_LIST	
Format:		OpCode	
15:9	Reserved		
	Project:	All	
	Format:	MBZ	
8:0	DWord Length		
	Format:	=n Total Length – 2	
		Format: Q1	
	Value	Name	Description
	3h	Excludes DWord (0,1) [Default]	Default value = 2(N-1)+3 h
1	31:16	Reserved	
		Project:	All
		Format:	MBZ
	15:12	Stream to Buffer Selects [3]	
		Project:	All
Format:		U4 bitmask	
		Index of SO Stream	



3DSTATE_SO_DECL_LIST															
	Identifies to which SO Buffers stream 3 outputs. See Stream To Buffer Selects [0] field description.														
11:8	Stream to Buffer Selects [2] <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U4 bitmask</td> </tr> </table>	Project:	All	Format:	U4 bitmask										
Project:	All														
Format:	U4 bitmask														
	Identifies to which SO Buffers stream 2 outputs. See Stream To Buffer Selects [0] field description.														
7:4	Stream to Buffer Selects [1] <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U4 bitmask</td> </tr> </table>	Project:	All	Format:	U4 bitmask										
Project:	All														
Format:	U4 bitmask														
	Identifies to which SO Buffers stream 1 outputs. See Stream To Buffer Selects [0] field description.														
3:0	Stream to Buffer Selects [0] <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U4 bitmask</td> </tr> </table> <p>Identifies to which SO Buffers stream 0 outputs (irrespective of whether those buffers are enabled via 3DSTATE_STREAMOUT). Software is required to scan the SO_DECL list in order to provide this summary information. Note: For “inactive” streams, software must program this field to all zero (no buffers written to) and the corresponding Num Entries field to zero (no valid SO_DECLs).</p> <table border="1"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td>1xxb</td> <td>SO Buffer 3</td> </tr> <tr> <td>x1xb</td> <td>SO Buffer 2</td> </tr> <tr> <td>xx1b</td> <td>SO Buffer 1</td> </tr> <tr> <td>xxx1b</td> <td>SO Buffer 0</td> </tr> </tbody> </table>	Project:	All	Format:	U4 bitmask	Value	Name	1xxb	SO Buffer 3	x1xb	SO Buffer 2	xx1b	SO Buffer 1	xxx1b	SO Buffer 0
Project:	All														
Format:	U4 bitmask														
Value	Name														
1xxb	SO Buffer 3														
x1xb	SO Buffer 2														
xx1b	SO Buffer 1														
xxx1b	SO Buffer 0														
2	31:24 Num Entries [3] <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U8 #entries</td> </tr> </table> <p>Specifies the number of valid SO_DECL entries for Stream 3. (See notes in Num Entries [0] field description).</p> <table border="1"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td>[0,128]</td> <td>entries</td> </tr> </tbody> </table>	Project:	All	Format:	U8 #entries	Value	Name	[0,128]	entries						
Project:	All														
Format:	U8 #entries														
Value	Name														
[0,128]	entries														
	23:16 Num Entries [2] <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U8 #entries</td> </tr> </table> <p>Specifies the number of valid SO_DECL entries for Stream 2. (See notes in Num Entries [0] field description).</p> <table border="1"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td>[0,128]</td> <td>entries</td> </tr> </tbody> </table>	Project:	All	Format:	U8 #entries	Value	Name	[0,128]	entries						
Project:	All														
Format:	U8 #entries														
Value	Name														
[0,128]	entries														
15:8	Num Entries [1]														



3DSTATE_SO_DECL_LIST

3DSTATE_SO_DECL_LIST									
	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U8 #entries</td> </tr> </table>	Project:	All	Format:	U8 #entries				
	Project:	All							
Format:	U8 #entries								
<p>Specifies the number of valid SO_DECL entries for Stream 1. (See notes in Num Entries [0] field description).</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 50%; text-align: center;">Value</th> <th style="width: 50%; text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">[0,128]</td> <td style="text-align: center;">entries</td> </tr> </tbody> </table>		Value	Name	[0,128]	entries				
Value	Name								
[0,128]	entries								
7:0	<p>Num Entries [0]</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U8 #entries</td> </tr> </table> <p>Specifies the number of valid SO_DECL entries for Stream 0. Note that the SO_DECLs are programmed in groups of four (one SO_DECL for each of the four streams). Therefore the number of 2-DWord groups of SO_DECLs supplied in this command is derived from the stream(s) with the most valid SO_DECLs. The NumEntries value specific to each stream will indicate how many SO_DECLs are valid for that particular stream. Any trailing invalid SO_DECLs supplied for streams with fewer valid SO_DECLs will be ignored. It is legal to specify Num Entries = 0 for all four streams simultaneously. In this case there will be no SO_DECLs included in the command (only DW 0-2). Note that all Stream to Buffer Selects bits must be zero in this case (as no streams produce output).</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 50%; text-align: center;">Value</th> <th style="width: 50%; text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">[0,128]</td> <td style="text-align: center;">entries</td> </tr> </tbody> </table>	Project:	All	Format:	U8 #entries	Value	Name	[0,128]	entries
Project:	All								
Format:	U8 #entries								
Value	Name								
[0,128]	entries								
3..4	<p>63:48 SO_DECL[3,1]</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>SO_DECL</td> </tr> </table> <p>This field contains Stream 3 SO_DECL 0</p>	Project:	All	Format:	SO_DECL				
	Project:	All							
	Format:	SO_DECL							
	<p>47:32 SO_DECL[2,1]</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>SO_DECL</td> </tr> </table> <p>This field contains Stream 2 SO_DECL 0</p>	Project:	All	Format:	SO_DECL				
Project:	All								
Format:	SO_DECL								
<p>31:16 SO_DECL[1,1]</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>SO_DECL</td> </tr> </table> <p>This field contains Stream 1 SO_DECL 0</p>	Project:	All	Format:	SO_DECL					
Project:	All								
Format:	SO_DECL								
<p>15:0 SO_DECL[0,1]</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>SO_DECL</td> </tr> </table> <p>This field contains Stream 0 SO_DECL 0</p>	Project:	All	Format:	SO_DECL					
Project:	All								
Format:	SO_DECL								
5..6	<p>63:48 SO_DECL[3,1]</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>SO_DECL</td> </tr> </table> <p>This field contains Stream 3 SO_DECL 1</p>	Project:	All	Format:	SO_DECL				
	Project:	All							
Format:	SO_DECL								
<p>47:32 SO_DECL[2,1]</p>									



3DSTATE_SO_DECL_LIST		
	Project: All	
	Format: SO_DECL	
	This field contains Stream 2 SO_DECL 1	
31:16	SO_DECL[1,1]	
	Project: All	
	Format: SO_DECL	
This field contains Stream 1 SO_DECL 1		
15:0	SO_DECL[0,1]	
	Project: All	
	Format: SO_DECL	
This field contains Stream 0 SO_DECL 1		
7..n	63:48	SO_DECL[3,n]
		Project: All
		Format: SO_DECL
This field contains Stream 3 SO_DECL n		
47:32		SO_DECL[2,n]
		Project: All
		Format: SO_DECL
This field contains Stream 2 SO_DECL n		
31:16		SO_DECL[1,n]
		Project: All
		Format: SO_DECL
This field contains Stream 1 SO_DECL n		
15:0		SO_DECL[0,n]
		Project: All
		Format: SO_DECL
This field contains Stream 0 SO_DECL n		



8.5.1 SO_DECL Structure Definition

SO_DECL			
Source:	RenderCS		
Default Value:	0x00000000		
<p>A list of SO_DECL structures are passed in the 3DSTATE_SO_DECL_LIST command. Each structure specifies either (a) the source and destination of an up-to-4-DWord appending write into an SO buffer, or (b) how many DWords to skip over in the destination SO buffer (i.e., a “hole” where the previous buffer contents are maintained). Workaround for IVBGT2:A0 (ends IVBGT2:B0): because of corruption in IVBGT2:A0, software needs to put a noop decl (Hole flag is 0, Component Mask is 0) as the first decl in every decl list.</p>			
DWord	Bit	Description	
0	15:14	Reserved	
		Project:	All
		Format:	MBZ
	13:12	Output Buffer Slot	
		Project:	All
		Format:	U2 Buffer Index
			This field selects the destination output buffer slot.
	11	Hole Flag	
		Project:	All
		Format:	Flag
			If set, the Component Mask field indirectly specifies a number of 32-bit locations to skip over (leave unmodified in memory) in the selected output buffer. The Register Index field is ignored. The only permitted Component Mask values are as follows:
			0x0 No Dwords are skipped over (SO_DECL performs no operation)
			0x1 (X) Skip 1 DWord
			0x3 (XY) Skip 2 DWords
			0x7 (XYZ) Skip 3 DWords
		0xF (XYZW) Skip 4 DWords	
10	Reserved		
	Project:	All	
	Format:	MBZ	
9:4	Register Index		
	Project:	All	
	Format:	U6 128-bit granular offset into the source vertex read data	
		If Hole Flag is clear, this field specifies the 128-bit offset into the source vertex data which supplies the source data to be written to the destination buffer, where the individual 32-component destination locations are selected by Component Mask. e.g., Register Index 0 corresponds with the first 128 bits of the data read from the vertex URB entry (as per corresponding Vertex Read Offset state)	
		There is only enough internal storage for the 128-bit vertex header and 32 128-bit vertex attributes.	

SO_DECL			
	Value	Name	
	[0,32]		
	0h	[Default]	
	Programming Notes		
	It is the responsibility of software to map any API-visible source data specifications (e.g., vertex register number) into 128-bit granular URB read offsets.		
3:0	Component Mask		
	Project:	All	
	Format:	MASK 4-bit Mask	
	<p>This field is a 4-bit bitmask that selects which contiguous 32-bit component(s) are either written or skipped-over in the destination buffer.</p> <p>If this field is zero the SO_DECL operation is effectively a no-op. No data will be appended to the destination and the destination buffer's write pointer will not be advanced.</p> <p>If the Hole Flag is set, this field (if non-zero) indirectly specifies how much the destination buffer's write pointer should be advanced. See Hole Flag description above for restrictions on this field.</p> <p>If the Hole Flag is clear, this field (if non-zero) selects which source components are to be written to the destination buffer. The components must be contiguous, e.g. YZW is legal, but XZW is not. The selected source components are written to the destination buffer starting at the current write pointer, and then the write pointer is advanced past the written data. E.g., if YZW is specified, the three (YZW) components of the source register will be written to the destination buffer at the current write pointer, and the write pointer will be advanced by 3 DWords.</p>		
	Value	Name	Project
	0h	[Default]	
	xxx1b	SO_DECL_COMPMASK_X	All
	xx1xb	SO_DECL_COMPMASK_Y	All
	x1xxb	SO_DECL_COMPMASK_Z	All
	1xxxb	SO_DECL_COMPMASK_W	All



8.6 3DSTATE_SO_BUFFER

The 3DSTATE_SO_BUFFER command specifies the location and characteristics of an SO buffer in memory.

3DSTATE_SO_BUFFER			
Source:		RenderCS	
Length Bias:		2	
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	1h 3DSTATE_NONPIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		18h 3DSTATE_SO_BUFFER	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	DWord Length		
	Default Value:	2h Excludes DWord (0,1)	
	Format:	=n	
	Total Length - 2		
1	31	Reserved	
		Project:	All
		Format:	MBZ
	30:29	SO Buffer Index	
		Project:	All
		Format:	U2
	Specifies which of the four SO Buffers is being defined.		
	28:25	SO Buffer Object Control State	
		Format:	MEMORY_OBJECT_CONTROL_STATE
		Specifies the memory object control state for the SO buffer.	
24:22	Reserved		
	Format:	MBZ	
21:12	Reserved		
	Project:	All	
	Format:	MBZ	
11:0	Surface Pitch		



3DSTATE_SO_BUFFER															
	<table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U12 Pitch in Bytes</td> </tr> <tr> <td colspan="2">This field specifies the pitch of the SO buffer in #Bytes.</td> </tr> <tr> <td style="text-align: center;">Value</td> <td style="text-align: center;">Name</td> </tr> <tr> <td>[0,2048]</td> <td>Must be 0 or a multiple of 4 Bytes.</td> </tr> <tr> <td colspan="2" style="text-align: center;">Programming Notes</td> </tr> <tr> <td colspan="2">A Surface Pitch of 0 indicates an un-bound buffer. No writes are performed. Surface Base Address is ignored.</td> </tr> </table>	Project:	All	Format:	U12 Pitch in Bytes	This field specifies the pitch of the SO buffer in #Bytes.		Value	Name	[0,2048]	Must be 0 or a multiple of 4 Bytes.	Programming Notes		A Surface Pitch of 0 indicates an un-bound buffer. No writes are performed. Surface Base Address is ignored.	
Project:	All														
Format:	U12 Pitch in Bytes														
This field specifies the pitch of the SO buffer in #Bytes.															
Value	Name														
[0,2048]	Must be 0 or a multiple of 4 Bytes.														
Programming Notes															
A Surface Pitch of 0 indicates an un-bound buffer. No writes are performed. Surface Base Address is ignored.															
2	<table border="1"> <tr> <td>31:2</td> <td>Surface Base Address</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>GraphicsAddress[31:2]</td> </tr> <tr> <td colspan="2">This field specifies the starting DWord address LSBs of the buffer in Graphics Memory.</td> </tr> </table>	31:2	Surface Base Address	Project:	All	Format:	GraphicsAddress[31:2]	This field specifies the starting DWord address LSBs of the buffer in Graphics Memory.							
	31:2	Surface Base Address													
Project:	All														
Format:	GraphicsAddress[31:2]														
This field specifies the starting DWord address LSBs of the buffer in Graphics Memory.															
1:0	<table border="1"> <tr> <td>Reserved</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Reserved	Project:	All	Format:	MBZ									
Reserved															
Project:	All														
Format:	MBZ														
3	<table border="1"> <tr> <td>31:2</td> <td>Surface End Address</td> </tr> <tr> <td>Format:</td> <td>GraphicsAddress[31:2]</td> </tr> <tr> <td colspan="2">This field specifies the ending DWord address of the buffer in Graphics Memory.</td> </tr> </table>	31:2	Surface End Address	Format:	GraphicsAddress[31:2]	This field specifies the ending DWord address of the buffer in Graphics Memory.									
	31:2	Surface End Address													
Format:	GraphicsAddress[31:2]														
This field specifies the ending DWord address of the buffer in Graphics Memory.															
1:0	<table border="1"> <tr> <td>Reserved</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Reserved	Project:	All	Format:	MBZ									
Reserved															
Project:	All														
Format:	MBZ														

8.7 Rendering Disable

Independent of SOL function enable, if rendering (i.e, 3D pipeline functions past the SOL stage) is enabled (via clearing the Rendering Disable bit), the SOL stage will pass topologies for a specific input stream (as selected by Render Stream Select) down the pipeline, with the exception of PATCHLIST_n topologies which are never passed downstream. Software must ensure that the vertices exiting the SOL stage include a vertex header and position value so that the topologies can be correctly processed by subsequent pipeline stages. Specifically, rendering must be disabled whenever 128-bit vertices are output from a GS thread.

If Rendering Disable is set, the SOL stage will prevent any topologies from exiting the SOL stage.

8.8 Statistics

The SOL stage controls the incrementing of two 64-bit statistics counter registers for each of the four output buffer slots, SO_NUM_PRIMS_WRITTEN[] and SO_PRIM_STORAGE_NEEDED[].



9. 3D Pipeline – Clip Stage

9.1 3D Pipeline – CLIP Stage Overview

The CLIP stage of the 3D Pipeline is similar to the GS stage in that it can be used to perform general processing on incoming 3D objects via spawned threads. However, the CLIP stage also includes specialized logic to perform a *ClipTest* function on incoming objects. These two usage models of the CLIP stage are outlined below.

Refer to the *Common 3D FF Unit Functions* subsection in the *3D Overview* chapter for a general description of a 3D Pipeline stage, as much of the CLIP stage operation and control falls under these “common” functions. I.e., many of the CLIP stage state variables and CLIP thread payload parameters are described in *3D Overview*, and although they are listed here for completeness, that chapter provides the detailed description of the associated functions.

Refer to this chapter for an overall description of the CLIP stage, details on the *ClipTest* function, and any exceptions the CLIP stage exhibits with respect to common FF unit functions.

9.1.1 Clip Stage – General-Purpose Processing

Numerous state variable controls are provided to tailor the *ClipTest* function as required by the API or primitive characteristics. These controls allow a mode where all objects are passed to CLIP threads, and in this regard the CLIP stage can be used as a second GS stage. However, unlike the GS stage, primitives output by CLIP threads will not be subject to 3D Clipping, and therefore any clip-testing/clipping of these primitives (if required) would need to be performed by the CLIP thread itself.

9.1.2 Clip Stage – 3D Clipping

The *ClipTest* fixed function is provided to optimize the CLIP stage for support of generalized *3D Clipping*. The CLIP FF unit examines the position of incoming vertices, performs a fixed function *VertexClipTest* on these positions, and then examines the results for the vertices of each independent object in *ClipDetermination*.

The results of *ClipDetermination* indicate whether an object is to be processed by a thread (*MustClip*), discarded (*TrivialReject*) or passed down the pipeline unmodified (*TrivialAccept*). In the *MustClip* case, the spawned thread is responsible for performing the actual 3D Clipping algorithm. The CLIP thread is passed the source object vertex data and is able to output a new, arbitrary 3D primitive (e.g., the clipped primitive), or no output at all. Note that the output primitive is independent in that it is comprised of newly-generated VUEs, and does not share vertices with the source primitive or other CLIP-generated primitives.

New vertices produced by the CLIP threads are stored in the URB. Their Vertex Headers are then read from the VUEs in order to insert the relevant information into the 3D pipeline. The CLIP unit maintains the proper ordering of CLIP-generated primitives and any surrounding trivially-accepted primitives. The CLIP unit also supports multiple concurrent CLIP threads and maintains the proper ordering of the thread outputs as dictated by the order of the source objects.

The outgoing primitive stream is sent down the pipeline to the Strip/Fan (SF) FF stage (now including the read-back VUE Vertex Header data such as Vertex Position (NDC or screen space), RTAIndex, VPIndex,



PointWidth) and control information (PrimType, PrimStart, PrimEnd) while the remainder of the vertex data remains in the VUE in the URB.

9.1.3 Fixed Function Clipper

The GPU supports Fixed Function Clipping.

9.2 Concepts

This section provides an overview of 3D clip-testing and clipping concepts. It is provided as background material: some of the concepts impact HW functionality while others impact CLIP kernel functionality.

9.2.1 The Clip Volume

3D objects are optionally clipped to the *clip volume*. The clip volume is defined as the *intersection* of a set of *clip half-spaces*. Six of these half-spaces define the view volume, while additional, user-defined half-spaces can be employed to perform clipping (or at least culling) within the view volume.

The CLIP stage design will permit the enable/disable of certain subsets of these clip half-spaces. This capability can be used, for example, to disable viewport, guardband, and near and far clipping as required by the API and other conditions.

9.2.1.1 View Volume

The intersection of the six view half-spaces defines the *view volume*. The view volume is defined in 4D clip space coordinates as:

View Clip Plane	'Outside' Condition	
	4D Clip Space	NDC space, positive w
XMIN (NDC Left)	$\text{clip.x} < -\text{clip.w}$	$\text{ndc.x} < -1$
XMAX (NDC Right)	$\text{clip.w} < \text{clip.x}$	$\text{ndc.x} > 1$
YMIN (NDC Bottom)	$\text{clip.y} < -\text{clip.w}$	$\text{ndc.y} < -1$
YMAX (NDC top)	$\text{clip.w} < \text{clip.y}$	$\text{ndc.y} > 1$
ZMIN (NDC Near)	OGL: $\text{clip.z} < -\text{clip.w}$	OGL: $\text{ndc.z} < -1.0$
ZMAX (NDC Far)	$\text{clip.w} < \text{clip.z}$	$\text{ndc.z} > 1.0$



Note that, since the 2D (X,Y) extent of the projected view volume is subsequently mapped to the 2D pixel space viewport, the terms “viewport” and “view volume” are used somewhat interchangeably in this discussion.

The CLIP unit will perform view volume clip test using NDC coordinates (the results of the speculative PerspectiveDivide). The treatment of negative `ndc.w` and invalid (NaN, +/-INF) coordinates is clarified below.

Negative W Coordinates

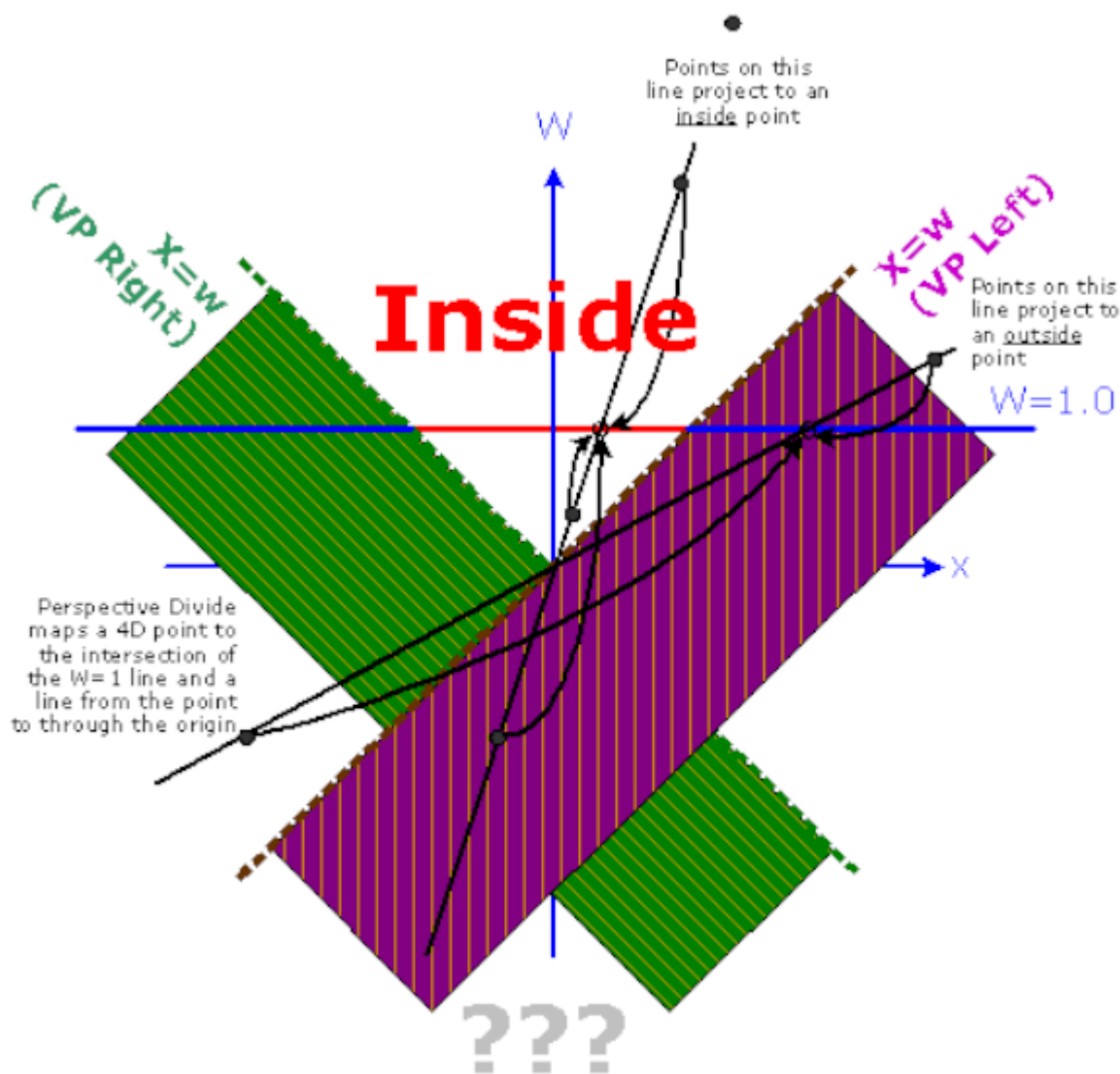
Consider for a moment vertices with a negative `clip.w` coordinate. Examination of the API definitions for “outside” shows that it is impossible for that vertex to be considered *inside* both the XMIN (NDC Left) and XMAX (NDC Right) planes. The `clip.x` coordinate would need to be greater than or equal to some positive value (`-clip.w`) to be considered inside the XMIN plane, while also being less than or equal to the negative (`clip.w`) value to be considered inside the XMAX plane. Obviously both these conditions cannot be met simultaneously, so a vertex with a negative `clip.w` coordinate will always appear outside.

Surprisingly, it is possible for a vertex to be *outside* both the XMIN and XMAX planes (and likewise for the Y axis). This arises when `clip.w` is negative and `clip.x` falls between `clip.w` and `-clip.w`. Note, however, that in NDC space (post perspective-divide), this same vertex would be considered *inside*. This disparity arises from the loss of information from the perspective divide operation, specifically the signs of the input operands. The CLIP stage will avoid this artifact by supporting an additional `clip.w=0` clip plane – a negative `ndc.rhw` value indicates the point is outside of the `clip.w=0` plane.

The assumption made in the Clip stage is that only the `w>0` portion of clip space is considered visible. The `VertexClipTest` function tests each incoming `1/w` value and, if negative, the vertex is tagged as being outside the `w=0` plane. These vertex outcodes are combined in `ClipDetermination` to determine TA/TR/MC status.

A negative `w` coordinate poses an additional issue due to the fact that `VertexClipTest` is performed using post-perspective-projection coordinates (NDC or screen space). This disparity arises from the loss of information from the perspective divide operation, specifically the signs of the input operands. For example, to test for $(x > w)$ using NDC coordinates, $(x/w > 1)$ must be used when $w > 0$, and $(x/w < 1)$ must be used when $w < 0$. The `VertexClipTest` function therefore uses the sign of the incoming `1/w` coordinate to select the appropriate comparison function for each of the VP and GB clip planes.

As the CLIP thread performs clipping in 4D clip space, only the truly visible portions of objects (i.e, meeting the 4D clip space visibility criteria) will be considered. The CLIP thread should not output negative `w` (clip or NDC) coordinates.



B6941-01

9.2.2 User-Specified Clipping

The various APIs define mechanisms by which objects can be clipped or culled according to some user-specified parameter(s) in addition to the implied viewport clipping. The HW support of these mechanisms is restricted to use of the 8 UserClipFlags (UCFs) of the VUE Vertex Header. Software is required to provide the remaining support (e.g., the JITTER including GEN4 instructions to cause a distance value to be computed, tested for visibility, and generation of the appropriate UCF bit.)

9.2.3 Guard Band

Note: Refer to Vertex X,Y Clamping and Quantization in the SF stage section for device-specific guardband size information.

3DClipping is time consuming. For cases where 2DClipping is sufficient, we are willing to forgo 3DClipping and instead apply 2DClipping during rendering. In the general case, this is possible only

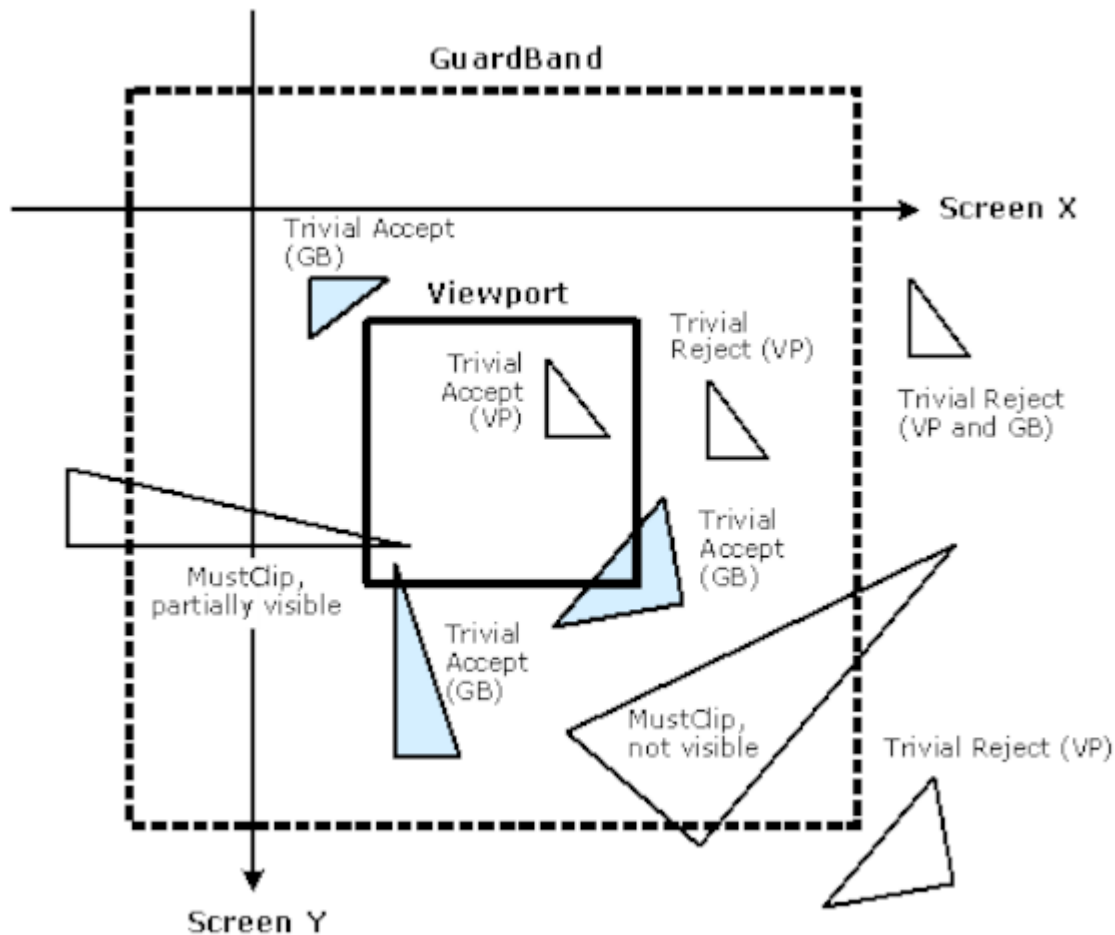
when an object is totally within the ZMin and ZMax planes, and only clipping to the view volume X/Y MIN/MAX clip planes is required, as 2DClipping is restricted to a screen-aligned 2D rectangle.

However, we must ensure that the 2D extent of these objects do not exceed the limitations of the renderer's coordinate space (see Vertex X,Y Clamping and Quantization in the SF section). Therefore we define a 2D *guardband* region corresponding to (though likely somewhat smaller than) the maximum 2D extent supported by the renderer. During VertexClipTest, vertices are (optionally) subjected to an additional visibility test based on the 2D guardband region.

During ClipDetermination, if an object is not trivially-rejected from the 2D viewport, the XMIN_GB, XMAX_GB, YMIN_GB and YMAX_GB guardband outcodes are used instead of the XMIN, XMAX, YMIN, YMAX view volume outcodes to determine trivial-accept. This will allow objects that fall within the guardband and possibly intersect the viewport to be trivially-accepted and passed down the pipeline.

The diagram below shows some examples of objects (triangles) in relation to the viewport and guardband. The shaded triangles are examples of triangles that are not trivially accepted to the viewport but trivially accepted to the guardband and therefore passed to down the pipeline. Without the guardband, these triangles would have to be submitted to a CLIP thread.

Normal Guardband Operation

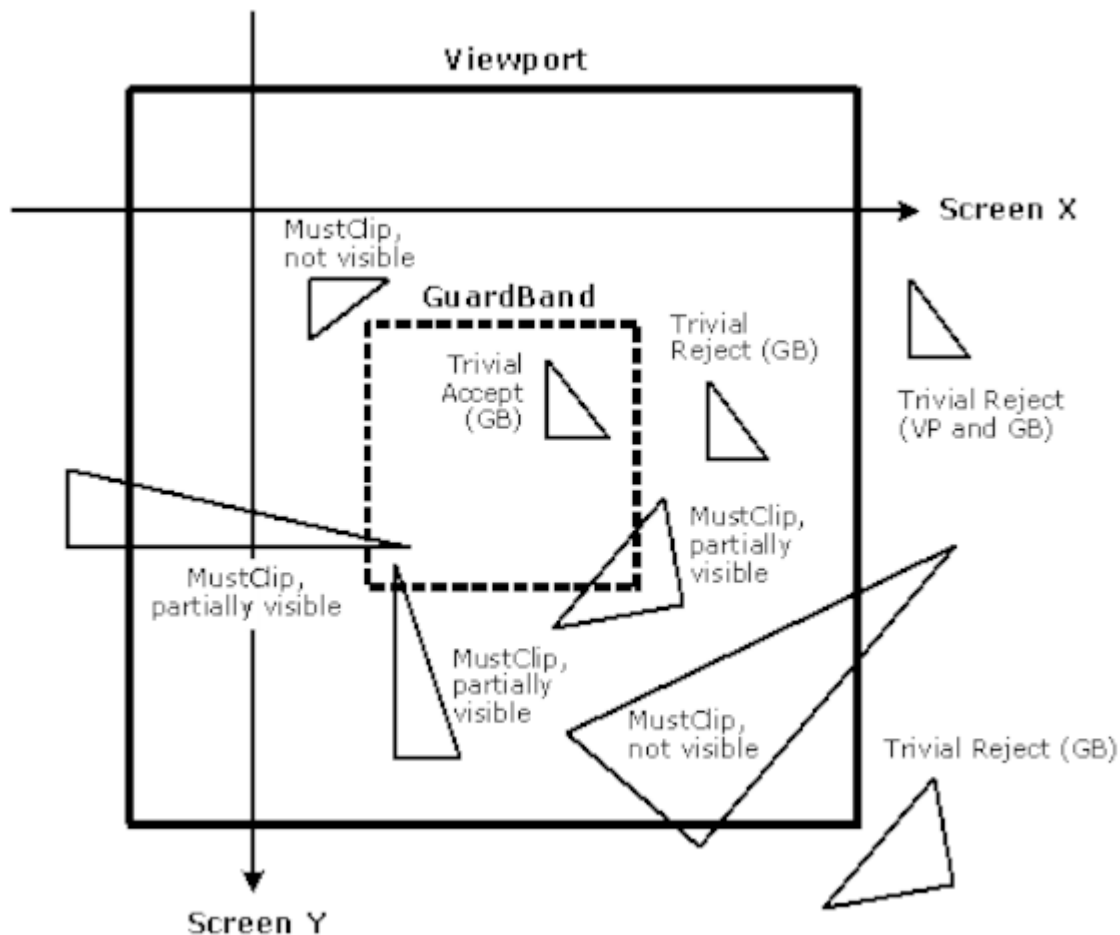


B.6822-01

The CLIP stage needs to handle the case where the viewport XY is larger than the screen space coordinate range supported by the SF and WM units. This condition may arise when the API defines an

implicit 2D clip between the viewport XY extent and the rendertarget. In the 3D pipeline, the guardband *must* be used to force explicit clipping in order to ensure legal coordinates are passed out of the CLIP stage. Therefore the CLIP unit supports a guardband that can be larger or smaller than the viewport (in any particular direction). The following diagram illustrates a case with a very large viewport, extending well beyond the guardband. Note that the only trivial accept case is where objects are completely within the guardband.

Very Large Viewport Case

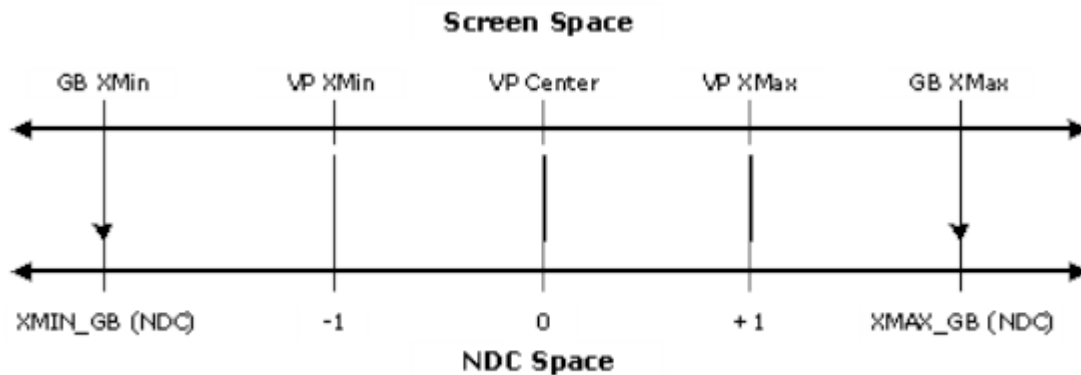


B.6842-01

9.2.3.1 NDC Guardband Parameters

Note: Refer to *Vertex X,Y Clamping and Quantization in the SF stage* section for device-specific guardband size information.

When the CLIP unit performs VertexClipTest in NDC space, the guardband limits must be provided as NDC coordinates. The diagram below shows how the guardband NDC coordinates are derived. Specifically, the XMIN_GB NDC coordinate is simply the ratio of the (screen space) distance from the screen space VP center to the screen space GB XMin boundary over the distance from the VP center to the VP XMin (left) boundary. A similar computation yields the XMAX_GB (right), YMIN_GB (bottom) and YMAX_GB (top) guardband NDC coordinates.



B.6843-01

As these guardband parameters are defined relative to the viewport, each of the up-to-16 sets of viewport specifications supported in the 3D pipeline will require a corresponding set of guardband parameters. These guardband parameters are provided as a separate memory-resident state structure (CLIP_VIEWPORT), and referenced via the **Clipper Viewport State Pointer** contained in the CLIP_STATE structure. Note that the CLIP_VIEWPORT structure has a different definition than the SF_VIEWPORT structure used by the SF unit.

9.2.4 Vertex-Based Clip Testing & Considerations

The CLIP unit performs clip test and determines whether objects need to be clipped based solely on information (position, UserClipFlags) provided at the *vertices* of the object as they arrive at the clip stage. Issues arise if and when the corresponding rendered object is not constrained to the convex hull of the object. Different APIs impose different treatment of these conditions.

In addition and in the more general case, a CLIP thread could be used to convert the object (as defined by its vertices) into some arbitrary output primitive. In this case, the CLIP unit's ClipTest/ClipDetermination logic may not be suitable for determination of when to reject/accept/clip objects. In this case the ClipMode can be used to route all (or all non-rejected) objects to CLIP threads, where the proper clip-test and clipping can occur in the CLIP kernel.

One issue that arises is whether a trivial-reject to the VPXY is suitable. If this were allowed, an object might be discarded even if it would have been partially visible in the viewport. A second issue is whether a TA against the GB is suitable. If this were allowed, portions of the rendered object might be visible in the VP even if the object should have been clipped out of the VP.

9.2.4.1 Triangle Objects

In the normal processing of triangle-based primitives (tristrip/trilist/polygon/etc.), the footprint of each triangle is constrained to the 2D convex hull. I.e., the rendering of these triangles will not produce pixels outside of the triangle. Therefore the normal operation of the CLIP unit functions will support the proper clip testing and clip determination for triangle objects:

- Both the VPXY and GB clip boundaries can be utilized (as described above). If the triangle is TR against the VP, it can be discarded. Otherwise, if the triangle is TA against the GB, it can be passed down the pipeline (assuming it is TA against VPZ, UCFs, etc.) and properly handled by 2DClipping.
- The GB parameters can be programmed to coincide with the maximum allowable screen space extent (though making the GB marginally smaller than this max extent is highly recommended).

9.2.4.2 Non-Wide Line Objects

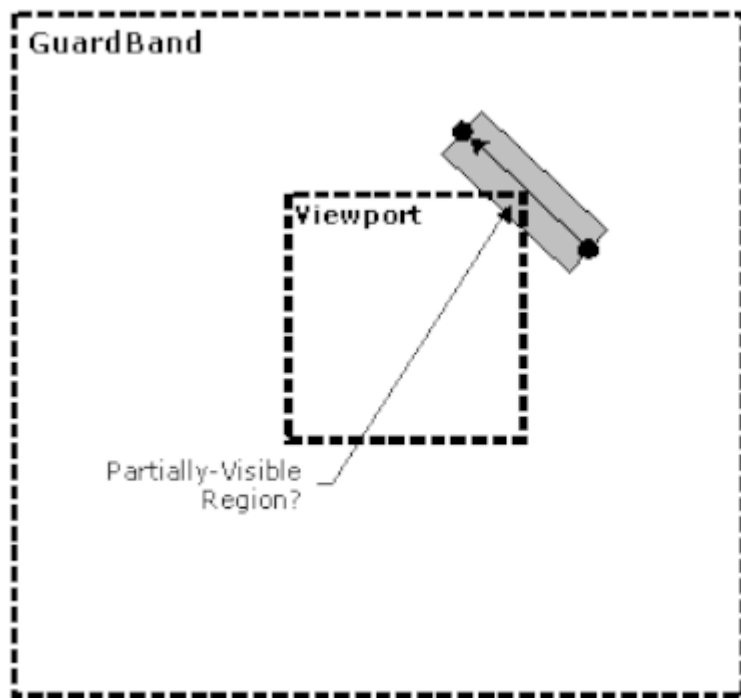
In the normal processing of non-wide, line-based primitives (linestrip/linelist/etc.), the footprint of each line is constrained to the 2D convex hull. I.e., the rendering of these lines will not produce pixels off of the line. Therefore the normal operation of the CLIP unit functions will support the proper clip testing and clip determination for non-wide line objects. (See Triangle Objects above).

9.2.4.3

9.2.4.4 Wide Line Objects

The rendering hardware supports wide lines (solid lines with a line width or anti-aliased lines). When rendered, pixels outside of the convex hull will be generated.

The following diagram shows an example of a wide line that normally would be TA against the GB. If the TA is allowed, the partially-visible region of the line would be rendered.



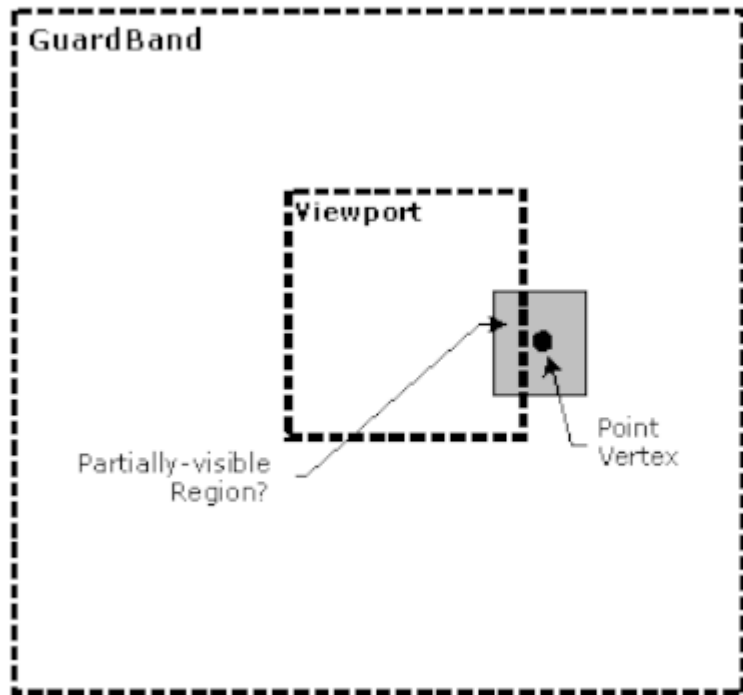
B6844-01

In general, OpenGL dictates that the partially-visible region must not be rendered. In this case the line must be clipped-out against the VPXY (not TA against the GB). To accomplish this, SW could disable the GB when drawing wide lines.

9.2.4.5 Wide Points

The GEN rendering hardware supports a width parameter for native line objects. When rendered, pixels surrounding the point (center) vertex will be generated.

The following diagram shows an example wide point that normally would be TR against the VPXY. If the TR is allowed, the partially-visible region of the point would not be rendered.



B.6845-01

In general, OpenGL dictates that the partially-visible region must not be rendered. In this case the point must be TR against the VPXY (not TA against the GB). To accomplish this, SW could disable the GB when drawing wide points.

9.2.4.6 RECTLIST

The CLIP unit treats RECTLIST exactly like TRILIST. No special consideration is made for the implied 4th vertex of each rectangle (although ViewportXY and Guardband VertexClipTest theoretically should be sufficient to drive ClipDetermination). Given this, and the fact that RECTLIST is primarily intended for driver-generated “BLT” functions, there are number of restrictions on the use of RECTLIST, especially regarding the CLIP unit. Refer to the RECTLIST definition in 3D Pipeline.

9.2.5 3D Clipping

If an object needs to be clipped, it will be passed to the CLIP thread. The CLIP thread will perform some (arbitrary) algorithm to clip the primitive, and subsequently output “new” vertices as a primitive defining the visible region of the input object (assuming there is a visible region). In the process of spawning the CLIP thread, the input vertices may be considered “consumed” and therefore dereferenced. Therefore the CLIP thread will need to copy (if required) any input VUE data to a new output VUE – there is no mechanism to “output” input vertices other than copying.

supports only Fixed function Clipping

9.3 CLIP Stage Input

As a stage of the 3D pipeline, the CLIP stage receives inputs from the previous (GS) stage. Refer to *3D Overview* for an overview of the various types of input to a 3D Pipeline stage. The remainder of this subsection describes the inputs specific to the CLIP stage.



9.3.1 State

9.3.1.1 3DSTATE CLIP

The state used by the Clip Stage is defined by this inline state packet.

3DSTATE_CLIP					
Source:		RenderCS			
Length Bias:		2			
DWord	Bit	Description			
0	31:29	Command Type			
		Default Value:	3h GFXPIPE		
		Format:	OpCode		
	28:27	Command SubType			
		Default Value:	3h GFXPIPE_3D		
		Format:	OpCode		
	26:24	3D Command Opcode			
		Default Value:	0h 3DSTATE_PIPELINE		
		Format:	OpCode		
	23:16	3D Command Sub Opcode			
	Default Value:	12h 3DSTATE_CLIP			
	Format:	OpCode			
15:8	Reserved				
	Project:	All			
	Format:	MBZ			
7:0	DWord Length				
	Default Value:	02h Excludes DWord (0,1)			
	Project:	All			
	Format:	=n Total Length - 2			
1	31:21	Reserved			
		Project:	All		
		Format:	MBZ		
	20	Front Winding			
		Project:	All		
	Determines whether a triangle object is considered "front facing" if the screen space vertex positions, when traversed in the order, result in a clockwise (CW) or counter-clockwise (CCW) winding order. Does not apply to points or lines.				
		Value	Name	Description	Project
		0h		FRONTWINDING_CW	All
		1h		FRONTWINDING_CCW	All
	19	Vertex Sub Pixel Precision Select			
		Project:	All		
		Format:	U1		
Selects the number of fractional bits maintained in the vertex data					
		Value	Name	Description	Project
	0h		8 sub pixel precision bits maintained	All	



3DSTATE_CLIP

	1h		4 sub pixel precision bits maintained	All
18	EarlyCull Enable			
	Project:		All	
	Format:		Enable	
	This field is used to enable/disable the EarlyCull function.			
	Programming Notes			Project
Workaround : Due to Hardware issue "EarlyCull" needs to be enabled only for the cases where the incoming primitive topology into the clipper guaranteed to be Trilist.				
17:16	Cull Mode			
	Project:		All	
	Format:		3D_CullMode	
	Controls removal (culling) of triangle objects based on orientation. The cull mode only applies to triangle objects and does not apply to lines, points or rectangles.			
	Value	Name	Description	Project
	0h	CULLMODE_BOTH	All triangles are discarded (i.e., no triangle objects are drawn)	All
	1h	CULLMODE_NONE	No triangles are discarded due to orientation	All
	2h	CULLMODE_FRONT	Triangles with a front-facing orientation are discarded	All
	3h	CULLMODE_BACK	Triangles with a back-facing orientation are discarded	All
	Programming Notes			
Orientation determination is based on the setting of the Front Winding state.				
15:11	Reserved			
	Project:		All	
	Format:		MBZ	
10	Clipper Statistics Enable			
	Project:		All	
	Format:		Enable	
	This bit controls whether Clip-unit-specific statistics register(s) can be incremented.			
	Value	Name	Description	Project
0h	Disable	CL_INVOCATIONS_COUNT cannot increment	All	
1h	Enable	CL_INVOCATIONS_COUNT can increment	All	
9:8	Reserved			
	Project:		All	
	Format:		MBZ	
7:0	User Clip Distance Cull Test Enable Bitmask			
	Project:		All	
	Format:		Enable[8]	
	This 8 bit mask field selects which of the 8 user clip distances against which trivial reject / trivial accept determination needs to be made (does not cause a must clip).DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances.			
2	31	CLIP Enable		
		Project:		All
		Format:		Enable
Specifies whether the CLIP function is enabled or disabled (pass-through).				
30	API Mode			



3DSTATE_CLIP

3DSTATE_CLIP			
	Project: All		
	Controls the definition of the NEAR clipping plane		
	Value	Name	Description
	0h	APIMODE_OGL	NEAR VP boundary == 0.0 (NDC)
			Project All
29	Reserved		
	Project: All		
	Format: MBZ		
28	Viewport XY ClipTest Enable		
	Project: All		
	Format: Enable		
	This field is used to control whether the Viewport X,Y extents are considered in VertexClipTest. See Tristrip Clipping Errata subsection.		
27	Viewport Z ClipTest Enable		
	Project: All		
	Format: Enable		
	This field is used to control whether the Viewport Z extents (near, far) are considered in VertexClipTest.		
26	Guardband ClipTest Enable		
	Project: All		
	Format: Enable		
	This field is used to control whether the Guardband X,Y extents are considered in VertexClipTest for non-point objects. If the Guardband ClipTest is DISABLED but the Viewport XY ClipTest is ENABLED, ClipDetermination operates as if the Guardband were coincident with the Viewport. If both the Guardband and Viewport XY ClipTest are DISABLED, all vertices are considered "visible" with respect to the XY directions.		
25:24	Reserved		
	Project: All		
	Format: MBZ		
23:16	User Clip Distance Clip Test Enable Bitmask		
	Project: All		
	Format: Enable[8]		
	This 8 bit mask field selects which of the 8 user clip distances against which trivial reject / trivial accept / must clip determination needs to be made. DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances.		
15:13	Clip Mode		
	Project: All		
	This field specifies a general mode of the CLIP unit, when the CLIP unit is ENABLED.		
	Value	Name	Description
	0h	CLIPMODE_NORMAL	TrivialAccept objects are passed down the pipeline, MustClip objects Clipped in the Fixed Function Clipper HW, TrivialReject and BAD objects are discarded
	1h	Reserved	
	2h	Reserved	
	3h	CLIPMODE_REJECT_ALL	All objects are discarded
	4h	CLIPMODE_ACCEPT_ALL	All objects (except BAD objects) are trivially accepted. This effectively disables the clip-test/clip-determination function.
			Project All



3DSTATE_CLIP

			Note that the CLIP unit will still filter out adjacency information, which may be required since the SF unit does not accept primitives with adjacency.	
	5h-7h	Reserved		All
12:10	Reserved			
	Project:			All
	Format:			MBZ
9	Perspective Divide Disable			
	Project:			All
	Format:			Disable
	<p>This field disables the Perspective Divide function performed on homogeneous position read from the URB. This feature can be used by software to submit pre-transformed “screen-space” geometry for rasterization. This likely requires the W component of positions to contain “rhw” (aka 1/w) in order to support perspective-correct interpolation of vertex attributes. Likewise, the X,Y,Z components will likely be required to be X/W, Y/W, Z/W. Note that the device does not support clipping when perspective divide is disabled. Software must specify CLIPMODE_ACCEPT_ALL whenever it disables perspective divide. This implies that software must ensure that object positions are completely contained within the “guardband” screen-space limits imposed by the SF unit (e.g., by clipping in CPU SW before submitting the objects).</p>			
8	Non-Perspective Barycentric Enable			
	Project:			All
	Format:			Enable
	<p>This field enables computation of non-perspective barycentric parameters in the clipper, which are sent to SF unit in the must clip case. This field must be enabled if any non-perspective barycentric parameters are enabled in the Windower.</p>			
7:6	Reserved			
	Project:			All
	Format:			MBZ
5:4	Triangle Strip/List Provoking Vertex Select			
	Project:			All
	Format:			U2 enumerated type
	<p>This field selects which vertex of a triangle (in a triangle strip or list primitive) is considered the “provoking vertex”.</p>			
	Value	Name	Project	
	0h	Vertex 0	All	
	1h	Vertex 1	All	
	2h	Vertex 2	All	
	3h	Reserved	All	
3:2	Line Strip/List Provoking Vertex Select			
	Project:			All
	Format:			U2 enumerated type
	<p>This field selects which vertex of a line (in a line strip or list primitive) is considered the “provoking vertex”.</p>			



3DSTATE_CLIP			
	Value	Name	Project
	0h	Vertex 0	All
	1h	Vertex 1	All
	2h	Reserved	All
	3h	Reserved	All
1:0	Triangle Fan Provoking Vertex Select		
	Project:	All	
	Format:	U32 enumerated type	
	This field selects which vertex of a triangle (in a triangle fan primitive) is considered the “provoking vertex”.		
	Value	Name	Project
	0h	Vertex 0	All
	1h	Vertex 1	All
	2h	Vertex 2	All
	3h	Reserved	All
3	31:28	Reserved	
	Project:	All	
	Format:	MBZ	
	27:17	Minimum Point Width	
	Project:	All	
	Format:	U8.3 pixels	
	This value is used to clamp read-back PointWidth values.		
	16:6	Maximum Point Width	
	Project:	All	
	Format:	U8.3 pixels	
	This value is used to clamp read-back PointWidth values.		
	5	Force Zero RTAIndex Enable	
	Project:	All	
	Format:	Enable	
	If set, the Clip unit will ignore the read-back RTAIndex and operate as if the value 0 was read-back. If clear, the read-back value is used.		
	4	Reserved	
	Project:	All	
	Format:	MBZ	
	3:0	Maximum VPIndex	
	Project:	All	
	Format:	U4-1 index value (# of viewports)	
	This field specifies the maximum valid VPIndex value, corresponding to the number of active viewports.		



3DSTATE_CLIP

If the source of the VPIndex exceeds this maximum value, a VPIndex value of 0 is passed down the pipeline. Note that this clamping does not affect a VPIndex value stored in the URB.

9.4 Object Staging

The CLIP unit's Object Staging Buffer (OSB) accepts streams of input vertex information packets, along with each vertex's VertexClipTest result (outCode). This information is buffered until a complete object or the last vertex of the primitive topology is received. The OSB then performs the ClipDetermination function on the object vertices, and takes the actions required by the results of that function.

9.4.1 Partial Object Removal

The OSB is responsible for removing incomplete LINESSTRIP and TRISTRIP objects that it may receive from the preceding stage (GS). Partial object removal is not supported for other primitive types due to either (a) the GS is not permitted to output those primitive types (e.g., primitives with adjacency info), and the VF unit will have removed the partial objects as part of 3DPRIMITIVE processing, or (b) although the GS thread is allowed to output the primitive type (e.g., LINELIST), it is assumed that the GS kernel will be correctly implemented to avoid outputting partial objects (or pipeline behavior is UNDEFINED).

An object is considered 'partial' if the last vertex of the primitive topology is encountered (i.e., PrimEnd is set) before a complete set of vertices for that object have been received. Given that only LINESSTRIP and TRISTRIP primitive types are subject to CLIP unit partial object removal, the only supported cases of partial objects are 1-vertex LINESSTRIPS and 1 or 2-vertex TRISTRIPS.

9.4.2 ClipDetermination Function

In ClipDetermination, the vertex outcodes of the primitive are combined in order to determine the clip status of the object (TR: trivially reject; TA: trivial accept; MC: must clip; BAD: invalid coordinate). Only those vertices included in the object are examined (3 vertices for a triangle, 2 for a line, and 1 for a point). The outcode bit arrays for the vertices are separately ANDed (intersection) and ORed (union) together (across vertices, not within the array) to yield objANDCode and objORCode bit arrays.

TR/TA against interesting boundary subsets are then computed. The TR status is computed as the logical OR of the appropriate objANDCode bits, as the vertices need only be outside of one common boundary to be trivially rejected. The TA status is computed as the logical NOR of the appropriate objORCode bits, as any vertex being outside of any of the boundaries prevents the object from being trivially accepted.

If any vertex contains a BAD coordinate, the object is considered BAD and any computed TR/TA results will effectively be ignored in the final action determination.

Next, the boundary subset TR/TA results are combined to determine an overall status of the object. If the object is TR against any viewport or enabled UC plane, the object is considered TR. Note that, by definition, being TR against a VPXY boundary implies that the vertices will be TR against the corresponding GB boundary, so computing TR_GB is unnecessary.

The treatment of the UCF outcodes is conditional on the UserClipFlags MustClip Enable state. If DISABLED, an object that is not TR against the UCFs is considered TA against them. Put another way, objects will only be culled (not clipped) with respect to the UCFs. If ENABLED, the UCF outcodes are



treated like the other outcodes, in that they are used to determine TR, TA or MC status, and an object can be passed to a CLIP thread simply based on it straddling a UCF.

Finally, the object is considered MC if it is neither TR or TA.

The following logic is used to compute the final TR, TA, and MC status.

```
//
// ClipDetermination
//
//
// Compute objANDCode and objORCode
//
switch (object type) {
case POINT:
{
objANDCode[...] = v0.outCode[...]
objORCode[...] = v0.outCode[...]
} break
case LINE:
{
objANDCode[...] = v0.outCode[...] & v1.outCode[...]
objORCode[...] = v0.outCode[...] | v1.outCode[...]
} break
case TRIANGLE:
{
objANDCode[...] = v0.outCode[...] & v1.outCode[...] & v2.outCode[...]
objORCode[...] = v0.outCode[...] | v1.outCode[...] | v2.outCode[...]
} break
//
// Determine TR/TA against interesting boundary subsets
//
TR_VPXY = (objANDCode[VP_L] | objANDCode[VP_R] | objANDCode[VP_T] |
objANDCode[VP_B])
TR_GB = (objANDCode[GB_L] | objANDCode[GB_R] | objANDCode[GB_T] |
objANDCode[GB_B])
TA_GB = !(objORCode[GB_L] | objORCode[GB_R] | objORCode[GB_T] |
objORCode[GB_B])
TA_VPZ = !(objORCode[VP_N] | objORCode[VP_Z])
```




```
TR_VPZ = (objANDCode[VP_N] | objANDCode[VP_Z])
TA_UC = !(objORCode[UC0] | objORCode[UC1] | ... | objORCode[UC7])
TR_UC = (objANDCode[UC0] | objANDCode[UC1] | ... | objANDCode[UC7])
BAD = objORCode[BAD]
TA_NEGW = !objORCode[NEGW]
TR_NEGW = objANDCode[NEGW]
//
// Trivial Reject
//
// An object is considered TR if all vertices are TR against any common
boundary
// Note that this allows the case of the VPXY being outside the GB
//
TR = TR_GB || TR_VPXY || TR_VPZ || TR_UC || TR_NEGW
//
// Trivial Accept
//
// For an object to be TA, it must be TA against the VPZ and GB, not TR,
// and considered TA against the UC planes and NEGW
// If the UCMC mode is disabled, an object is considered TA against the UC
// as long as it isn't TR against the UC.
// If the UCMC mode is enabled, then the object really has to be TA against
the UC
// to be considered TA
// In this way, enabling the UCMC mode will force clipping if the object is
neither
// TA or TR against the UC
//
TA = !TR && TA_GB && TA_VPZ && TA_NEGW
UCMC = CLIP_STATE.UserClipFlagsMustClipEnable
TA = TA && ( (UCMC && TA_UC) || (!UCMC && !TR_UC) )
//
// MustClip
// This is simply defined as not TA or TR
// Note that exactly one of TA, TR and MC will be set
//
```



```
MC = !(TA || TR)
```

9.4.3 ClipMode

The ClipMode state determines what action the CLIP unit takes given the results of ClipDetermination. The possible actions are:

- PASSTHRU: Pass the object directly down the pipeline. A CLIP thread is `not` spawned.
- DISCARD: Remove the object from the pipeline and dereference object vertices as required (i.e., dereferencing will not occur if the vertices are shared with other objects).
- SPAWN: Pass the object to a CLIP thread. In the process of initiating the thread, the object vertices may be dereferenced.

The following logic is used to determine what to do with the object (PASSTHRU or DISCARD or SPAWN).

```
//  
// Use the ClipMode to determine the action to take  
//  
switch (CLIP_STATE.ClipMode) {  
case NORMAL: {  
    PASSTHRU = TA && !BAD  
    DISCARD = TR || BAD  
    SPAWN    = MC && !BAD  
}  
  
case CLIP_ALL: {  
    PASSTHRU = 0  
    DISCARD = 0  
    SPAWN    = 1  
}  
  
case CLIP_NOT_REJECT: {  
PASSTHRU = 0  
    DISCARD = TR || BAD  
    SPAWN    = !(TR || BAD)  
}  
  
case REJECT_ALL: {  
    PASSTHRU = 0  
    DISCARD = 1  
    SPAWN    = 0  
}  
  
case ACCEPT_ALL: {  
    PASSTHRU = !BAD
```



```
DISCARD = BAD
SPAWN   = 0

}
} endswitch
```

9.4.3.1 NORMAL ClipMode

In NORMAL mode, objects will be discarded if TR or BAD, passed through if TA, and passed to a CLIP thread if MC. This mode is typically used when the CLIP kernel is only used to perform 3D Clipping (the expected usage model).

9.4.3.2 CLIP_ALL ClipMode

In CLIP_ALL mode, all objects (regardless of classification) will be passed to CLIP threads. Note that this includes BAD objects. This mode can be used to perform arbitrary processing in the CLIP thread, or as a backup if for some reason the CLIP unit fixed functions (VertexClipTest, ClipDetermination) are not sufficient for controlling 3D Clipping.

9.4.3.3 CLIP_NON_REJECT ClipMode

This mode is similar to CLIP_ALL mode, but TR and BAD objects are discarded and all other (TA, MC) objects are passed to CLIP threads. Usage of this mode assumes that the CLIP unit fixed functions (VertexClipTest, ClipDetermination) are sufficient at least in respect to determining trivial reject.

9.4.3.4 REJECT_ALL ClipMode

In REJECT_ALL mode, all objects (regardless of classification) are discarded. This mode effectively clips out all objects.

9.4.3.5 ACCEPT_ALL ClipMode

In ACCEPT_ALL mode, all non-BAD objects are passed directly down the pipeline. This mode partially disables the CLIP stage. BAD objects will still be discarded, and incomplete primitives (generated by a GS thread) will be discarded.

Primitive topologies with adjacency are also handled, in that the adjacent-only vertices are dereferenced and only non-adjacent objects are passed down the pipeline. This condition can arise when primitive topologies with adjacency are generated but the GS stage is disabled. If this condition is allowed, the CLIP stage must not be completely disabled – as this would allow adjacent vertices to pass through the CLIP stage and lead to UNPREDICATBLE results as the rest of the pipeline does not comprehend adjacency.

9.5 Object Pass-Through

Depending on ClipMode, objects may be passed directly down the pipeline. The PrimTopologyType associated with the output objects may differ from the input PrimTopologyType, as shown in the table below.

Programming Note: The CLIP unit does `not` tolerate primitives with adjacency that have “dangling vertices”. This should not be an issue under normal conditions, as the VF unit will not generate these



sorts of primitives and the GS thread is restricted (though by specification only) to not output these sorts of primitives.

Input PrimTopologyType	Pass-Through Output PrimTopologyType	Notes
POINTLIST	POINTLIST	
POINTLIST_BF	POINTLIST_BF	
LINELIST	LINELIST	
LINELIST_ADJ	LINELIST	Adjacent vertices removed.
LINESTRIP	LINESTRIP	
LINESTRIP_ADJ	LINESTRIP	Adjacent vertices removed.
LINESTRIP_BF	LINESTRIP_BF	
LINESTRIP_CONT	LINESTRIP_CONT	
LINESTRIP_CONT_BF	LINESTRIP_CONT_BF	
LINELOOP	N/A	Not supported after GS.
TRILIST	TRILIST	
RECTLIST	RECTLIST	
TRILIST_ADJ	TRILIST	Adjacent vertices removed.
TRISTRIP	TRISTRIP or TRISTRIP_REV	Depends on where the incoming strip is broken (if at all) by discarded or clipped objects See Tristrip Clipping Errata subsection.
TRISTRIP_REV	TRISTRIP or TRISTRIP_REV	Depends on where the incoming strip is broken (if at all) by discarded or clipped objects See Tristrip Clipping Errata subsection.
TRISTRIP_ADJ	TRISTRIP or TRISTRIP_REV	Depends on where the incoming strip is broken (if at all) by discarded or clipped objects Adjacent vertices removed. See Tristrip Clipping Errata subsection.
TRIFAN	TRIFAN	
TRIFAN_NOSTIPPLE	TRIFAN_NOSTIPPLE	
POLYGON	POLYGON	
QUADLIST	N/A	Not supported after GS.
QUADSTRIP	N/A	Not supported after GS.

9.6 Primitive Output

(This section refers to output from the CLIP unit to the pipeline, not output from the CLIP thread)

The CLIP unit will output primitives (either passed-through or generated by a CLIP thread) in the proper order. This includes the buffering of a concurrent CLIP thread's output until the preceding CLIP thread terminates. Note that the requirement to buffer subsequent CLIP thread output until the preceding CLIP



thread terminates has ramifications on determining the number of VUEs allocated to the CLIP unit and the number of concurrent CLIP threads allowed.

9.7 Other Functionality

9.7.1 Statistics Gathering

The CLIP unit includes logic to assist in the gathering of certain pipeline statistics, primarily in support of the Asynchronous Query function of the APIs. The statistics take the form of MI counter registers (see Memory Interface Registers), where the CLIP unit provides signals causing those counters to increment.

Software is responsible for controlling (enabling) these counters in order to provide the required statistics at the DDI level. For example, software might need to disable the statistics gathering before submitting non-API-visible objects (e.g., RECTLISTs) for processing.

The CLIP unit must be ENABLED (via the CLIP Enable bit of PIPELINED_STATE_POINTERS) in order to it to affect the statistics counters. This might lead to a pathological case where the CLIP unit needs to be ENABLED simply to provide statistics gathering. If no clipping functionality is desired, Clip Mode can be set to ACCEPT_ALL to effectively inhibit clipping while leaving the CLIP stage ENABLED.

The two statistics the CLIP unit affects (if enabled) are:

- CL_INVOCATION_COUNT:
 - Incremented for every object received from the GS stage.

9.7.1.1 CL_INVOCATION_COUNT

If the **Statistics Enable** bit (CLIP_STATE) is set, the CLIP unit increments the CL_INVOCATION_COUNT register for every `complete` object received from the GS stage.

In order to maintain a count of application-generated objects, software will need to clear the CLIP unit's **Statistic Enable** whenever driver-generated objects are rendered.



10. 3D Pipeline - Strips and Fans (SF) Stage

10.1 Overview

The Strips and Fan (SF) stage of the 3D pipeline is responsible for performing “setup” operations required to rasterize 3D objects.

10.1.1 Inputs from CLIP

The following table describes the per-vertex inputs passed to the SF unit from the previous (CLIP) stage of the pipeline.

SF’s Vertex Pipeline Inputs

Variable	Type	Description
primType	enum	Type of primitive topology the vertex belongs to. <i>Primitive Assembly</i> for a list of primitive types supported by the SF unit. See <i>3D Pipeline</i> for descriptions of these topologies. Notes: The CLIP unit will convert any primitive with adjacency (3DPRIMxxx_ADJ) it receives from the pipeline into the corresponding primitive without adjacency (3DPRIMxxx). QUADLIST, QUADSTRIP, LINELOOP primitives are not supported by the SF unit. Software must use a GS thread to convert these to some other (supported) primitive type. If an object is clipped by the hardware clipper, the CLunit would force this field to “3DPRIM_POLYGON”. SFunit would process this incoming object just as it would any other “3DPRIM_POLYGON”. SFunit selects vertex 0 as the provoking vertex.
primStart,primEnd	boolean	Indicate vertex’s position within the primitive topology
vInX[]	float	Vertex X position (screen space or NDC space)
vInY[]	float	Vertex Y position (screen space or NDC space)
vInZ[]	float	Vertex Z position (screen space or NDC space)
vInInvW[]	float	Reciprocal of Vertex homogeneous (clip space) W
hVUE[]	URB address	Points to the vertex’s data stored in the URB (one VUE handle per vertex)
renderTargetArrayIndex	uint	Index of the render target (array element or 3D slice), clamped to 0 by the GS unit if the max value was exceeded. If this vertex is the leading vertex of an object within the primitive topology, this value will be associated with that object in subsequent processing.
viewPortIndex	uint	Index of a viewport transform matrix within the SF_VIEWPORT structure used to perform Viewport Transformation on object vertices and scissor operations on an object. If this vertex is the leading vertex of an object within the primitive topology, this



Variable	Type	Description
		value will be associated with that object in the Viewport Transform and Scissor subfunctions, otherwise the value is ignored. Note that for primitive topologies with vertices shared between objects, this means a shared vertex may be subject to multiple Viewport Transformation operations if the viewPortIndex varies within the topology.
pointSize	uint	If this vertex is within a POINTLIST[_BF] primitive topology, this value specifies the screen space size (width,height) of the square point to be rasterized about the vertex position. Otherwise the value is ignored.

10.1.2 Attribute Setup/Interpolation Process

The following sections describe the Attribute Setup/Interpolation Process.

10.1.2.1 Attribute Setup/Interpolation Process

Hardware computes all needed parameters, as there is no setup thread.

10.1.3 Outputs to WM

The outputs from the SF stage to the WM stage are mostly comprised of implementation-specific information required for the rasterization of objects. The types of information is summarized below, but as the interface is not exposed to software a detailed discussion is not relevant to this specification.

- PrimType of the object
- VPIndex, RTAIndex associated with the object
- Coefficients for Z, 1/W, perspective and non-perspective b1 and b2 per vertex, and attribute vertex deltas a0, a1, and a2 per attribute.
- Information regarding the X,Y extent of the object (e.g., bounding box, etc.).
- Edge or line interpolation information (e.g., edge equation coefficients, etc.).
- Information on where the WM is to start rasterization of the object.
- Object orientation (front/back-facing).
- Last Pixel indication (for line drawing).

10.2 Primitive Assembly

The first subfunction within the SF unit is *Primitive Assembly*. Here 3D primitive vertex information is buffered and, when a sufficient number of vertices are received, converted into basic 3D objects which are then passed to the Viewport Transformation subfunction.

The number of vertices passed with each primitive is constrained by the primitive type. *Primitive Assembly*. Passing any other number of vertices results in UNDEFINED behavior. Note that this restriction only applies to primitive output by GS threads (which is under control of the GS kernel). See the Vertex Fetch chapter for details on how the VF unit automatically removes incomplete objects resulting from processing a 3DPRIMITIVE command.

SF-Supported Primitive Types & Vertex Count Restrictions

primType	VertexCount Restriction
3DPRIM_TRILIST	nonzero multiple of 3



primType	VertexCount Restriction
3DPRIM_TRISTRIP 3DPRIM_TRISTRIP_REVERSE	>=3
3DPRIM_TRIFAN 3DPRIM_TRIFAN_NOSTIPPLE 3DPRIM_POLYGON	>=3
3DPRIM_LINELIST	nonzero multiple of 2
3DPRIM_LINESTRIP 3DPRIM_LINESTRIP_CONT 3DPRIM_LINESTRIP_BF 3DPRIM_LINESTRIP_CONT_BF	>=2
3DPRIM_RECTLIST	nonzero multiple of 3
3DPRIM_POINTLIST 3DPRIM_POINTLIST_BF	nonzero

Primitive Assembly for a list of the 3D object types.

3D Object Types

objectType	generated by primType	Vertices/Object
3DOBJ_POINT	3DPRIM_POINTLIST 3DPRIM_POINTLIST_BF	1
3DOBJ_LINE	3DPRIM_LINELIST 3DPRIM_LINESTRIP 3DPRIM_LINESTRIP_CONT 3DPRIM_LINESTRIP_BF 3DPRIM_LINESTRIP_CONT_BF	2
3DOBJ_TRIANGLE	3DPRIM_TRILIST 3DPRIM_TRISTRIP 3DPRIM_TRISTRIP_REVERSE 3DPRIM_TRIFAN 3DPRIM_TRIFAN_NOSTIPPLE 3DPRIM_POLYGON	3
3DOBJ_RECTANGLE	3DPRIM_RECTLIST	3 (expanded to 4 in RectangleCompletion)

Primitive Assembly for the outputs of Primitive Decomposition.

Primitive Decomposition Outputs

Variable	Type	Description
-----------------	-------------	--------------------



Variable	Type	Description
objectType	enum	Type of object. <i>Primitive Assembly</i>
nV	uint	The number of object vertices passed to Object Setup. <i>Primitive Assembly</i>
v[0..nV-1]*	various	Data arrays associated with <code>object</code> vertices. Data in the array consists of X, Y, Z, <code>invW</code> and a pointer to the other vertex attributes. These additional attributes are not used by directly by the 3D fixed functions but are made available to the SF thread. The number of valid vertices depends on the object type. <i>Primitive Assembly</i>
invertOrientation	enum	Indicates whether the orientation (CW or CCW winding order) of the vertices of a triangle object should be inverted. Ignored for non-triangle objects.
backFacing	enum	Valid only for points and line objects, indicates a back facing object. This is used later for culling.
provokingVtx	uint	Specifies the index (into the <code>v[]</code> arrays) of the vertex considered the “provoking” vertex (for flat shading). The selection of the provoking vertex is programmable via <code>SF_STATE (xxx Provoking Vertex Select</code> state variables.)
polyStippleEnable	boolean	TRUE if Polygon Stippling is enabled. FALSE for <code>TRIFAN_NOSTIPPLE</code> . Ignored for non-triangle objects.
continueStipple	boolean	Only applies to line objects. TRUE if Line Stippling should be continued (i.e., not reset) from where the previous line left off. If FALSE, Line Stippling is reset for each line object.
renderTargetIndex	uint	Index of the render target (array element or 3D slice), clamped to 0 by the GS unit if the max value was exceeded. This value is simply passed in SF thread payloads and not used within the SF unit.
viewportIndex	uint	Index of a viewport transform matrix within the <code>SF_VIEWPORT</code> structure used to perform Viewport Transformation on object vertices and scissor operations on an object.
pointSize	unit	For point objects, this value specifies the screen space size (width,height) of the square point to be rasterized about the vertex position. Otherwise the value is ignored.

The following table defines, for each primitive topology type, which vertex’s `VPIndex/RTAIndex` applies to the objects within the topology.

VPIndex/RTAIndex Selection

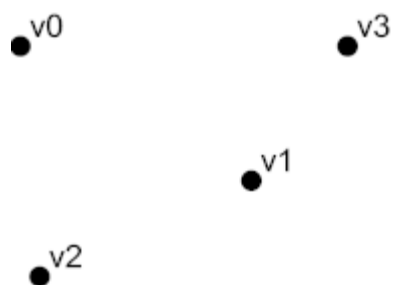
PrimTopologyType	Viewport Index Usage
POINTLIST POINTLIST_BF	Each vertex supplies the <code>VPIndex</code> for the corresponding point object
LINELIST	The leading vertex of each line supplies the <code>VPIndex</code> for the corresponding line object. <code>V0.VPIndex</code> □ Line(<code>V0,V1</code>) <code>V2.VPIndex</code> □ Line(<code>V2,V3</code>) ...
LINESTRIP LINESTRIP_BF LINESTRIP_CONT LINESTRIP_CONT_BF	The leading vertex of each line segment supplies the <code>VPIndex</code> for the corresponding line object. <code>V0.VPIndex</code> □ Line(<code>V0,V1</code>) <code>V1.VPIndex</code> □ Line(<code>V1,V2</code>) ... NOTE: If the <code>VPIndex</code> changes within the topology, shared vertices will be processed

	(mapped) multiple times.
TRILIST RECTLIST	The leading vertex of each triangle/rect supplies the VPIndex for the corresponding triangle/rect objects. V0.VPIndex \square Tri(V0,V1,V2) V3.VPIndex \square Tri(V3,V4,V5) ...
TRISTRIP TRISTRIP_REVERSE	The leading vertex of each triangle supplies the VPIndex for the corresponding triangle object. V0.VPIndex \square Tri(V0,V1,V2) V1.VPIndex \square Tri(V1,V2,V3) ... NOTE: If the VPIndex changes within the primitive, shared vertices will be processed (mapped) multiple times.
TRIFAN TRIFAN_NOSTIPPLE POLYGON	The first vertex (V0) supplies the VPIndex for all triangle objects.

10.2.1 Point List Decomposition

The 3DPRIM_POINTLIST and 3DPRIM_POINTLIST_BACKFACING primitives specify a list of independent points.

3DPRIM_POINTLIST Primitive



The decomposition process divides the list into a series of basic 3DOBJ_POINT objects that are then passed individually and in order to the Object Setup subfunction. The *provokingVertex* of each object is, by definition, v[0].

Points have no winding order, so the primitive command is used to explicitly state whether they are back-facing or front-facing points. Primitives of type 3DPRIM_POINTLIST_BACKFACING are decomposed exactly the same way as 3DPRIM_POINTLIST primitives, but the *backFacing* variable is set for resulting point objects being passed on to object setup.

```
PointListDecomposition() {
```

```
objectType = 3DOBJ_POINT
```

```

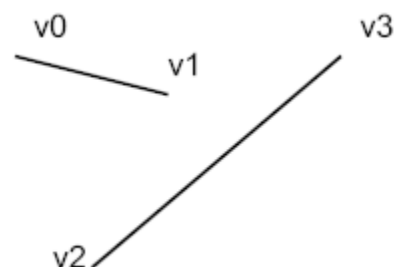
nV = 1
provokingVtx = 0
if (primType == 3DPRIM_POINTLIST)
    backFacing = FALSE
else // primType == 3DPRIM_POINTLIST_BACKFACING
    backFacing = TRUE
for each (vertex I in [0..vertexCount-1]) {
    v[0] [] vIn[i] // copy all arrays (e.g., v[X], v[Y], etc.)
    ObjectSetup()
}
}

```

10.2.2 Line List Decomposition

The 3DPRIM_LINELIST primitive specifies a list of independent lines.

3DPRIM_LINELIST Primitive



The decomposition process divides the list into a series of basic 3DOBJ_LINE objects that are then passed individually and in order to the Object Setup stage. The lines are generated with the following object vertex order: v0, v1; v2, v3; and so on. The *provokingVertex* of each object is taken from the **Line List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

```

LineListDecomposition() {
    objectType = 3DOBJ_LINE
    nV = 2
    provokingVtx = Line List/Strip Provoking Vertex Select
    continueStipple = FALSE
    for each (vertex I in [0..vertexCount-2] by 2) {
        v[0] arrays [] vIn[i] arrays
        v[1] arrays [] vIn[i+1] arrays
        ObjectSetup()
    }
}

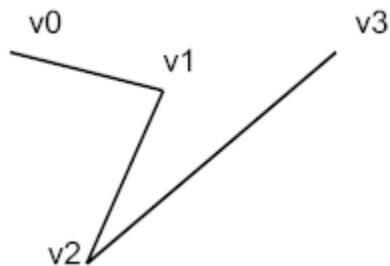
```

}

10.2.3 Line Strip Decomposition

The 3DPRIM_LINESTRIP, 3DPRIM_LINESTRIP_CONT, 3DPRIM_LINESTRIP_BF, and 3DPRIM_LINESTRIP_CONT_BF primitives specify a list of connected lines.

3DPRIM_LINESTRIP_xxx Primitive



The decomposition process divides the strip into a series of basic 3DOBJ_LINE objects that are then passed individually and in order to the Object Setup stage. The lines are generated with the following object vertex order: v0,v1; v1,v2; and so on. The *provokingVertex* of each object is taken from the **Line List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

Lines have no winding order, so the primitive command is used to explicitly state whether they are back-facing or front-facing lines. Primitives of type 3DPRIM_LINESTRIP[_CONT]_BF are decomposed exactly the same way as 3DPRIM_LINESTRIP[_CONT] primitives, but the *backFacing* variable is set for the resulting line objects being passed on to object setup. Likewise 3DPRIM_LINESTRIP_CONT[_BF] primitives are decomposed identically to basic line strips, but the *continueStipple* variable is set to true so that the line stipple pattern will pick up from where it left off with the last line primitive, rather than being reset.

```

LineStripDecomposition() {
objectType = 3DOBJ_LINE

  nV = 2

  provokingVtx = Line List/Strip Provoking Vertex Select

  if (primType == 3DPRIM_LINESTRIP) {
    backFacing = FALSE
    continueStipple = FALSE
  } else if (primType == 3DPRIM_LINESTRIP_BF) {
    backFacing = TRUE
    continueStipple = FALSE
  } else if (primType == 3DPRIM_LINESTRIP_CONT) {
    backFacing = FALSE
    continueStipple = TRUE
  } else if (primType == 3DPRIM_LINESTRIP_CONT_BF) {

```

```

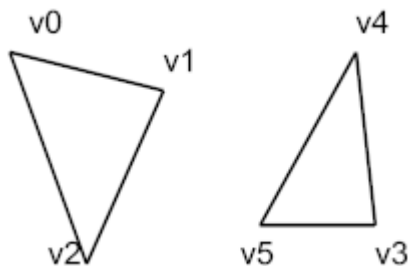
backFacing = TRUE
continueStipple = TRUE
}
for each (vertex l in [0..vertexCount-1]) {
v[0] arrays □□v[n[i]] arrays
v[1] arrays □□v[n[i+1]] arrays
ObjectSetup()
continueStipple = TRUE
}
}

```

10.2.4 Triangle List Decomposition

The 3DPRIM_TRILIST primitive specifies a list of independent triangles.

3DPRIM_TRILIST Primitive



The decomposition process divides the list into a series of basic 3DOBJ_TRIANGLE objects that are then passed individually and in order to the Object Setup stage. The triangles are generated with the following object vertex order: v0,v1,v2; v3,v4,v5; and so on. The *provokingVertex* of each object is taken from the **Triangle List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

```

TriangleListDecomposition() {
objectType = 3DOBJ_TRIANGLE
nV = 3
invertOrientation = FALSE
provokingVtx = Triangle List/Strip Provoking Vertex Select
polyStippleEnable = TRUE
for each (vertex l in [0..vertexCount-3] by 3) {
v[0] arrays □□v[n[i]] arrays
v[1] arrays □□v[n[i+1]] arrays
v[2] arrays □□v[n[i+2]] arrays
}
}

```

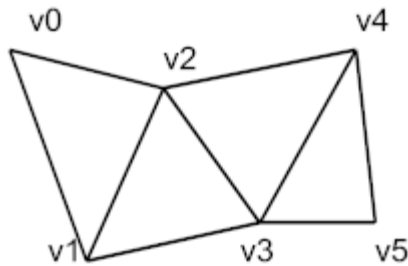
ObjectSetup()

```
}
}
```

10.2.5 Triangle Strip Decomposition

The 3DPRIM_TRISTRIP and 3DPRIM_TRISTRIP_REVERSE primitives specify a series of triangles arranged in a strip, as illustrated below.

3DPRIM_TRISTRIP[_REVERSE] Primitive



The decomposition process divides the strip into a series of basic 3DOBJ_TRIANGLE objects that are then passed individually and in order to the Object Setup stage. The triangles are generated with the following object vertex order: v0,v1,v2; v1,v2,v3; v2,v3,v4; and so on. Note that the *winding order* of the vertices alternates between CW (clockwise), CCW (counter-clockwise), CW, etc. The *provokingVertex* of each object is taken from the **Triangle List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

The 3D pipeline uses the winding order of the vertices to distinguish between front-facing and back-facing triangles (*Triangle Orientation Face Culling* below). Therefore, the 3D pipeline must account for the alternation of winding order in strip triangles. The *invertOrientation* variable is generated and used for this purpose.

To accommodate the situation where the driver is forced to break an input strip primitive into multiple trisrip primitive commands (e.g., due to ring or batch buffer size restrictions), two trisrip primitive types are supported. 3DPRIM_TRISTRIP is used for the initial section of a strip, and wherever a continuation of a strip starts with a triangle with a CW winding order. 3DPRIM_TRISTRIP_REVERSE is used for a continuation of a strip that starts with a triangle with a CCW winding order.

```
TriangleStripDecomposition() {
```

```
objectType = 3DOBJ_TRIANGLE
```

```
  nV = 3
```

```
provokingVtx = Triangle List/Strip Provoking Vertex Select
```

```
if (primType == 3DPRIM_TRISTRIP)
```

```
  invertOrientation = FALSE
```

```
else // primType == 3DPRIM_TRISTRIP_REVERSE
```

```
  invertOrientation = TRUE
```

```
polyStippleEnable = TRUE
```

```

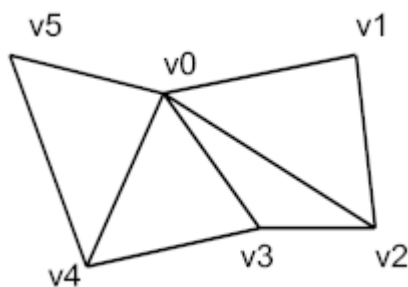
for each (vertex I in [0..vertexCount-3]) {
  v[0] arrays □□vIn[i] arrays
  v[1] arrays □□vIn[i+1] arrays
  v[2] arrays □□vIn[i+2] arrays
  ObjectSetup()
  invertOrientation = ! invertOrientation
}
}

```

10.2.6 Triangle Fan Decomposition

The 3DPRIM_TRIFAN and 3DPRIM_TRIFAN_NOSTIPPLE primitives specify a series of triangles arranged in a fan, as illustrated below.

3DPRIM_TRIFAN Primitive



The decomposition process divides the fan into a series of basic 3DOBJ_TRIANGLE objects that are then passed individually and in order to the Object Setup stage. The triangles are generated with the following object vertex order: v0,v1,v2; v0,v2,v3; v0,v3,v4; and so on. As there is no alternation in the vertex winding order, the *invertOrientation* variable is output as FALSE unconditionally. The *provokingVertex* of each object is taken from the **Triangle Fan Provoking Vertex** state variable, as programmed via SF_STATE.

Primitives of type 3DPRIM_TRIFAN_NOSTIPPLE are decomposed exactly the same way, except the *polyStippleEnable* variable is FALSE for the resulting objects being passed on to object setup. This will inhibit polygon stipple for these triangle objects.

```

TriangleFanDecomposition() {
  objectType = 3DOBJ_TRIANGLE

  nV = 3

  invertOrientation = FALSE

  provokingVtx = Triangle Fan Provoking Vertex Select

  if (primType == 3DPRIM_TRIFAN)
    polyStippleEnable = TRUE
  else // primType == 3DPRIM_TRIFAN_NOSTIPPLE

```

```

    polyStippleEnable = FALSE
v[0] arrays [] vIn[0] arrays // the 1st vertex is common
for each (vertex I in [1..vertexCount-2]) {
v[1] arrays [] vIn[i] arrays
v[2] arrays [] vIn[i+1] arrays
ObjectSetup()
}
}

```

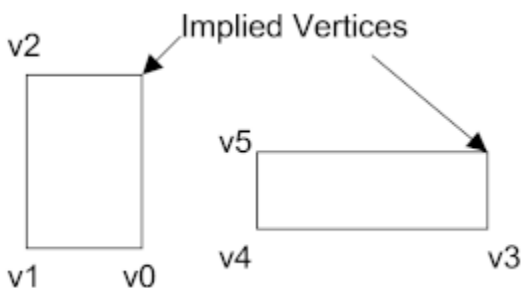
10.2.7 Polygon Decomposition

The 3DPRIM_POLYGON primitive is identical to the 3DPRIM_TRIFAN primitive with the exception that the *provokingVtx* is overridden with 0. This support has been added specifically for OpenGL support, avoiding the need for the driver to change the provoking vertex selection when switching between trifan and polygon primitives.

10.2.8 Rectangle List Decomposition

The 3DPRIM_RECTLIST primitive command specifies a list of independent, axis-aligned rectangles. Only the lower right, lower left, and upper left vertices (in that order) are included in the command – the upper right vertex is derived from the other vertices (in Object Setup).

3DPRIM_RECTLIST Primitive



The decomposition of the 3DPRIM_RECTLIST primitive is identical to the 3DPRIM_TRILIST decomposition, with the exception of the *objectType* variable.

```

RectangleListDecomposition() {
objectType = 3DOBJ_RECTANGLE

nV = 3

invertOrientation = FALSE
provokingVtx = 0
for each (vertex I in [0..vertexCount-3] by 3) {
v[0] arrays [] vIn[i] arrays

```




$v[1]$ arrays □□ $v[n+i+1]$ arrays

$v[2]$ arrays □□ $v[n+i+2]$ arrays

ObjectSetup()

```
}  
}
```

10.3 Object Setup

The Object Setup subfunction of the SF stage takes the post-viewport-transform data associated with each vertex of a basic object and computes various parameters required for scan conversion. This includes generation of implied vertices, translations and adjustments on vertex positions, and culling (removal) of certain classes of objects. The final object information is passed to the Windower/Masker (WM) stage where the object is rasterized into pixels.

10.3.1 Invalid Position Culling (Pre/Post-Transform)

At input to the SF stage, any objects containing a floating-point NaN value for Position X, Y, Z, or RHW will be unconditionally discarded. Note that this occurs on an object (not primitive) basis.

If Viewport Transformation is enabled, any objects containing a floating-point NaN value for post-transform Position X, Y or Z will be unconditionally discarded.

10.3.2 Viewport Transformation

If the **Viewport Transform Enable** bit of SF_STATE is ENABLED, a viewport transformation is applied to each vertex of the object.

The VPIndex associated with the leading vertex of the object is used to obtain the **Viewport Matrix Element** data from the corresponding element of the SF_VIEWPORT structure in memory. For each object vertex, the following scale and translate transformation is applied to the position coordinates:

$$x' = m00 * x + m30$$

$$y' = m11 * y + m31$$

$$z' = m22 * z + m32$$

Software is responsible for computing the matrix elements from the viewport information provided to it from the API.

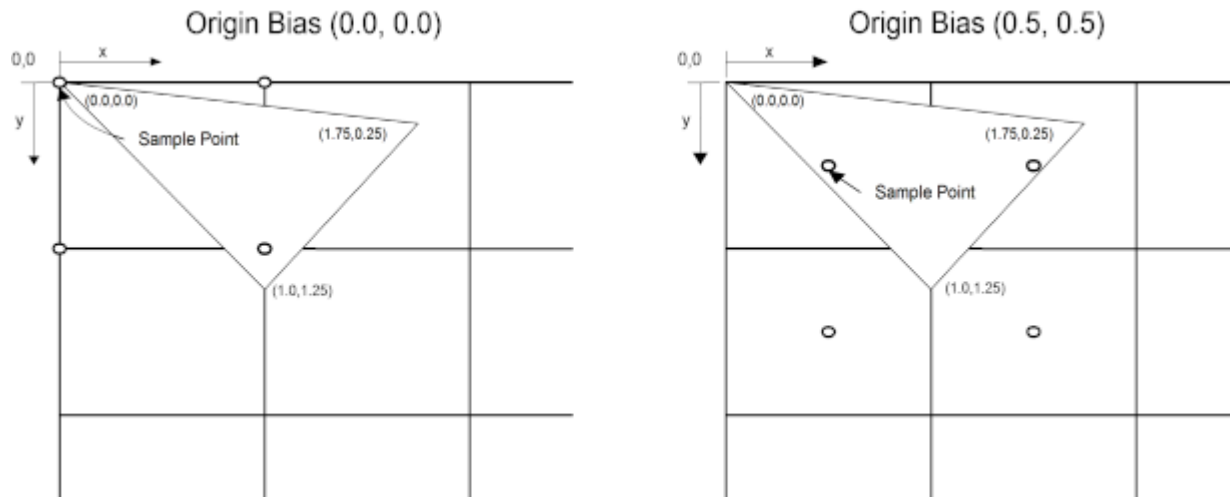
10.3.3 Destination Origin Bias

The positioning of the pixel sampling grid is programmable and is controlled by the **Destination Origin Horizontal/Vertical Bias** state variables (set via SF_STATE). If these bias values are both 0, pixels are sampled on an integer grid. Pixel (0,0) will be considered inside the object if the sample point at XY coordinate (0,0) falls within the primitive.

If the bias values are both 0.5, pixels are sampled on a “half” integer grid (i.e., X.5, Y.5). Pixel (0,0) will be considered inside the object if the sample point at XY coordinate (0.5,0.5) falls within the primitive. This positioning of the sample grid corresponds with the OpenGL rasterization rules, where “fragment centers” lay on a half-integer grid. It also corresponds with the Intel740 rasterizer (though that device did not employ “top left” rules).

Note that subsequent descriptions of rasterization rules for the various objects will be with reference to the pixel sampling grid.

Destination Origin Bias

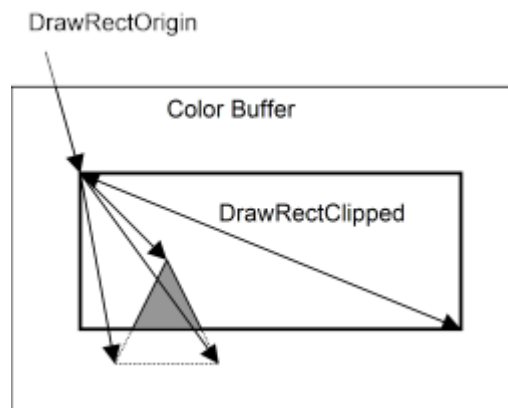


10.3.4 Point Rasterization Rule Adjustment

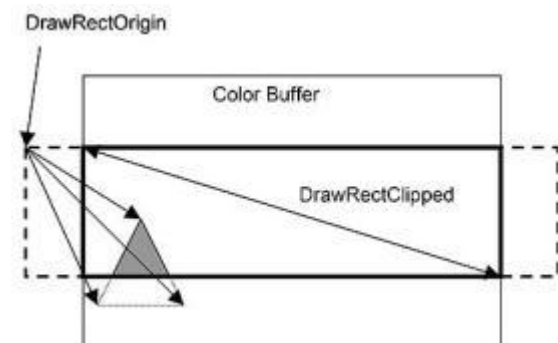
POINT objects are rasterized as square RECTANGLES, with one exception: The **Point Rasterization Rule** state variable (in SF_STATE) controls the rendering of point object edges that fall directly on pixel sample points, as the treatment of these edge pixels varies between APIs.

The following diagram shows the rasterization of a 2-pixel wide point centered at (2,2) given current DX rasterization rules (where changed the rasterization of points to match the rasterization of an identical (square) polygon). Here the pixel sample grid coincides with the integer pixel coordinates, and the **Point Rasterization Rule** is set to RASTRULE_UPPER_LEFT. Note that the pixels which lie only on the upper and/or left edges are lit.

Onscreen Draw Rectangle



Partially-offscreen Draw Rectangle



10.3.5.1 3DSTATE_DRAWING_RECTANGLE

3DSTATE_DRAWING_RECTANGLE		
Project:	All	
Source:	RenderCS	
Length Bias:	2	
The 3DSTATE_DRAWING_RECTANGLE command is used to set the 3D drawing rectangle and related state.		
DWord	Bit	Description
0	31:29	Command Type
		Default Value: 3h GFXPIPE
		Format: OpCode
	28:27	Command SubType
	Default Value: 3h GFXPIPE_3D	
	Format: OpCode	
26:24		3D Command Opcode
		Default Value: 1h 3DSTATE_NONPIPELINED
		Format: OpCode
23:16		3D Command Sub Opcode
		Default Value: 00h 3DSTATE_DRAWING_RECTANGLE
		Format: OpCode



3DSTATE_DRAWING_RECTANGLE				
	15:14	Reserved		
		Format: MBZ		
	13:8	Reserved		
		Format: MBZ		
	7:0	DWord Length		
		Default Value: 2h Excludes DWord (0,1)		
		Project: All		
		Format: =n Total Length - 2		
1	31:16	Clipped Drawing Rectangle Y Min		
		Project: All		
		Format: U16 in Pixels from Color Buffer origin (upper left corner)		
		Specifies Ymin value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with Y coordinates less than Ymin will be clipped out.		
		Value	Name	Project
		[0,16383]	Device ignores bits 31:30	
		Programming Notes		Project
		This value can be larger than Clipped Drawing Rectangle Y Max. If Ymin>Ymax, the clipped drawing rectangle is null, all polygons are discarded. If Ymin==Ymax, the clipped drawing rectangle is 1 pixel wide in the Y direction.		
		15:0	Clipped Drawing Rectangle X Min	
			Project: All	
		Format: U16 in Pixels from Color Buffer origin (upper left corner)		
	Specifies Xmin value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with X coordinates less than Xmin will be clipped out.			
	Value	Name	Project	
	[0,16383]	Device ignores bits 15:14		
	Programming Notes		Project	
	This value can be larger than Clipped Drawing Rectangle X Max. If Xmin>Xmax, the clipped drawing rectangle is null, all polygons are discarded. If Xmin==Xmax, the clipped drawing rectangle is 1 pixel wide in the X direction.			
2	31:16	Clipped Drawing Rectangle Y Max		
		Project: All		
		Format: U16 in Pixels from Color Buffer origin (upper left corner)		
		Specifies Ymax value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with coordinates greater than Ymax will be clipped out.		
	Value	Name	Project	
	[0,16383]	Device ignores bits 31:30		



3DSTATE_DRAWING_RECTANGLE		
		Programming Notes
		This value can be less than Clipped Drawing Rectangle Y Min. If Ymax<Ymin, the clipped drawing rectangle is null, all polygons are discarded. If Ymin==Ymax, the clipped drawing rectangle is 1 pixel wide in the Y direction.
	Project	
15:0	Clipped Drawing Rectangle X Max	
	Project:	All
	Format:	U16 in Pixels from Color Buffer origin (upper left corner)
	Specifies Xmax value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with coordinates greater than Xmax will be clipped out.	
	Value	Name
	[0,16383]	Device ignores bits 15:14
		Project
		Programming Notes
		This value can be less than Clipped Drawing Rectangle X Min. If Xmax<Xmin, the clipped drawing rectangle is null, all polygons are discarded. If Xmin==Xmax, the clipped drawing rectangle is 1 pixel wide in the X direction.
		Project
3	31:16	Drawing Rectangle Origin Y
	Project:	All
	Format:	S15 in Pixels from Color Buffer origin (upper left corner).
		Description
		Range: [-16384,16383] (Bit 31 should be a sign extension)
		Specifies Y origin of Drawing Rectangle (in whole pixels) relative to origin of the Color Buffer, used to map incoming (Draw Rectangle-relative) vertex positions to the Color Buffer space.
		Project
	15:0	Drawing Rectangle Origin X
	Project:	All
	Format:	S15 in Pixels from Color Buffer origin (upper left corner).
		Description
		Range: [-16384,16383] (Bit 15 should be a sign extension)
		Specifies X origin of Drawing Rectangle (in whole pixels) relative to origin of the Color Buffer, used to map incoming (Draw Rectangle-relative) vertex positions to the Color Buffer space.
		Project

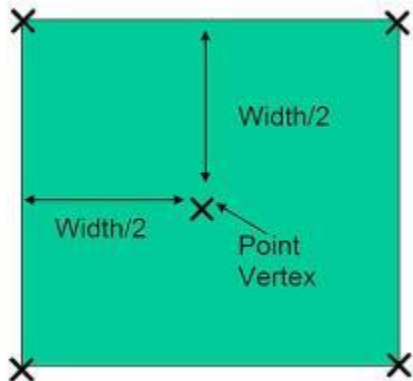
10.3.6 Point Width Application

This stage of the pipeline applies only to 3DOBJ_POINT objects. Here the point object is converted from a single vertex to four vertices located at the corners of a square centered at the point's X,Y position. The width and height of the square are specified by a *point width* parameter. The **Use Point Width State** value in SF_STATE determines the source of the point width parameter: the point width is either taken from the

Point Width value programmed in SF_STATE or the PointWidth specified with the vertex (as read back from the vertex VUE earlier in the pipeline).

The corner vertices are computed by adding and subtracting one half of the point width. *Point Width Application.*

Point Width Application

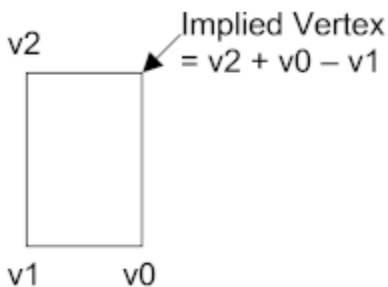


Z and W vertex attributes are copied from the single point center vertex to each of the four corner vertices.

10.3.7 Rectangle Completion

This stage of the pipeline applies only to 3DOBJ_RECTANGLE objects. Here the X,Y coordinates of the 4th (upper right) vertex of the rectangle object is computed from the first 3 vertices as shown in the following diagram. The other vertex attributes assigned to the implied vertex (v[3]) are UNDEFINED as they are not used. The Object Setup subfunction will use the values at only the first 3 vertices to compute attribute interpolants used across the entire rectangle.

Rectangle Completion



10.3.8 Vertex X,Y Clamping and Quantization

At this stage of the pipeline, vertex X and Y positions are in continuous screen (pixel) coordinates. These positions are quantized to subpixel precision by rounding the incoming values to the nearest subpixel (using round-to-nearest-or-even rules). The device supports rasterization with either 4 or 8 fractional (subpixel) position bits, as specified by the **Vertex SubPixel Precision Select** bit of SF_STATE.



The vertex X and Y screenspace coordinates are also **clamped** to the fixed-point “guardband” range supported by the rasterization hardware, as listed in the following table:

Per-Device Guardband Extents

Supported X,Y Screenspace “Guardband” Extent	Maximum Post-Clamp Delta (X or Y)
[-32K,32K-1]	N/A

Note that this clamping occurs after the Drawing Rectangle Origin has been applied and objects have been expanded (i.e., points have been expanded to squares, etc.). In almost all circumstances, if an object’s vertices are actually modified by this clamping (i.e., had X or Y coordinates outside of the guardband extent the rendered object will not match the intended result. Therefore software should take steps to ensure that this does not happen – e.g., by clipping objects such that they do not exceed these limits after the Drawing Rectangle is applied.

In addition, in order to be correctly rendered, objects must have a screenspace bounding box not exceeding 8K in the X or Y direction. This additional restriction must also be comprehended by software, i.e., enforced by use of clipping.

10.3.9 Degenerate Object Culling

At this stage of the pipeline, “degenerate” objects are discarded. This operation is automatic and cannot be disabled. (The object rasterization rules would by definition cause these objects to be “invisible” – this culling operation is mentioned here to reinforce that the device implementation optimizes these degeneracies as early as possible).

Degenerate Object Culling for definitions of degenerate objects.

Degenerate Objects

objType	Degenerate Object Definition
3DOBJ_POINT	Two or more corner vertices are coincident (i.e., the radius quantized to zero)
3DOBJ_LINE	The endpoints are coincident
3DOBJ_TRIANGLE	All three vertices are collinear or any two vertices are coincident and SOLID fill mode applies to the triangle
3DOBJ_RECTANGLE	Two or more corner vertices are coincident

10.3.10 Triangle Orientation (Face) Culling

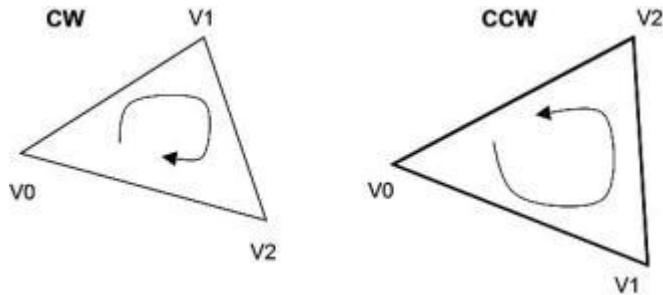
At this stage of the pipeline, 3DOBJ_TRIANGLE objects can be optionally discarded based on the “face orientation” of the object. This culling operation does not apply to the other object types.

This operation is typically called “back face culling”, though front facing objects (or all 3DOBJ_TRIANGLE objects) can be selected to be discarded as well. Face culling is typically used to eliminate triangles facing away from the viewer, thus reducing rendering time.

The “winding order” of a triangle is defined by the the triangle vertex’s 2D (X,Y) screen space position when traversed from v0 to v1 to v2. That traversal will proceed in either a clockwise (CW) or counter-clockwise (CCW) direction. The “winding order” of a triangle is defined by the the triangle vertex’s 2D (X,Y) screen space position when traversed from v0 to v1 to v2. That traversal will proceed in either a clockwise (CW) or counter-clockwise (CCW) direction. (A degenerate triangle is considered “backfacing”, regardless of the FrontWinding state.

(A degenerate triangle is considered “backfacing”, regardless of the FrontWinding state.

Triangle Winding Order



The **Front Winding** state variable in SF_STATE controls whether CW or CCW triangles are considered as having a “front-facing” orientation (at which point non-front-facing triangles are considered “back-facing”). The internal variable *invertOrientation* associated with the triangle object is then used to determine whether the orientation of a that triangle should be inverted. Recall that this variable is set in the Primitive Decomposition stage to account for the alternating orientations of triangles in strip primitives resulting from the ordering of the vertices used to process them.

The **Cull Mode** state variable in SF_STATE specifies how triangles are to be discarded according to their resultant orientation. *Degenerate Object Culling*.

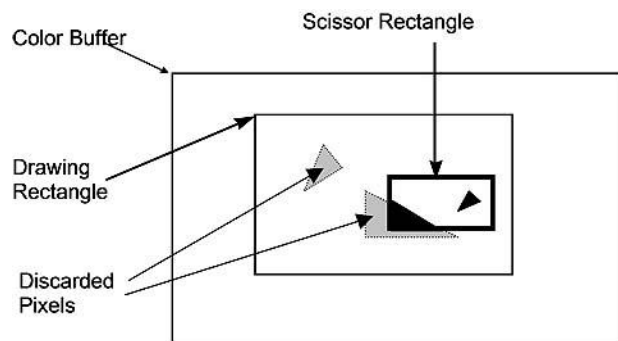
Cull Mode

CullMode	Definition
CULLMODE_NONE	The face culling operation is disabled
CULLMODE_FRONT	Triangles with “front facing” orientation are discarded
CULLMODE_BACK	Triangles with “back facing” orientation are discarded
CULLMODE_BOTH	All triangles are discarded

10.3.11 Scissor Rectangle Clipping

A *scissor* operation can be used to restrict the extent of rendered pixels to a screen-space aligned rectangle. If the scissor operation is enabled, portions of objects falling outside of the intersection of the scissor rectangle and the clipped draw rectangle are clipped (pixels discarded).

The scissor operation is enabled by the **Scissor Rectangle Enable** state variable in SF_STATE. If enabled, the VPIndex associated with the leading vertex of the object is used to select the corresponding SF_VIEWPORT structure. Up to 16 structures are supported. The **Scissor Rectangle X,Y Min,Max** fields of the SF_VIEWPORT structure defines a scissor rectangle as a rectangle in integer pixel coordinates. The scissor rectangle is defined relative to the Drawing Rectangle to better support the OpenGL API. (OpenGL specifies the “Scissor Box” in window-relative coordinates). This allows instruction buffers with embedded Scissor Rectangle definitions to remain valid even after the destination window (drawing rectangle) moves.



Specifying either scissor rectangle $x_{min} > x_{max}$ or $y_{min} > y_{max}$ will cause all polygons to be discarded for a given viewport (effectively a null scissor rectangle).

10.3.12 Line Rasterization

The device supports three styles of line rendering: *zero-width (cosmetic)* lines, *non-antialiased* lines, and *antialiased* lines. (These rules also satisfy the OpenGL conformance requirements.) Non-antialiased lines are rendered as a polygon having a specified width as measured parallel to the major axis of the line. Antialiased lines are rendered as a rectangle having a specified width measured perpendicular to the line connecting the vertices.

The functions required to render lines is split between the SF and WM units. The SF unit is responsible for computing the overall geometry of the object to be rendered, including the pixel-exact bounding box, edge equations, etc., and therefore is provided with the screen-geometry-related state variables. The WM unit performs the actual scan conversion, determining the exact pixel included/excluded and coverage value for anti-aliased lines.

10.3.12.1 Zero-Width (Cosmetic) Line Rasterization

(The specification of zero-width line rasterization would be more correctly included in the WM Unit chapter, though is being included here to keep it with the rasterization details of the other line types).

When the **Line Width** is set to zero, the device will use special rules to rasterize zero-width (“cosmetic”) lines. The **Anti-Aliasing Enable** state variable is ignored when **Line Width** is zero.

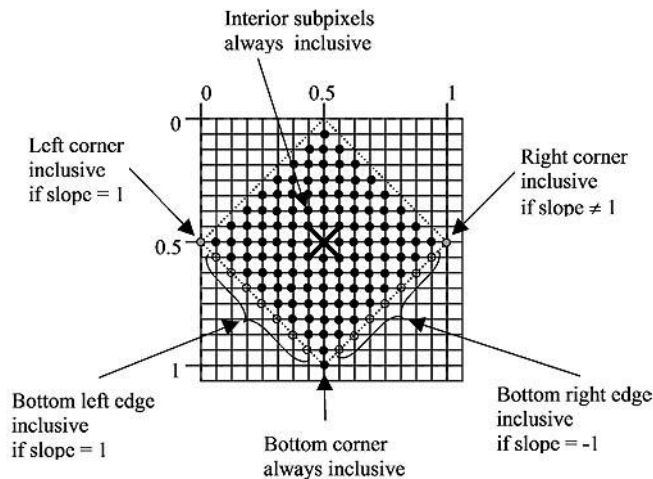
When the *LineWidth* is set to zero, the device will use special rules to rasterize “cosmetic” lines. The rasterization rules also comply with the OpenGL conformance requirements (for 1-pixel wide non-smooth lines). Refer to the appropriate API specifications for details on these requirements.

The GIQ rules basically intersect the directed, ideal line connecting two endpoints with an array of diamond-shaped areas surrounding pixel sample points. Wherever the line intersects a diamond (including passing through a diamond), the corresponding pixel is lit. Special rules are used to define the subpixel locations which are considered interior to the diamonds, as a function of the slope of the line. When a line ends in a diamond (and therefore does not exit that diamond), the corresponding pixel is not drawn. When a line starts in a diamond and exits that diamond, the corresponding pixel is drawn.

10.3.12.2 1GIQ (Diamond) Sampling Rules – Legacy Mode

When the **Legacy Line Rasterization Enable** bit in WM_STATE is 0x00000001 , zero-width lines are rasterized according to the algorithm presented in this subsection. Also note that the **Last Pixel Enable** bit of SF_STATE controls whether the last pixel of the last line in a LINESTRIP_xxx primitive or the last pixel of each line in a LINELIST_xxx primitive is rendered.

Refer to the following figure, which shows the neighborhood of subpixels around a given pixel sample point. Note that the device divides a pixel into a 16x16 array of subpixels, referenced by their upper left corners.



The solid-colored subpixels are considered “interior” to the diamond centered on the pixel sample point. Here the Manhattan distance to the pixel sample point (center) is less than $\frac{1}{2}$.

The subpixels falling on the edges of the diamond (Manhattan distance = $\frac{1}{2}$) are exclusive, with the following exceptions:

1. **The bottom corner subpixel is always inclusive.** This is to ensure that lines with slopes in the open range $(-1, 1)$ touch a diamond even when they cross exactly between pixel diamonds.
2. **The right corner subpixel is inclusive as long as the line slope is not exactly one, in which case the left corner subpixel is inclusive.** Including the right corner subpixel ensures that lines with slopes in the range $(1, +\infty]$ or $[-\infty, -1)$ touch a diamond even when they cross exactly between pixel diamonds. Including the left corner on slope=1 lines is required for proper handling of slope=1 lines (see (3) below) – where if the right corner was inclusive, a slope=1 line falling exactly between pixel centers would wind up lighting pixel on both sides of the line (not desired).
3. **The subpixels along the bottom left edge are inclusive only if the line slope = 1.** This is to correctly handle the case where a slope=1 line falls enters the diamond through a left or bottom corner and ends on the bottom left edge. One does not consider this “passing through” the diamond (where the normal rules would have us light the pixel). This is to avoid the following case: One slope=1 line segment enters through one corner and ends on the edge, and another (continuation) line segments starts at that point on the edge and exits through the other corner. If simply passing through a corner caused the pixel to be lit, this case would case the pixel to be lit twice – breaking the rule that connected line segments should not cause double-hits or missing pixels. So, by considering the entire bottom left edge as “inside” for slope=1 lines, we will only light the pixel when a line passes through the entire edge, or starts on the edge (or the left or bottom corner) and exits the diamond.
4. **The subpixels along the bottom right edge are inclusive only if the line slope = -1.** Similar case as (3), except slope=-1 lines require the bottom right edge to be considered inclusive.

The following equation determines whether a point (point.x, point.y) is inside the diamond of the pixel sample point (sample.x, sample.y), given additional information about the slope (slopePosOne, slopeNegOne).

























$$\text{delta_x} = \text{point.x} - \text{sample.x}$$

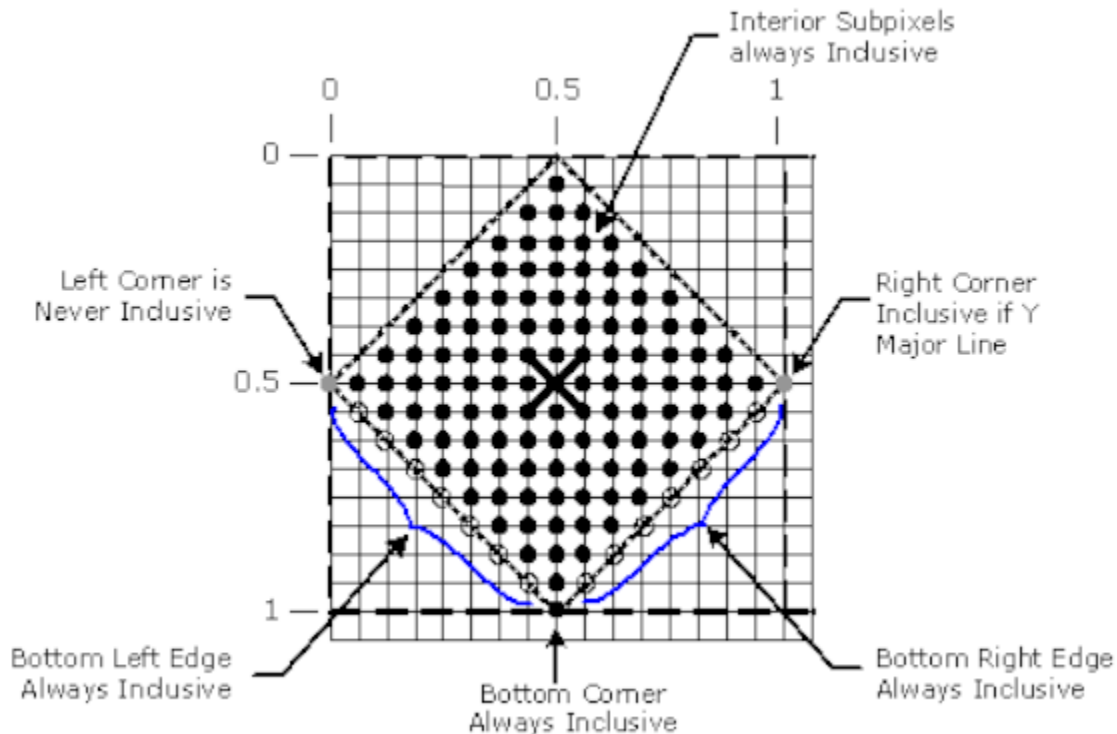


```
delta_y = point.y - sample.y
distance = abs(delta_x) + abs(delta_y)
interior = (distance < 0.5)
bottom_corner = (delta_x == 0.0) && (delta_y == 0.5)
left_corner   = (delta_x == -0.5) && (delta_y == 0.0)
right_corner  = (delta_x == 0.5) && (delta_y == 0.0)
bottom_left_edge = (distance == 0.5) && (delta_x < 0) && (delta_y > 0)
bottom_right_edge = (distance == 0.5) && (delta_x > 0) && (delta_y > 0)

inside = interior ||
bottom_corner ||
(slopePosOne ? left_corner : right_corner) ||
(slopePosOne && left_edge) ||
(slopeNegOne && right_edge)
```

10.3.12.3 GIQ (Diamond) Sampling Rules – DX10 Mode

When the **Legacy Line Rasterization Enable** bit in WM_STATE is                        



B6849-01

The solid-colored subpixels are considered “interior” to the diamond centered on the pixel sample point. Here the Manhattan distance to the pixel sample point (center) is less than $\frac{1}{2}$.

The subpixels falling on the edges of the diamond (Manhattan distance = $\frac{1}{2}$) are exclusive, with the following exceptions:

1. **The bottom corner subpixel is always inclusive.** This is to ensure that lines with slopes in the open range $(-1,1)$ touch a diamond even when they cross exactly between pixel diamonds.
2. **The right corner subpixel is inclusive as long as the line is not X Major (X Major is defined as $-1 \leq \text{slope} \leq 1$).** Including the right corner subpixel ensures that lines with slopes in the range $(>1, +\infty]$ or $[-\infty, <-1)$ touch a diamond even when they cross exactly between pixel diamonds.
3. **The left corner subpixel is never inclusive.** For Y Major lines, having the right corner subpixel as always inclusive requires that the left corner subpixel should never be inclusive, since a line falling exactly between pixel centers would wind up lighting pixel on both sides of the line (not desired).
4. **The subpixels along the bottom left edge are always inclusive.** This is to correctly handle the case where a line enters the diamond through a left or bottom corner and ends on the bottom left edge. One does not consider this “passing through” the diamond (where the normal rules would have us light the pixel). This is to avoid the following case: One line segment enters through one corner and ends on the edge, and another (continuation) line segments starts at that point on the edge and exits through the other corner. If simply passing through a corner caused the pixel to be lit, this case would cause the pixel to be lit twice – breaking the rule that connected line segments should not cause double-hits or missing pixels. So, by considering the entire bottom left edge as “inside”, we will only light the pixel when a line passes through the entire edge, or starts on the edge (or the left or bottom corner) and exits the diamond.



5. **The subpixels along the bottom right edge are always inclusive.** Same as case as (4), except slope=-1 lines require the bottom right edge to be considered inclusive.

The following equation determines whether a point (point.x, point.y) is inside the diamond of the pixel sample point (sample.x, sample.y), given additional information about the slope (XMajor).

$\text{delta_x} = \text{point.x} - \text{sample.x}$

$\text{delta_y} = \text{point.y} - \text{sample.y}$

$\text{distance} = \text{abs}(\text{delta_x}) + \text{abs}(\text{delta_y})$

$\text{interior} = (\text{distance} < 0.5)$

$\text{bottom_corner} = (\text{delta_x} == 0.0) \ \&\& \ (\text{delta_y} == 0.5)$

$\text{left_corner} = (\text{delta_x} == -0.5) \ \&\& \ (\text{delta_y} == 0.0)$

$\text{right_corner} = (\text{delta_x} == 0.5) \ \&\& \ (\text{delta_y} == 0.0)$

$\text{bottom_left_edge} = (\text{distance} == 0.5) \ \&\& \ (\text{delta_x} < 0) \ \&\& \ (\text{delta_y} > 0)$

$\text{bottom_right_edge} = (\text{distance} == 0.5) \ \&\& \ (\text{delta_x} > 0) \ \&\& \ (\text{delta_y} > 0)$

$\text{inside} = \text{interior} \ ||$

$\text{bottom_corner} \ ||$

$(!\text{XMajor} \ \&\& \ \text{right_corner}) \ ||$

$(\text{bottom_left_edge}) \ ||$

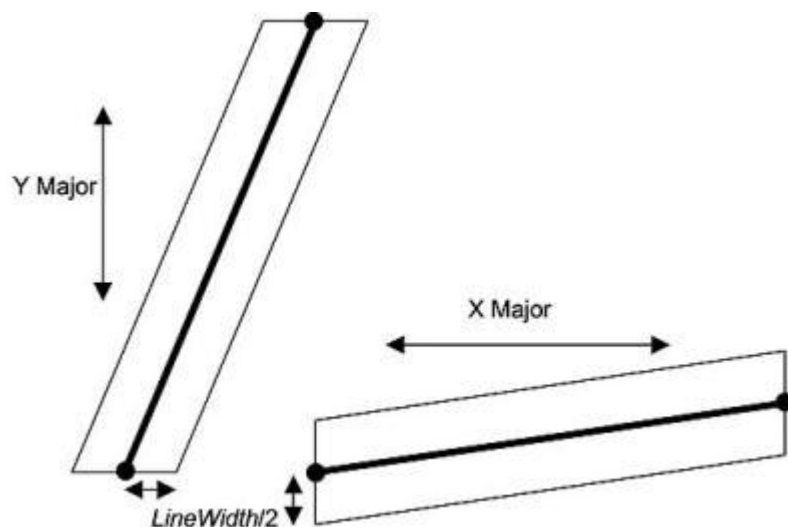
$(\text{bottom_right_edge})$

10.3.12.4 Non-Antialiased Wide Line Rasterization

Non-anti-aliased, non-zero-width lines are rendered as parallelograms that are centered on, and aligned to, the line joining the endpoint vertices. Pixels sampled interior to the parallelogram are rendered; pixels sampled exactly on the parallelogram edges are rendered according to the polygon “top left” rules.

The parallelogram is formed by first determining the major axis of the line (diagonal lines are considered x-major). The corners of the parallelogram are computed by translating the line endpoints by +/- (**Line Width** / 2) in the direction of the minor axis, as shown in the following diagram.

Non-Antialiased Line Rasterization

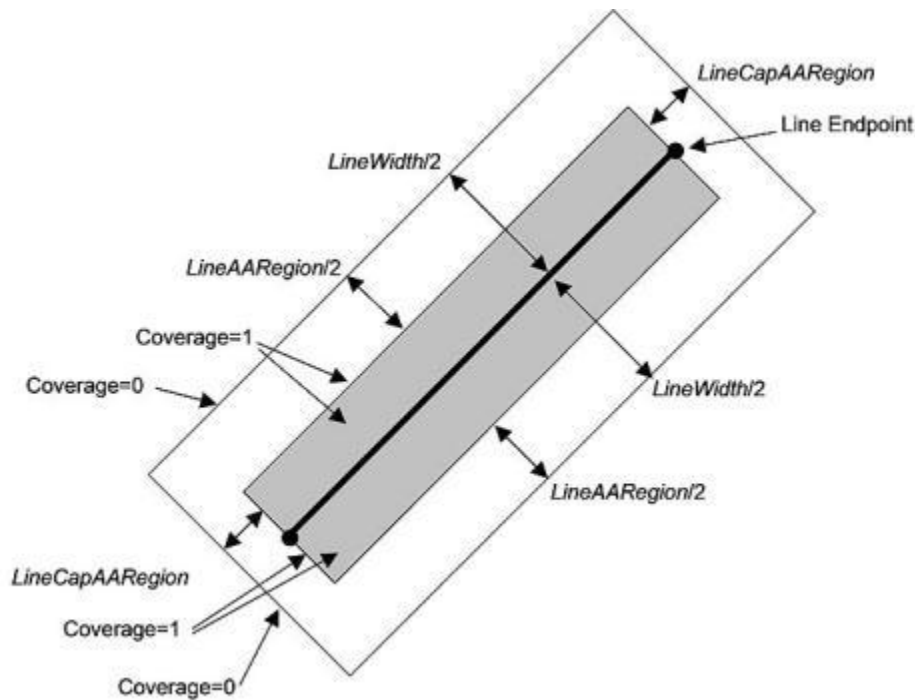


10.3.12.5 Anti-aliased Line Rasterization

Anti-aliased lines are rendered as rectangles that are centered on, and aligned to, the line joining the endpoint vertices. For each pixel in the rectangle, a fractional coverage value (referred to as Antialias Alpha) is computed – this coverage value will normally be used to attenuate the pixel’s alpha in the pixel shader thread. The resultant alpha value is therefore available for use in those downstream pixel pipeline stages in order to generate the desired effect (e.g., use the attenuated alpha value to modulate the pixel’s color, and add the result to the destination color, etc.). Note that software is required to explicitly program the pixel shader and pixel pipeline to obtain the desired anti-aliasing effect – the device will simply make the coverage-attenuated pixel alpha values available for use in the pixel shader.

The dimensions of the rendered rectangle, and the parameters controlling the coverage value computation, are programmed via the **Line Width**, **Line AA Region**, and **Line Cap AA Region** state variables, as shown below. The edges parallel to the line are located at the distance ($LineWidth/2$) from the line (measured in screen pixel units perpendicular to the line). The end-cap edges are perpendicular to the line and located at the distance ($LineCapAARegion$) from the endpoints.

Anti-aliased Line Rasterization



Along the parallel edges, the coverage values ramp from the value 0 at the very edges of the rectangle to the value 1 at the perpendicular distance ($LineAARegion/2$) from a given edge (in the direction of the line). A pixel's coverage value is computed with respect to the closest edge. In the cases where $(LineAARegion/2) < (LineWidth/2)$, this results in a region of fractional coverage values near the edges of the rectangle, and a region of "fully-covered" coverage values (i.e., the value 1) at the interior of the line. When $(LineAARegion/2) == (LineWidth/2)$, only pixel sample points falling exactly on the line can generate fully-covered coverage values. If $(LineAARegion/2) > (LineWidth/2)$, no pixels can be fully-covered (it is expected that this case is not typically desired).

Along the end cap edges, the coverage values ramp from the value 1 at the line endpoint to the value 0 at the cap edge – itself at a perpendicular distance ($LineCapAARegion$) from the endpoint. Note that, unlike the line-parallel edges, there is only a single parameter ($LineCapAARegion$) controlling the extension of the line at the end caps and the associated coverage ramp.

The regions near the corners of the rectangle have coverage values influenced by distances from both the line-parallel and end cap edges – here the two coverage values are multiplied together to provide a composite coverage value.

The computed coverage value for each pixel is passed through the Windower Thread Dispatch payload. The Pixel Shader kernel should be passed (unmodified) by the shader to the Render Cache as part of its output message.



10.3.13 3DSTATE_SF

3DSTATE_SF				
Source:		RenderCS		
Length Bias:		2		
DWord	Bit	Description		
0	31:29	Command Type		
		Default Value:	3h GFXPIPE	
		Format:	OpCode	
	28:27	Command SubType		
		Default Value:	3h GFXPIPE_3D	
		Format:	OpCode	
	26:24	3D Command Opcode		
		Default Value:	0h 3DSTATE	
		Format:	OpCode	
	23:16	3D Command Sub Opcode		
	Default Value:	13h 3DSTATE_SF		
	Format:	OpCode		
15:8	Reserved			
	Project:	All		
	Format:	MBZ		
7:0	DWord Length			
	Default Value:	5h Excludes DWord (0,1)		
	Project:	All		
	Format:	=n Total Length - 2		
1	31:15	Reserved		
		Format:	MBZ	
14:12	Depth Buffer Surface Format			
	Project:	All		
	Format:	U3 Enumerated Type		
	Specifies the format of the depth buffer. This must exactly match the Surface Format programmed via 3DSTATE_DEPTH_BUFFER. The SF requires this information in order to compute Global Depth Bias.			
	Value	Name	Description	Project
	0h	D32_FLOAT_S8X24_UINT	D32_FLOAT_S8X24_UINT	All
	1h	D32_FLOAT	D32_FLOAT	All
	2h	D24_UNORM_S8_UINT	D24_UNORM_S8_UINT	All
	3h	D24_UNORM_X8_UINT	D24_UNORM_X8_UINT	All
	4h	Reserved	Reserved	All
5h	D16_UNORM	D16_UNORM	All	
6h-7h	Reserved	Reserved	All	
11	Legacy Global Depth Bias Enable			



3DSTATE_SF		
	Project:	All
	Format:	Enable
	Enables the SF to use the Global Depth Offset Constant state unmodified. If this bit is not set, the SF will scale the Global Depth Offset Constant as described in section Error! Reference source not found. of this document.	
	Programming Notes	
	This bit should be set whenever non zero depth bias (Slope, Bias) values are used. Setting this bit may have some degradation of performance for some workloads.	
10	Statistics Enable	
	Project:	All
	Format:	Enable
	If ENABLED, this FF unit will increment CL_PRIMITIVES_COUNT on behalf of the CLIP stage. If DISABLED, CL_PRIMITIVES_COUNT will be left unchanged.	
	Programming Notes	
	This bit should be set whenever clipping is enabled and the Statistics Enable bit is set in CLIP_STATE. It should be cleared if clipping is disabled or Statistics Enable in CLIP_STATE is clear.	
9	Global Depth Offset Enable Solid	
	Project:	All
	Format:	Enable
	Enables computation and application of Global Depth Offset for SOLID objects.	
	Programming Notes	
	This bit should be set whenever non zero depth bias (Slope, Bias) values are used. Setting this bit may have some degradation of performance for some workloads.	
	Due to an HW issue driver needs to send a pipe control with stall when ever there is state change in depth bias related state	
	Project	
8	Global Depth Offset Enable Wireframe	
	Project:	All
	Format:	Enable
	Enables computation and application of Global Depth Offset when triangles are rendered in WIREFRAME mode.	
	Programming Notes	
	This bit should be set whenever non zero depth bias (Slope, Bias) values are used. Setting this bit may have some degradation of performance for some workloads.	
	Due to an HW issue driver needs to send a pipe control with stall when ever there is state change in depth bias related state	
	Project	
7	Global Depth Offset Enable Point	
	Project:	All
	Format:	Enable
	Enables computation and application of Global Depth Offset when triangles are rendered in POINT mode.	
	Programming Notes	
	This bit should be set whenever non zero depth bias (Slope, Bias) values are used. Setting this bit may have some degradation of performance for some workloads.	
	Due to an HW issue driver needs to send a pipe control with stall when ever there is state change in depth bias related state	
	Project	
6:5	FrontFace Fill Mode	
	Project:	All
	Format:	U2 enumerated type



3DSTATE_SF

3DSTATE_SF			
This state controls how front-facing triangle and rectangle objects are rendered.			
Value	Name	Description	Project
0h	SOLID	Any triangle or rectangle object found to be front-facing is rendered as a solid object. This setting is required when rendering rectangle (RECTLIST) objects.	All
1h	WIREFRAME	Any triangle object found to be front-facing is rendered as a series of lines along the triangle boundaries (as determined by the topology type and controlled by the vertex EdgeFlags).	All
2h	POINT	Any triangle object found to be front-facing is rendered as a set of point primitives at the triangle vertices (as determined by the topology type and controlled by the vertex EdgeFlags).NOTE: If the triangle is clipped, points will not be rendered at clip-inserted vertices. Point will only be rendered at original vertices (if visible).	
3h	Reserved		
4:3	BackFace Fill Mode		
	Project:	All	
	Format:	U2 enumerated type	
This state controls how back-facing triangle and rectangle objects are rendered.			
Value	Name	Description	Project
0h	SOLID	Any triangle or rectangle object found to be back-facing is rendered as a solid object. This setting is required when rendering rectangle (RECTLIST) objects.	All
1h	WIREFRAME	Any triangle object found to be back-facing is rendered as a series of lines along the triangle boundaries (as determined by the topology type and controlled by the vertex EdgeFlags).	All
2h	POINT	Any triangle object found to be back-facing is rendered as a set of point primitives at the triangle vertices (as determined by the topology type and controlled by the vertex EdgeFlags).NOTE: If the triangle is clipped, points will not be rendered at clip-inserted vertices. Point will only be rendered at original vertices (if visible).	
3h	Reserved		
2	Reserved		
	Project:	All	
	Format:	MBZ	
1	View Transform Enable		
	Project:	All	
	Format:	Enable	
This bit controls the Viewport Transform function.			
0	Front Winding		
	Project:	All	
Determines whether a triangle object is considered "front facing" if the screen space vertex positions, when traversed in the order, result in a clockwise (CW) or counter-clockwise (CCW) winding order. Does not apply to points or lines.			
2	31	Anti-Aliasing Enable	
	Project:	All	



3DSTATE_SF

		Format:	Enable
		This field enables "alpha-based" line anti-aliasing.	
		Programming Notes	
		This field must be disabled if any of the render targets have integer (UINT or SINT) surface format.	
30:29		Cull Mode	
		Project:	All
		Format:	3D_CullMode
		Controls removal (culling) of triangle objects based on orientation. The cull mode only applies to triangle objects and does not apply to lines, points or rectangles.	
		Value	Name
		Description	Project
	0h	CULLMODE_BOTH	All triangles are discarded (i.e., no triangle objects are drawn)
	1h	CULLMODE_NONE	No triangles are discarded due to orientation
	2h	CULLMODE_FRONT	Triangles with a front-facing orientation are discarded
	3h	CULLMODE_BACK	Triangles with a back-facing orientation are discarded
		Programming Notes	
		Orientation determination is based on the setting of the Front Winding state.	
27:18		Line Width	
		Project:	All
		Format:	U3.7
		Range: [0.0, 7.9921875]	
		Controls width of line primitives. Setting a Line Width of 0.0 specifies the rasterization of the "thinnest" (one-pixel-wide), non-antialiased lines. Note that this effectively overrides the effect of AAEnable (though the AAEnable state variable is not modified). Lines rendered with zero Line Width are rasterized using GIQ (Grid Intersection Quantization) rules as specified by the GDI APIs.	
		Programming Notes	
		Software must not program a value of 0.0 when running in MSRASTMODE_ON_xxx modes – zero-width lines are not available when multisampling rasterization is enabled.	
17:16		Line End Cap Antialiasing Region Width	
		Project:	All
		Format:	U2
		This field specifies the distances over which the coverage of anti-aliased line end caps are computed.	
		Value	Name
		Description	Project
	0h		0.5 pixels
	1h		1.0 pixels
	2h		2.0 pixels
	3h		4.0 pixels
15		Reserved	
		Project:	All
		Format:	MBZ
14		Reserved	
		Format:	MBZ
11		Scissor Rectangle Enable	



3DSTATE_SF																																																																											
	<table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> <tr> <td colspan="2">Enables operation of Scissor Rectangle.</td> </tr> </table>	Project:	All	Format:	Enable	Enables operation of Scissor Rectangle.																																																																					
Project:	All																																																																										
Format:	Enable																																																																										
Enables operation of Scissor Rectangle.																																																																											
10	<table border="1"> <tr> <td colspan="2">Reserved</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Reserved		Format:	MBZ																																																																						
Reserved																																																																											
Format:	MBZ																																																																										
9:8	<table border="1"> <tr> <td colspan="2">Multisample Rasterization Mode</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U2 enumerated type</td> </tr> <tr> <td colspan="2">This state is duplicated in 3DSTATE_WM and both must be set to the same value. See the field in 3DSTATE_WM for definition details.</td> </tr> </table>	Multisample Rasterization Mode		Project:	All	Format:	U2 enumerated type	This state is duplicated in 3DSTATE_WM and both must be set to the same value. See the field in 3DSTATE_WM for definition details.																																																																			
Multisample Rasterization Mode																																																																											
Project:	All																																																																										
Format:	U2 enumerated type																																																																										
This state is duplicated in 3DSTATE_WM and both must be set to the same value. See the field in 3DSTATE_WM for definition details.																																																																											
7:0	<table border="1"> <tr> <td colspan="2">Reserved</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Reserved		Project:	All	Format:	MBZ																																																																				
Reserved																																																																											
Project:	All																																																																										
Format:	MBZ																																																																										
3	<table border="1"> <tr> <td>31</td> <td> <table border="1"> <tr> <td colspan="2">Last Pixel Enable</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> <tr> <td colspan="2">If ENABLED, the last pixel of a diamond line will be lit. This state will only affect the rasterization of Diamond lines (will not affect wide lines or anti-aliased lines).</td> </tr> <tr> <td colspan="2" style="text-align: center;">Programming Notes</td> </tr> <tr> <td colspan="2">Last pixel is applied to all lines of a LINELIST, and only the last line of a LINESSTRIP.</td> </tr> </table> </td> </tr> <tr> <td>30:29</td> <td> <table border="1"> <tr> <td colspan="2">Triangle Strip/List Provoking Vertex Select</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>0-based vertex index</td> </tr> <tr> <td colspan="2">Selects which vertex of a triangle (in a triangle strip or list primitive) is considered the "provoking vertex". Used for flat shading of primitives. Does current implementation send provoking vertex first?</td> </tr> <tr> <td style="text-align: center;">Value</td> <td style="text-align: center;">Name</td> </tr> <tr> <td>0h</td> <td>Vertex 0</td> </tr> <tr> <td>1h</td> <td>Vertex 1</td> </tr> <tr> <td>2h</td> <td>Vertex 2</td> </tr> <tr> <td>3h</td> <td>Reserved</td> </tr> </table> </td> </tr> <tr> <td>28:27</td> <td> <table border="1"> <tr> <td colspan="2">Line Strip/List Provoking Vertex Select</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>0-based vertex index</td> </tr> <tr> <td colspan="2">Selects which vertex of a line (in a line strip or list primitive) is considered the "provoking vertex".</td> </tr> <tr> <td style="text-align: center;">Value</td> <td style="text-align: center;">Name</td> <td style="text-align: center;">Description</td> <td style="text-align: center;">Project</td> </tr> <tr> <td>0h</td> <td></td> <td>Vertex 0</td> <td>All</td> </tr> <tr> <td>1h</td> <td></td> <td>Vertex 1</td> <td>All</td> </tr> <tr> <td>2h</td> <td></td> <td>Reserved</td> <td>All</td> </tr> <tr> <td>3h</td> <td></td> <td>Reserved</td> <td>All</td> </tr> </table> </td> </tr> <tr> <td>26:25</td> <td> <table border="1"> <tr> <td colspan="2">Triangle Fan Provoking Vertex Select</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>0-based vertex index</td> </tr> <tr> <td colspan="2">Selects which vertex of a triangle (in a triangle fan primitive) is considered the "provoking vertex".</td> </tr> </table> </td> </tr> </table>	31	<table border="1"> <tr> <td colspan="2">Last Pixel Enable</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> <tr> <td colspan="2">If ENABLED, the last pixel of a diamond line will be lit. This state will only affect the rasterization of Diamond lines (will not affect wide lines or anti-aliased lines).</td> </tr> <tr> <td colspan="2" style="text-align: center;">Programming Notes</td> </tr> <tr> <td colspan="2">Last pixel is applied to all lines of a LINELIST, and only the last line of a LINESSTRIP.</td> </tr> </table>	Last Pixel Enable		Project:	All	Format:	Enable	If ENABLED, the last pixel of a diamond line will be lit. This state will only affect the rasterization of Diamond lines (will not affect wide lines or anti-aliased lines).		Programming Notes		Last pixel is applied to all lines of a LINELIST, and only the last line of a LINESSTRIP.		30:29	<table border="1"> <tr> <td colspan="2">Triangle Strip/List Provoking Vertex Select</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>0-based vertex index</td> </tr> <tr> <td colspan="2">Selects which vertex of a triangle (in a triangle strip or list primitive) is considered the "provoking vertex". Used for flat shading of primitives. Does current implementation send provoking vertex first?</td> </tr> <tr> <td style="text-align: center;">Value</td> <td style="text-align: center;">Name</td> </tr> <tr> <td>0h</td> <td>Vertex 0</td> </tr> <tr> <td>1h</td> <td>Vertex 1</td> </tr> <tr> <td>2h</td> <td>Vertex 2</td> </tr> <tr> <td>3h</td> <td>Reserved</td> </tr> </table>	Triangle Strip/List Provoking Vertex Select		Project:	All	Format:	0-based vertex index	Selects which vertex of a triangle (in a triangle strip or list primitive) is considered the "provoking vertex". Used for flat shading of primitives. Does current implementation send provoking vertex first?		Value	Name	0h	Vertex 0	1h	Vertex 1	2h	Vertex 2	3h	Reserved	28:27	<table border="1"> <tr> <td colspan="2">Line Strip/List Provoking Vertex Select</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>0-based vertex index</td> </tr> <tr> <td colspan="2">Selects which vertex of a line (in a line strip or list primitive) is considered the "provoking vertex".</td> </tr> <tr> <td style="text-align: center;">Value</td> <td style="text-align: center;">Name</td> <td style="text-align: center;">Description</td> <td style="text-align: center;">Project</td> </tr> <tr> <td>0h</td> <td></td> <td>Vertex 0</td> <td>All</td> </tr> <tr> <td>1h</td> <td></td> <td>Vertex 1</td> <td>All</td> </tr> <tr> <td>2h</td> <td></td> <td>Reserved</td> <td>All</td> </tr> <tr> <td>3h</td> <td></td> <td>Reserved</td> <td>All</td> </tr> </table>	Line Strip/List Provoking Vertex Select		Project:	All	Format:	0-based vertex index	Selects which vertex of a line (in a line strip or list primitive) is considered the "provoking vertex".		Value	Name	Description	Project	0h		Vertex 0	All	1h		Vertex 1	All	2h		Reserved	All	3h		Reserved	All	26:25	<table border="1"> <tr> <td colspan="2">Triangle Fan Provoking Vertex Select</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>0-based vertex index</td> </tr> <tr> <td colspan="2">Selects which vertex of a triangle (in a triangle fan primitive) is considered the "provoking vertex".</td> </tr> </table>	Triangle Fan Provoking Vertex Select		Project:	All	Format:	0-based vertex index	Selects which vertex of a triangle (in a triangle fan primitive) is considered the "provoking vertex".	
31	<table border="1"> <tr> <td colspan="2">Last Pixel Enable</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> <tr> <td colspan="2">If ENABLED, the last pixel of a diamond line will be lit. This state will only affect the rasterization of Diamond lines (will not affect wide lines or anti-aliased lines).</td> </tr> <tr> <td colspan="2" style="text-align: center;">Programming Notes</td> </tr> <tr> <td colspan="2">Last pixel is applied to all lines of a LINELIST, and only the last line of a LINESSTRIP.</td> </tr> </table>	Last Pixel Enable		Project:	All	Format:	Enable	If ENABLED, the last pixel of a diamond line will be lit. This state will only affect the rasterization of Diamond lines (will not affect wide lines or anti-aliased lines).		Programming Notes		Last pixel is applied to all lines of a LINELIST, and only the last line of a LINESSTRIP.																																																															
Last Pixel Enable																																																																											
Project:	All																																																																										
Format:	Enable																																																																										
If ENABLED, the last pixel of a diamond line will be lit. This state will only affect the rasterization of Diamond lines (will not affect wide lines or anti-aliased lines).																																																																											
Programming Notes																																																																											
Last pixel is applied to all lines of a LINELIST, and only the last line of a LINESSTRIP.																																																																											
30:29	<table border="1"> <tr> <td colspan="2">Triangle Strip/List Provoking Vertex Select</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>0-based vertex index</td> </tr> <tr> <td colspan="2">Selects which vertex of a triangle (in a triangle strip or list primitive) is considered the "provoking vertex". Used for flat shading of primitives. Does current implementation send provoking vertex first?</td> </tr> <tr> <td style="text-align: center;">Value</td> <td style="text-align: center;">Name</td> </tr> <tr> <td>0h</td> <td>Vertex 0</td> </tr> <tr> <td>1h</td> <td>Vertex 1</td> </tr> <tr> <td>2h</td> <td>Vertex 2</td> </tr> <tr> <td>3h</td> <td>Reserved</td> </tr> </table>	Triangle Strip/List Provoking Vertex Select		Project:	All	Format:	0-based vertex index	Selects which vertex of a triangle (in a triangle strip or list primitive) is considered the "provoking vertex". Used for flat shading of primitives. Does current implementation send provoking vertex first?		Value	Name	0h	Vertex 0	1h	Vertex 1	2h	Vertex 2	3h	Reserved																																																								
Triangle Strip/List Provoking Vertex Select																																																																											
Project:	All																																																																										
Format:	0-based vertex index																																																																										
Selects which vertex of a triangle (in a triangle strip or list primitive) is considered the "provoking vertex". Used for flat shading of primitives. Does current implementation send provoking vertex first?																																																																											
Value	Name																																																																										
0h	Vertex 0																																																																										
1h	Vertex 1																																																																										
2h	Vertex 2																																																																										
3h	Reserved																																																																										
28:27	<table border="1"> <tr> <td colspan="2">Line Strip/List Provoking Vertex Select</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>0-based vertex index</td> </tr> <tr> <td colspan="2">Selects which vertex of a line (in a line strip or list primitive) is considered the "provoking vertex".</td> </tr> <tr> <td style="text-align: center;">Value</td> <td style="text-align: center;">Name</td> <td style="text-align: center;">Description</td> <td style="text-align: center;">Project</td> </tr> <tr> <td>0h</td> <td></td> <td>Vertex 0</td> <td>All</td> </tr> <tr> <td>1h</td> <td></td> <td>Vertex 1</td> <td>All</td> </tr> <tr> <td>2h</td> <td></td> <td>Reserved</td> <td>All</td> </tr> <tr> <td>3h</td> <td></td> <td>Reserved</td> <td>All</td> </tr> </table>	Line Strip/List Provoking Vertex Select		Project:	All	Format:	0-based vertex index	Selects which vertex of a line (in a line strip or list primitive) is considered the "provoking vertex".		Value	Name	Description	Project	0h		Vertex 0	All	1h		Vertex 1	All	2h		Reserved	All	3h		Reserved	All																																														
Line Strip/List Provoking Vertex Select																																																																											
Project:	All																																																																										
Format:	0-based vertex index																																																																										
Selects which vertex of a line (in a line strip or list primitive) is considered the "provoking vertex".																																																																											
Value	Name	Description	Project																																																																								
0h		Vertex 0	All																																																																								
1h		Vertex 1	All																																																																								
2h		Reserved	All																																																																								
3h		Reserved	All																																																																								
26:25	<table border="1"> <tr> <td colspan="2">Triangle Fan Provoking Vertex Select</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>0-based vertex index</td> </tr> <tr> <td colspan="2">Selects which vertex of a triangle (in a triangle fan primitive) is considered the "provoking vertex".</td> </tr> </table>	Triangle Fan Provoking Vertex Select		Project:	All	Format:	0-based vertex index	Selects which vertex of a triangle (in a triangle fan primitive) is considered the "provoking vertex".																																																																			
Triangle Fan Provoking Vertex Select																																																																											
Project:	All																																																																										
Format:	0-based vertex index																																																																										
Selects which vertex of a triangle (in a triangle fan primitive) is considered the "provoking vertex".																																																																											



3DSTATE_SF			
	Value	Name	Project
	0h	Vertex 0	All
	1h	Vertex 1	All
	2h	Vertex 2	All
	3h	Reserved	All
24:15	Reserved		
	Project:	All	
	Format:	MBZ	
14	AA Line Distance Mode		
	Project:	All	
	Format:	U1	
	This bit controls the distance computation for antialiased lines.		
	Value	Name	Description
	1h	AALINEDISTANCE_TRUE	True distance computation. This is the normal setting which should yield WHQL compliance.
13	Reserved		
	Project:	All	
	Format:	MBZ	
12	Vertex Sub Pixel Precision Select		
	Project:	All	
	Format:	U1	
	Selects the number of fractional bits maintained in the vertex data		
	Value	Name	Description
	0h	Disable	8 sub pixel precision bits maintained
	1h	Enable	4 sub pixel precision bits maintained
11	Use Point Width State		
	Project:	All	
	Format:	U1	
	Controls whether the point width passed on the vertex or from state is used for rendering point primitives.		
	Value	Name	Description
	0h		Use Point Width on Vertex
	1h		Use Point Width from State
10:0	Point Width		
	Project:	All	
	Format:	U8.3	
	Range: [0.125, 255.875] pixels		
	This field specifies the size (width) of point primitives in pixels. This field is overridden (though not overwritten) whenever point width information is passed in the FVF		
4	31:0	Global Depth Offset Constant	
	Project:	All	
	Format:	IEEE_FP	
	Specifies the constant term in the Global Depth Offset function.		
5	31:0	Global Depth Offset Scale	
	Project:	All	
	Format:	IEEE_FP	



3DSTATE_SF		
		Specifies the scale term used in the Global Depth Offset function.
6	31:0	Global Depth Offset Clamp
		Project: All
		Format: IEEE_FP
		Specifies the clamp term used in the Global Depth Offset function.

10.3.14 3DSTATE_SBE

The state used by “setup backend” is defined by this inline state packet.

3DSTATE_SBE			
Source:		RenderCS	
Length Bias:		2	
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
		Default Value:	1Fh 3DSTATE_SBE
		Format:	OpCode
	15:8	Reserved	
		Project:	All
Format:		MBZ	
7:0	DWord Length	Default Value:	0Ch Excludes DWord (0,1)
		Project:	All
		Format:	=n
		Total Length - 2	
1	31:29	Reserved	
		Project:	All
		Format:	MBZ
28	Attribute Swizzle Control Mode		
		Format:	U1 enumerated type



3DSTATE_SBE

3DSTATE_SBE			
		<p>When Attribute Swizzle Enable is ENABLED, this bit controls whether attributes 0-15 or 16-31 are subject to the following swizzle controls:</p> <ul style="list-style-type: none"> • Attribute n Component Override X/Y/Z/W • Attribute n Constant Source • Attribute n Swizzle Select • Attribute n Source Attribute • Attribute n Wrap Shortest Enables <p>Note that the Number of SF Output Attributes field specifies how many attributes are output.</p> <p>Note: This field does not impact any functions which provide separate states for all 32 attributes (e.g., Point sprite, Constant interpolation).</p>	
		Value	Name
		Description	Project
	0h	SWIZ_0_15	Attributes 0-15 are subject to swizzling, and attributes 16-31 are not.
	1h	SWIZ_16_31	Attributes 16-31 are subject to swizzling, and attributes 0-15 are not. Only valid when 16 or more attributes are output.
27:22	Number of SF Output Attributes		
	Format:	U6 count of attributes	
	Specifies the number of vertex attributes passed from the SF stage to the WM stage (does not include Position).		
	Value	Name	
	[0,32]		
21	Attribute Swizzle Enable		
	Project:	All	
	Format:	Enable	
	Enables the SF to perform swizzling on (up to the first 16) vertex attributes. If DISABLED, all vertex attributes are passed through.		
20	Point Sprite Texture Coordinate Origin		
	Project:	All	
	Format:	U1 enumerated type	
	This state controls how Point Sprite Texture Coordinates are generated (when enabled on a per-attribute basis by Point Sprite Texture Coordinate Enable).		
	Value	Name	Description
	0h	UPPERLEFT	Top Left = (0,0,0,1)Bottom Left = (0,1,0,1)Bottom Right = (1,1,0,1)
	1h	LOWERLEFT	Top Left = (0,1,0,1)Bottom Left = (0,0,0,1)Bottom Right = (1,0,0,1)
19:16	Reserved		
	Project:	All	
	Format:	MBZ	
15:11	Vertex URB Entry Read Length		
	Project:	All	



3DSTATE_SBE

		Format:	U5 Specifies the amount of URB data read for each Vertex URB entry, in 256-bit register increments.
		Value	Name
		[1,16]	
		Programming Notes	
		It is UNDEFINED to set this field to 0 indicating no Vertex URB data to be read. This field should be set to the minimum length required to read the maximum source attribute. The maximum source attribute is indicated by the maximum value of the enabled Attribute # Source Attribute if Attribute Swizzle Enable is set, Number of Output Attributes-1 if enable is not set. $read_length = \text{ceiling}((\text{max_source_attr}+1)/2)$	
	10	Reserved	
		Project:	All
	9:4	Vertex URB Entry Read Offset	
		Project:	All
		Format:	U6
		Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB.	
		Value	Name
		[0,63]	
	3:0	Reserved	
		Project:	All
		Format:	MBZ
2..9	31	Attribute [2n+1] Component Override W	
		Project:	All
		Format:	Enable
		If set, the W component of output Attribute 1 is overridden by the W component of the constant vector specified by ConstantSource[1].	
	30	Attribute [2n+1] Component Override Z	
		Project:	All
		Format:	Enable
		If set, the Z component of output Attribute 1 is overridden by the Z component of the constant vector specified by ConstantSource[1].	
	29	Attribute [2n+1] Component Override Y	
		Project:	All
		Format:	Enable
		If set, the Y component of output Attribute 1 is overridden by the Y component of the constant vector specified by ConstantSource[1].	
	28	Attribute [2n+1] Component Override X	
		Project:	All
		Format:	Enable
		If set, the X component of output Attribute 1 is overridden by the X component of the constant vector	

3DSTATE_SBE				
	specified by ConstantSource[1].			
27	Reserved			
	Project:	All		
	Format:	MBZ		
26:25	Attribute [2n+1] Constant Source			
	Project:	All		
	Format:	U2 enumerated type		
	This state selects a constant vector which can be used to override individual components of Attribute 1			
	Value	Name	Description	Project
	0h	CONST_0000	Constant.xyzw = 0.0,0.0,0.0,0.0	All
	1h	CONST_0001_FLOAT	Constant.xyzw = 0.0,0.0,0.0,1.0	All
	2h	CONST_1111_FLOAT	Constant.xyzw = 1.0,1.0,1.0,1.0	All
	3h	PRIM_ID	Constant.xyzw = PrimID (replicated)	All
	24	Reserved		
Project:		All		
Format:		MBZ		
23:22	Attribute [2n+1] Swizzle Select			
	Project:	All		
	Format:	U2 enumerated type		
	This state, along with Attribute 1 Source Attribute, specifies the source for output Attribute 1.			
	Value	Name	Description	Project
	0h	INPUTATTR	This attribute is sourced from AttrInputReg[SourceAttribute]	All
	1h	INPUTATTR_FACING	If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute]. If the object is back-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1].	All
	2h	INPUTATTR_W	This attribute is sourced from AttrInputReg[SourceAttribute]. The W component is copied to the X component.	All
	3h	INPUTATTR_FACING_W	If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute]. If the object is back-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1]. The W component is copied to the X component.	All
	21	Reserved		
Project:		All		
Format:		MBZ		
20:16	Attribute [2n+1] Source Attribute			
	Project:	All		
	Format:	U5		
This field selects the source attribute for Attribute 1. Source attribute 0 corresponds to the first 128 bits of data indicated by Vertex URB Entry Read Offset				
15	Attribute [2n] Component Override W			
	Project:	All		
	Format:	Enable		
If set, the W component of output Attribute 0 is overridden by the W component of the constant vector				



3DSTATE_SBE

	specified by ConstantSource[1].		
14	Attribute [2n] Component Override Z		
	Project:	All	
	Format:	Enable	
	If set, the Z component of output Attribute 0 is overridden by the Z component of the constant vector specified by ConstantSource[1].		
13	Attribute [2n] Component Override Y		
	Project:	All	
	Format:	Enable	
	If set, the Y component of output Attribute 0 is overridden by the Y component of the constant vector specified by ConstantSource[1].		
12	Attribute [2n] Component Override X		
	Project:	All	
	Format:	Enable	
	If set, the X component of output Attribute 0 is overridden by the X component of the constant vector specified by ConstantSource[1].		
11	Reserved		
	Project:	All	
	Format:	MBZ	
10:9	Attribute [2n] Constant Source		
	Project:	All	
	Format:	U2 enumerated type	
	This state selects a constant vector which can be used to override individual components of Attribute 0		
	Value	Name	Description
	0h	CONST_0000	Constant.xyzw = 0.0,0.0,0.0,0.0
	1h	CONST_0001_FLOAT	Constant.xyzw = 0.0,0.0,0.0,1.0
	2h	CONST_1111_FLOAT	Constant.xyzw = 1.0,1.0,1.0,1.0
	3h	PRIM_ID	Constant.xyzw = PrimID (replicated)
		Project	
			All
			All
			All
			All
8	Reserved		
	Project:	All	
	Format:	MBZ	
7:6	Attribute [2n] Swizzle Select		
	Project:	All	
	Format:	U2 enumerated type	
	This state, along with Attribute 0 Source Attribute, specifies the source for output Attribute 0.		
	Value	Name	Description
	0h	INPUTATTR	This attribute is sourced from AttrInputReg[SourceAttribute]
	1h	INPUTATTR_FACING	If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute].If the object is back-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1].
	2h	INPUTATTR_W	This attribute is sourced from AttrInputReg[SourceAttribute].
		Project	
			All
			All
			All



3DSTATE_SBE			
			The W component is copied to the X component.
	3h	INPUTATTR_FACING_W	If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute]. If the object is back-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1]. The W component is copied to the X component.
	5	Reserved	
		Project:	All
		Format:	MBZ
	4:0	Attribute [2n] Source Attribute	
		Project:	All
		Format:	U5
		This field selects the source attribute for Attribute 0. Source attribute 0 corresponds to the first 128 bits of data indicated by Vertex URB Entry Read Offset	
10	31:0	Point Sprite Texture Coordinate Enable	
		Project:	All
		Format:	32-bit bitmask
		When processing point primitives, the attributes from the incoming point vertex are typically copied to the point object corner vertices. However, if a bit is set in this field, the corresponding Attribute is selected as a Point Sprite Texture Coordinate, in which case each corner vertex is assigned a pre-defined texture coordinate as defined by the Point Sprite Texture Coordinate Origin state bit. Bit 0 corresponds to output Attribute 0. This field must be programmed to 0 when non-point primitives are rendered.	
11	31:0	Constant Interpolation Enable[31:0]	
		Project:	All
		This field is a bitmask containing a Constant Interpolation Enable bit for each corresponding attribute. If a bit is set, that attribute will undergo constant interpolation, and the corresponding WrapShortest Enable bits (if defined) will be ignored. If a bit is clear, components which are not enabled for WrapShortest interpolation (if defined) will be linearly interpolated.	
12	31:28	Attribute 7 WrapShortest Enables	
		Project:	All
		Format:	Enable[4]
		This state selects which components (if any) of Attribute 7 are to be interpolated in a "wrap shortest" fashion. Operation is UNDEFINED if any of these bits are set and the Constant Interpolation Enable bit associated with this attribute is set. Note that wrap-shortest interpolation is only supported for Attributes 0-15. Bit 0: WrapShortest X Component Bit 1: WrapShortest Y Component Bit 2: WrapShortest Z Component Bit 3: WrapShortest W Component	
	27:24	Attribute 6 WrapShortest Enables	
		Project:	All
		(See above).	
	23:20	Attribute 5 WrapShortest Enables	
		Project:	All
		(See above).	
	19:16	Attribute 4 WrapShortest Enables	
		Project:	All



3DSTATE_SBE		
	(See above).	
15:12	Attribute 3 WrapShortest Enables	
	Project: All	
	(See above).	
11:8	Attribute 2 WrapShortest Enables	
	Project: All	
	(See above).	
7:4	Attribute 1 WrapShortest Enables	
	Project: All	
	(See above).	
3:0	Attribute 0 WrapShortest Enables	
	Project: All	
	(See above).	
13	Attribute 15 WrapShortest Enables	
	Project: All	
	Format: Enable[4]	
	This state selects which components (if any) of Attribute 15 are to be interpolated in a “wrap shortest” fashion. Operation is UNDEFINED if any of these bits are set and the Constant Interpolation Enable bit associated with this attribute is set. Bit 0: WrapShortest X Component Bit 1: WrapShortest Y Component Bit 2: WrapShortest Z Component Bit 3: WrapShortest W Component	
	Attribute 14 WrapShortest Enables	
	Project: All	
	(See above).	
	Attribute 13 WrapShortest Enables	
	Project: All	
	(See above).	
Attribute 12 WrapShortest Enables		
Project: All		
(See above).		
Attribute 11 WrapShortest Enables		
Project: All		
(See above).		
Attribute 10 WrapShortest Enables		
Project: All		
(See above).		
Attribute 9 WrapShortest Enables		
Project: All		



3DSTATE_SBE	
	(See above).
3:0	Attribute 8 WrapShortest Enables
	Project: All
	(See above).

10.3.15 SF_CLIP_VIEWPORT

The viewport-specific state used by both the SF and CL units (SF_CLIP_VIEWPORT) is stored as an array of up to 16 elements, each of which contains the DWords described below. The start of each element is spaced 16 DWords apart. The location of first element of the array, as specified by both **Pointer to SF_VIEWPORT** and **Pointer to CLIP_VIEWPORT**, is aligned to a 64-byte boundary.

SF_CLIP_VIEWPORT		
Source:	RenderCS	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	Bit	Description
0	31:0	Viewport Matrix Element m00 Format: IEEE_Float
1	31:0	Viewport Matrix Element m11 Format: IEEE_Float
2	31:0	Viewport Matrix Element m22 Format: IEEE_Float Total Length - 2
3	31:0	Viewport Matrix Element m30 Format: IEEE_Float Total Length - 2
4	31:0	Viewport Matrix Element m31 Format: IEEE_Float
5	31:0	Viewport Matrix Element m32 Format: IEEE_Float
6	31:0	Reserved
7	31:0	Reserved Project: All Format: MBZ
8	31:0	X Min Clip Guardband Default Value: 0h Excludes DWord (0,1) Format: FLOAT32 . This 32-bit float represents the XMin guardband boundary (normalized to Viewport.XMin == -1.0f). This corresponds to the left boundary of the NDC guardband.
9	31:0	X Max Clip Guardband



SF_CLIP_VIEWPORT		
		Default Value: 0h Excludes DWord (0,1) Format: FLOAT32 This 32-bit float represents the XMax guardband boundary (normalized to Viewport..XMax == 1.0f). This corresponds to the right boundary of the NDC guardband.
10	31:0	Y Min Clip Guardband Format: FLOAT32 This 32-bit float represents the YMin guardband boundary (normalized to Viewport.YMin == -1.0f). This corresponds to the bottom boundary of the NDC guardband.
11	31:0	Y Max Clip Guardband: Format: FLOAT32 This 32-bit float represents the YMax guardband boundary (normalized to Viewport.YMax == 1.0f). This corresponds to the top boundary of the NDC guardband.
12		
12..15	31:0	Reserved Format: MBZ

10.3.16 SCISSOR_RECT

SCISSOR_RECT								
Source: RenderCS								
Default Value: 0x00000000, 0x00000000								
The viewport-specific state used by the SF unit (SCISSOR_RECT) is stored as an array of up to 16 elements, each of which contains the DWords described below. The start of each element is spaced 2 DWords apart. The location of first element of the array, as specified by Pointer to SCISSOR_RECT, is aligned to a 32-byte boundary.								
DWord	Bit	Description						
0	31:16	Scissor Rectangle Y Min Project: All Format: U16 Pixels from Drawing Rectangle origin (upper left corner) Specifies Y Min coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) Y coordinates less than Y Min will be clipped out if Scissor Rectangle is enabled. NOTE: If Y Min is set to a value greater than Y Max, all primitives will be discarded for this viewport. <table border="1" style="width: 100%; margin-top: 5px;"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> <th style="text-align: center;">Project</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">[0,16383]</td> <td></td> <td></td> </tr> </tbody> </table>	Value	Name	Project	[0,16383]		
Value	Name	Project						
[0,16383]								
	15:0	Scissor Rectangle X Min Project: All Format: U16 Pixels from Drawing Rectangle origin (upper left corner) Specifies X Min coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) X coordinates less than X Min will be clipped out if Scissor Rectangle is enabled.						



SCISSOR_RECT								
		NOTE: If X Min is set to a value greater than X Max, all primitives will be discarded for this viewport.						
		<table border="1"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> <th style="text-align: center;">Project</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">[0,16383]</td> <td></td> <td></td> </tr> </tbody> </table>	Value	Name	Project	[0,16383]		
Value	Name	Project						
[0,16383]								
1	31:16	Scissor Rectangle Y Max Project: All Format: U16 Pixels from Drawing Rectangle origin (upper left corner) Specifies Y Max coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) Y coordinates greater than Y Max will be clipped out if Scissor Rectangle is enabled. <table border="1"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> <th style="text-align: center;">Project</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">[0,16383]</td> <td></td> <td></td> </tr> </tbody> </table>	Value	Name	Project	[0,16383]		
Value	Name	Project						
[0,16383]								
	15:0	Scissor Rectangle X Max Project: All Format: U16 Pixels from Drawing Rectangle origin (upper left corner) Specifies X Max coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) Y coordinates greater than X Max will be clipped out if Scissor Rectangle is enabled. <table border="1"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> <th style="text-align: center;">Project</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0-16383</td> <td></td> <td></td> </tr> </tbody> </table>	Value	Name	Project	0-16383		
Value	Name	Project						
0-16383								

10.4 Attribute Interpolation Setup

With the attribute interpolation setup function being implemented in hardware for, a number of state fields in 3DSTATE_SF are utilized to control interpolation setup.

Number of SF Output Attributes sets the number of attributes that will be output from the SF stage, not including position. This can be used to specify up to 32, and may differ from the number of input attributes. The number of input attributes is derived from the **Vertex URB Entry Read Length** field. Note that this field is also used to specify whether swizzling is to be performed on Attributes 0-15 or Attributes 16-32. See the state field definition for details.

10.4.1 Attribute Swizzling

The first or last set of 16 attributes can be swizzled according to certain state fields. **Attribute Swizzle Enable** enables the swizzling for all 16 of these attributes, and each of the attributes has a 2-bit **Swizzle Select** field that controls swizzling with the following settings:

- INPUTATTR – This attribute is sourced from AttrInputReg[SourceAttribute].
- INPUTATTR_FACING – This attribute is sourced from AttrInputReg[SourceAttribute] if the object is front-facing, otherwise it is sourced from AttrInputReg[SourceAttribute+1].
- INPUTATTR_W – This attribute is sourced from AttrInputReg[SourceAttribute]. WYZW (the W component of the source is copied to the X component of the destination).
- INPUTATTR_FACING – If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute]. WYZW (the W component of the source is copied to the X



component of the destination). If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1]. WYZW.

Each of the first or last set of 16 attributes also has a 5-bit **Source Attribute** field which specify, per output attribute (not component), which input attribute sources the output attribute when INPUTATTR is selected for **Swizzle Select**. A **Source Attribute** value of 0 corresponds to the 128-bit attribute immediately following the vertex 4D position. If INPUTATTR_FACING is selected, this specifies the first of two consecutive (front,back) input attributes, where the SourceAttribute value can be an odd or even number (just not 31, as that would place the back-face input attribute past the end of the input max complement of input attributes).

Constant overriding is also available for the first or last set of 16 attributes. Each attribute has a **Constant Source** field which specifies the constant values per swizzled attribute, with the following settings available:

- XYZW = 0000
- XYZW = 0001
- XYZW = 1111

Each channel of each attribute has a **Component Override** field to control whether the corresponding channel is overridden with the constant value defined in **Constant Source**.

10.4.2 1Interpolation Modes

All 32 attributes have a **Constant Interpolation Enable** state field bit to specify whether all components of the attribute are to be interpolated as constant values (not varying over the pixels of the object). If set, the attribute at the provoking vertex is copied to a0, and a1 and a2 are set to zero – this results in a constant interpolation of the provoking vertex value. If clear, the attribute is linearly interpolated. Attributes 0-15 are further subjected to Wrap Shortest processing on a per-component basis, via the **Attribute WrapShortest Enables** state bitfields. WrapShortest processing modifies the a1 and/or a2 values depending on attribute deltas. All

The table below indicates the output values of a0, a1, and a2 depending on interpolation mode settings.

	a0	a1	a2
Constant	A0	0.0	0.0
Linear	A0	A1-A0	A2-A0
Wrap Shortest	A0	(A1-A0)+1 (A1-A0) <= -0.5	(A2-A0)+1 (A2-A0) <= -0.5
		(A1-A0)-1 (A1-A0) >= 0.5	(A2-A0)-1 (A2-A0) >= 0.5
		(A1-A0) otherwise	(A2-A0) otherwise

10.4.3 Point Sprites

Normally all vertex attributes (including texture coordinates) other than position are simply replicated from the incoming point center vertex to the generated point object (corner) vertices. However, both DX9 and OGL support “sprite points”, where some/all texture coordinates are replaced with full-scale 2D texture coordinates.

A 32-bit **PointSprite TextureCoordinate Enable** bit mask controls whether the corresponding vertex attribute is to be replaced by a sprite point texture coordinate. The global (not per-attribute) **Point Sprite TextureCoordinate Origin** field controls how the point object vertex (top/bottom, left/right) texture coordinates are generated:



UPPERLEFT	Left	Right
Top	(0,0,0,1)	(1,0,0,1)
Bottom	(0,1,0,1)	(1,1,0,1)

LOWERLEFT	Left	Right
Top	(0,1,0,1)	(1,1,0,1)
Bottom	(0,0,0,1)	(1,0,0,1)

10.5 Depth Offset

The state for depth offset in 3DSTATE_SF controls the depth offset function. Since this function was previously contained in the Windower stage, refer to the “Depth Offset” section in the Windower chapter for more details on this function.

10.6 Other SF Functions

10.6.1 Statistics Gathering

The SF stage itself does not have any associated pipeline statistics; however, it counts the number of objects being output by the clipper on the clipper’s behalf, since it is less feasible to have the CLIP unit figure out how many objects have been output by a clip thread. It is easy for the SF unit to count the number of objects it receives from the CLIP stage since it is decomposing the output primitive topologies into objects anyway.

If the **Statistics Enable** bit is set in SF_STATE, then SF will increment the CL_PRIMITIVES_COUNT Register (see Memory Interface Registers in Volume Ia, GPU) once for each object in each primitive topology it receives from the CLIP stage. This bit should always be set if clipping is enabled and pipeline statistics are desired.

Software should always clear the **Statistics Enable** bit in SF_STATE if the clipper is disabled since objects SF receives are not considered “primitives output by the clipper” unless the clipper is enabled. Note that the clipper can be disabled either using bypass mode via a PIPELINE_STATE_POINTERS command with **Clip Enable** clear or by setting **Clip Mode** in CLIP_STATE to CLIPMODE_ACCEPT_ALL.

11. 3D Pipeline – Windower (WM) Stage

11.1 Overview

As mentioned in the *SF Unit* chapter, the SF stage prepares an object for scan conversion by the Window/Masker (WM) unit. Refer to the *SF Unit* chapter for details on the screen-space geometry of objects to be rendered. The WM unit uses the parameters provided by the SF unit in the object-specific rasterization algorithms.

The WM stage of the 3D pipeline performs the following operations (at a high level)

- Pre-scan-conversion modification of some primitive attributes, including
 - Application of Depth Offset to the position Z attribute
- Scan-conversion of the various primitive types, including
 - 2D clipping to the scissor/draw rectangle intersection
- Spawning of Pixel Shader (PS) threads to process the pixels resulting from scan-conversion

The spawned Pixel Shader (PS) threads are responsible for the following (high-level) operations

- interpolation of vertex attributes (other than X,Y,Z) to the pixel location
- performing any “Pixel Shader” operations dictated by the API PS program
 - Using the Sampler shared function to sample data from “texture” surfaces
 - Using the DataPort to perform general memory I/O
- Submitting the shaded pixel results to the DataPort for any subsequent “blending” (aka Output Merger) operation and write to the RenderCache.

The WM unit keeps a scoreboard of pixels being processed in outstanding PS threads in order to guarantee in-order rasterization results. This allows the WM unit to overlap processing of several objects.

11.1.1 Inputs from SF to WM

The outputs from the SF stage to the WM stage are mostly comprised of implementation-specific information required for the rasterization of objects. The types of information is summarized below, but as the interface is not exposed to software a detailed discussion is not relevant to this specification.

- PrimType of the object
- VPIndex, RTAIndex associated with the object
- Handle of the Primitive URB Entry (PUE) that was written by the SF (Setup) thread. This handle will be passed to all WM (PS) threads spawned from the WM's rasterization process.
- Information regarding the X,Y extent of the object (e.g., bounding box, etc.)
- Edge or line interpolation information (e.g., edge equation coefficients, etc.)
- Information on where the WM is to start rasterization of the object
- Object orientation (front/back-facing)
- Last Pixel indication (for line drawing)



11.2 Windower Pipelined State

{WA}: The driver must make sure a PIPE_CONTROL with the **Depth Stall Enable** bit set after all the following states are programmed:

- 3DSTATE_PS
- 3DSTATE_VIEWPORT_STATE_POINTERS_CC
- 3DSTATE_CONSTANT_PS
- 3DSTATE_BINDING_TABLE_POINTERS_PS
- 3DSTATE_SAMPLER_STATE_POINTERS_PS
- 3DSTATE_CC_STATE_POINTERS
- 3DSTATE_BLEND_STATE_POINTERS
- 3DSTATE_DEPTH_STENCIL_STATE_POINTERS

11.2.1 3DSTATE_WM

3DSTATE_WM		
Source: RenderCS		
Length Bias: 2		
DWord	Bit	Description
0	31:29	Command Type
		Default Value: 3h GFXPIPE
		Format: OpCode
	28:27	Command SubType
		Default Value: 3h GFXPIPE_3D
		Format: OpCode
	26:24	3D Command Opcode
		Default Value: 0h 3DSTATE_PIPELINED
		Format: OpCode
	23:16	3D Command Sub Opcode
	Default Value: 14h 3DSTATE_WM	
	Format: OpCode	
15:8	Reserved	
	Project:	All
	Format:	MBZ
7:0	DWord Length	
	Default Value:	01h Excludes DWord (0,1)
	Project:	All
	Format:	=n
		Total Length - 2
1	31	Statistics Enable
	Project:	All
	Format:	Enable
		If ENABLED, the Windower and pixel pipeline will engage in statistics gathering. If DISABLED, statistics information associated with this FF stage will be left unchanged. See Statistics Gathering.



3DSTATE_WM				
30	Depth Buffer Clear			
	<table border="1"><tr><td>Project:</td><td>All</td></tr><tr><td>Format:</td><td>Enable</td></tr></table>	Project:	All	Format:
Project:	All			
Format:	Enable			
When set, the depth buffer is initialized as a side-effect of rendering pixels.				
Programming Notes				
If this field is enabled,				
the Depth Test Enable field in DEPTH_STENCIL_STATE must be disabled.				
3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set.				
3DSTATE_DEPTH_BUFFER::Stencil Write Enable must be set if 3DSTATE_STENCIL_BUFFER::Stencil buffer enable is set. Additionally the following must be set to the correct values.				
B. DEPTH_STENCIL_STATE::Stencil Write Mask must be 0xFF				
C. DEPTH_STENCIL_STATE::Stencil Test Mask must be 0xFF				
D. DEPTH_STENCIL_STATE::Back Face Stencil Write Mask must be 0xFF				
E. DEPTH_STENCIL_STATE::Back Face Stencil Test Mask must be 0xFF				
Refer to section 0 "Depth Buffer Clear" for additional restrictions when this field is enabled. If this field is enabled, Pixel Shader Kill Pixel must be disabled.				
29	Thread Dispatch Enable			
	<table border="1"><tr><td>Project:</td><td>All</td></tr><tr><td>Format:</td><td>Enable</td></tr></table>	Project:	All	Format:
Project:	All			
Format:	Enable			
This bit, if set, indicates that it is possible for a PS thread to modify a render target, i.e., at least one render target is enabled (is not of type SURFTYPE_NULL and has at least one channel enabled for writes) and the PS kernel contains a code path that may issue a write to that/those enabled RTs.				
Programming Notes				
This bit is used for performance optimizations and does not directly control writing to render targets. If this bit is DISABLED, no pixel shader threads will be dispatched. For correct behavior, this bit must be set consistently with the behavior of the PS kernel, i.e. if this bit is DISABLED the PS kernel must not write color or depth to any render targets. If this field is disabled, Pixel Shader Kill Pixel must be disabled.				
28	Depth Buffer Resolve Enable			
	<table border="1"><tr><td>Project:</td><td>All</td></tr><tr><td>Format:</td><td>Enable</td></tr></table>	Project:	All	Format:
Project:	All			
Format:	Enable			
When set, the depth buffer is made to be consistent with the hierarchical depth buffer as a side-effect of rendering pixels. This is intended to be used when the depth buffer is to be used as a surface outside of the 3D rendering operation.				
Programming Notes				
If this field is enabled,				
the Depth Buffer Clear and Hierarchical Depth Buffer Resolve Enable fields must both be disabled.				

3DSTATE_WM									
	<p>3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set.</p> <p>Refer to section 11.5.4.2 "Depth Buffer Resolve" for additional restrictions when this field is enabled. If Hierarchical Depth Buffer Enable is disabled, enabling this field will have no effect.</p>								
27	<p>Hierarchical Depth Buffer Resolve Enable</p> <table border="1" style="width: 100%;"> <tr> <td style="width: 80%;">Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> </table> <p>When set, the hierarchical depth buffer is made to be consistent with the depth buffer as a side-effect of rendering pixels. This is intended to be used when the depth buffer has been modified outside of the 3D rendering operation.</p> <table border="1" style="width: 100%;"> <thead> <tr> <th style="text-align: center;">Programming Notes</th> <th style="text-align: center;">Project</th> </tr> </thead> <tbody> <tr> <td style="width: 80%;"> <p>If this field is enabled,</p> <p>the Depth Buffer Clear and Depth Buffer Resolve Enable fields must both be disabled.</p> <p>3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set.</p> <p>Refer to section 11.5.4.3 "Hierarchical Depth Buffer Resolve" for additional restrictions when this field is enabled.</p> <p>If Hierarchical Depth Buffer Enable is disabled, enabling this field will have no effect.</p> <p>Performance Note: expect the hierarchical depth buffer's impact on performance to be reduced for some period of time after this operation is performed, as the hierarchical depth buffer is initialized to a state that makes it ineffective. Further rendering will tend to bring the hierarchical depth buffer back to a more effective state.</p> <p>Software needs to do an ambiguate after allocating the surface for the first time if the depth buffer width and height are NOT aligned to 8 and 4 respectively.</p> </td> <td style="width: 20%;"></td> </tr> </tbody> </table>	Project:	All	Format:	Enable	Programming Notes	Project	<p>If this field is enabled,</p> <p>the Depth Buffer Clear and Depth Buffer Resolve Enable fields must both be disabled.</p> <p>3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set.</p> <p>Refer to section 11.5.4.3 "Hierarchical Depth Buffer Resolve" for additional restrictions when this field is enabled.</p> <p>If Hierarchical Depth Buffer Enable is disabled, enabling this field will have no effect.</p> <p>Performance Note: expect the hierarchical depth buffer's impact on performance to be reduced for some period of time after this operation is performed, as the hierarchical depth buffer is initialized to a state that makes it ineffective. Further rendering will tend to bring the hierarchical depth buffer back to a more effective state.</p> <p>Software needs to do an ambiguate after allocating the surface for the first time if the depth buffer width and height are NOT aligned to 8 and 4 respectively.</p>	
Project:	All								
Format:	Enable								
Programming Notes	Project								
<p>If this field is enabled,</p> <p>the Depth Buffer Clear and Depth Buffer Resolve Enable fields must both be disabled.</p> <p>3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set.</p> <p>Refer to section 11.5.4.3 "Hierarchical Depth Buffer Resolve" for additional restrictions when this field is enabled.</p> <p>If Hierarchical Depth Buffer Enable is disabled, enabling this field will have no effect.</p> <p>Performance Note: expect the hierarchical depth buffer's impact on performance to be reduced for some period of time after this operation is performed, as the hierarchical depth buffer is initialized to a state that makes it ineffective. Further rendering will tend to bring the hierarchical depth buffer back to a more effective state.</p> <p>Software needs to do an ambiguate after allocating the surface for the first time if the depth buffer width and height are NOT aligned to 8 and 4 respectively.</p>									
26	<p>Legacy Diamond Line Rasterization</p> <table border="1" style="width: 100%;"> <tr> <td style="width: 80%;">Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> </table> <p>This bit, if ENABLED, indicates that the Windower will rasterize zero width lines using the DX9 rasterization rules. If DISABLED, the Windower will rasterize zero width lines using the DX10 rasterization rules (see Strips Fans chapter).</p>	Project:	All	Format:	Enable				
Project:	All								
Format:	Enable								
25	<p>Pixel Shader Kill Pixel</p> <table border="1" style="width: 100%;"> <tr> <td style="width: 80%;">Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> </table> <p>This bit, if ENABLED, indicates that the PS kernel or color calculator has the ability to kill (discard) pixels or samples, other than due to depth or stencil testing. This bit is required to be ENABLED in the following situations:</p> <p>The API pixel shader program contains "killpix" or "discard" instructions, or other code in the pixel shader kernel that can cause the final pixel mask to differ from the pixel mask received on dispatch.</p> <p>A sampler with chroma key enabled with kill pixel mode is used by the pixel shader.</p>	Project:	All	Format:	Enable				
Project:	All								
Format:	Enable								

3DSTATE_WM

Any render target has **Alpha Test Enable** or **AlphaToCoverage Enable** enabled.

The pixel shader kernel generates and outputs oMask.

Note: As ClipDistance clipping is fully supported in hardware and therefore not via PS instructions, there should be no need to ENABLE this bit due to ClipDistance clipping.

24:23 Pixel Shader Computed Depth Mode

Format: U2 Enumerated Type

This field specifies the computed depth mode for the pixel shader.

Value	Name	Description	Project
0h	PSCDEPTH_OFF	Pixel shader does not compute depth	All
1h	PSCDEPTH_ON	Pixel shader computes depth with no guarantee as to its value	All
2h	PSCDEPTH_ON_GE	Pixel shader computes depth and guarantees that oDepth >= SourceDepth	All
3h	PSCDEPTH_ON_LE	Pixel shader computes depth and guarantees that oDepth <= SourceDepth	All

Programming Notes

When bit 5 is set in WM_STATE (i.e. RT independent rasterization is enabled), this field can not be programmed to values: 2h or 3h.

22:21 Early Depth/Stencil Control

Format: U2 Enumerated Type

This field specifies the behavior of early depth/stencil test.

Value	Name	Description	Project
0h	EDSC_NORMAL	Depth/Stencil Test/Write behaves as if it happens post-shader, however the pixel shader is not necessarily executed if the pixel fails depth or stencil test (this is the legacy behavior)	All
1h	EDSC_PSEXEC	Depth/Stencil Test/Write behaves as if it happens post-shader, and the pixel shader is executed if the pixel fails depth or stencil test (although pre-shader actions such as primitive inclusion, stipple, etc. will still cause the shader not to execute)	All
2h	EDSC_PREPS	Depth/Stencil Test/Write behaves as if it happens pre-shader. The pixel shader is not executed if the pixel fails depth or stencil test. Depth and stencil writes occur even if the pixel is killed by the shader or post-shader by alpha test, etc. Depth output by the pixel shader is ignored.	All
3h	Reserved		All



3DSTATE_WM			
Programming Notes			
If EDSC_PSEXEC mode is selected, Thread Dispatch Enable must be set.			
Errata			
	Description		Project
	When value of "2h" is programmed, PS_INVOCATIONS_COUNT may not be accurate.		
20	Pixel Shader Uses Source Depth		
	Project:	All	
	Format:	Enable	
	This bit, if ENABLED, indicates that the PS kernel requires the source depth value (vPos.z) to be passed in the payload. The source depth value is interpolated according to the Position ZW Interpolation Mode state.		
19	Pixel Shader Uses Source W		
	Project:	All	
	Format:	Enable	
	This bit, if ENABLED, indicates that the PS kernel requires the interpolated source W value (vPos.w) to be passed in the payload. The W value is interpolated according to the Position ZW Interpolation Mode state.		
18:17	Position ZW Interpolation Mode		
	Project:	All	
	Format:	U2 Enumerated Type	
	This field elects "interpolation mode" associated with the Position Z (source depth) and W coordinates passed in the PS payload when the PS requires Position as input. This field does not determine whether these coordinates are actually included in the payload (see Pixel Shader Requires Depth, Pixel Shader Requires W).		
	Value	Name	Description
	0h	INTERP_PIXEL	Evaluate Z & W at the pixel center or UL corner (as specified by Pixel Location of 3DSTATE_MULTISAMPLE)
	1h	Reserved	
	2h	INTERP_CENTROID	
	3h	INTERP_SAMPLE	
			Project
			All
			All
			All
			All
Programming Notes			
When bit 5 is set in WM_STATE, value of 3h is not defined for this field. Programming Note: When bit 5 in dword 1 (RT Independent Rasterization Enable) is set and bit 30 in dword 2 (PS UAV-only) is not set in WM_STATE, value of 3h is not defined for this field.			
16:11	Barycentric Interpolation Mode		
	Project:	All	
	Format:	Enable[6]	
	Controls which barycentric interpolation terms must be passed into the pixel shader kernel. Bit 0: Perspective Pixel Location barycentric is required Bit 1: Perspective Centroid barycentric is required Bit 2: Perspective Sample barycentric is required		



3DSTATE_WM			
	Bit 3: Non-perspective Pixel Location barycentric is required Bit 4: Non-perspective Centroid barycentric is required Bit 5: Non-perspective Sample barycentric is required		
	Programming Notes		
	If contiguous dispatch modes are enabled, only bit 3 (non-perspective pixel location) can be set, all other bits in this field must be zero. Pixel Location below refers to either the upper left corner or pixel center depending on the Pixel Location state of 3DSTATE_MULTISAMPLING). MSDISPMODE_PERSAMPLE is required in order to select Perspective Sample or Non-perspective Sample barycentric coordinates. Errata: When Centroid Barycentric mode is required, HW may produce incorrect interpolation results when a 2X2 pixels have unlit pixels.		
10	Pixel Shader Uses Input Coverage Mask		
	Project:	All	
	Format:	Enable	
	This bit, if ENABLED, indicates that the PS kernel requires the input coverage mask to be passed in the payload.		
9:8	Line End Cap Antialiasing Region Width		
	Project:	All	
	Format:	U2	
	This field specifies the distances over which the coverage of anti-aliased line end caps are computed.		
	Value	Name	Description
	0h		0.5 pixels
	1h		1.0 pixels
	2h		2.0 pixels
	3h		4.0 pixels
7:6	Line Antialiasing Region Width		
	Project:	All	
	Format:	U2	
	This field specifies the distance over which the anti-aliased line coverage is computed.		
	Value	Name	Description
	0h		0.5 pixels
	1h		1.0 pixels
	2h		2.0 pixels
	3h		4.0 pixels
5	Reserved		
	Format:	MBZ	
4	Polygon Stipple Enable		
	Project:	All	
	Format:	Enable	
	Enables the Polygon Stipple function.		
3	Line Stipple Enable		
	Project:	All	
	Format:	Enable	
	Enables the Line Stipple function.		



3DSTATE_WM				
2	Point Rasterization Rule			
	Project:	All		
	Format:	3D_RasterizationRule		
	This field specifies the rasterization rules to be applied whenever the edges of a point primitive fall exactly on a pixel sampling point.			
	Value	Name	Description	
	0h	RASTRULE_UPPER_LEFT	To match "normal" upper left rules for surface primitives	
	1h	RASTRULE_UPPER_RIGHT	To match OpenGL point rasterization rules (round to + infinity, where this is the upper right direction wrt OpenGL screen origin of lower left).	
			Project	
			All	
			All	
1:0	Multisample Rasterization Mode			
	Project:	All		
	Format:	U2 enumerated type		
	This field determines whether multisample rasterization is turned on/off, and how the pixel sample point(s) are defined. Software sets this according to the API, the API's multisample enable state setting (if any), and whether 1X or 4X MSRTs are bound. This state is duplicated in 3DSTATE_SF and both must be set to the same value. Refer to the "Multisampling" section for details on the settings of this field.			
	Value	Name	Project	
	0h	MSRASTMODE_OFF_PIXEL	All	
	1h	MSRASTMODE_OFF_PATTERN	All	
2h	MSRASTMODE_ON_PIXEL	All		
3h	MSRASTMODE_ON_PATTERN	All		
2	31	Multisample Dispatch Mode		
		Project:	All	
		Format:	U1 Enumerated Type	
		This bit, along with Number of Multisamples , determines how PS threads are dispatched. Software programs this bit depending on the per-pixel v.s per-sample PS execution requirement. When RT Independent Rasterization Enable = 1 , value of 0h for this field is not allowed.		
	Value	Name	Description	Project
	0h	MSDISPMODE_PERSAMPLE	This is the high-quality DX10.1 multisample mode where (over and above PERPIXEL mode) the PS is run for each covered sample. This mode is also used for "normal" non-multisample rendering (aka 1X), given Number of Multisamples is programmed to NUMSAMPLES_1.	All
	1h	MSDISPMODE_PERPIXEL	This is the classic multisample mode of operation, typically used for both antialiasing and transparency. Setup and rasterization operate in full multisample mode, testing coverage and depth/stencil test at the sample level but only running the PS once per pixel.	All
	30:0	Reserved		
		Format:	MBZ	



11.2.2 3DSTATE_PS

This command is used to set state used by the pixel shader dispatch stage.

3DSTATE_PS				
Source:		RenderCS		
Length Bias:		2		
DWord	Bit	Description		
0	31:29	Command Type		
		Default Value:	3h GFXPIPE	
		Format:	OpCode	
	28:27	Command SubType		
		Default Value:	3h GFXPIPE_3D	
		Format:	OpCode	
	26:24	3D Command Opcode		
		Default Value:	0h 3DSTATE_PIPELINED	
		Format:	OpCode	
	23:16	3D Command Sub Opcode		
	Default Value:	20h 3DSTATE_PS		
	Format:	OpCode		
15:8	Reserved			
	Project:	All		
	Format:	MBZ		
7:0	DWord Length			
	Default Value:	06h Excludes DWord (0,1)		
	Project:	All		
	Format:	=n		
	Total Length - 2			
1	31:6	Kernel Start Pointer[0]		
		Project:	All	
		Format:	InstructionBaseOffset[31:6]Kernel	
	Specifies the 64-byte aligned address offset of the first instruction in the kernel[0]. This pointer is relative to the Instruction Base Address.			
5:0	Reserved			
	Project:	All		
	Format:	MBZ		
2	31	Single Program Flow (SPF)		
		Project:	All	
	Specifies the initial condition of the kernel program as either a single program flow (SIMDn _{xm} with m = 1) or as multiple program flows (SIMDn _{xm} with m > 1). See CR0 description in ISA Execution Environment.			
		Value	Name	Description
		Project		
	0h	Multiple	Multiple Program Flows	All
	1h	Single	Single Program Flows	All



3DSTATE_PS				
30	Vector Mask Enable (VME)			
	Project:	All		
	Format:	U1 Enumerated Type		
	When SPF=0, VME specifies which mask to use to initialize the initial channel enables. When SPF=1, VME specifies which mask to use to generate execution channel enables.			
	Value	Name	Description	Project
0h	Dmask	Channels are enabled based on the dispatch mask	All	
1h	Vmask	Channels are enabled based on the vector mask	All	
29:27	Sampler Count			
	Project:	All		
	Format:	U3		
	Specifies how many samplers (in multiples of 4) the pixel shader 0 kernel uses. Used only for prefetching the associated sampler state entries.			
	Value	Name	Description	Project
	[0,4]			
	0h		no samplers used	All
	1h		between 1 and 4 samplers used	All
	2h		between 5 and 8 samplers used	All
	3h		between 9 and 12 samplers used	All
	4h		between 13 and 16 samplers used	All
5h-7h		Reserved	All	
26	Denormal Mode			
	Project:	All		
	Specifies the denormal mode used by the dispatched thread.			
	Value	Name	Description	Project
0h	FTZ	Denormals are flushed to zero	All	
1h	RET	Denormals are retained	All	
25:18	Binding Table Entry Count			
	Project:	All		
	Format:	U8		
	Specifies how many binding table entries the kernel uses. Used only for prefetching of the binding table entries and associated surface state. Note: For kernels using a large number of binding table entries, it may be advantageous to set this field to zero to avoid prefetching too many entries and thrashing the state cache.			
	This field is ignored if VS Function Enable is DISABLED.			
	Value	Name		
	[0,255]			
	Programming Notes			
	When HW binding table bit is set, it is assumed that the Binding Table Entry Count field will be generated at JIT time.			
	17	Reserved		
Format:		MBZ		
16	Floating Point Mode			



3DSTATE_PS			
		Project:	All
		Specifies the floating point mode used by the dispatched thread.	
	Value	Name	Description
	0h	IEEE-754	Use IEEE-754 rules
	1h	Alt	Use alternate rules
15:14	Rounding Mode		
		Project:	All
		Specifies the rounding mode used by the dispatched thread.	
	Value	Name	Description
	0h	RTNE	Round to Nearest Even
	1h	RU	Round toward +infinity
	2h	RD	Round toward -infinity
	3h	RTZ	Round toward zero
13	Illegal Opcode Exception Enable		
		Project:	All
		Format:	Enable
		This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment.	
12	Reserved		
		Project:	All
		Format:	MBZ
11	MaskStack Exception Enable		
		Project:	All
		Format:	Enable
		This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment.	
10:8	Reserved		
		Project:	All
		Format:	MBZ
7	Software Exception Enable		
		Project:	All
		Format:	Enable
		This bit gets loaded into EU CR0.1[13] (note the bit # difference). See Exceptions and ISA Execution Environment.	
6:0	Reserved		
		Project:	All
		Format:	MBZ
3	31:10	Scratch Space Base Pointer	
		Project:	All
		Format:	GeneralStateOffset[31:10]ScratchSpace
		Specifies the 1k-byte aligned address offset to scratch space for use by the kernel. This pointer is relative to the General State Base Address .	
	9:4	Reserved	
		Project:	All



3DSTATE_PS			
		Format:	MBZ
3:0	Per Thread Scratch Space		
	Project:	All	
	Format:	U4	
	Specifies the amount of scratch space allowed to be used by each thread. The driver must allocate enough contiguous scratch space, pointed to by the Scratch Space Pointer, to ensure that the Maximum Number of Threads each get Per Thread Scratch Space size without exceeding the driver-allocated scratch space.		
	Value	Name	
	[0,11]	indicating [1k bytes, 2M bytes] in powers of two	
4	31:24	Maximum Number of Threads	
	Format:	U8-1 representing thread count	
	Description		Project
	Range: WIZ Hashing Disable in GT_MODE register enabled: Range = [7,171] --> [8,172] threads. Only odd values are allowed (resulting in even max number of threads) WIZ Hashing Disable in GT_MODE register disabled: Range = [3,85] --> [4,86] threads. Only odd values are allowed (resulting in even max number of threads)		
	Specifies the maximum number of simultaneous threads allowed to be active. Used to avoid using up the scratch space, or to avoid potential deadlock.		
	Value	Name	Description
	[3h,2fh]	Range	[4,48] threads
	Programming Notes		
	If this field is changed between 3DPRIMITIVE commands, a PIPE_CONTROL command with Stall at Pixel Scoreboard set is required to be issued. This field must have an odd value so that the max number of PS threads is even.		
23:12	Reserved		
	Format:	MBZ	
11	Push Constant Enable		
	Project:	All	
	Format:	Enable	
	This field must be enabled if the sum of the PS Constant Buffer [3:0] Read Length fields in 3DSTATE_CONSTANT_PS is nonzero, and must be disabled if the sum is zero.		
10	Attribute Enable		
	Project:	All	
	Format:	Enable	
	This field must be enabled if the Number of SF Output Attributes field in 3DSTATE_SBE is nonzero, and must be disabled if that field is zero.		



3DSTATE_PS				
9	oMask Present to RenderTarget			
	Project:	All		
	Format:	Enable		
	This bit is inserted in the PS payload header and made available to the DataPort (either via the message header or via header bypass) to indicate that oMask data (one or two phases) is included in Render Target Write messages. If present, the oMask data is used to mask off samples.			
8	Render Target Fast Clear Enable			
	Project:	All		
	Format:	Enable		
	This field is set to enable fast clear of the bound render targets. See “Render Target Fast Clear” for restrictions on enabling this field.			
7	Dual Source Blend Enable			
	Project:	All		
	Format:	Enable		
	This field is set if dual source blend is enabled. If this bit is disabled, the data port dual source message reverts to a single source message using source 0.			
6	Render Target Resolve Enable			
	Project:	All		
	Format:	Enable		
	This field is set to enable clear value resolve on non-multisampled render targets. See “Render Target Resolve” for restrictions on enabling this field.			
5	Reserved			
	Format:	MBZ		
4:3	Position XY Offset Select			
	Project:	All		
	Format:	U2 Enumerated Type		
	This field specifies if/what Position XY Offset values are passed in the PS payload. Note that these are per-slot (pixel/sample) offsets, and therefore separate from the subspan XY coordinates passed in R1.			
	Value	Name	Description	Project
	0h	POSOFFSET_NONE	No Position XY Offsets are included in the PS payload.	All
	1h	Reserved		All
	2h	POSOFFSET_CENTROID	Position XY Offsets will be passed in the PS payload, and these will reflect the Centroid position(s).	All
	3h	POSOFFSET_SAMPLE	Position XY Offsets will be passed in the PS payload, and these will reflect the multisample position(s).	All
	Programming Notes			
SW Recommendation: If the PS kernel needs the Position Offsets to compute a Position XY value, this field should match Position ZW Interpolation Mode to ensure a consistent position.xyzw computation				
If the PS kernel does not need the Position XY Offsets to compute a Position Value, then this field should be programmed to POSOFFSET_NONE, as the PS kernel should be using the various				



3DSTATE_PS																	
	<p>barycentric inputs to evaluate other-than-position attributes. However, this field can be used to pass Centroid or Sample offsets in the payload for special test modes (e.g., where barycentric coordinates are computed in the PS vs. being HW-generated and passed in the payload).</p> <p>MSDISPMODE_PERSAMPLE is required in order to select POSOFFSET_SAMPLE.</p>																
2	<p>32 Pixel Dispatch Enable</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> </table> <table border="1"> <thead> <tr> <th style="text-align: center;">Description</th> <th style="text-align: center;">Project</th> </tr> </thead> <tbody> <tr> <td>Enables the Windower to dispatch 8 subspans in one payload.</td> <td></td> </tr> <tr> <td>Note: in the table below, the Valid column indicates which products that combination is supported on. Combinations of dispatch enables not listed in the table are not available on any product. A: Valid B: Valid D: Valid, except when in non-1x PERSAMPLE mode. E: Valid, except when in PERSAMPLE mode with number of multisamples >= 8. F: Valid.</td> <td></td> </tr> <tr> <td>Each of the three KSP values are separately specified.</td> <td></td> </tr> <tr> <td>In addition, each kernel has a separately-specified GRF register count.</td> <td></td> </tr> <tr> <td>Variable Pixel Dispatch Section: Pixel Grouping (Dispatch size) control for valid pixel dispatch combinations.</td> <td></td> </tr> </tbody> </table>	Project:	All	Format:	Enable	Description	Project	Enables the Windower to dispatch 8 subspans in one payload.		Note: in the table below, the Valid column indicates which products that combination is supported on. Combinations of dispatch enables not listed in the table are not available on any product. A: Valid B: Valid D: Valid, except when in non-1x PERSAMPLE mode. E: Valid, except when in PERSAMPLE mode with number of multisamples >= 8. F: Valid.		Each of the three KSP values are separately specified.		In addition, each kernel has a separately-specified GRF register count.		Variable Pixel Dispatch Section: Pixel Grouping (Dispatch size) control for valid pixel dispatch combinations.	
Project:	All																
Format:	Enable																
Description	Project																
Enables the Windower to dispatch 8 subspans in one payload.																	
Note: in the table below, the Valid column indicates which products that combination is supported on. Combinations of dispatch enables not listed in the table are not available on any product. A: Valid B: Valid D: Valid, except when in non-1x PERSAMPLE mode. E: Valid, except when in PERSAMPLE mode with number of multisamples >= 8. F: Valid.																	
Each of the three KSP values are separately specified.																	
In addition, each kernel has a separately-specified GRF register count.																	
Variable Pixel Dispatch Section: Pixel Grouping (Dispatch size) control for valid pixel dispatch combinations.																	
1	<p>16 Pixel Dispatch Enable</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> </table> <table border="1"> <thead> <tr> <th style="text-align: center;">Description</th> <th style="text-align: center;">Project</th> </tr> </thead> <tbody> <tr> <td>Enables the Windower to dispatch 4 subspans in one payload.</td> <td></td> </tr> <tr> <td>Note: in the table below, the Valid column indicates which products that combination is supported on. Combinations of dispatch enables not listed in the table are not available on any product. A: Valid B: Valid D: Valid, except when in non-1x PERSAMPLE mode. E: Valid, except when in PERSAMPLE mode with number of multisamples >= 8. F: Valid.</td> <td></td> </tr> <tr> <td>Each of the three KSP values are separately specified.</td> <td></td> </tr> <tr> <td>In addition, each kernel has a separately-specified GRF register count.</td> <td></td> </tr> <tr> <td>Variable Pixel Dispatch Section: Pixel Grouping (Dispatch size) control for valid pixel dispatch combinations.</td> <td></td> </tr> </tbody> </table>	Project:	All	Format:	Enable	Description	Project	Enables the Windower to dispatch 4 subspans in one payload.		Note: in the table below, the Valid column indicates which products that combination is supported on. Combinations of dispatch enables not listed in the table are not available on any product. A: Valid B: Valid D: Valid, except when in non-1x PERSAMPLE mode. E: Valid, except when in PERSAMPLE mode with number of multisamples >= 8. F: Valid.		Each of the three KSP values are separately specified.		In addition, each kernel has a separately-specified GRF register count.		Variable Pixel Dispatch Section: Pixel Grouping (Dispatch size) control for valid pixel dispatch combinations.	
Project:	All																
Format:	Enable																
Description	Project																
Enables the Windower to dispatch 4 subspans in one payload.																	
Note: in the table below, the Valid column indicates which products that combination is supported on. Combinations of dispatch enables not listed in the table are not available on any product. A: Valid B: Valid D: Valid, except when in non-1x PERSAMPLE mode. E: Valid, except when in PERSAMPLE mode with number of multisamples >= 8. F: Valid.																	
Each of the three KSP values are separately specified.																	
In addition, each kernel has a separately-specified GRF register count.																	
Variable Pixel Dispatch Section: Pixel Grouping (Dispatch size) control for valid pixel dispatch combinations.																	
0	<p>8 Pixel Dispatch Enable</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> </table> <table border="1"> <thead> <tr> <th style="text-align: center;">Description</th> <th style="text-align: center;">Project</th> </tr> </thead> <tbody> <tr> <td>Enables the Windower to dispatch 2 subspans in one payload.</td> <td></td> </tr> </tbody> </table>	Project:	All	Format:	Enable	Description	Project	Enables the Windower to dispatch 2 subspans in one payload.									
Project:	All																
Format:	Enable																
Description	Project																
Enables the Windower to dispatch 2 subspans in one payload.																	



3DSTATE_PS					
	<p>Note: in the table below, the Valid column indicates which products that combination is supported on. Combinations of dispatch enables not listed in the table are not available on any product.</p> <p>A: Valid B: Valid D: Valid, except when in non-1x PERSAMPLE mode. E: Valid, except when in PERSAMPLE mode with number of multisamples >= 8. F: Valid..</p> <p>Each of the three KSP values are separately specified.</p> <p>In addition, each kernel has a separately-specified GRF register count.</p> <p>Variable Pixel Dispatch Section: Pixel Grouping (Dispatch size) control for valid pixel dispatch combinations.</p>				
5	31:23 Reserved Project: All Format: MBZ				
	22:16 Dispatch GRF Start Register for Constant/Setup Data [0] Project: All Format: U7 Specifies the starting GRF register number for the Constant/Setup portion of the thread payload for kernel[0].				
	<table border="1" style="width: 100%;"> <thead> <tr> <th style="width: 50%;">Value</th> <th style="width: 50%;">Name</th> </tr> </thead> <tbody> <tr> <td>[0,127]</td> <td></td> </tr> </tbody> </table>	Value	Name	[0,127]	
	Value	Name			
	[0,127]				
	15 Reserved Project: All Format: MBZ				
	14:8 Dispatch GRF Start Register for Constant/Setup Data [1] Project: All Format: U7 Specifies the starting GRF register number for the Constant/Setup portion of the thread payload for kernel[1].				
	<table border="1" style="width: 100%;"> <thead> <tr> <th style="width: 50%;">Value</th> <th style="width: 50%;">Name</th> </tr> </thead> <tbody> <tr> <td>[0,127]</td> <td></td> </tr> </tbody> </table>	Value	Name	[0,127]	
	Value	Name			
	[0,127]				
7 Reserved Project: All Format: MBZ					
6:0 Dispatch GRF Start Register for Constant/Setup Data [2] Project: All Format: U7 Specifies the starting GRF register number for the Constant/Setup portion of the thread payload for kernel[2].					
<table border="1" style="width: 100%;"> <thead> <tr> <th style="width: 50%;">Value</th> <th style="width: 50%;">Name</th> </tr> </thead> <tbody> <tr> <td>[0,127]</td> <td></td> </tr> </tbody> </table>	Value	Name	[0,127]		
Value	Name				
[0,127]					
6	31:6 Kernel Start Pointer[1] Project: All Format: InstructionBaseOffset[31:6]Kernel Specifies the 64-byte aligned address offset of the first instruction in kernel[1]. This pointer is relative to the Instruction Base Address.				



3DSTATE_PS		
	5:0	Reserved
		Project: All
		Format: MBZ
7	31:6	Kernel Start Pointer[2]
		Project: All
		Format: InstructionBaseOffset[31:6]Kernel
		Specifies the 64-byte aligned address offset of the first instruction in kernel[2]. This pointer is relative to the Instruction Base Address .
	5:0	Reserved
		Project: All
		Format: MBZ



11.2.3 3DSTATE_CONSTANT_PS

3DSTATE_CONSTANT_PS				
Source:		RenderCS		
Length Bias:		2		
This command sets pointers to the push constants for the PS unit. The constant data pointed to by this command is loaded into the PS unit's push constant buffer (PCB).				
Programming Notes			Project	
It is invalid to execute this command more than once between 3D_PRIMITIVE commands.				
Constant buffers must be enabled in order from Constant Buffer 0 to Constant Buffer 3 within this command. For example, it is not allowed to enable Constant Buffer 1 by programming a non-zero value in the PS Constant Buffer 1 Read Length without a non-zero value in PS Constant Buffer 0 Read Length.				
DWord	Bit	Description		
0	31:29	Command Type		
		Default Value:	3h GFXPIPE	
		Format:	OpCode	
	28:27	Command SubType		
		Default Value:	3h GFXPIPE_3D	
		Format:	OpCode	
	26:24	3D Command Opcode		
		Default Value:	0h 3DSTATE_PIPELINED	
		Format:	OpCode	
	23:16	3D Command Sub Opcode		
Default Value:		17h 3DSTATE_CONSTANT_PS		
Format:		OpCode		
15:8	Reserved			
	Format:	MBZ		
7:0	Dword Length			
	Project:	All		
	Format:	=n Total Length – 2		
		Value	Name	Project
		5h	Excludes DWord (0,1) [Default]	
1..6	191:0	Constant Body		
		Format:	3DSTATE_CONSTANT(Body)	
Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS				



11.2.4 3DSTATE_PUSH_CONSTANT_ALLOC_PS

3DSTATE_PUSH_CONSTANT_ALLOC_PS			
Source:	RenderCS		
Length Bias:	2		
Description			
This command sets up the URB configuration for PS Push Constant Buffer.			
A PIPE_CONTROL command with the CS Stall bit set must be programmed in the ring after this instruction.			
Programming Notes			
Restriction:			
<p>The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size.</p> <p>The sum of the constant length programmed in 3DSTATE_CONSTANT_PS must be equal or smaller than the size of the allocated space in the URB including the buffering for half cachelines. See Push Constant URB Allocation section for more details.</p> <p>The 3DSTATE_CONSTANT_PS must be reprogrammed prior to the next 3DPRIMITIVE command after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_PS.</p>			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	1h 3DSTATE_NONPIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		16h 3DSTATE_PUSH_CONSTANT_ALLOC_PS	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	Dword Length		
	Default Value:	0h Excludes Dword (0,1)	
	Project:	All	
	Format:	=n Total Length – 2	
1	31:20	Reserved	
		Format:	MBZ



3DSTATE_PUSH_CONSTANT_ALLOC_PS		
19:16	Constant Buffer Offset	
	Format: U5	
	Specifies the offset of the PS constant buffer into the URB.	
	Value	Name
	[0,15] 0h	(0KB - 15KB) 0KB [Default]
15:5	Reserved	
	Format: MBZ	
4:0	Constant Buffer Size	
	Format: U5	
	Specifies the size of the PS constant buffer. This value will determine the amount of data the command stream can pre-fetch before the buffer is full. Value of zero is only valid when constants are not enabled for PS.	
	Value	Name
	[0,15] 0h	(0KB – 15KB) Increments of 1KB 0KB [Default]

11.2.5 3DSTATE_SAMPLE_MASK

The sample mask state used by the windower stage is defined with this inline state packet.

3DSTATE_SAMPLE_MASK		
Source:	RenderCS	
Length Bias:	2	
DWord	Bit	Description
0	31:29	Command Type
		Default Value: 3h GFXPIPE Format: OpCode
	28:27	Command SubType
		Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	3D Command Opcode
Default Value: 0h 3DSTATE_PIPELINED Format: OpCode		
23:16	3D Command Sub Opcode	
	Default Value: 18h 3DSTATE_SAMPLE_MASK Format: OpCode	
15:8	Reserved	
	Project: All Format: MBZ	
7:0	Dword Length	

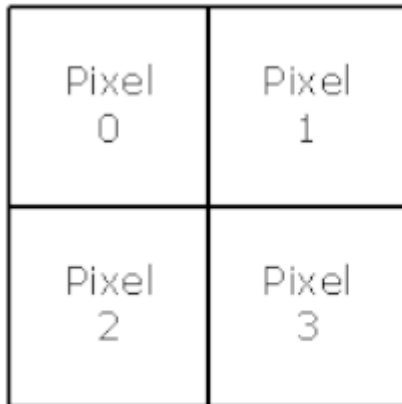
3DSTATE_SAMPLE_MASK																	
	<table border="1"> <tr> <td>Default Value:</td> <td>0h Excludes Dword (0,1)</td> </tr> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>=n Total Length – 2</td> </tr> </table>	Default Value:	0h Excludes Dword (0,1)	Project:	All	Format:	=n Total Length – 2										
Default Value:	0h Excludes Dword (0,1)																
Project:	All																
Format:	=n Total Length – 2																
1	<table border="1"> <tr> <td>31:8</td> <td>Reserved</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> <tr> <td>7:0</td> <td>Sample Mask</td> </tr> <tr> <td>Format:</td> <td>8 bit mask Right-justified bitmask (Bit 0 = Sample0). Number of bits that are used is determined by Num Multisamples (3DSTATE_MULTISAMPLE)</td> </tr> <tr> <td colspan="2"> <p>A per-multisample-position mask state variable that is immediately and unconditionally ANDed with the sample coverage mask as part of the rasterization process. This mask is applied prior to centroid selection.</p> </td> </tr> <tr> <td colspan="2" style="text-align: center;">Programming Notes</td> </tr> <tr> <td colspan="2"> <p>If Number of Multisamples is NUMSAMPLES_1, bits 7:1 of this field must be zero.</p> </td> </tr> <tr> <td colspan="2"> <p>If Number of Multisamples is NUMSAMPLES_4, bits 7:4 of this field must be zero.</p> </td> </tr> </table>	31:8	Reserved	Format:	MBZ	7:0	Sample Mask	Format:	8 bit mask Right-justified bitmask (Bit 0 = Sample0). Number of bits that are used is determined by Num Multisamples (3DSTATE_MULTISAMPLE)	<p>A per-multisample-position mask state variable that is immediately and unconditionally ANDed with the sample coverage mask as part of the rasterization process. This mask is applied prior to centroid selection.</p>		Programming Notes		<p>If Number of Multisamples is NUMSAMPLES_1, bits 7:1 of this field must be zero.</p>		<p>If Number of Multisamples is NUMSAMPLES_4, bits 7:4 of this field must be zero.</p>	
31:8	Reserved																
Format:	MBZ																
7:0	Sample Mask																
Format:	8 bit mask Right-justified bitmask (Bit 0 = Sample0). Number of bits that are used is determined by Num Multisamples (3DSTATE_MULTISAMPLE)																
<p>A per-multisample-position mask state variable that is immediately and unconditionally ANDed with the sample coverage mask as part of the rasterization process. This mask is applied prior to centroid selection.</p>																	
Programming Notes																	
<p>If Number of Multisamples is NUMSAMPLES_1, bits 7:1 of this field must be zero.</p>																	
<p>If Number of Multisamples is NUMSAMPLES_4, bits 7:4 of this field must be zero.</p>																	

11.3 Rasterization

The WM unit uses the setup computations performed by the SF unit to rasterize objects into the corresponding set of pixels. Most of the controls regarding the screen-space geometry of rendered objects are programmed via the SF unit.

The rasterization process generates pixels in 2x2 groups of pixels called *subspans* (see UNRESOLVED CROSS REFERENCE, Pixels with a SubSpan) which, after being subjected to various inclusion/discard tests, are grouped and passed to spawned Pixel Shader (PS) threads for subsequent processing. Once these PS threads are spawned, the WM unit provides only bookkeeping functions on the pixels. Note that the WM unit can proceed on to rasterize subsequent objects while PS threads from previous objects are still executing.

Pixels with a SubSpan



B6850-01

11.3.1 Drawing Rectangle Clipping

The Drawing Rectangle defines the maximum extent of pixels which can be rendered. Portions of objects falling outside the Drawing Rectangle will be clipped (pixels discarded). Implementations will typically discard objects falling completely outside of the Drawing Rectangle as early in the pipeline as possible. There is no control to turn off Drawing Rectangle clipping – it is unconditional.

For the purposes of clipping, the Drawing Rectangle must itself be clipped to the destination buffer extents (The Drawing Rectangle Origin, used to offset relative X,Y coordinates earlier in the pipeline, is permitted to lie offscreen). The **Clipped Drawing Rectangle X,Y Min,Max** state variables (programmed via `3DSTATE_DRAWING_RECTANGLE` – See *SF Unit*) defines the intersection of the Drawing Rectangle and the Color Buffer. It is specified with non-negative integer pixel coordinates relative to the Destination Buffer upper-left origin.

Pixels with coordinates *outside* of the Drawing Rectangle cannot be rendered (i.e., the rectangle is inclusive). For example, to render to a full-screen 1280x1024 buffer, the following values would be required: `Xmin=0`, `Ymin=0`, `Xmax=1279` and `Ymax=1023`.

For “full screen” rendering, the Drawing Rectangle coincides with the screen-sized buffer. For “front-buffer windowed” rendering it coincides with the destination “window”.

11.3.2 Line Rasterization

See *SF Unit* chapter for details on the screen-space geometry of the various line types.

11.3.2.1 Coverage Values for Anti-Aliased Lines

The WM unit is provided with both the **Line Anti-Aliasing Region Width** and **Line End Cap Anti-Aliasing Region Width** state variables (in `WM_STATE`) in order to compute the coverage values for anti-aliased lines.



11.3.2.2 3DSTATE_AA_LINE_PARAMS

3DSTATE_AA_LINE_PARAMETERS			
Source:	RenderCS		
Length Bias:	2		
The 3DSTATE_AA_LINE_PARAMS command is used to specify the slope and bias terms used in the improved alpha coverage computation (specifically for DX WHQL compliance). Note that in these devices the coverage values passed to PS threads are full U0.8 values.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	1h 3DSTATE_NONPIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		0Ah 3DSTATE_AA_LINE_PARAMS	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	Dword Length		
	Default Value:	1h Excludes Dword (0,1)	
	Project:	All	
	Format:	=n Total Length – 2	
1	31:24	Reserved	
		Format:	MBZ
	23:16	AA Coverage Bias	
		Project:	All
		Format:	U0.8
	This field specifies the bias term to be used in the aa coverage computation for edges 0 and 3.		
	15:8	Reserved	
		Format:	MBZ
	7:0	AA Coverage Slope	
		Project:	All
Format:		U0.8	
This field specifies the slope term to be used in the aa coverage computation for edges 0 and 3. If this field is zero, the Windower will revert to legacy aa line coverage computation (though still output expanded U0.8 coverage values).			



3DSTATE_AA_LINE_PARAMETERS		
2	31:24	Reserved
		Format: MBZ
	23:16	AA Coverage EndCap Bias
		Project: All
		Format: U0.8 This field specifies the bias term to be used in the aa coverage computation for edges 1 and 2.
	15:8	Reserved
		Format: MBZ
	7:0	AA Coverage EndCap Slope
		Project: All
		Format: U0.8 This field specifies the slope term to be used in the aa coverage computation for edges 1 and 2.

The slope and bias values should be computed to closely match the reference rasterizer results Based on empirical data, the following recommendations are offered:

The final alpha for the center of the line needs to be 148 to match the reference rasterizer In this case, the Lo to edge 0 and edge 3 will be the same Since the alpha for each edge is multiplied together, we get:

$$\text{edge0alpha} * \text{edge1alpha} = 148/255 = 0.580392157$$

Since $\text{edge0alpha} = \text{edge3alpha}$ we get:

$$(\text{edge0alpha})^2 = 0.580392157$$

$$\text{edge0alpha} = \sqrt{0.580392157} = 0.761834731 \text{ at the center pixel}$$

$$\text{The desired alpha for pixel 1} = 54/255 = 0.211764706$$

$$\text{The slope is } (0.761834731 - 0.211764706) = 0.550070025$$

Since we are using 8 bit precision, the slope becomes

$$\text{AA Coverage [EndCap] Slope} = 0.55078125$$

The alpha value for Lo = 0 (second pixel from center) determines the bias term and is equal to

$$(0.211764706 - 0.550070025) = -0.338305319$$

With 8 bits of precision the programmed bias value

$$\text{AA Coverage [EndCap] Bias} = 0.33984375$$

11.3.2.3 Line Stipple

Line stipple, controlled via the **Line Stipple Enable** state variable in WM_STATE, discards certain pixels that are produced by non-AA line rasterization.

The line stipple rule is specified via the following state variables programmed via 3DSTATE_LINE_STIPPLE: the 16-bit **Line Stipple Pattern** (p), **Line Stipple Repeat Count** l, and **Line**



Stipple Inverse Repeat Count. Software must compute **Line Stipple Inverse Repeat Count** as $1.0f / \text{Line Stipple Repeat Count}$ and then converted from float to the required fixed point encoding (see 3DSTATE_LINE_STIPPLE).

The WM unit maintains an internal Line Stipple Counter state variable (s) The initial value of s is zero; s is incremented after production of each pixel of a line segment (pixels are produced in order, beginning at the starting point and working towards the ending point). S is reset to 0 whenever a new primitive is processed (unless the primitive type is LINESTRIP_CONT or LINESTRIP_CONT_BF), and before every line segment in a group of independent segments (LINELIST primitive).

During the rasterization of lines, the WM unit computes:

$$b = \lfloor s/r \rfloor \bmod 16,$$

A pixel is rendered if the bth bit of p is 1, otherwise it is discarded. The bits of p are numbered with 0 being the least significant and 15 being the most significant.

11.3.2.4 3DSTATE_LINE_STIPPLE

3DSTATE_LINE_STIPPLE			
Source:	RenderCS		
Length Bias:	2		
The 3DSTATE_LINE_STIPPLE command is used to specify state variables used in the Line Stipple function.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	1h 3DSTATE_NONPIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
Default Value:		08h 3DSTATE_LINE_STIPPLE	
Format:		OpCode	
15:8	Reserved		
	Project:	All	
	Format:	MBZ	
7:0	Dword Length		
	Default Value:	1h Excludes Dword (0,1)	
	Project:	All	
	Format:	=n Total Length – 2	
1	31	Modify Enable (Current Repeat Counter, Current Stipple Index)	
		Project:	All
		Format:	Enable
		Modify enable for Current Repeat Counter and Current Stipple Index fields.	
		Programming Notes	



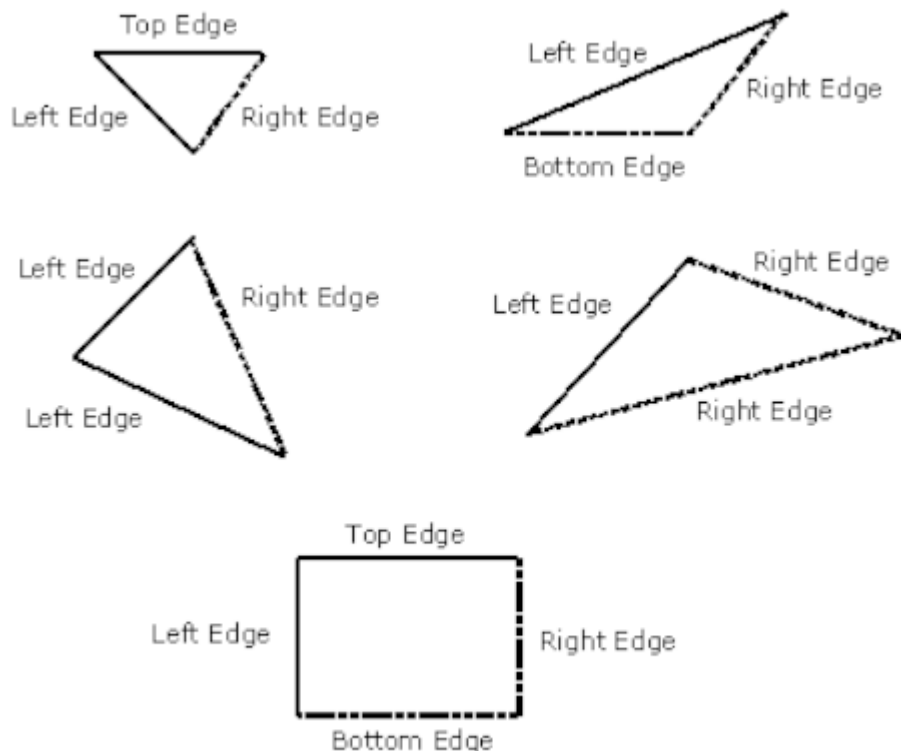
3DSTATE_LINE_STIPPLE	
	Software should never set this field to enabled. It is provided only for HW-generated commands as part of context save/restore.
30	Reserved
	Project: All Format: MBZ
29:21	Current Repeat Counter
	Project: All Format: U9 This field sets the HW-internal repeat counter state. Note: Software should never attempt to set this value – this state is only provided for HW-generated commands as part of context save/restore.
20	Reserved
	Project: All Format: MBZ
19:16	Current Stipple Index
	Project: All Format: U4 This field sets the HW-internal stipple pattern index. Note: Software should never attempt to set this value – this state is only provided for HW-generated commands as part of context save/restore.
15:0	Line Stipple Pattern
	Project: All Format: 16 bit mask Bit 15 = most significant bit, Bit 0 = least significant bit Specifies a pattern used to mask out bit specific pixels while rendering lines.
2	31:15 Line Stipple Inverse Repeat Count
	Project: All Format: U1.16 Range: [0.00390625, 1.0] Specifies the inverse (truncated) of the repeat count for the line stipple function.
14:9	Reserved
	Project: All Format: MBZ
8:0	Line Stipple Repeat Count
	Project: All Format: U9 Specifies the repeat count for the line stipple function.
	Value
	Name
	[1, 256]

11.3.3 Polygon (Triangle and Rectangle) Rasterization

The rasterization of LINE, TRIANGLE, and RECTANGLE objects into pixels requires a “pixel sampling grid” to be defined. This grid is defined as an axis-aligned array of pixel sample points spaced exactly 1 pixel unit apart. If a sample point falls within one of these objects, the pixel associated with the sample point is considered “inside” the object, and information for that pixel is generated and passed down the pipeline.

For TRIANGLE and RECTANGLE objects, if a sample point intersects an edge of the object, the associated pixel is considered “inside” the object if the intersecting edge is a “left” or “top” edge (or, more exactly, the intersected edge is not a “right” or “bottom” edge). Note that “top” and “bottom” edges are by definition exactly horizontal. See UNRESOLVED CROSS REFERENCE, TRIANGLE and RECTANGLE Edge Types, for the edge types for representative TRIANGLE and RECTANGLE objects (solid edges are inclusive, dashed edges are exclusive).

TRIANGLE and RECTANGLE Edge Types



B-6851-01



11.3.3.1 Polygon Stipple

The *Polygon Stipple* function, controlled via the **Polygon Stipple Enable** state variable in WM_STATE, allows only selected pixels of a repeated 32x32 pixel pattern to be rendered Polygon stipple is applied only to the following primitive types:

3DPRIM_POLYGON
3DPRIM_TRIFAN
3DPRIM_TRILIST
3DPRIM_TRISTRIP
3DPRIM_TRISTRIP_REVERSE

Note that the 3DPRIM_TRIFAN_NOSTIPPLE object is never subject to polygon stipple.

The stipple pattern is defined as a 32x32 bit pixel mask via the 3DSTATE_POLY_STIPPLE_PATTERN command. This is a non-pipelined command which incurs an implicit pipeline flush when executed.

The origin of the pattern is specified via **Polygon Stipple X,Y Offset** state variables programmed via the 3DSTATE_POLY_STIPPLE_OFFSET command The offsets are pixel offsets from the Color Buffer origin to the upper left corner of the stipple pattern. This is a non-pipelined command which incurs an implicit pipeline flush when executed.

11.3.3.2 3DSTATE_POLY_STIPPLE_OFFSET

3DSTATE_POLY_STIPPLE_OFFSET			
Project:	All		
Source:	RenderCS		
Length Bias:	2		
The 3DSTATE_POLY_STIPPLE_OFFSET command is used to specify the origin of the repeated screen-space Polygon Stipple Pattern as an X,Y offset from the Color Buffer origin.			
DWord	Bit	Description	
0	31:29	Command Type	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	Command SubType	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	3D Command Opcode	
		Default Value:	1h 3DSTATE_NONPIPELINED
		Format:	OpCode
	23:16	3D Command Sub Opcode	
		Default Value:	06h 3DSTATE_POLY_STIPPLE_OFFSET
		Format:	OpCode
	15:8	Reserved	
		Project:	All
		Format:	MBZ
7:0	Dword Length		



3DSTATE_POLY_STIPPLE_OFFSET			
		Default Value: 0h Excludes Dword (0,1) Project: All Format: =n Total Length – 2	
1	31:13	Reserved	
		Project: All	
		Format: MBZ	
	12:8	Polygon Stipple X Offset	
		Project: All	
		Format: U5	
		Specifies a 5 bit x address offset in the poly stipple pattern	
		Value	Name
		[0,31]	
	7:5	Reserved	
Project: All			
Format: MBZ			
4:0	Polygon Stipple Y Offset		
	Project: All		
	Format: U5		
	Specifies a 5 bit y address offset in the poly stipple pattern		
	Value	Name	
	[0,31]		

11.3.3.3 3DSTATE_POLY_STIPPLE_PATTERN

3DSTATE_POLY_STIPPLE_PATTERN		
Project:	All	
Source:	RenderCS	
Length Bias:	2	
The 3DSTATE_POLY_STIPPLE_PATTERN command is used to specify the 32x32 Polygon Stipple Pattern used in the Polygon Stipple function of the WM unit.		
DWord	Bit	Description
0	31:29	Command Type
		Default Value: 3h GFXPIPE
		Format: OpCode
	28:27	Command SubType
		Default Value: 3h GFXPIPE_3D
		Format: OpCode
	26:24	3D Command Opcode
		Default Value: 1h 3DSTATE_NONPIPELINED
		Format: OpCode
	23:16	3D Command Sub Opcode
		Default Value: 07h 3DSTATE_POLY_STIPPLE_PATTERN
		Format: OpCode



3DSTATE_POLY_STIPPLE_PATTERN		
	15:8	Reserved
		Project: All Format: MBZ
	7:0	Dword Length
		Default Value: 1Fh Excludes Dword (0,1) Project: All Format: =n Total Length – 2
1	31:0	Polygon Stipple Pattern Row 1 (top most) Project: All Format: 32 bit mask Bit 31 = upper left corner, Bit 0 = upper right corner of first row. Specifies a pattern used by Polygon Stipple to mask out specific pixels of every 32x32 area rendered.
2..32	31:0	Polygon Stipple Pattern Rows 2-32 (bottom most) Project: All Format: 32 bit mask Bit 31 = upper left corner, Bit 0 = upper right corner of first row. Specifies a pattern used by Polygon Stipple to mask out specific pixels of every 32x32 area rendered.

11.4 Multisampling

The multisampling function has two components:

- **Multisample Rasterization:** multisample rasterization occurs at a subpixel level, wherein each pixel consists of a number of “samples” at state-defined positions within the pixel footprint Coverage of the primitive as well as color calculator operations (stencil test, depth test, color buffer blending, etc.) are done at the sample level In addition the pixel shader itself can optionally run at the sample level depending on a separate state field.
- **Multisample Render Targets (MSRT):** The render targets, as well as the depth and stencil buffers, now have the ability to store per-sample values When combined with multisample rasterization, color calculator operations such as stencil test, depth test, and color buffer blending are done with the destination surface containing potentially different values per sample.

11.4.1 Multisample Modes/State

A number of state variables control the operation of the multisampling function. The following list indicates the state and their location. Refer to the state definition for more details.

- **Multisample Rasterization Mode (3DSTATE_SF and 3DSTATE_WM):** controls whether rasterization of non-lines is performed on a pixel or sample basis (PIXEL vs. PATTERN), and whether multisample rasterization of lines enabled (OFF vs. ON).



- **Multisample Dispatch Mode** (3DSTATE_WM): controls whether the pixel shader is executed per pixel or per sample.
- **Number of Multisamples** (3DSTATE_MULTISAMPLE and SURFACE_STATE): indicates the number of samples per pixel contained on the surface. This field in 3DSTATE_MULTISAMPLE must match the corresponding field in SURFACE_STATE for each render target. The depth, hierarchical depth, and stencil buffers inherit this field from 3DSTATE_MULTISAMPLE.
- **Pixel Location** (3DSTATE_MULTISAMPLE): indicates the subpixel location where values specified as “pixel” are sampled. This is either the upper left corner or the center.
- **MSAA Sample Offsets** (3DSTATE_MULTISAMPLE]): for each of the N samples, specifies the subpixel location of each sample.

11.4.2 3DSTATE_MULTISAMPLE

3DSTATE_MULTISAMPLE		
Source:	RenderCS	
Length Bias:	2	
<p>The 3DSTATE_MULTISAMPLE command is used to specify multisample state associated with the current render target/depth buffer. This is non-pipelined state.</p> <p>Programming Restriction: Driver must hierarchy that all the caches in the depth pipe are flushed before this command is parsed. This requires driver to send a PIPE_CONTROL with a CS stall along with a Depth Flush prior to this command.</p> <p>When this command is issued, the currently active depth buffer, hierarchical depth buffer, stencil buffer, and render target(s) must be cleared (meaning that every pixel must be overwritten). Alternatively, other surfaces can be activated before issuing the next 3DPRIMITIVE that were previously rendered with the same values of all state fields in this command. In other words, it is illegal to render to these surfaces with multiple different values of the state fields in this command.</p>		
Programming Notes		
<p>When programming the sample offsets (for NUMSAMPLES_4 or _8 and MSRASTMODE_xxx_PATTERN), the order of the samples 0 to 3 (or 7 for 8X) must have monotonically increasing distance from the pixel center. This is required to get the correct centroid computation in the device.</p>		
DWord	Bit	Description
0	31:29	Command Type
		Default Value: 3h GFXPIPE Format: OpCode
	28:27	Command SubType
		Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	3D Command Opcode
		Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	3D Command Sub Opcode
		Default Value: 0Dh 3DSTATE_MULTISAMPLE Format: OpCode
	15:8	Reserved
		Project: All



3DSTATE_MULTISAMPLE				
		Format:	MBZ	
	7:0	Dword Length		
		Project:	All	
		Format:	=n Total Length – 2	
		Excludes Dword (0,1)		
		Value	Name	
		2h	[Default]	
1	31:6	Reserved		
		Project:	All	
		Format:	MBZ	
	5	Reserved		
		Format:	MBZ	
	4	Pixel Location		
		Project:	All	
		Format:	U1	
		This field specifies where the device evaluates “pixel” (vs. centroid or sample) values/attributes.		
		Value	Name	Description
0h	PIXLOC_CENTER	Use the pixel center (0.5, 0.5 offset)	All	
1h	PIXLOC_UL_CORNER	Use the pixel upper-left corner	All	
	Programming Notes			
	The programming of this field is assumed to be a function of the API being supported. Specifically, it is expected that OpenGL and DX10+ APIs require CENTER selection, while DX9- APIs require UL_CORNER selection.			
3:1	Number of Multisamples			
	Project:	All		
	Format:	U3 enumerated value		
	This field specifies how many samples/pixel exist in all RTs and the Depth Buffer, as log2(#samples). This field is valid regardless of the setting of Multisample Rasterization Mode .			
	Value	Name	Description	Project
	0h	NUMSAMPLES_1	1 sample/pixel	All
	1h	Reserved		All
	2h	NUMSAMPLES_4	4 samples/pixel	All
	3h	NUMSAMPLES_8	8 samples/pixel	All
	[4h,7h]	Reserved		All
	Programming Notes			
	Setting Multisample Rasterization Mode to MSRASTMODE_xxx_PATTERN when Number of Multisamples == NUMSAMPLES_1 is UNDEFINED.			
	The setting of this field must match the Number of Multisamples field in SURFACE_STATE of all			



3DSTATE_MULTISAMPLE									
	bound render targets.								
0	Reserved Project: All Format: MBZ								
2	31:28 Sample3 X Offset Project: All Format: U0.4 <table border="1"> <thead> <tr> <th>Description</th> <th>Project</th> </tr> </thead> <tbody> <tr> <td>Subpixel X offset of Sample 3 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1</td> <td></td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </tbody> </table>	Description	Project	Subpixel X offset of Sample 3 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1		Value	Name	[0,15]	[0,0.9375]
Description	Project								
Subpixel X offset of Sample 3 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1									
Value	Name								
[0,15]	[0,0.9375]								
	27:24 Sample3 Y Offset Project: All Format: U0.4 <table border="1"> <thead> <tr> <th>Description</th> <th>Project</th> </tr> </thead> <tbody> <tr> <td>Subpixel Y offset of Sample 3 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1</td> <td></td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </tbody> </table>	Description	Project	Subpixel Y offset of Sample 3 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1		Value	Name	[0,15]	[0,0.9375]
Description	Project								
Subpixel Y offset of Sample 3 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1									
Value	Name								
[0,15]	[0,0.9375]								
	23:20 Sample2 X Offset Project: All Format: U0.4 <table border="1"> <thead> <tr> <th>Description</th> <th>Project</th> </tr> </thead> <tbody> <tr> <td>Subpixel X offset of Sample 2 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1</td> <td></td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </tbody> </table>	Description	Project	Subpixel X offset of Sample 2 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1		Value	Name	[0,15]	[0,0.9375]
Description	Project								
Subpixel X offset of Sample 2 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1									
Value	Name								
[0,15]	[0,0.9375]								
	19:16 Sample2 Y Offset Project: All Format: U0.4								



3DSTATE_MULTISAMPLE																		
		<table border="1"> <thead> <tr> <th>Description</th> <th>Project</th> </tr> </thead> <tbody> <tr> <td>Subpixel Y offset of Sample <u>2</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1</td> <td></td> </tr> <tr> <td></td> <td></td> </tr> <tr> <th>Value</th> <th>Name</th> </tr> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </tbody> </table>	Description	Project	Subpixel Y offset of Sample <u>2</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1				Value	Name	[0,15]	[0,0.9375]						
Description	Project																	
Subpixel Y offset of Sample <u>2</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1																		
Value	Name																	
[0,15]	[0,0.9375]																	
15:12	Sample1 X Offset	<table border="1"> <tbody> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U0.4</td> </tr> <tr> <td></td> <td></td> </tr> <tr> <th>Description</th> <th>Project</th> </tr> <tr> <td>Subpixel X offset of Sample <u>1</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1</td> <td></td> </tr> <tr> <td></td> <td></td> </tr> <tr> <th>Value</th> <th>Name</th> </tr> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </tbody> </table>	Project:	All	Format:	U0.4			Description	Project	Subpixel X offset of Sample <u>1</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1				Value	Name	[0,15]	[0,0.9375]
Project:	All																	
Format:	U0.4																	
Description	Project																	
Subpixel X offset of Sample <u>1</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1																		
Value	Name																	
[0,15]	[0,0.9375]																	
11:8	Sample1 Y Offset	<table border="1"> <tbody> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U0.4</td> </tr> <tr> <td></td> <td></td> </tr> <tr> <th>Description</th> <th>Project</th> </tr> <tr> <td>Subpixel Y offset of Sample <u>1</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1</td> <td></td> </tr> <tr> <td></td> <td></td> </tr> <tr> <th>Value</th> <th>Name</th> </tr> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </tbody> </table>	Project:	All	Format:	U0.4			Description	Project	Subpixel Y offset of Sample <u>1</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1				Value	Name	[0,15]	[0,0.9375]
Project:	All																	
Format:	U0.4																	
Description	Project																	
Subpixel Y offset of Sample <u>1</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1																		
Value	Name																	
[0,15]	[0,0.9375]																	
7:4	Sample0 X Offset	<table border="1"> <tbody> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U0.4</td> </tr> <tr> <td></td> <td></td> </tr> <tr> <th>Description</th> <th>Project</th> </tr> <tr> <td>Subpixel X offset of Sample <u>0</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1</td> <td></td> </tr> <tr> <td></td> <td></td> </tr> <tr> <th>Value</th> <th>Name</th> </tr> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </tbody> </table>	Project:	All	Format:	U0.4			Description	Project	Subpixel X offset of Sample <u>0</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1				Value	Name	[0,15]	[0,0.9375]
Project:	All																	
Format:	U0.4																	
Description	Project																	
Subpixel X offset of Sample <u>0</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode. Valid when NUMSAMPLES_1																		
Value	Name																	
[0,15]	[0,0.9375]																	
3:0	Sample0 Y Offset	<table border="1"> <tbody> <tr> <td>Project:</td> <td>All</td> </tr> </tbody> </table>	Project:	All														
Project:	All																	



3DSTATE_MULTISAMPLE															
	<table border="1"> <tr> <td>Format:</td> <td>U0.4</td> </tr> <tr> <td colspan="2" style="text-align: center;">Description</td> </tr> <tr> <td colspan="2">Subpixel Y offset of Sample 0 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.</td> </tr> <tr> <td colspan="2">Valid when NUMSAMPLES_1</td> </tr> <tr> <td colspan="2" style="text-align: center;">Value</td> </tr> <tr> <td colspan="2" style="text-align: center;">Name</td> </tr> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </table>	Format:	U0.4	Description		Subpixel Y offset of Sample 0 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.		Valid when NUMSAMPLES_1		Value		Name		[0,15]	[0,0.9375]
Format:	U0.4														
Description															
Subpixel Y offset of Sample 0 relative to the UL pixel origin. Valid only when NUMSAMPLES_4 or _8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.															
Valid when NUMSAMPLES_1															
Value															
Name															
[0,15]	[0,0.9375]														
3	<table border="1"> <tr> <td>31:28</td> <td>Sample7 X Offset</td> </tr> <tr> <td>Format:</td> <td>U0.4</td> </tr> <tr> <td colspan="2">Subpixel X offset of Sample 7 relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.</td> </tr> <tr> <td colspan="2" style="text-align: center;">Value</td> </tr> <tr> <td colspan="2" style="text-align: center;">Name</td> </tr> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </table>	31:28	Sample7 X Offset	Format:	U0.4	Subpixel X offset of Sample 7 relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.		Value		Name		[0,15]	[0,0.9375]		
	31:28	Sample7 X Offset													
	Format:	U0.4													
	Subpixel X offset of Sample 7 relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.														
	Value														
Name															
[0,15]	[0,0.9375]														
27:24	<table border="1"> <tr> <td>Sample7 Y Offset</td> <td></td> </tr> <tr> <td>Format:</td> <td>U0.4</td> </tr> <tr> <td colspan="2">Subpixel Y offset of Sample 7 relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.</td> </tr> <tr> <td colspan="2" style="text-align: center;">Value</td> </tr> <tr> <td colspan="2" style="text-align: center;">Name</td> </tr> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </table>	Sample7 Y Offset		Format:	U0.4	Subpixel Y offset of Sample 7 relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.		Value		Name		[0,15]	[0,0.9375]		
Sample7 Y Offset															
Format:	U0.4														
Subpixel Y offset of Sample 7 relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.															
Value															
Name															
[0,15]	[0,0.9375]														
23:20	<table border="1"> <tr> <td>Sample6 X Offset</td> <td></td> </tr> <tr> <td>Format:</td> <td>U0.4</td> </tr> <tr> <td colspan="2">Subpixel X offset of Sample 6 relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.</td> </tr> <tr> <td colspan="2" style="text-align: center;">Value</td> </tr> <tr> <td colspan="2" style="text-align: center;">Name</td> </tr> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </table>	Sample6 X Offset		Format:	U0.4	Subpixel X offset of Sample 6 relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.		Value		Name		[0,15]	[0,0.9375]		
Sample6 X Offset															
Format:	U0.4														
Subpixel X offset of Sample 6 relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.															
Value															
Name															
[0,15]	[0,0.9375]														
19:16	<table border="1"> <tr> <td>Sample6 Y Offset</td> <td></td> </tr> <tr> <td>Format:</td> <td>U0.4</td> </tr> <tr> <td colspan="2">Subpixel Y offset of Sample 6 relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.</td> </tr> <tr> <td colspan="2" style="text-align: center;">Value</td> </tr> <tr> <td colspan="2" style="text-align: center;">Name</td> </tr> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </table>	Sample6 Y Offset		Format:	U0.4	Subpixel Y offset of Sample 6 relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.		Value		Name		[0,15]	[0,0.9375]		
Sample6 Y Offset															
Format:	U0.4														
Subpixel Y offset of Sample 6 relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.															
Value															
Name															
[0,15]	[0,0.9375]														
15:12	<table border="1"> <tr> <td>Sample5 X Offset</td> <td></td> </tr> <tr> <td>Format:</td> <td>U0.4</td> </tr> <tr> <td colspan="2">Subpixel X offset of Sample 5 relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.</td> </tr> <tr> <td colspan="2" style="text-align: center;">Value</td> </tr> <tr> <td colspan="2" style="text-align: center;">Name</td> </tr> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </table>	Sample5 X Offset		Format:	U0.4	Subpixel X offset of Sample 5 relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.		Value		Name		[0,15]	[0,0.9375]		
Sample5 X Offset															
Format:	U0.4														
Subpixel X offset of Sample 5 relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.															
Value															
Name															
[0,15]	[0,0.9375]														
11:8	Sample5 Y Offset														

3DSTATE_MULTISAMPLE											
	<table border="1"> <tr> <td>Format:</td> <td>U0.4</td> </tr> <tr> <td colspan="2">Subpixel Y offset of Sample <u>5</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.</td> </tr> <tr> <td style="text-align: center;">Value</td> <td style="text-align: center;">Name</td> </tr> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </table>	Format:	U0.4	Subpixel Y offset of Sample <u>5</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.		Value	Name	[0,15]	[0,0.9375]		
Format:	U0.4										
Subpixel Y offset of Sample <u>5</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.											
Value	Name										
[0,15]	[0,0.9375]										
7:4	<table border="1"> <tr> <td colspan="2">Sample4 X Offset</td> </tr> <tr> <td>Format:</td> <td>U0.4</td> </tr> <tr> <td colspan="2">Subpixel X offset of Sample <u>4</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.</td> </tr> <tr> <td style="text-align: center;">Value</td> <td style="text-align: center;">Name</td> </tr> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </table>	Sample4 X Offset		Format:	U0.4	Subpixel X offset of Sample <u>4</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.		Value	Name	[0,15]	[0,0.9375]
Sample4 X Offset											
Format:	U0.4										
Subpixel X offset of Sample <u>4</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.											
Value	Name										
[0,15]	[0,0.9375]										
3:0	<table border="1"> <tr> <td colspan="2">Sample4 Y Offset</td> </tr> <tr> <td>Format:</td> <td>U0.4</td> </tr> <tr> <td colspan="2">Subpixel Y offset of Sample <u>4</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.</td> </tr> <tr> <td style="text-align: center;">Value</td> <td style="text-align: center;">Name</td> </tr> <tr> <td>[0,15]</td> <td>[0,0.9375]</td> </tr> </table>	Sample4 Y Offset		Format:	U0.4	Subpixel Y offset of Sample <u>4</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.		Value	Name	[0,15]	[0,0.9375]
Sample4 Y Offset											
Format:	U0.4										
Subpixel Y offset of Sample <u>4</u> relative to the UL pixel origin. Valid only when NUMSAMPLES_8. Setting ignored when not in MSRASTMODE_xxx_PATTERN mode.											
Value	Name										
[0,15]	[0,0.9375]										

11.5 Early Depth/Stencil Processing

The Windower/IZ unit provides the Early Depth Test function, a major performance-optimization feature where an attempt is made to remove pixels that fail the Depth and Stencil Tests prior to pixel shading. This requires the WM unit to perform the interpolation of pixel (“source”) depth values, read the current (“destination”) depth values from the cached depth buffer, and perform the Depth and Stencil Tests. As the WM unit has per-pixel source and destination Z values, these values are passed in the PS thread payload, if required.

11.5.1 Depth Offset

The depth offset function is contained in SF unit, thus the state to control it is also contained in SF unit.

There are occasions where the Z position of some objects need to be slightly offset in order to reduce artifacts due to coplanar or near-coplanar primitives. A typical example is drawing the edges of triangles as wireframes – the lines need to be drawn slightly closer to the viewer to ensure they will not be occluded by the underlying polygon. Another example is drawing objects on a wall – without a bias on the z positions, they might be fully or partially occluded by the wall.

The device supports *global* depth offset, applied only to triangles, that bases the offset on the object’s z slope. Note that there is no clamping applied at this stage after the Z position is offset – clamping to [0,1] can be performed later after the Z position is interpolated to the pixel. This is preferable to clamping prior to interpolation, as the clamping would change the Z slope of the entire object.

The Global Depth Offset function is controlled by the **Global Depth Offset Enable** state variable in WM_STATE. Global Depth Offset is only applied to 3DOBJ_TRIANGLE objects.



When Global Depth Offset Enable is ENABLED, the pipeline will compute:

$\text{MaxDepthSlope} = \max(\text{abs}(dz/dx), \text{abs}(dz/dy))$ // approximation of max depth slope for polygon

When UNORM Depth Buffer is at Output Merger (or no Depth Buffer):

$\text{Bias} = \text{GlobalDepthOffsetConstant} * r + \text{GlobalDepthOffsetScale} * \text{MaxDepthSlope}$

Where r is the minimum representable value > 0 in the depth buffer format, converted to float32 (note: If state bit **Legacy Global Depth Bias Enable** is set, the r term will be forced to 1.0)

When Floating Point Depth Buffer at Output Merger:

$\text{Bias} = \text{GlobalDepthOffsetConstant} * 2^{(\text{exponent}(\text{max } z \text{ in primitive}) - r)} + \text{GlobalDepthOffsetScale} * \text{MaxDepthSlope}$

Where r is the # of mantissa bits in the floating point representation (excluding the hidden bit), e.g. 23 for float32 (note: If state bit Legacy Global Depth Bias Enable is set, no scaling is applied to the GlobalDepthOffsetConstant).

Adding Bias to z:

```
if (GlobalDepthOffsetClamp > 0)
    Bias = min(DepthBiasClamp, Bias)
else if (GlobalDepthOffsetClamp < 0)
    Bias = max(DepthBiasClamp, Bias)
// else if GlobalDepthOffsetClamp == 0, no clamping occurs
z = z + Bias
```

Biasing is constant for a given primitive The biasing formulas are performed with float32 arithmetic Global Depth Bias is not applied to any point or line primitives

11.5.2 Early Depth Test/Stencil Test/Write

When **Early Depth Test Enable** is ENABLED, the WM unit will attempt to discard depth-occluded pixels during scan conversion (before processing them in the Pixel Shader) Pixels are only discarded when the WM unit can ensure that they would have no impact to the ColorBuffer or DepthBuffer This function is therefore only a performance feature.

Note: The **Early Depth Test Enable** bit is no longer present. This function is always enabled.

If some pixels within a subspan are discarded, only the pixel mask is affected indicating that the discarded pixels are not active If all pixels within a subspan are discarded, that subspan will not even be dispatched.

11.5.2.1 Software-Provided PS Kernel Info

In order for the WM unit to properly perform Early Depth Test and supply the proper information in the PS thread payload (and even determine if a PS thread needs to be dispatched), it requires information regarding the PS kernel operation This information is provided by a number of state bits in WM_STATE, as summarized in the following table.



State Bit	Description
Pixel Shader Kill Pixel	This must be set when there is a chance that valid pixels passed to a PS thread may be discarded. This includes the discard of pixels by the PS thread resulting from a “killpixel” or “alphatest” function or as dictated by the results of the sampling of a “chroma-keyed” texture. The WM unit needs this information to prevent early depth/stencil writes for pixels which might be killed by the PS thread, etc. See WM_STATE/3DSTATE_WM for more information.
Pixel Shader Computed Depth	This must be set when the PS thread computes the “source” depth value (i.e., from the API POV, writes to the “oDepth” output). In this case the WM unit can’t make any decisions based on the WM-interpolated depth value. See WM_STATE/3DSTATE_WM for more information.
Pixel Shader Uses Source Depth	Must be set if the PS thread requires the WM-interpolated source depth value. This will force the source depth to be passed in the thread payload where otherwise the WM unit would not have seen it as required. See WM_STATE/3DSTATE_WM for more information.

11.5.3 Hierarchical Depth Buffer

A hierarchical depth buffer is supported beginning with to reduce memory traffic due to depth buffer accesses. This buffer is supported only in Tile Y memory.

The **Surface Type, Height, Width, Depth, Minimum Array Element, Render Target View Extent,** and **Depth Coordinate Offset X/Y** of the hierarchical depth buffer are inherited from the depth buffer. The height and width of the hierarchical depth buffer that must be allocated are computed by the following formulas, where HZ is the hierarchical depth buffer and Z is the depth buffer. The Z_Height, Z_Width, and Z_Depth values given in these formulas are those present in 3DSTATE_DEPTH_BUFFER incremented by one. : The value of Z_Height and Z_Width must each be multiplied by 2 before being applied to the table below if **Number of Multisamples** is set to NUMSAMPLES_4. The value of Z_Height must be multiplied by 2 and Z_Width must be multiplied by 4 before being applied to the table below if **Number of Multisamples** is set to NUMSAMPLES_8..



Since Hierarchical Depth Buffer supports multiple LODs. The HZ_height is different as shown in the table below:

Surface Type	HZ_Width (bytes)	HZ_Height (rows)
SURFTYPE_1D	ceiling(Z_Width / 16) * 16	Ceiling ((Q_pitch * Z_depth/2) /8) * 8
SURFTYPE_2D	ceiling(Z_Width / 16) * 16	Ceiling ((Q_pitch * Z_depth/2) /8) * 8
SURFTYPE_3D	ceiling(Z_Width / 16) * 16	see below
SURFTYPE_CUBE	ceiling(Z_Width / 16) * 16	Ceiling ((Q_pitch * Z_depth * 6/2) /8) * 8

where, Qpitch is computed using vertical alignment j=8, please refer to the GPU overview volume for Qpitch definition.

The minimum HZ_Height required for a 3D surface must be computed based on h_L parameters documented in the GPU Overview volume, and the maximum LOD m:

$$HZ_Height = \frac{1}{2} \left[\sum_{i=0}^m h_i * \max \left(1, \text{floor} \left(\frac{Z_Depth}{2^i} \right) \right) \right]$$

In order to compute the minimum QPitch for the HZ surface, the height of each LOD in pixels is determined using the equations for h_L in the GPU Overview volume, using a vertical alignment j=8. The following equation gives the minimum HZ_QPitch based on largest LOD m defined in the surface:

$$HZ_QPitch = \frac{1}{2} \left[h_0 + \max \left(h_1, \sum_{i=2}^m h_i \right) \right]$$

If m is less than 2, treat all h_L with L > m as zero and use the above equation.

The minimum HZ_Height required for a 3D surface must be computed based on h_L parameters documented in the GPU Overview volume, and the maximum LOD m:

$$HZ_Height = \frac{1}{2} \left[\sum_{i=0}^m h_i * \text{floor} \left(\frac{Z_Depth}{2^i} \right) \right]$$

The format of the data in the hierarchical depth buffer is not documented here, as this surface needs only to be allocated by software Hardware will read and write this surface during operation and its contents are discarded once the last primitive is rendered that uses the hierarchical depth buffer.

The hierarchical depth buffer can be enabled whenever a depth buffer is defined, with its effect being invisible other than generally higher performance The only exception is the hierarchical depth buffer must be disabled when using software tiled rendering.

If HiZ is enabled, you must initialize the clear value by either

- a. Perform a depth clear pass to initialize the clear value.
- b. Send a 3dstate_clear_params packet with valid = 1



Without one of these events, context switching will fail, as it will try to save off a clear value even though no valid clear value has been set. When context restore happens, HW will restore an uninitialized clear value.

11.5.3.1 Depth Buffer Clear

With the hierarchical depth buffer enabled, performance is generally improved by using the special clear mechanism described here to clear the hierarchical depth buffer and the depth buffer. This is enabled through the **Depth Buffer Clear** field in WM_STATE or 3DSTATE_WM. This bit can be used to clear the depth buffer in the following situations:

- Complete depth buffer clear
- Partial depth buffer clear with the clear value the same as the one used on the previous clear
- Partial depth buffer clear with the clear value different than the one used on the previous clear can use this mechanism if a depth buffer resolve is performed first.

The following is required when performing a depth buffer clear with this field:

- If other rendering operations have preceded this clear, a PIPE_CONTROL with depth cache flush enabled, Depth Stall bit enabled must be issued before the rectangle primitive used for the depth buffer clear operation.
- The fields in 3DSTATE_CLEAR_PARAMS are set to indicate the source of the clear value and (if source is in this command) the clear value itself.
- A rectangle primitive representing the clear area is delivered. The primitive must adhere to the following restrictions on size:
 - If **Number of Multisamples** is NUMSAMPLES_1, the rectangle must be aligned to an 8x4 pixel block relative to the upper left corner of the depth buffer, and contain an integer number of these pixel blocks, and all 8x4 pixels must be lit.
 - If **Number of Multisamples** is NUMSAMPLES_4, the rectangle must be aligned to a 4x2 pixel block (8x4 sample block) relative to the upper left corner of the depth buffer, and contain an integer number of these pixel blocks, and all samples of the 4x2 pixels must be lit.
 - If **Number of Multisamples** is NUMSAMPLES_8, the rectangle must be aligned to a 2x2 pixel block (8x4 sample block) relative to the upper left corner of the depth buffer, and contain an integer number of these pixel blocks, and all samples of the 2x2 pixels must be lit.
- **Depth Test Enable** must be disabled and **Depth Buffer Write Enable** must be enabled (if depth is being cleared).
- Stencil buffer clear can be performed at the same time by enabling Stencil Buffer Write Enable. Stencil Test Enable must be enabled and Stencil Pass Depth Pass Op set to REPLACE, and the clear value that is placed in the stencil buffer is the **Stencil Reference Value** from COLOR_CALC_STATE.
- Note also that stencil buffer clear can be performed without depth buffer clear. For stencil only clear, **Depth Test Enable** and **Depth Buffer Write Enable** must be disabled.
- **Pixel Shader Dispatch**, **Alpha Test**, **Pixel Shader Kill Pixel** and **Pixel Shader Computed Depth** must all be disabled.

Several cases exist where **Depth Buffer Clear** cannot be enabled (the legacy method of clearing must be performed):



- If the depth buffer format is D32_FLOAT_S8X24_UINT or D24_UNORM_S8_UINT.
- If stencil test is enabled but the separate stencil buffer is disabled.

11.5.3.2 Depth Buffer Resolve

If the hierarchical depth buffer is enabled, the depth buffer may contain incorrect results after rendering is complete. If the depth buffer is retained and used for another purpose (i.e. as input to the sampling engine as a shadow map), it must first be “resolved”. This is done by setting the **Depth Buffer Resolve Enable** field in WM_STATE or 3DSTATE_WM and rendering a full render target sized rectangle. Once this is complete, the depth buffer will contain the same contents as it would have had the rendering been performed with the hierarchical depth buffer disabled. In a typical usage model, depth buffer needs to be resolved after rendering on it and before using a depth buffer as a source for any consecutive operation. Depth buffer can be used as a source in three different cases: using it as a texture for the next rendering sequence, honoring a lock on the depth buffer to the host OR using the depth buffer as a blit source.

The following is required when performing a depth buffer resolve:

- A rectangle primitive of the same size as the previous depth buffer clear operation must be delivered, and depth buffer state cannot have changed since the previous depth buffer clear operation.
- **Depth Test Enable** must be enabled with the **Depth Test Function** set to NEVER. **Depth Buffer Write Enable** must be enabled. **Stencil Test Enable** and **Stencil Buffer Write Enable** must be disabled.
- **Pixel Shader Dispatch**, **Alpha Test**, **Pixel Shader Kill Pixel** and **Pixel Shader Computed Depth** must all be disabled.

11.5.3.3 Hierarchical Depth Buffer Resolve

If the hierarchical depth buffer is enabled, the hierarchical depth buffer may contain incorrect results if the depth buffer is written to outside of the 3D rendering operation. If this occurs, the hierarchical depth buffer must be “resolved” to avoid incorrect device behavior. This is done by setting the Hierarchical Depth Buffer Resolve Enable field in WM_STATE or 3DSTATE_WM and rendering a full render target sized rectangle. Once this is complete, the hierarchical depth buffer will contain contents such that rendering will give the same results as it would have had the rendering been performed with the hierarchical depth buffer disabled.

The following is required when performing a hierarchical depth buffer resolve:

- A rectangle primitive covering the full render target must be delivered.
- **Depth Test Enable** must be disabled. **Depth Buffer Write Enable** must be enabled. **Stencil Test Enable** and **Stencil Buffer Write Enable** must be disabled.
- **Pixel Shader Dispatch**, **Alpha Test**, **Pixel Shader Kill Pixel** and **Pixel Shader Computed Depth** must all be disabled.

11.5.4 Separate Stencil Buffer

The separate stencil buffer is always enabled, thus the field in 3DSTATE_DEPTH_BUFFER to explicitly enable the separate stencil buffer has been removed. Surface formats with interleaved depth and stencil are no longer supported.



The stencil buffer has a format of S8_UINT, and shares **Surface Type, Height, Width, and Depth, Minimum Array Element, Render Target View Extent, Depth Coordinate Offset X/Y, LOD, and Depth Buffer Object Control State** fields of the depth buffer.

11.5.5 Depth/Stencil Buffer State

11.5.5.1 3DSTATE_DEPTH_BUFFER

DWord		Bit	Description	Project
3DSTATE_DEPTH_BUFFER				
Source:		RenderCS		
Length Bias:		2		
The depth buffer surface state is delivered as a pipelined state packet. However, the state change pipelining isn't completely transparent (see restriction below).				
Programming Notes				Project
Restriction: Prior to changing Depth/Stencil Buffer state (i.e., any combination of 3DSTATE_DEPTH_BUFFER, 3DSTATE_CLEAR_PARAMS, 3DSTATE_STENCIL_BUFFER, 3DSTATE_HIER_DEPTH_BUFFER) SW must first issue a pipelined depth stall (PIPE_CONTROL with Depth Stall bit set), followed by a pipelined depth cache flush (PIPE_CONTROL with Depth Flush Bit set), followed by another pipelined depth stall (PIPE_CONTROL with Depth Stall Bit set), unless SW can otherwise guarantee that the pipeline from WM onwards is already flushed (e.g., via a preceding MI_FLUSH).				
3DSTATE_DEPTH_BUFFER must always be programmed along with the other Depth/Stencil state commands (i.e. 3DSTATE_CLEAR_PARAMS, 3DSTATE_STENCIL_BUFFER, or 3DSTATE_HIER_DEPTH_BUFFER).				
Driver must send a least one PIPE_CONTROL command with CS Stall and a post sync operation prior to the group of depth commands (3DSTATE_DEPTH_BUFFER, 3DSTATE_CLEAR_PARAMS, 3DSTATE_STENCIL_BUFFER, and 3DSTATE_HIER_DEPTH_BUFFER).				
The depth buffer is always Tile-Y				
DWord	Bit	Description		
0	31:29	Command Type		
		Default Value:	3h GFXPIPE	
		Format:	OpCode	
	28:27	Command SubType		
		Default Value:	3h GFXPIPE_3D	
		Format:	OpCode	
	26:24	3D Command Opcode		
		Default Value:	0h 3DSTATE_PIPELINED	
		Format:	OpCode	
	23:16	3D Command Sub Opcode		
		Default Value:	05h 3DSTATE_DEPTH_BUFFER	
		Format:	OpCode	
15:8	Reserved			
	Project:	All		
	Format:	MBZ		
7:0	Dword Length			
	Default Value:	0h Excludes Dword (0,1)		
	Project:	All		
	Format:	=n Total Length – 2		



3DSTATE_DEPTH_BUFFER					
	0h	Reserved	Reserved		
	1h	D32_FLOAT	D32_FLOAT		
	2h	Reserved	Reserved		
	3h	D24_UNORM_X8_UINT	D24_UNORM_X8_UINT		
	4h	Reserved	Reserved		
	5h	D16_UNORM	D16_UNORM		
	6h-7h	Reserved	Reserved		
17:0	Surface Pitch				
	Project:	All			
	Format:	U18-1 Pitch in Bytes			
	This field specifies the pitch of the depth buffer in (#Bytes – 1).				
	Value	Name	Description		
	[127, 3FFFFh]		corresponding to [128B, 256KB] also restricted to a multiple of 128B		
	Programming Notes				
	The pitch specified must be a multiple of the tile pitch, in the range [128B, 128KB].				
	2	31:0	Surface Base Address		
			Project:	All	
Format:			GraphicsAddress[31:0]Depth_Buffer		
This field specifies the starting Dword address of the buffer in mapped Graphics Memory.					
Programming Notes					
The Depth Buffer can only be mapped to Main Memory (uncached). If the surface is tiled, the base address must conform to the Per-Surface Tiling Alignment Rules. If the buffer is linear, the surface must be 64-byte aligned.					
3	31:18	Height			
		Project:	All		
		Format:	U14		
		Range: SURFTYPE_1D: must be zero SURFTYPE_2D: height of surface – 1 (y/v dimension) [0,16383] SURFTYPE_3D: height of surface – 1 (y/v dimension) [0,2047] SURFTYPE_CUBE: height of surface – 1 (y/v dimension) [0, 16383]			
		This field specifies the height of the surface. If the surface is MIP-mapped, this field contains the height of the base MIP level.			
		Programming Notes			
		The Height of the depth buffer must be the same as the Height of the render target(s) (defined in SURFACE_STATE), unless Surface Type is SURFTYPE_1D or SURFTYPE_2D with Depth = 0 (non-array) and LOD = 0 (non-mip mapped).			
		17:4	Width	Project:	All
				Format:	U14-1
				Range: SURFTYPE_1D: width of surface – 1 (x/u dimension) [0, 16383] SURFTYPE_2D: width of surface – 1 (x/u dimension) [0, 16383] SURFTYPE_3D: width of surface – 1 (x/u dimension)	

3DSTATE_DEPTH_BUFFER																			
	<p>[0,2047]SURFTYPE_CUBE: width of surface – 1 (x/u dimension) [0, 16383]</p> <p>This field specifies the width of the surface. If the surface is MIP-mapped, this field specifies the width of the base MIP level. The width is specified in units of pixels.</p> <p style="text-align: center;">Programming Notes</p> <p>The Width specified by this field must be less than or equal to the surface pitch (specified in bytes via the Surface Pitch field). For cube maps, Width must be set equal to Height. The Width of the depth buffer must be the same as the Width of the render target(s) (defined in SURFACE_STATE), unless Surface Type is SURFTYPE_1D or SURFTYPE_2D with Depth = 0 (non-array) and LOD = 0 (non-mip mapped).</p>																		
3:0	<p>LOD</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U4 in LOD units</td> </tr> </table> <p>This field defines the MIP level that is currently being rendered into.</p> <table border="1"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td>[0, 14]</td> <td></td> </tr> </tbody> </table> <p style="text-align: center;">Programming Notes</p> <p>The LOD of the depth buffer must be the same as the LOD of the render target(s) (defined in SURFACE_STATE)</p>	Project:	All	Format:	U4 in LOD units	Value	Name	[0, 14]											
Project:	All																		
Format:	U4 in LOD units																		
Value	Name																		
[0, 14]																			
4	<p>31:21 Depth</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U11-1</td> </tr> </table> <p>This field specifies the total number of levels for a volume texture or the number of array elements allowed to be accessed starting at the Minimum Array Element for arrayed surfaces. If the volume texture is MIP-mapped, this field specifies the depth of the base MIP level.</p> <table border="1"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td>[0, 2047]</td> <td>SURFTYPE_1D number of array elements – 1</td> </tr> <tr> <td>[0, 2047]</td> <td>SURFTYPE_2D number of array elements – 1</td> </tr> <tr> <td>[0, 2047]</td> <td>SURFTYPE_3D depth of surface – 1 (r/z dimension)</td> </tr> <tr> <td>0</td> <td>SURFTYPE_CUBE (must be zero)</td> </tr> </tbody> </table> <p style="text-align: center;">Programming Notes</p> <p>The Depth of the depth buffer must be the same as the Depth of the render target(s) (defined in SURFACE_STATE).</p> <p>20:10 Minimum Array Element</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U11</td> </tr> </table> <p>For 1D and 2D Surfaces: This field indicates the minimum array element that can be accessed as part of this surface. The delivered array index is added to this field before being used to address the surface.</p> <p>For 3D Surfaces: This field indicates the minimum 'R' coordinate on the LOD currently being rendered to. This field is added to the delivered array index before it is used to address the surface.</p> <p>For Other Surfaces:</p>	Project:	All	Format:	U11-1	Value	Name	[0, 2047]	SURFTYPE_1D number of array elements – 1	[0, 2047]	SURFTYPE_2D number of array elements – 1	[0, 2047]	SURFTYPE_3D depth of surface – 1 (r/z dimension)	0	SURFTYPE_CUBE (must be zero)	Project:	All	Format:	U11
Project:	All																		
Format:	U11-1																		
Value	Name																		
[0, 2047]	SURFTYPE_1D number of array elements – 1																		
[0, 2047]	SURFTYPE_2D number of array elements – 1																		
[0, 2047]	SURFTYPE_3D depth of surface – 1 (r/z dimension)																		
0	SURFTYPE_CUBE (must be zero)																		
Project:	All																		
Format:	U11																		



3DSTATE_DEPTH_BUFFER							
	This field is ignored.						
	<table border="1"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> </tr> </thead> <tbody> <tr> <td>[0, 2047]</td> <td>SURFTYPE_1D/2D</td> </tr> <tr> <td>[0, 2047]</td> <td>SURFTYPE_3D</td> </tr> </tbody> </table>	Value	Name	[0, 2047]	SURFTYPE_1D/2D	[0, 2047]	SURFTYPE_3D
Value	Name						
[0, 2047]	SURFTYPE_1D/2D						
[0, 2047]	SURFTYPE_3D						
9:4	<p>Reserved</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Project:	All	Format:	MBZ		
Project:	All						
Format:	MBZ						
3:0	<p>Depth Buffer Object Control State</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MEMORY_OBJECT_CONTROL_STATE</td> </tr> </table> <p>Specifies the memory object control state for the depth buffer.</p>	Project:	All	Format:	MEMORY_OBJECT_CONTROL_STATE		
Project:	All						
Format:	MEMORY_OBJECT_CONTROL_STATE						
5	<p>31:16 Depth Coordinate Offset Y</p> <table border="1"> <tr> <td>Format:</td> <td>S15 in Screen Space (pixels)(3 LSBs MBZ)</td> </tr> </table> <p>Range: [-8192,8191] Bits 31:30 should be a sign extension Specifies a signed pixel offset to be added to the RenderTarget Y coordinate in order to generate a DepthBuffer Y coordinate. (See Depth Coordinate in Windower).</p> <p style="text-align: center;">Programming Notes</p> <p>The 3 LSBs of both offsets must be zero to ensure correct alignmentSoftware must ensure that the resulting (sum) coordinate value is non-negative This field must be zero when rendering to field-mode (interlaced) Color Buffers (i.e., when Surface State's VerticalLineStride==1). This field can only be nonzero when rendering to surfaces of type SURFTYPE_1D and SURFTYPE_2D with Depth = 0 (non-array) and LOD = 0 (non-mip mapped). The offsets need to be aligned to the hashing mode set for WM in the GT_MODE register (0x7008) bits[12:10]. For eg if the hashing mode is set to 16x16, the Depth Coordinate Y offset needs to be aligned to the 16x16 pixel block.</p>	Format:	S15 in Screen Space (pixels)(3 LSBs MBZ)				
Format:	S15 in Screen Space (pixels)(3 LSBs MBZ)						
15:0	<p>Depth Coordinate Offset X</p> <table border="1"> <tr> <td>Format:</td> <td>S15 in Screen Space (pixels)(3 LSBs MBZ)</td> </tr> </table> <p>Range: [-8192,8191] Bits 15:14 should be a sign extension Specifies a signed pixel offset to be added to the RenderTarget X coordinate in order to generate a DepthBuffer X coordinate. (See Depth Coordinate in Windower).</p> <p style="text-align: center;">Programming Notes</p> <p>The 3 LSBs of both offsets must be zero to ensure correct alignmentSoftware must ensure that the resulting (sum) coordinate value is non-negative. This field must be zero when rendering to field-mode (interlaced) Color Buffers (i.e., when Surface State's VerticalLineStride==1). This field can only be nonzero when rendering to surfaces of type SURFTYPE_1D and</p>	Format:	S15 in Screen Space (pixels)(3 LSBs MBZ)				
Format:	S15 in Screen Space (pixels)(3 LSBs MBZ)						



3DSTATE_DEPTH_BUFFER					
	<p>SURFTYPE_2D with Depth = 0 (non-array) and LOD = 0 (non-mip mapped). The offsets need to be aligned to the hashing mode set for WM in the GT_MODE register (0x7008) bits[12:10]. For eg if the hashing mode is set to 16x16, the Depth Coordinate X offset needs to be aligned to the 16x16 pixel block.</p>				
6	<p>31:21 Render Target View Extent</p> <table border="1" style="width: 100%;"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U11</td> </tr> </table> <p>Range: SURFTYPE_1D/2D: same value as Depth field Range: SURFTYPE_3D: [0, 2047] to indicate extent of [1, 2048]</p> <p>For 3D Surfaces: This field indicates the extent of the accessible 'R' coordinates minus 1 on the LOD currently being rendered to.</p> <p>For 1D and 2D Surfaces: This field must be set to the same value as the Depth field.</p> <p>For Other Surfaces: This field is ignored.</p>	Project:	All	Format:	U11
Project:	All				
Format:	U11				
	<p>20:0 Reserved</p> <table border="1" style="width: 100%;"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Project:	All	Format:	MBZ
Project:	All				
Format:	MBZ				

11.5.5.2 3DSTATE_STENCIL_BUFFER

3DSTATE_STENCIL_BUFFER	
Source:	RenderCS
Length Bias:	2
This command sets the surface state of the separate stencil buffer, delivered as a pipelined state command. However, the state change pipelining isn't completely transparent (see restriction below).	
Programming Notes	
Restriction: Prior to changing Depth/Stencil Buffer state (i.e., any combination of 3DSTATE_DEPTH_BUFFER, 3DSTATE_CLEAR_PARAMS, 3DSTATE_STENCIL_BUFFER, 3DSTATE_HIER_DEPTH_BUFFER) SW must first issue a pipelined depth stall (PIPE_CONTROL with Depth Stall bit set, followed by a pipelined depth cache flush (PIPE_CONTROL with Depth Flush Bit set, followed by another pipelined depth stall (PIPE_CONTROL with Depth Stall Bit set), unless SW can otherwise guarantee that the pipeline from WM onwards is already flushed (e.g., via a preceding MI_FLUSH).	
3DSTATE_STENCIL_BUFFER must always be programmed in the along with the other Depth/Stencil state commands(i.e. 3DSTATE_DEPTH_BUFFER, 3DSTATE_CLEAR_PARAMS, or 3DSTATE_HIER_DEPTH_BUFFER)	
Driver must send a least one PIPE_CONTROL command with CS Stall and a post sync operation prior to the group of depth commands(3DSTATE_DEPTH_BUFFER, 3DSTATE_CLEAR_PARAMS, 3DSTATE_STENCIL_BUFFER, and 3DSTATE_HIER_DEPTH_BUFFER).	
The stencil buffer is always Tile-Y	
DWord	Bit
Description	
0	31:29 Command Type



3DSTATE_STENCIL_BUFFER		
	Default Value:	3h GFXPIPE
	Format:	OpCode
28:27	Command SubType	
	Default Value:	3h GFXPIPE_3D
	Format:	OpCode
26:24	3D Command Opcode	
	Default Value:	0h 3DSTATE_PIPELINED
	Format:	OpCode
23:16	3D Command Sub Opcode	
	Default Value:	06h 3DSTATE_STENCIL_BUFFER
	Format:	OpCode
15:8	Reserved	
	Project:	All
	Format:	MBZ
7:0	Dword Length	
	Project:	All
	Format:	=n Total Length – 2
	Value	Name
	1h	Excludes Dword (0,1) [Default]
1	31	Reserved
	Format:	MBZ
	30:29	Reserved
	Project:	All
	Format:	MBZ
	28:25	Stencil Buffer Object Control State
	Format:	MEMORY_OBJECT_CONTROL_STATE
		Description
		Specifies the memory object control state for the stencil buffer. Stencil Buffer Object Control State [3:0]
		This field is not context save and restored by hardware. If this field is programmed to any value other than zero, it must be programmed after the following commands or events:
		<ul style="list-style-type: none"> MI_SET_CONTEXT MI_WAIT_FOR_EVENT (Specifically waits on vblank or display flip) Render engine goes IDLE due to head point equal to tail pointer
		Project
	24:22	Reserved
	Format:	MBZ
	21:17	Reserved



3DSTATE_STENCIL_BUFFER											
	<table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Project:	All	Format:	MBZ						
Project:	All										
Format:	MBZ										
16:0	<p>Surface Pitch</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>U17-1 Pitch in Bytes</td> </tr> </table> <p>This field specifies the pitch of the stencil buffer in (#Bytes – 1).</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>[127, 3FFFFh]</td> <td></td> <td>corresponding to [128B, 128KB]also restricted to a multiple of 128B</td> </tr> </tbody> </table> <p style="text-align: center;">Programming Notes</p> <p>Since this surface is tiled, the pitch specified must be a multiple of the tile pitch, in the range [128B, 128KB].</p>	Project:	All	Format:	U17-1 Pitch in Bytes	Value	Name	Description	[127, 3FFFFh]		corresponding to [128B, 128KB]also restricted to a multiple of 128B
Project:	All										
Format:	U17-1 Pitch in Bytes										
Value	Name	Description									
[127, 3FFFFh]		corresponding to [128B, 128KB]also restricted to a multiple of 128B									
2	<p>31:0 Surface Base Address_low</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>GraphicsAddress[31:0]Stencil_Buffer</td> </tr> </table> <p>This field specifies the starting Dword address of the buffer in mapped Graphics Memory.</p> <p style="text-align: center;">Programming Notes</p> <p>The Stencil Buffer can only be mapped to Main Memory (uncached).</p> <p>Since this surface is tiled, the base address must conform to the Per-Surface Tiling Alignment Rules.</p>	Project:	All	Format:	GraphicsAddress[31:0]Stencil_Buffer						
Project:	All										
Format:	GraphicsAddress[31:0]Stencil_Buffer										

11.5.5.3 3DSTATE_HIER_DEPTH_BUFFER

3DSTATE_HIER_DEPTH_BUFFER	
Source:	RenderCS
Length Bias:	2
This command sets the surface state of the hierarchical depth buffer, delivered as a pipelined state command. However, the state change pipelining isn't completely transparent (see restriction below).	
Programming Notes	
<p>Restriction: Prior to changing Depth/Stencil Buffer state (i.e., any combination of 3DSTATE_DEPTH_BUFFER, 3DSTATE_CLEAR_PARAMS, 3DSTATE_STENCIL_BUFFER, 3DSTATE_HIER_DEPTH_BUFFER) SW must first issue a pipelined depth stall (PIPE_CONTROL with Depth Stall bit set, followed by a pipelined depth cache flush (PIPE_CONTROL with Depth Flush Bit set, followed by another pipelined depth stall (PIPE_CONTROL with Depth Stall Bit set), unless SW can otherwise guarantee that the pipeline from WM onwards is already flushed (e.g., via a preceding MI_FLUSH).</p>	
3DSTATE_HIER_DEPTH_BUFFER must always be programmed in the along with the other Depth/Stencil state commands(i.e. 3DSTATE_DEPTH_BUFFER, 3DSTATE_CLEAR_PARAMS, or 3DSTATE_STENCIL_BUFFER)	
Driver must send a least one PIPE_CONTROL command with CS Stall and a post sync operation prior to the group of depth commands(3DSTATE_DEPTH_BUFFER, 3DSTATE_CLEAR_PARAMS, 3DSTATE_STENCIL_BUFFER, and 3DSTATE_HIER_DEPTH_BUFFER).	
DWord	Bit
Description	
0	31:29
Command Type	
Default Value: 3h GFXPIPE	



3DSTATE_HIER_DEPTH_BUFFER		
	Format:	OpCode
28:27	Command SubType	
	Default Value:	3h GFXPIPE_3D
	Format:	OpCode
26:24	3D Command Opcode	
	Default Value:	0h 3DSTATE_PIPELINED
	Format:	OpCode
23:16	3D Command Sub Opcode	
	Default Value:	07h 3DSTATE_HIER_DEPTH_BUFFER
	Format:	OpCode
15:8	Reserved	
	Project:	All
	Format:	MBZ
7:0	Dword Length	
	Project:	All
	Format:	=n Total Length – 2
	Value	Name
	1h	Excludes Dword (0,1) [Default]
1	31:29	Reserved
		Format:
28:25	Hierarchical Depth Buffer Object Control State	
	Format:	MEMORY_OBJECT_CONTROL_STATE
	Description	Project
	Specifies the memory object control state for the hierarchical depth buffer.	
	This field is not context save and restored by hardware. If this field is programmed to any value other than zero, it must be programmed after the following commands or events:	
	<ul style="list-style-type: none"> • MI_SET_CONTEXT • MI_WAIT_FOR_EVENT (Specifically waits on vblank or display flip) • Render engine goes IDLE due to head point equal to tail pointer 	
24:17	Reserved	
	Project:	All
	Format:	MBZ
16:0	Surface Pitch	
	Project:	All
	Format:	U17-1 Pitch in Bytes
	This field specifies the pitch of the hierarchical depth buffer in (#Bytes – 1).	
	Value	Name



3DSTATE_HIER_DEPTH_BUFFER	
	[127, 3FFFFh] corresponding to [128B, 128KB] also restricted to a multiple of 128B
	Programming Notes
	Since this surface is tiled, the pitch specified must be a multiple of the tile pitch, in the range [128B, 128KB].
2	31:0 Surface Base Address [31:0]
	Project: All
	Format: GraphicsAddress[31:0]HierarchicalDepthBuffer
	This field specifies the starting Dword address of the buffer in mapped Graphics Memory.
	Programming Notes
	The Hierarchical Depth Buffer can only be mapped to Main Memory (uncached).
	Since this surface is tiled, the base address must conform to the Per-Surface Tiling Alignment Rules.

11.5.5.4 3DSTATE_CLEAR_PARAMS

3DSTATE_CLEAR_PARAMS	
Source:	RenderCS
Length Bias:	2
This command defines the depth clear value delivered as a pipelined state command. However, the state change pipelining isn't completely transparent (see restriction below).	
Programming Notes	
Restriction: Prior to changing Depth/Stencil Buffer state (i.e., any combination of 3DSTATE_DEPTH_BUFFER, 3DSTATE_CLEAR_PARAMS, 3DSTATE_STENCIL_BUFFER, 3DSTATE_HIER_DEPTH_BUFFER) SW must first issue a pipelined depth stall (PIPE_CONTROL with Depth Stall bit set), followed by a pipelined depth cache flush (PIPE_CONTROL with Depth Flush Bit set, followed by another pipelined depth stall (PIPE_CONTROL with Depth Stall Bit set), unless SW can otherwise guarantee that the pipeline from WM onwards is already flushed (e.g., via a preceding MI_FLUSH).	
3DSTATE_CLEAR_PARAMS must always be programmed in the along with the other Depth/Stencil state commands(i.e. 3DSTATE_DEPTH_BUFFER, 3DSTATE_STENCIL_BUFFER, or 3DSTATE_HIER_DEPTH_BUFFER)	
Driver must send a least one PIPE_CONTROL command with CS Stall and a post sync operation prior to the group of depth commands(3DSTATE_DEPTH_BUFFER, 3DSTATE_CLEAR_PARAMS, 3DSTATE_STENCIL_BUFFER, and 3DSTATE_HIER_DEPTH_BUFFER).	
DWord	Bit
Description	
0	31:29 Command Type
	Default Value: 3h GFXPIPE
	Format: OpCode
	28:27 Command SubType
	Default Value: 3h GFXPIPE_3D
	Format: OpCode
	26:24 3D Command Opcode
	Default Value: 0h 3DSTATE_PIPELINED
	Format: OpCode
	23:16 3D Command Sub Opcode



3DSTATE_CLEAR_PARAMS		
		Default Value: 04h 3DSTATE_CLEAR_PARAMS Format: OpCode
	15:8	Reserved Format: MBZ
	7:0	Dword Length Default Value: 1h Excludes Dword (0,1) Format: =n Total Length – 2
1	31:0	Depth Clear Value Format: for Surface Format of depth buffer: D32_FLOAT_S8X24_UINT: IEEE_FloatD32_FLOAT: IEEE_FloatD24_UNORM_S8_UINT: U24 UNORM in bits [23:0] D24_UNORM_X8_UINT: U24 UNORM in bits [23:0] D16_UNORM: U16 UNORM in bits [15:0] This field defines the clear value that will be applied to the depth buffer if the Depth Buffer Clear field is enabled. It is valid only if Depth Buffer Clear Value Valid is set.
2	31:1	Reserved Format: MBZ
	0	Depth Clear Value Valid Format: Boolean This field enables the Depth Clear Value . If clear, the depth clear value is obtained from interpolated depth of an arbitrary pixel of the primitive rendered with Depth Buffer Clear set in WM_STATE or 3DSTATE_WM. If set, the depth clear value is obtained from the Depth Clear Value field of this command.

11.6 Barycentric Attribute Interpolation

Given hardware clipper and setup, some of the previous flexibility in the algorithm used to interpolate attributes is no longer available. Hardware uses barycentric parameters to aid in attribute interpolation, and these parameters are computed in hardware per-pixel (or per-sample) and delivered in the thread payload to the pixel shader. Also delivered in the payload are a set of vertex deltas (a0, a1, and a2) per channel of each attribute.

There are six different barycentric parameters that can be enabled for delivery in the pixel shader payload. These are enabled via the **Barycentric Interpolation Mode** bits in 3DSTATE_WM.

In the pixel shader kernel, the following computation is done for each attribute channel of each pixel/sample given the corresponding attribute channel a0/a1/a2 and the pixel/sample's b1/b2 barycentric parameters, where A is the value of the attribute channel at that pixel/sample:

$$A = a_0 + (a_1 * b_1) + (a_2 * b_2)$$

11.7 MCS Buffer for Render Target(s)

MCS buffer can be enabled for two purposes described below. MCS buffer can be controlled using two mechanisms: 1) MMIO bit Cache Mode 1 (0x2124) register bit 5 and 2) RT surface state. Following table summarizes modes of operation related to MCS buffer.



Cache Mode MMIO Bit (Please refer to Vol 1c)	MSC Enable (Surface State)	Operation
1 (feature disable)	X	Normal mode of operation i.e. no MSAA compression and no color clear
0	0	Normal mode of operation i.e. no MSAA compression and no color clear
0	1	Depending on the Number of multi-samples, either MSAA compression OR color clear is enabled

- MSAA Compression:** Multi-sample render target is bound to the pipeline and MSAA compression feature is enabled. In this case, MCS buffer stores the information required for MSAA compression algorithm. The size and layout of the MCS buffer is based on per-pixel RT. For 4X and 8X MSAA, MCS buffer element is 8bpp and 32bpp respectively. Height, width and layout of MCS buffer in this case needs must match with Render Target height, width and layout. MCS buffer is tiledY. When MCS buffer is enabled and bound to MSRT, it is required that it is cleared prior to any rendering. A clear value can be specified optionally in the surface state of the corresponding RT. Clear pass for this case requires that scaled down primitive is sent down with upper left co-ordinate to coincide with actual rectangle being cleared. For MSAA, clear rectangle's height and width need to as show in the following table in terms of (width,height) of the RT.

MSAA	Width of Clear Rect	Height of Clear Rect
4X	Ceil($1/8 * \text{width}$)	Ceil($1/2 * \text{height}$)
8X	Ceil($1/2 * \text{width}$)	Ceil($1/2 * \text{height}$)

- Fast Color Clear:** When non multi-sample render target is bond to the pipeline and MSC buffer is enabled, MCS buffer is used as an intermediate (coarse granular) buffer per RT. Hence, MCS buffer is used to improve render target clear. When MCS is buffer is used for color clear of non-multisampler render target, the following restrictions apply.
 - Support is limited to tiled render targets.
 - Support is for non-mip-mapped and non-array surface types only.
 - Clear is supported only on the full RT; i.e., no partial clear or overlapping clears.



The following table describes the RT alignment

	Pixels	Lines
TiledY RT CL		
bpp		
32	8	4
64	4	4
128	2	4
TiledX RT CL		
bpp		
32	16	2
64	8	2
128	4	2

- MCS buffer for non-MSRT is supported only for RT formats 32bpp, 64bpp and 128bpp.
- Clear pass must have a clear rectangle that must follow alignment rules in terms of pixels and lines as shown in the table below. Further, the clear-rectangle height and width must be multiple of the following dimensions. If the height and width of the render target being cleared do not meet these requirements, an MCS buffer can be created such that it follows the requirement and covers the RT.

	Pixels	Lines
TiledY RT		
bpp		
32	128	128
64	64	128
128	32	128
TiledX RT		
bpp		
32	256	64
64	128	64
128	64	64

In order to optimize the performance MCS buffer (when bound to 1X RT) clear similarly to MCS buffer clear for MSRT case, clear rect is required to be scaled by the following factors in the horizontal and vertical directions:

	Horizontal scale down factor	Vertical scale down factor
MCS CL for TiledY RCC		
bpp		
32	64	64
64	32	64
128	16	64
MCS CL for TiledX		



RCC		
bpp		
32	128	32
64	64	32
128	32	32

Following are the SW requirements for MCS buffer clear functionality:

- At the time of Render Target creation, SW needs to create clear-buffer; i.e., MCS buffer.
- At the clear time, clear value for that RT must be programmed and clear enable bit must be set in the surface state of the corresponding RT.
- SW must clear the RT with setting a RT clear bit set in the PS state during the clear pass as described in the following sub-section.
- Since only one RT is bound with a clear pass, only one RT can be cleared at a time. In order to clear multiple RTs, multiple clear passes are required.
- Before binding the “cleared” RT to texture OR honoring a CPU lock OR submitting for flip, SW must ensure a resolve pass. Such a resolve pass is described in the following sub-section.

11.8 Render Target Fast Clear

Fast clear of the render target is performed by setting the **Render Target Fast Clear Enable** field in 3DSTATE_PS and rendering a rectangle. The size of the rectangle is related to the size of the MCS.

The following is required when performing a render target fast clear:

- The render target(s) is/are bound as they normally would be, with the MCS surface defined in SURFACE_STATE.
- A rectangle primitive of the same size as the MCS surface is delivered.
- The pixel shader kernel requires no attributes, and delivers a value of 0xFFFFFFFF in all channels of the render target write message. The replicated color message should be used.
- **Depth Test Enable, Depth Buffer Write Enable, Stencil Test Enable, Stencil Buffer Write Enable, and Alpha Test Enable** must all be disabled.
- After Render target fast clear, pipe-control with color cache write-flush must be issued before sending any DRAW commands on that render target.

11.9 Render Target Resolve

If the MCS is enabled on a non-multisampled render target, the render target must be resolved before being used for other purposes (display, texture, CPU lock). The clear value from SURFACE_STATE is written into pixels in the render target indicated as clear in the MCS. This is done by setting the **Render Target Resolve Enable** field in 3DSTATE_PS and rendering a full render target sized rectangle. Once this is complete, the render target will contain the same contents as it would have had the rendering been performed with MCS surface disabled. In a typical usage model, the render target(s) need to be resolved after rendering and before using it as a source for any consecutive operation.

The following is required when performing a render target resolve:



- PIPE_CONTROL with end of pipe sync must be delivered.
- A rectangle primitive must be scaled down by the following factors with respect to render target being resolved.

Resolve rectangle scaling for TiledY RCC		
	width scale down factor	height scale down factor
bpp		
32	4	2
64	2	2
128	1	2
Resolve rectangle scaling for TiledX RCC		
bpp		
32	8	1
64	4	1
128	2	1

- : The pixel shader kernel requires no attributes, but must deliver a render target write message covering all pixels and all render targets desired to be resolved The color data in these messages is ignored (the replicated color message is required).
- **Depth Test Enable, Depth Buffer Write Enable, Stencil Test Enable, Stencil Buffer Write Enable, and Alpha Test Enable** must all be disabled.

Note that this render target resolve procedure is not supported on multisampled render targets. Unresolved multisampled render targets are directly supported by the sampling engine, which resolves clear values in addition to decompressing the surface This applies to both *ld2dms* and *sample2dms* messages.

11.10 Pixel Shader Thread Generation

After a group of object pixels have been rasterized, the Pixel Shader function is invoked to further compute pixel color/depth information and cause results to be written to rendertargets and/or depth buffers For each pixel, the Pixel Shader calculates the values of the various vertex attributes that are to be interpolated across the object using the interpolation coefficients It then executes an API-supplied Pixel Shader Program Instructions in this program permit the accessing of texture map data, where Texture Samplers are employed to sample and filter texture maps (see the *Shared Functions* chapter) Arithmetic operations can be performed on the texture data, input pixel information and Pixel Shader Constants in order to compute the resultant pixel color/depth The Pixel Shader program also allows the pixel to be discarded from further processing For pixels that are not discarded, the pixel shader must send messages to update one or more render targets with the pixel results.

11.10.1 Pixel Grouping (Dispatch Size) Control

The WM unit can pass a grouping of 2 subspans (8 pixels), 4 subspans (16 pixels) or 8 subspans (32 pixels) to a Pixel Shader thread Software should take into account the following considerations when determining which groupings to support/enable during operation This determination involves a tradeoff of these likely conflicting issues Note that the size of the dispatch has significant impact on the kernel program (it is certainly not transparent to the kernel) Also note that there is no implied spatial relationship between the subspans passed to a PS thread, other than the fact that they come from the same object.

1. **Thread Efficiency:** In general, there is some amount of overhead involved with PS thread dispatch, and if this can be amortized over a larger number of pixels, efficiency will likely



increase This is especially true for very short PS kernels, as may be used for desktop composition, etc.

2. **GRF Consumption:** Processing more pixels per thread will require a larger thread payload and likely more temporary register usage, both of which translate into a requirement for a larger GRF register allocation for the threads. If this increased GRF usage could lead to increased use of scratch space (for spill/fill, etc.) and possibly less efficient use of the Eus (as it would be less likely to find an EU with enough free physical GRF registers to service the thread).
3. **Object Size:** If the number of very small objects (e.g., covering 2 subspans or fewer) is expected to comprise a significant portion of the workload, supporting the 8-pixel dispatch mode may be advantageous Otherwise there could be a large number of 16-pixel dispatches with only 1 or 2 valid subspans, resulting in low efficiency for those threads.
4. **Intangibles:** Kernel footprint & Instruction Cache impact; Complexity;

The groupings of subspans that the WM unit is allowed to include in a PS thread payload is controlled by the **32,16,8 Pixel Dispatch Enable** state variables programmed in WM_STATE. Using these state variables, the WM unit will attempt to dispatch the largest allowed grouping of subspans The following table lists the possible combinations of these state variables.

Please note that, the valid column in table indicates which products supports the combination dispatch. Combinations that are not listed in the table are not available on any product.

A: Valid on all products

B: Valid only on Not valid on if 4x PERPIXEL mode with pixel shader computed depth.

D: Valid on all products, except when in non-1x PERSAMPLE mode (applies to only) .

F: Valid on all products.

Variable Pixel Dispatch

Contiguous 64 Pixel Dispatch Enable	Contiguous 32 Pixel Dispatch Enable	32 Pixel Dispatch Enable	16 Pixel Dispatch Enable	8 Pixel Dispatch Enable	Valid	IP for n-pixel Contiguous Dispatch		IP for n-pixel Dispatch (KSP offsets are in 128-bit instruction units)		
						n=64	n=32	n=32	n=16	n=8
0	0	0	0	1	A					KSP[0]
0	0	0	1	0	F				KSP[0]	
0	0	0	1	1	D				KSP[2]	KSP[0]
0	0	1	0	0	B			KSP[0]		
0	0	1	1	0	E			KSP[1]	KSP[2]	
0	0	1	1	1	D			KSP[1]	KSP[2]	KSP[0]
0	1	0	0	0	C		KSP[0]			
0	1	1	0	0	C		KSP[1]	KSP[0]		
0	1	1	1	0	D		KSP[2]	KSP[1]	KSP[0]	
1	0	0	0	0	C	KSP[0]				
1	0	1	0	0	C	KSP[1]		KSP[0]		
1	0	1	1	0	D	KSP[2]		KSP[1]	KSP[0]	
1	1	0	0	0	C	KSP[1]	KSP[0]			
1	1	1	0	0	C	KSP[2]	KSP[1]	KSP[0]		



:

Each of the four KSP values is separately specified (three for). In addition, each kernel has a separately-specified GRF register count, whereas on , all kernels share the same GRF register count field, with the one with the maximum register count required applying to all

Depending on the subspan grouping selected, the WM unit will modify the starting PS Instruction Pointer (derived from the Kernel Start Pointer in WM_STATE) as a means to inform the PS kernel of the number of subspans included in the payload. The modified IP is a function of the enabled modes and the dispatch size, as shown in the table below.

The driver must ensure that the PS kernel begins with a corresponding jump table to properly handle the number of subspans dispatched. The WM unit will “OR” in the two lsb’s of the Kernel Pointer (bits 5:4) to create an instruction level address (note that the pointer from WM_STATE is 64 byte aligned which hierarchica to four instructions).

If only one dispatch mode is enabled, the Jitter should not include any jump table entries at the beginning of the PS kernel. If multiple dispatch modes are enabled, a two entry jump table should always be inserted, regardless of which modes are enabled (jump table entry for 8 pixel dispatch, followed by jump table entry for 32 pixel dispatch).

Note that for a 32 pixel dispatch, the Windower will hierarch the **Dispatch GRF Start Register for URB Data** state by 2 to account for the extra payload data required. The Pixel Shader kernel needs to comprehend this modification for the 32 pixel kernel code.

```
if (32PixelDispatchEnable && n>7)  
Dispatch 32 Pixels  
else if (16PixelDispatchEnable && (n>2 || !8PixelDispatchEnable))  
Dispatch 16 Pixels  
else  
Dispatch 8 Pixels
```

11.10.1.1 Contiguous Dispatch Modes

There are three cases to consider depending on which dispatch modes are enabled based on the legal combinations in the table above:

Only normal dispatch modes are enabled. This is the normal operating mode in which all features are supported.

Only contiguous dispatch modes are enabled. In this case, software must ensure that the fast composite restrictions are met.

Both normal and contiguous dispatch modes are enabled In this case, a combination of software and the setup kernel must check all of the restrictions required by the contiguous dispatch pixel shader code. The result of the check in the setup kernel is indicated in the message descriptor of the URB write message. The windower then chooses a dispatch mode from either the normal category or the contiguous category depending on whether the restriction check fails or passes, respectively.



If both the 32- and 64-pixel contiguous dispatch modes are enabled together, the windower will choose which one to use based on whether at least one pixel from the upper and lower 8x4 halves of the 8x8 block is active. If one half has no pixel active, the half that does have pixels active will be dispatched as a 32-pixel thread.

The following logic describes how the windower chooses the dispatch mode based on which modes are enabled:

d32 = normal 32-pixel dispatch mode enabled

d16 = normal 16-pixel dispatch mode enabled

d8 = normal 8-pixel dispatch mode enabled

c64 = contiguous 64-pixel dispatch mode enabled

c32 = contiguous 32-pixel dispatch mode enabled

ContiguousSelect = (c64 || c32) &&

[!(d32 || d16 || d8) || RestrictionCheckPass]

For ContiguousSelect true:

<i>contiguous area available</i>	<i>first priority</i>	<i>second priority</i>
<i>both superspan halves</i>	<i>c64</i>	<i>c32</i>
<i>one superspan half</i>	<i>c32</i>	<i>c64</i>

For ContiguousSelect false:

<i>subspans available</i>	<i>first priority</i>	<i>second priority</i>	<i>third priority</i>
<i>s >= 4</i>	<i>d32</i>	<i>d16</i>	<i>d8</i>
<i>4 > s >= 2</i>	<i>d16</i>	<i>d8</i>	<i>d32</i>
<i>2 > s >= 1</i>	<i>d8</i>	<i>d16</i>	<i>d32</i>

11.10.2 Multisampling Effects on Pixel Shader Dispatch

The pixel shader payloads are defined in terms of subspans and pixels. The slots in the pixel shader thread previously mapped 1:1 with pixels. With multisampling, a slot could contain a pixel or may just contain a single sample, depending on the mode. Payload definitions now refer to “slot” to make the definition independent of multisampling mode.

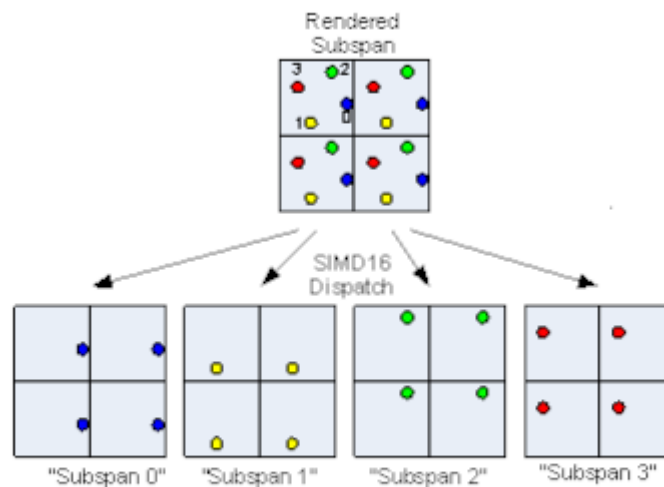
11.10.2.1 MSDISPMODE_PERPIXEL Thread Dispatch

In PERPIXEL mode, the pixel shader kernel still works on 2/4/8 separate subspans, depending on dispatch mode. The fact that rasterization and the depth/stencil tests are being performed on a per-sample (not per-pixel) basis is transparent to the pixel shader kernel.

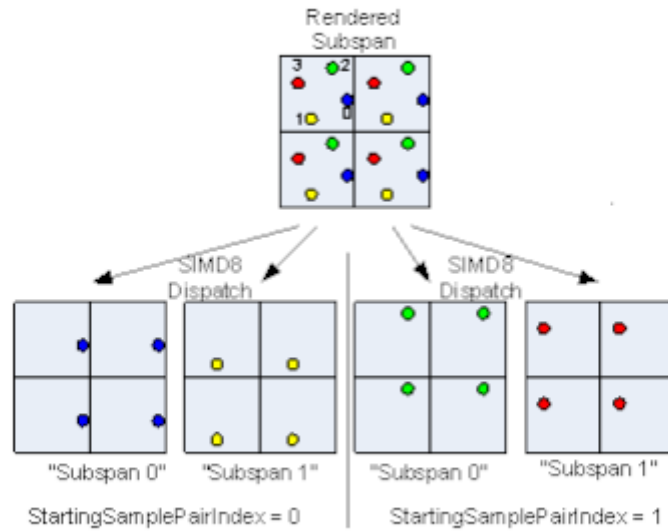
11.10.2.2 MSDISPMODE_PERSAMPLE Thread Dispatch

In PERSAMPLE mode, the pixel shader needs to operate on a sample vs. pixel basis (although this collapses in NUMSAMPLES_1 mode) Instead of processing strictly different subspans in parallel, the PS kernel processes different sample indices of one or more subspans in parallel For example, a SIMD16 dispatch in PERSAMPLE/NUMSAMPLES_4 mode would operate on a single subspan, with the usual "4 Subspan0 pixel slots" used for the "4 Sample0 locations of the (single) subspan" Subspan1 slots would be used for the Sample1 locations, and so on This layout allows the pixel shader to compute derivatives/LOD based on deltas between corresponding sample locations in the subspan in the same fashion as LEGACY pixel shader execution, and as required by DX10.1.

Depending on the dispatch mode (8/16/32 pixels) and multisampling mode (1X/4X), there are different mappings of subspans/samples onto dispatches and slots-within-dispatch In some cases, more than one subspan may be included in a dispatch, while in other cases multiple dispatches are required to process all samples for a single subspan In the latter case, the **StartingSamplePairIndex** value is included in the payload header so the Render Target Write message will access the correct samples with each message.



PERSAMPLE SIMD16 4X Dispatch



PERSAMPLE SIMD8 4X Dispatch

The following table provides the complete dispatch/slot mappings for all the MS/Dispatch combinations.

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
SIMD32	1X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0] Slot[11:8] = Subspan[2].Pixel[3:0].Sample[0] Slot[15:12] = Subspan[3].Pixel[3:0].Sample[0] Slot[19:16] = Subspan[4].Pixel[3:0].Sample[0] Slot[23:20] = Subspan[5].Pixel[3:0].Sample[0] Slot[27:24] = Subspan[6].Pixel[3:0].Sample[0] Slot[31:28] = Subspan[7].Pixel[3:0].Sample[0]
	2X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[1].Pixel[3:0].Sample[0] Slot[15:12] = Subspan[1].Pixel[3:0].Sample[1] Slot[19:16] = Subspan[2].Pixel[3:0].Sample[0] Slot[23:20] = Subspan[2].Pixel[3:0].Sample[1] Slot[27:24] = Subspan[3].Pixel[3:0].Sample[0] Slot[31:28] = Subspan[3].Pixel[3:0].Sample[1]
	4X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]



Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
		Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2] Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3] Slot[19:16] = Subspan[1].Pixel[3:0].Sample[0] Slot[23:20] = Subspan[1].Pixel[3:0].Sample[1] Slot[27:24] = Subspan[1].Pixel[3:0].Sample[2] Slot[31:28] = Subspan[1].Pixel[3:0].Sample[3]
	8X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2] Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3] Slot[19:16] = Subspan[0].Pixel[3:0].Sample[4] Slot[23:20] = Subspan[0].Pixel[3:0].Sample[5] Slot[27:24] = Subspan[0].Pixel[3:0].Sample[6] Slot[31:28] = Subspan[0].Pixel[3:0].Sample[7]
SIMD16	1X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0] Slot[11:8] = Subspan[2].Pixel[3:0].Sample[0] Slot[15:12] = Subspan[3].Pixel[3:0].Sample[0]
	2X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[1].Pixel[3:0].Sample[0] Slot[15:12] = Subspan[1].Pixel[3:0].Sample[1]
	4X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[1].Pixel[3:0].Sample[0] Slot[15:12] = Subspan[1].Pixel[3:0].Sample[1]
	8X	Dispatch[i]: (i=0, 2) SSPI = i Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1] Slot[11:8] = Subspan[0].Pixel[3:0].Sample[SSPI*2+2]



Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
		<i>Slot[15:12] = Subspan[0].Pixel[3:0].Sample[SSPI*2+3]</i>
	16X	<i>Dispatch[i]: (i=0, 2, 4, 6)</i> <i>SSPI = i</i> <i>Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0]</i> <i>Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]</i> <i>Slot[11:8] = Subspan[0].Pixel[3:0].Sample[SSPI*2+2]</i> <i>Slot[15:12] = Subspan[0].Pixel[3:0].Sample[SSPI*2+3]</i>
SIMD8	1X	<i>Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]</i> <i>Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0]</i>
	2X	<i>Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]</i> <i>Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1]</i>
	4X	<i>Dispatch[i]: (i=0..1)</i> <i>SSPI = i</i> <i>Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0]</i> <i>Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]</i>
	8X	<i>Dispatch[i]: (i=0, 1, 2, 3)</i> <i>SSPI = i</i> <i>Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0]</i> <i>Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]</i>
	16X	<i>Dispatch[i]: (i=0, 1, 2, 3, 4, 5, 6, 7)</i> <i>SSPI = i</i> <i>Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0]</i> <i>Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]</i>

11.10.3 PS Thread Payload for Normal Dispatch

The following table lists all possible contents included in a PS thread payload, in the order they are provided. Certain portions of the payload are optional, in which case the corresponding phase is skipped.

This payload does not apply to the contiguous dispatch modes. The payload for these modes are documented in the section titled *PS Thread Payload for Contiguous Dispatch*.

11.10.3.1 PS Thread Payload for Normal Dispatch

The following payload (UNRESOLVED CROSS REFERENCE, PS Thread Payload for Normal Dispatch) applies to all registers are numbered starting at 0, but many registers are skipped depending on



configuration This causes all registers below to be renumbered to fill in the skipped locations The only case where actual registers may be skipped is immediately before the constant data and again before the setup data.

PS Thread Payload for Normal Dispatch

Dword	Bit	Description
R0.7	31	
	30:24	Reserved
	23:0	<p>Primitive Thread ID: This field contains the primitive thread count passed to the Windower from the Strips Fans Unit.</p> <p>Format: Reserved for HW Implementation Use.</p>
R0.6	31:24	Reserved
	23:0	<p>Thread ID: This field contains the thread count which is incremented by the Windower for every thread that is dispatched.</p> <p>Format: Reserved for HW Implementation Use.</p>
R0.5	31:10	<p>Scratch Space Pointer: Specifies the 1K-byte aligned pointer to the scratch space available for this PS thread This is specified as an offset to the General State Base Address.</p> <p>Format = GeneralStateOffset[31:10]</p>
	9:8	Reserved
	7:0	<p>FFTID: This ID is assigned by the WM unit and is a identifier for the thread It is used to free up resources used by the thread upon thread completion.</p> <p>Format: Reserved for HW Implementation Use.</p>
R0.4	31:5	<p>Binding Table Pointer: Specifies the 32-byte aligned pointer to the Binding Table It is specified as an offset from the Surface State Base Address.</p> <p>Format = SurfaceStateOffset[31:5]</p>
	4:0	Reserved
R0.3	31:5	<p>Sampler State Pointer: Specifies the 32-byte aligned pointer to the Sampler State table It is specified as an offset from the Dynamic State Base Address.</p> <p>Format = DynamicStateOffset[31:5]</p>
	4	Reserved
	3:0	<p>Per Thread Scratch Space: Specifies the amount of scratch space allowed to be used by this thread.</p> <p>Programming Notes:</p> <ul style="list-style-type: none"> This amount is available to the kernel for information only It will be passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access



Dword	Bit	Description														
		<p>messages, but the Data Port will ignore it</p> <p>Format = U4</p> <p>Range = [0,11] indicating [1k bytes, 2M bytes] in powers of two</p>														
R0.2	31:0	Reserved : delivered as zeros (reserved for message header fields)														
R0.1	31:6	<p>Color Calculator State Pointer: Specifies the 64-byte aligned pointer to the Color Calculator state (COLOR_CALC_STATE structure in memory) It is specified as an offset from the Dynamic State Base Address This value is eventually passed to the ColorCalc function in the DataPort and is used to fetch the corresponding CC_STATE data.</p> <p>Format = DynamicStateOffset[31:5]</p>														
	5:0	Reserved														
R0.0	31	Reserved														
	30:27	<p>Viewport Index Specifies the index of the viewport currently being used.</p> <p>Format = U4</p> <p>Range = [0,15]</p>														
	26:16	<p>Render Target Array Index: Specifies the array index to be used for the following surface types:</p> <p>SURFTYPE_1D: specifies the array index Range = [0,2047]</p> <p>SURFTYPE_2D: specifies the array index Range = [0,2047]</p> <p>SURFTYPE_3D: specifies the “r” coordinate Range = [0,2047]</p> <p>SURFTYPE_CUBE: specifies the face identifier Range = [0,5]</p> <table border="1" data-bbox="488 1297 821 1520"> <thead> <tr> <th>face</th> <th>Render Target Array Index</th> </tr> </thead> <tbody> <tr> <td>+x</td> <td>0</td> </tr> <tr> <td>-x</td> <td>1</td> </tr> <tr> <td>+y</td> <td>2</td> </tr> <tr> <td>-y</td> <td>3</td> </tr> <tr> <td>+z</td> <td>4</td> </tr> <tr> <td>-z</td> <td>5</td> </tr> </tbody> </table> <p>Format = U11</p>	face	Render Target Array Index	+x	0	-x	1	+y	2	-y	3	+z	4	-z	5
face	Render Target Array Index															
+x	0															
-x	1															
+y	2															
-y	3															
+z	4															
-z	5															
	15	<p>Front/Back Facing Polygon: Determines whether the polygon is front or back facing Used by the render cache to determine which stencil test state to use.</p> <p>0: Front Facing</p> <p>1: Back Facing</p>														
	14	Reserved														



Dword	Bit	Description
	13	Source Depth to Render Target: Indicates that source depth will be sent to the render target
	12	oMask to Render Target: Indicates that oMask will be sent to the render target
	11:9	Reserved
	8	Reserved for expansion of Starting Sample Pair Index
	7:6	Starting Sample Pair Index: indicates the index of the first sample pair of the dispatch Format = U2 Range = [0,3]
	5	Reserved
	4:0	Primitive Topology Type: This field identifies the Primitive Topology Type associated with the primitive spawning this object The WM unit does not modify this value (e.g., objects within POINTLIST topologies see POINTLIST). Format: (See 3DPRIMITIVE command in <i>3D Pipeline</i>)
R1.7	31:16	Pixel/Sample Mask (SubSpan[3:0]) : Indicates which pixels within the four subspans are lit If 32 pixel dispatch is enabled, this field contains the pixel mask for the first four subspans. Note: This is not a duplicate of the Dispatch Mask that is delivered to the thread The dispatch mask has all pixels within a subspan as active if any of them are lit to enable LOD calculations to occur correctly. This field must not be modified by the Pixel Shader kernel.
	15:0	Pixel/Sample Mask Copy (SubSpan[3:0]) : This is a duplicate copy of the pixel mask This copy can be modified as the pixel shader thread executes in order to turn off pixels based on kill instructions.
R1.6	31:0	Reserved
R1.5	31:16	Y3: Y coordinate (screen space) for upper-left pixel of subspan 3 (slot 12) Format = U16
	15:0	X3: X coordinate (screen space) for upper-left pixel of subspan 3 (slot 12) Format = U16
R1.4	31:16	Y2 : Y coordinate (screen space) for upper-left pixel of subspan 2 (slot 8) Format = U16
	15:0	X2 : X coordinate (screen space) for upper-left pixel of subspan 2 (slot 8) Format = U16
R1.3	31:16	Y1 : Y coordinate (screen space) for upper-left pixel of subspan 1 (slot 4)



Dword	Bit	Description
		Format = U16
	15:0	X1 : X coordinate (screen space) for upper-left pixel of subspan 1 (slot 4) Format = U16
R1.2	31:16	Y0 : Y coordinate (screen space) for upper-left pixel of subspan 0 (slot 0) Format = U16
	15:0	X0 : X coordinate (screen space) for upper-left pixel of subspan 0 (slot 0) Format = U16
R1.1	31:0	Reserved
R1.0	31:16	Reserved
	15:12	Slot 3 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7] 16X MSAA range [0..15]
	11:8	Slot 2 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7] 16X MSAA range [0..15]
	7:4	Slot 1 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3]



Dword	Bit	Description
		8X MSAA range [0..7] 16X MSAA range [0..15]
	3:0	Slot 0 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7] 16X MSAA range [0..15]
		R2: delivered only if this is a <i>32-pixel dispatch</i> .
R2.7	31:16	Pixel/Sample Mask (SubSpan[7:4]) : Indicates which pixels within the upper four subspans are lit This field is valid only when the 32 pixel dispatch state is enabled This field must not be modified by the pixel shader thread. Note: This is not a duplicate of the dispatch mask that is delivered to the thread The dispatch mask has all pixels within a subspan as active if any of them are lit to enable LOD calculations to occur correctly. This field must not be modified by the Pixel Shader kernel.
	15:0	Pixel/Sample Mask Copy (SubSpan[7:4]) : This is a duplicate copy of pixel mask for the upper 16 pixels This copy will be modified as the pixel shader thread executes to turn off pixels based on kill instructions.
R2.6	31:0	Reserved
R2.5	31:16	Y7: Y coordinate (screen space) for upper-left pixel of subspan 7 (slot 28) Format = U16
	15:0	X7: X coordinate (screen space) for upper-left pixel of subspan 7 (slot 28) Format = U16
R2.4	31:16	Y6
	15:0	X6
R2.3	31:16	Y5
	15:0	X5
R2.2	31:16	Y4
	15:0	X4
R2.1	31:0	Reserved
R2.0	31:15	Reserved



Dword	Bit	Description
	15:12	Slot 7 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7] 16X MSAA range [0..15]
	11:8	Slot 6 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7] 16X MSAA range [0..15]
	7:4	Slot 5 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7] 16X MSAA range [0..15]
	3:0	Slot 4 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7] 16X MSAA range [0..15]



Dword	Bit	Description
		R3-R26: delivered only if the corresponding Barycentric Interpolation Mode bit is set Register phases containing Slot 8-15 data are not delivered in <i>8-pixel dispatch</i> mode.
R3.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 7 This and the next register phase is only included if the corresponding enable bit in Barycentric Interpolation Mode is set. Format = IEEE_Float
R3.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 6
R3.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 5
R3.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 4
R3.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 3
R3.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 2
R3.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 1
R3.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 0
R4		Perspective Pixel Location Barycentric[2] for Slots 7:0
R5.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 15
R5.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 14
R5.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 13
R5.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 12
R5.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 11
R5.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 10
R5.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 9
R5.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 8
R6		Perspective Pixel Location Barycentric[2] for Slots 15:8
R7:10		Perspective Centroid Barycentric
R11:14		Perspective Sample Barycentric
R15:18		Linear Pixel Location Barycentric
R19:22		Linear Centroid Barycentric
R23:26		Linear Sample Barycentric
		R27: delivered only if Pixel Shader Uses Source Depth is set.
R27.7	31:0	Interpolated Depth for Slot 7 Format = IEEE_Float This and the next register phase is only included if Pixel Shader Uses Source Depth (WM_STATE) is set.
R27.6	31:0	Interpolated Depth for Slot 6
R27.5	31:0	Interpolated Depth for Slot 5
R27.4	31:0	Interpolated Depth for Slot 4
R27.3	31:0	Interpolated Depth for Slot 3
R27.2	31:0	Interpolated Depth for Slot 2
R27.1	31:0	Interpolated Depth for Slot 1
R27.0	31:0	Interpolated Depth for Slot 0
		R28: delivered only if Pixel Shader Uses Source Depth is set and this is not an <i>8-pixel dispatch</i> .
R28.7	31:0	Interpolated Depth for Slot 15



Dword	Bit	Description
R28.6	31:0	Interpolated Depth for Slot 14
R28.5	31:0	Interpolated Depth for Slot 13
R28.4	31:0	Interpolated Depth for Slot 12
R28.3	31:0	Interpolated Depth for Slot 11
R28.2	31:0	Interpolated Depth for Slot 10
R28.1	31:0	Interpolated Depth for Slot 9
R28.0	31:0	Interpolated Depth for Slot 8
		R29: delivered only if Pixel Shader Uses Source W is set.
R29.7	31:0	Interpolated W for Slot 7 Format = IEEE_Float This and the next register phase is only included if Pixel Shader Uses Source W (WM_STATE) is set
R29.6	31:0	Interpolated W for Slot 6
R29.5	31:0	Interpolated W for Slot 5
R29.4	31:0	Interpolated W for Slot 4
R29.3	31:0	Interpolated W for Slot 3
R29.2	31:0	Interpolated W for Slot 2
R29.1	31:0	Interpolated W for Slot 1
R29.0	31:0	Interpolated W for Slot 0
		R30: delivered only if Pixel Shader Uses Source W is set and this is not an <i>8-pixel dispatch</i> .
R30.7	31:0	Interpolated W for Slot 15
R30.6	31:0	Interpolated W for Slot 14
R30.5	31:0	Interpolated W for Slot 13
R30.4	31:0	Interpolated W for Slot 12
R30.3	31:0	Interpolated W for Slot 11
R30.2	31:0	Interpolated W for Slot 10
R30.1	31:0	Interpolated W for Slot 9
R30.0	31:0	Interpolated W for Slot 8
		R31: delivered only if Position XY Offset Select is either POSOFFSET_CENTROID or POSOFFSET_SAMPLE
R31.7	31:24	Position Offset Y for Slot 15 This field contains either the CENTROID or SAMPLE position offset for Y, depending on the state of Position XY Offset Select . Format = U4.4 Range = [0.0,1.0)
	23:16	Position Offset X for Slot 15 This field contains either the CENTROID or SAMPLE position offset for X, depending on the state of Position XY Offset Select . Format = U4.4 Range = [0.0,1.0)



Dword	Bit	Description
	15:8	Position Offset Y for Slot 14
	7:0	Position Offset X for Slot 14
R31.6	31:24	Position Offset Y for Slot 13
	23:16	Position Offset X for Slot 13
	15:8	Position Offset Y for Slot 12
	7:0	Position Offset X for Slot 12
R31.5:4		Position Offset X/Y for Slot[11:8]
R31.3:2		Position Offset X/Y for Slot[7:4]
R31.1:0		Position Offset X/Y for Slot[3:0]
		R32: delivered only if Pixel Shader Uses Input Coverage Mask is set.
R32.7	31:0	Input Coverage Mask for Slot 7 Format = U32 This and the next register phase is only included if Pixel Shader Uses Input Coverage Mask (3DSTATE_PS) is set.
R32.6	31:0	Input Coverage Mask for Slot 6
R32.5	31:0	Input Coverage Mask for Slot 5
R32.4	31:0	Input Coverage Mask for Slot 4
R32.3	31:0	Input Coverage Mask for Slot 3
R32.2	31:0	Input Coverage Mask for Slot 2
R32.1	31:0	Input Coverage Mask for Slot 1
R32.0	31:0	Input Coverage Mask for Slot 0
		R33: delivered only if Pixel Shader Uses Input Coverage Mask is set and this is not an 8-pixel dispatch.
R33.7	31:0	Input Coverage Mask for Slot 15
R33.6	31:0	Input Coverage Mask for Slot 14
R33.5	31:0	Input Coverage Mask for Slot 13
R33.4	31:0	Input Coverage Mask for Slot 12
R33.3	31:0	Input Coverage Mask for Slot 11
R33.2	31:0	Input Coverage Mask for Slot 10
R33.1	31:0	Input Coverage Mask for Slot 9
R33.0	31:0	Input Coverage Mask for Slot 8
		R34-R57: delivered only if the corresponding Barycentric Interpolation Mode bit is set and this is a 32-pixel dispatch.
R34.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 23 This and the next register phase is only included if the corresponding enable bit in Barycentric Interpolation Mode is set. Format = IEEE_Float
R34.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 22
R34.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 21
R34.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 20
R34.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 19
R34.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 18
R34.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 17



Dword	Bit	Description
R34.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 16
R35		Perspective Pixel Location Barycentric[2] for Slots 23:16
R36.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 31
R36.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 30
R36.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 29
R36.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 28
R36.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 27
R36.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 26
R36.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 25
R36.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 24
R37		Perspective Pixel Location Barycentric[2] for Slots 31:24
R38:41		Perspective Centroid Barycentric
R42:45		Perspective Sample Barycentric
R46:49		Linear Pixel Location Barycentric
R50:53		Linear Centroid Barycentric
R54:57		Linear Sample Barycentric
		R58-R59: delivered only if Pixel Shader Uses Source Depth is set and this is a <i>32-pixel dispatch</i> .
R58.7	31:0	Interpolated Depth for Slot 23 Format = IEEE_Float This and the next register phase is only included if Pixel Shader Uses Source Depth (WM_STATE) bit is set.
R58.6	31:0	Interpolated Depth for Slot 22
R58.5	31:0	Interpolated Depth for Slot 21
R58.4	31:0	Interpolated Depth for Slot 20
R58.3	31:0	Interpolated Depth for Slot 19
R58.2	31:0	Interpolated Depth for Slot 18
R58.1	31:0	Interpolated Depth for Slot 17
R58.0	31:0	Interpolated Depth for Slot 16
R59.7	31:0	Interpolated Depth for Slot 31
R59.6	31:0	Interpolated Depth for Slot 30
R59.5	31:0	Interpolated Depth for Slot 29
R59.4	31:0	Interpolated Depth for Slot 28
R59.3	31:0	Interpolated Depth for Slot 27
R59.2	31:0	Interpolated Depth for Slot 26
R59.1	31:0	Interpolated Depth for Slot 25
R59.0	31:0	Interpolated Depth for Slot 24
		R60-R61: delivered only if Pixel Shader Uses Source W is set and this is a <i>32-pixel dispatch</i> .
R60.7	31:0	Interpolated W for Slot 23 Format = IEEE_Float This and the next register phase is only included if Pixel Shader Uses Source W (WM_STATE) bit is set.
R60.6	31:0	Interpolated W for Slot 22
R60.5	31:0	Interpolated W for Slot 21



Dword	Bit	Description
R60.4	31:0	Interpolated W for Slot 20
R60.3	31:0	Interpolated W for Slot 19
R60.2	31:0	Interpolated W for Slot 18
R60.1	31:0	Interpolated W for Slot 17
R60.0	31:0	Interpolated W for Slot 16
R61.7	31:0	Interpolated W for Slot 31
R61.6	31:0	Interpolated W for Slot 30
R61.5	31:0	Interpolated W for Slot 29
R61.4	31:0	Interpolated W for Slot 28
R61.3	31:0	Interpolated W for Slot 27
R61.2	31:0	Interpolated W for Slot 26
R61.1	31:0	Interpolated W for Slot 25
R61.0	31:0	Interpolated W for Slot 24
		R62: delivered only if Position XY Offset Select is either POSOFFSET_CENTROID or POSOFFSET_SAMPLE and this is a <i>32-pixel dispatch</i> .
R62.7	31:24	Position Offset Y for Slot 31 This field contains either the CENTROID or SAMPLE position offset for Y, depending on the state of Position XY Offset Select . Format = U4.4 Range = [0.0,1.0)
	23:16	Position Offset X for Slot 31 This field contains either the CENTROID or SAMPLE position offset for X, depending on the state of Position XY Offset Select . Format = U4.4 Range = [0.0,1.0)
	15:8	Position Offset Y for Slot 30
	7:0	Position Offset X for Slot 30
R62.6	31:24	Position Offset Y for Slot 29
	23:16	Position Offset X for Slot 29
	15:8	Position Offset Y for Slot 28
	7:0	Position Offset X for Slot 28
R62.5:4		Position Offset X/Y for Slot[27:24]
R62.3:2		Position Offset X/Y for Slot[23:20]
R62.1:0		Position Offset X/Y for Slot[19:16]
		R63-R64: delivered only if Pixel Shader Uses Input Coverage Mask is set and this is a <i>32-pixel dispatch</i> .
R63.7	31:0	Input Coverage Mask for Slot 23 Format = U32 This and the next register phase is only included if Pixel Shader Uses Input Coverage Mask (3DSTATE_PS) is set.
R63.6	31:0	Input Coverage Mask for Slot 22



Dword	Bit	Description
R63.5	31:0	Input Coverage Mask for Slot 21
R63.4	31:0	Input Coverage Mask for Slot 20
R63.3	31:0	Input Coverage Mask for Slot 19
R63.2	31:0	Input Coverage Mask for Slot 18
R63.1	31:0	Input Coverage Mask for Slot 17
R63.0	31:0	Input Coverage Mask for Slot 16
R64.7	31:0	Input Coverage Mask for Slot 31
R64.6	31:0	Input Coverage Mask for Slot 30
R64.5	31:0	Input Coverage Mask for Slot 29
R64.4	31:0	Input Coverage Mask for Slot 28
R64.3	31:0	Input Coverage Mask for Slot 27
R64.2	31:0	Input Coverage Mask for Slot 26
R64.1	31:0	Input Coverage Mask for Slot 25
R64.0	31:0	Input Coverage Mask for Slot 24
		Optional Padding before the Start of Constant/Setup Data The locations between the end of the Optional Payload Header and the location programmed via Dispatch GRF Start Register for Constant/Setup Data are considered “padding” and Reserved (see below)
optional, multiple of 8 DWs	31:0	Reserved
		The Dispatch GRF Start Register for Constant/Setup Data state variable in 3DSTATE_WM is used to define the starting location of the constant and setup data within the PS thread payload This control is provided to allow this data to be located at a fixed location within thread payloads, regardless of the amount of data in the Optional Payload Header This permits the kernel to use direct GRF addressing to access the constant/setup data, regardless of the optional parameters being passed (as these are determined on-the-fly by the WM unit)
		Constant Data (optional) : Some amount of constant data (possible none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_PS command (taking the buffer enables into account). The Constant Data arrives in a non-interleaved format.
Optional, multiple of 8 DWs	31:0	Constant Data
		Setup Data (Attribute Vertex Deltas) Output data from the SF stage is delivered in the PS thread payload The amount of data is determined by the Number of Output Attributes field Each register contains two channels of one attribute Thus, the total number of registers sent is equal to 2 * Number of Output Attributes.
Rp.7	31:0	a0[0].y – a0 vertex delta for Attribute0.y Format = IEEE_Float



Dword	Bit	Description
Rp.6	31:0	Reserved
Rp.5	31:0	a2[0].y – a2 vertex delta for Attribute0.y Format = IEEE_Float
Rp.4	31:0	a1[0].y – a1 vertex delta for Attribute0.y Format = IEEE_Float
Rp.3	31:0	a0[0].x – a0 vertex delta for Attribute0.x
Rp.2	31:0	Reserved
Rp.1	31:0	a2[0].x – a2 vertex delta for Attribute0.x
Rp.0	31:0	a1[0].x – a1 vertex delta for Attribute0.x
R(p+1).7	31:0	a0[0].w – a0 vertex delta for Attribute0.w
R(p+1).6	31:0	Reserved
R(p+1).5	31:0	a2[0].w – a2 vertex delta for Attribute0.w
R(p+1).4	31:0	a1[0].w – a1 vertex delta for Attribute0.w
R(p+1).3	31:0	a0[0].z – a0 vertex delta for Attribute0.z
R(p+1).2	31:0	Reserved
R(p+1).1	31:0	a2[0].z – a2 vertex delta for Attribute0.z
R(p+1).0	31:0	a1[0].z – a1 vertex delta for Attribute0.z
R(p+2):Rq		Vertex deltas for additional attributes in numerical order See definition of Rp and R(p+1) for formats.

11.11 Other WM Functions

11.11.1 Statistics Gathering

If **Statistics Enable** is set in WM_STATE or 3DSTATE_WM, the Windower increments the PS_INVOCATIONS_COUNT register once for each unmasked pixel (or sample) that is *dispatched* to a Pixel Shader thread.

PS_INVOCATIONS_COUNT register counts all the pixels/samples present in a 2X2 dispatched to Pixel Shader.

If **Early Depth Test Enable** is set it is possible for pixels or samples to be discarded prior to reaching the Pixel Shader due to failing the depth or stencil test PS_INVOCATIONS_COUNT will still be incremented for these pixels or samples since the depth test occurs after the pixel shader from the point of view of SW.



A0 Erratum BWT004 states that there is no way to indicate a true “null” pixel shader (in the sense that the pixel shader dispatch will be skipped). The “dummy” PS thread required for a “null” pixel shader will still cause PS_INVOCATIONS_COUNT to increment on pixel dispatches; if the “null” pixel dispatches are not to be counted, **Statistics Enable** must be *cleared* when changing to a “null” pixel shader. Clearing **Statistics Enable** may also prevent PS_DEPTH_COUNT from incrementing properly. Therefore, in certain pipeline configurations, it may be *impossible* to maintain both PS_INVOCATIONS_COUNT and PS_DEPTH_COUNT accurately.

When Early Depth Test is forced and when Statistics Enable is set, PS_INVOCATIONS_COUNT register may not have the correct value.

12. 3D Pipeline – Color Calculator (Output Merger)

12.1 Overview

Note:The Color Calculator logic resides in the Render Cache backing Data Port (DAP) shared function. It is described in this chapter as the Color Calc functions are naturally an extension of the 3D pipeline past the WM stage. See the DataPort chapter for details on the messages used by the Pixel Shader to invoke Color Calculator functionality.

The *Color Calculator* (referred to as “Output Merger in the DX Spec) function within the Data Port shared function completes the processing of rasterized pixels after the pixel color and depth have been computed by the Pixel Shader. This processing is initiated when the pixel shader thread sends a Render Target Write message (see *Shared Functions*) to the Render Cache. (Note that a single pixel shader thread may send multiple Render Target Write messages, with the result that multiple render targets get updated). The pixel variables pass through a pipeline of fixed (yet programmable) functions, and the results are conditionally written into the appropriate buffers.

The word “pixel” used in this section is effectively replaced with the word “sample” if multisample rasterization is enabled.

Pipeline Stage	Description
Alpha Coverage	It generates coverage masks using AlphaToCoverage AND/OR AlphaToOne functions based on src0.alpha.
Alpha Test	Compare pixel alpha with reference alpha and conditionally discard pixel
Stencil Test	Compare pixel stencil value with reference and forward result to Buffer Update stage
Depth Test	Compare pix.Z with corresponding Z value in the Depth Buffer and forward result to Buffer Update stage
Color Blending	Combine pixel color with corresponding color in color buffer according to programmable function
Gamma Correction	Adjust pixel’s color according to gamma function for SRGB destination surfaces.
Color Quantization	Convert “full precision” pixel color values to fixed precision of the color buffer format
Logic Ops	Combine pixel color logically with existing color buffer color (mutually exclusive with Color Blending)
Buffer Update	Write final pixel values to color and depth buffers or discard pixel without update

The following logic describes the high-level operation of the Pixel Processing pipeline:

```
PixelProcessing() {
AlphaCoverage()
AlphaTest()
DepthBufferCoordinateOffsetDisable
StencilTest()
DepthTest()
ColorBufferBlending()
}
```



```
GammaCorrection()  
ColorQuantization()  
LogicalOps()  
BufferUpdate()  
}
```

12.1.1 Alpha Coverage

Alpha coverage logic is supported and can be controlled using three state variables:

- **AlphaToCoverage Enable**, when enabled Color Calculator modifies the sample mask. This function (along with AlphaToOne) come at the top of the pixel pipeline. The sample's Source0.Alpha value (possibly being replicated from the pixel's Source0.Alpha) is used to compute a (optionally dithered) 1/2/4-bit mask (depending on NumSamples).
- The **AlphaToCoverage Dither Enable** SV is used to control the dithering of the AlphaToCoverage mask. The bit corresponding to the sample# is then ANDed with the sample's incoming mask bits – allowing the sample to be masked off depending on alpha.
- **AlphaToOne Enable**, when enabled, Color Calculator must replace Source0.Alpha (if present) with 1.0f.
- If AlphaToCoverage is disabled, AlphaToCoverage Dither does not have any impact.

NOTE:

- Src0.alpha needs to be first multiplied with AA alpha before applying AlphaToCoverage and AlphaToOne functions.
- An alpha value of NaN results in a no coverage (zero) mask.
- Alpha values from the pixel shader are treated as FLOAT32 format for computing the AlphaToCoverage Mask.

12.1.2 Alpha Test

The Alpha Test function can be used to discard pixels based on a comparison between the incoming pixel's alpha value and the **Alpha Test Reference** state variable in COLOR_CALC_STATE. This operation can be used to remove transparent or nearly-transparent pixels, though other uses for the alpha channel and alpha test are certainly possible.

This function is enabled by the **Alpha Test Enable** state variable in COLOR_CALC_STATE. If ENABLED, this function compares the incoming pixel's alpha value (*pixColor.Alpha*) and the reference alpha value specified by via the **Alpha Test Reference** state variable in COLOR_CALC_STATE. The comparison performed is specified by the **Alpha Test Function** state variable in COLOR_CALC_STATE.

The **Alpha Test Format** state variable is used to specify whether Alpha Test is performed using fixed-point (UNORM8) or FLOAT32 values. Accordingly, it determines whether the **Alpha Reference Value** is passed in a UNORM8 or FLOAT32 format. If UNORM8 is selected, the pixel's alpha value will be converted from floating-point to UNORM8 before the comparison.

Pixels that pass the Alpha Test proceed for further processing. Those that fail are discarded at this point in the pipeline.

If **Alpha Test Enable** is DISABLED, this pipeline stage has no effect.



The Alpha Test function is supported in conjunction with Multiple Render Targets (MRTs). If delivered in the incoming render target write message, source 0 alpha is used to perform the alpha test. If source 0 alpha is not delivered, the normal alpha value is used to perform the alpha test.

12.1.3 Depth Coordinate Offset

The Depth Coordinate Offset function applies a programmable constant offset to the RenderTarget X,Y screen space coordinates in order to generate DepthBuffer coordinates.

The function has been specifically added to allow the OpenGL driver to deal with a RenderTarget and DepthBuffer of differing sizes.

In contrast, OpenGL defines a lower-left screen coordinate origin. This requires the driver to incorporate a “Y coordinate flipping” transformation into the viewport mapping function. The Y extent of the RT is used in this flipping transformation. If the DepthBuffer extent is different, the wrong pixel Y locations within the DepthBuffer will be accessed.

The least expensive solution is to provide a translation offset to be applied to the post-viewport-mapped DepthBuffer Y pixel coordinate, effectively allowing the alignment of the lower-left origins of the RT and DepthBuffer. [Note that the previous DBCOD feature performed an optional translation of post-viewport-mapping RT pixel (screen) coordinates to generate DepthBuffer pixel (window) coordinates. Specifically, the Draw Rect Origin X,Y state could be subtracted from the RT pixel coordinates.]

This function uses **Depth Coordinate Offset X,Y** state (signed 16-bit values in 3DSTATE_DEPTH_RECTANGLE) that is *unconditionally* added to the RT pixel coordinates to generate DepthBuffer pixel coordinates.

The previous DBCOD feature can be supported by having the driver program Depth Coordinate X,Y Offset to the two’s complement of the the Draw Rect Origin. By programming Depth Coordinate X,Y Offset to zeros, the current “normal” operation (DBCOD disabled) can be achieved.

Programming Restrictions:

- Only simple 2D RTs are supported (no mipmaps)
- Software must ensure that the resultant DepthBuffer Coordinate X,Y values are non-negative.
- There are alignment restrictions – see 3DSTATE_DEPTH_BUFFER command.

12.1.4 Stencil Test

The Stencil Test function can be used to discard pixels based on a comparison between the [**Backface**] **Stencil Test Reference** state variable and the pixel’s stencil value. This is a general purpose function used for such effects as shadow volumes, per-pixel clipping, etc. The result of this comparison is used in the Stencil Buffer Update function later in the pipeline.

This function is enabled by the **Stencil Test Enable** state variable. If ENABLED, the current stencil buffer value for this pixel is read.

Programming Note:

- If the Depth Buffer is either undefined or does *not* have a surface format of D32_FLOAT_S8X24_UINT or D24_UNORM_S8_UINT and separate stencil buffer is disabled, **Stencil Test Enable** must be DISABLED.

A 2nd set of the stencil test state variables is provided so that pixels from back-facing objects, assuming they are not culled, can have a stencil test performed on them separate from the test for normal front-facing objects. The separate stencil test for back-facing objects can be enabled via the **Double Sided**



Stencil Enable state variable. Otherwise, non-culled back-facing objects will use the same test function, mask and reference value as front-facing objects. The 2nd stencil state for back-facing objects is most commonly used to improve the performance of rendering shadow volumes which require a different stencil buffer operation depending on whether pixels rendered are from a front-facing or back-facing object. The backface stencil state removes the requirement to render the shadow volumes in 2 passes or sort the objects into front-facing and back-facing lists.

The remainder of this subsection describes the function in term of **[Backface] <state variable name>**. The Backface set of state variables are only used if Double Sided Stencil Enable is ENABLED and the object is considered back-facing. Otherwise the normal (front-facing) state variables are used.

This function then compares the **[Backface] Stencil Test Reference** value and the pixel's stencil value value after logically ANDing both values by **[Backface] Stencil Test Mask**. The comparison performed is specified by the **[Backface] Stencil Test Function** state variable. The result of the comparison is passed down the pipeline for use in the Stencil Buffer Update function. The Stencil Test function does not in itself discard pixels.

If **Stencil Test Enable** is DISABLED, a result of "stencil test passed" is propagated down the pipeline.

12.1.5 Depth Test

The Depth Test function can be used to discard pixels based on a comparison between the incoming pixel's depth value and the current depth buffer value associated with the pixel. This function is typically used to perform the "Z Buffer" hidden surface removal. The result of this pipeline function is used in the Stencil Buffer Update function later in the pipeline.

This function is enabled by the **Depth Test Enable** state variable. If enabled, the pixel's ("source") depth value is first computed. After computation the pixel's depth value is clamped to the range defined by **Minimum Depth** and **Maximum Depth** in the selected CC_VIEWPORT state. Then the current ("destination") depth buffer value for this pixel is read.

This function then compares the source and destination depth values. The comparison performed is specified by the **Depth Test Function** state variable.

The result of the comparison is propagated down the pipeline for use in the subsequent Depth Buffer Update function. The Depth Test function does not in itself discard pixels.

If **Depth Test Enable** is DISABLED, a result of "depth test passed" is propagated down the pipeline.

Programming Note:

- Enabling the Depth Test function without defining a Depth Buffer is UNDEFINED.

12.1.6 Pre-Blend Color Clamping

Pre-Blend Color Clamping, controlled via **Pre-Blend Color Clamp Enable** OR Pre-Blend Source Only Clamp Enable and **Color Clamp Range** states in COLOR_CALC_STATE, is affected by the enabling of Color Buffer Blend as described below.

The following table summarizes the requirements involved with Pre-/Post-Blend Color Clamping.

Blending	RT Format	Pre-Blend Color Clamp	Post-Blend Color Clamp
Off	UNORM, UNORM_SRGB,YCRCB	Must be enabled with range = RT range or [0,1] (same function)	n/a, state ignored
	SNORM	Must be enabled with range = RT range or [-1,1] (same function)	n/a, state ignored
	FLOAT (except for	Must be enabled (with any desired	n/a, state ignored



Blending	RT Format	Pre-Blend Color Clamp	Post-Blend Color Clamp
	R11G11B10_FLOAT)	range)	
	R11G11B10_FLOAT	Must be enabled with either [0,1] or RT range	n/a, state ignored
	UINT, SINT	State ignored, implied clamp to RT range	n/a, state ignored
On (where permitted)	UNORM, UNORM_SRGB	Must be enabled with range = RT range or [0,1] (same function)	Must be enabled with range = RT range or [0,1] (same function)
	SNORM	Must be enabled with range = RT range or [-1,1] (same function)	Must be enabled with range = RT range or [-1,1] (same function)
	FLOAT (except for R11G11B10_FLOAT)	Can be disabled or enabled (with any desired range)	Must be enabled (with any desired range)
	R11G11B10_FLOAT	Can be disabled or enabled (with any desired range)	Must be enabled with either [0,1] or RT range

12.1.6.1 Pre-Blend Color Clamping when Blending is Disabled

The clamping of source color components is controlled by **Pre-Blend Color Clamp Enable**. If ENABLED, all source color components are clamped to the range specified by **Color Clamp Range**. If DISABLED, no clamping is performed.

Programming Notes:

- Given the possibility of writing UNPREDICTABLE values to the Color Buffer, it is expected and highly recommended that, when blending is disabled, software set **Pre-Blend Color Clamp Enable** to ENABLED and select an appropriate **Color Clamp Range**.
- When using SINT or UINT rendertarget surface formats, **Blending must** be DISABLED. The **Pre-Blend Color Clamp Enable** and **Color Clamp Range** fields are ignored, and an implied clamp to the rendertarget surface format is performed.

12.1.6.2 Pre-Blend Color Clamping when Blending is Enabled

The clamping of source, destination and constant color components is controlled by **Pre-Blend Color Clamp Enable**. If ENABLED, all these color components are clamped to the range specified by **Color Clamp Range**. If DISABLED, no clamping is performed on these color components prior to blending.

12.1.7 Color Buffer Blending

The Color Buffer Blending function is used to combine one or two incoming “source” pixel color+alpha values with the “destination” color+alpha read from the corresponding location in a RenderTarget.

Blending is enabled on a global basis by the **Color Buffer Blend Enable** state variable (in COLOR_CALC_STATE). If DISABLED, Blending and Post-Blend Clamp functions are disabled for all RenderTargets, and the pixel values (possibly subject to Pre-Blend Clamp) are passed through unchanged.

The Color Buffer Blend Enable is in the per-render-target BLEND_STATE, and the field in SURFACE_STATE is no longer supported.

Programming Notes:



- Color Buffer Blending and Logic Ops must not be enabled simultaneously, or behavior is UNDEFINED.
- Dual source blending:
- Only certain surface formats support Color Buffer Blending. Refer to the Surface Format tables in *Sampling Engine*. Blending must be disabled on a RenderTarget if blending is not supported.

The incoming “source” pixel values are modulated by a selected “source” blend factor, and the possibly gamma-decorrected “destination” values are modulated by a “destination” blend factor. These terms are then combined with a “blend function”. In general:

$$\text{src_term} = \text{src_blend_factor} * \text{src_color}$$

$$\text{dst_term} = \text{dst_blend_factor} * \text{dst_color}$$

$$\text{color output} = \text{blend_function}(\text{src_term}, \text{dst_term})$$

If there is no alpha value contained in the Color Buffer, a default value of 1.0 is used and, correspondingly, there is no alpha component computed by this function.

Dual Source Blending: When using “Dual Source” Render Target Write messages, the Source1 pixel color+alpha passed in the message can be selected as a src/dst blend factor. See *Color Buffer Blending*. In single-source mode, those blend factor selections are invalid. If SRC1 is included in a src/dst blend factor and a DualSource RT Write message is not utilized, results are UNDEFINED. (This reflects the same restriction in DX APIs, where undefined results are produced if “o1” is not written by a PS – there are no default values defined). If SRC1 is not included in a src/dst blend factor, dual source blending must be disabled.

The blending of the color and alpha components is controlled with two separate (color and alpha) sets of state variables. However, if the **Independent Alpha Blend Enable** state variable in COLOR_CALC_STATE is DISABLED, then the “color” (rather than “alpha”) set of state variables is used for both color and alpha. Note that this is the only use of the **Independent Alpha Blend Enable** state – it does not control whether Blending occurs, only how.

Per **Render Target Blend State:** Blend state is selected based on **Render Target Index** contained in the message header, and appropriate blend state is applied to Render Target Write messages.

The following table describes the color source and destination blend factors controlled by the **Source [Alpha] Blend Factor** and **Destination [Alpha] Blend Factor** state variables in COLOR_CALC_STATE. Note that the blend factors applied to the R,G,B channels are always controlled by the **Source/Destination Blend Factor**, while the blend factor applied to the alpha channel is controlled either by **Source/Destination Blend Factor** or **Source/Destination Alpha Blend Factor**.

Color Buffer Blend Color Factors

Blend Factor Selection	Blend Factor Applied for R,G,B,A channels (oN = output from PS to RT#N) (o1 = 2 nd output from PS in Dual-Source mode only) (rtN = destination color from RT#N) (CC = Constant Color)
BLENDFACTOR_ZERO	0.0, 0.0, 0.0, 0.0
BLENDFACTOR_ONE	1.0, 1.0, 1.0, 1.0
BLENDFACTOR_SRC_COLOR	oN.r, oN.g, oN.b, oN.a
BLENDFACTOR_INV_SRC_COLOR	1.0-oN.r, 1.0-oN.g, 1.0-oN.b, 1.0-oN.a
BLENDFACTOR_SRC_ALPHA	oN.a, oN.a, oN.a, oN.a
BLENDFACTOR_INV_SRC_ALPHA	1.0-oN.a, 1.0-oN.a, 1.0-oN.a, 1.0-oN.a



Blend Factor Selection	Blend Factor Applied for R,G,B,A channels (oN = output from PS to RT#N) (o1 = 2 nd output from PS in Dual-Source mode only) (rtN = destination color from RT#N) (CC = Constant Color)
BLENDFACTOR_SRC1_COLOR	o1.r, o1.g, o1.b, o1.a
BLENDFACTOR_INV_SRC1_COLOR	1.0-o1.r, 1.0-o1.g, 1.0-o1.b, 1.0-o1.a
BLENDFACTOR_SRC1_ALPHA	o1.a, o1.a, o1.a, o1.a
BLENDFACTOR_INV_SRC1_ALPHA	1.0-o1.a, 1.0-o1.a, 1.0-o1.a, 1.0-o1.a
BLENDFACTOR_DST_COLOR	rtN.r, rtN.g, rtN.b, rtN.a
BLENDFACTOR_INV_DST_COLOR	1.0-rtN.r, 1.0-rtN.g, 1.0-rtN.b, 1.0-rtN.a
BLENDFACTOR_DST_ALPHA	rtN.a, rtN.a, rtN.a, rtN.a
BLENDFACTOR_INV_DST_ALPHA	1.0-rtN.a, 1.0-rtN.a, 1.0-rtN.a, 1.0-rtN.a
BLENDFACTOR_CONST_COLOR	CC.r, CC.g, CC.b, CC.a
BLENDFACTOR_INV_CONST_COLOR	1.0-CC.r, 1.0-CC.g, 1.0-CC.b, 1.0-CC.a
BLENDFACTOR_CONST_ALPHA	CC.a, CC.a, CC.a, CC.a
BLENDFACTOR_INV_CONST_ALPHA	1.0-CC.a, 1.0-CC.a, 1.0-CC.a, 1.0-CC.a
BLENDFACTOR_SRC_ALPHA_SATURATE	f,f,f,1.0 where f = min(1.0 – rtN.a, oN.a)

The following table lists the supported blending operations defined by the **Color Blend Function** state variable and the **Alpha Blend Function** state variable (when in independent alpha blend mode).

Color Buffer Blend Functions

Blend Function	Operation (<i>for each color component</i>)
BLENDFUNCTION_ADD	SrcColor*SrcFactor + DstColor*DstFactor
BLENDFUNCTION_SUBTRACT	SrcColor*SrcFactor - DstColor*DstFactor
BLENDFUNCTION_REVERSE_SUBTRACT	DstColor*DstFactor - SrcColor*SrcFactor
BLENDFUNCTION_MIN	min (SrcColor*SrcFactor, DstColor*DstFactor) Programming Note: This is a superset of the OpenGL “min” function.
BLENDFUNCTION_MAX	max (SrcColor*SrcFactor, DstColor*DstFactor) Programming Note: This is a superset of the OpenGL “max” function.

12.1.8 Post-Blend Color Clamping

(See *Pre-Blend Color Clamping* above for a summary table regarding clamping)

Post-Blend Color clamping is available only if Blending is enabled.

If Blending is enabled, the clamping of blending output color components is controlled by **Post-Blend Color Clamp Enable**. If ENABLED, the color components output from blending are clamped to the range specified by **Color Clamp Range**. If DISABLED, no clamping is performed at this point.

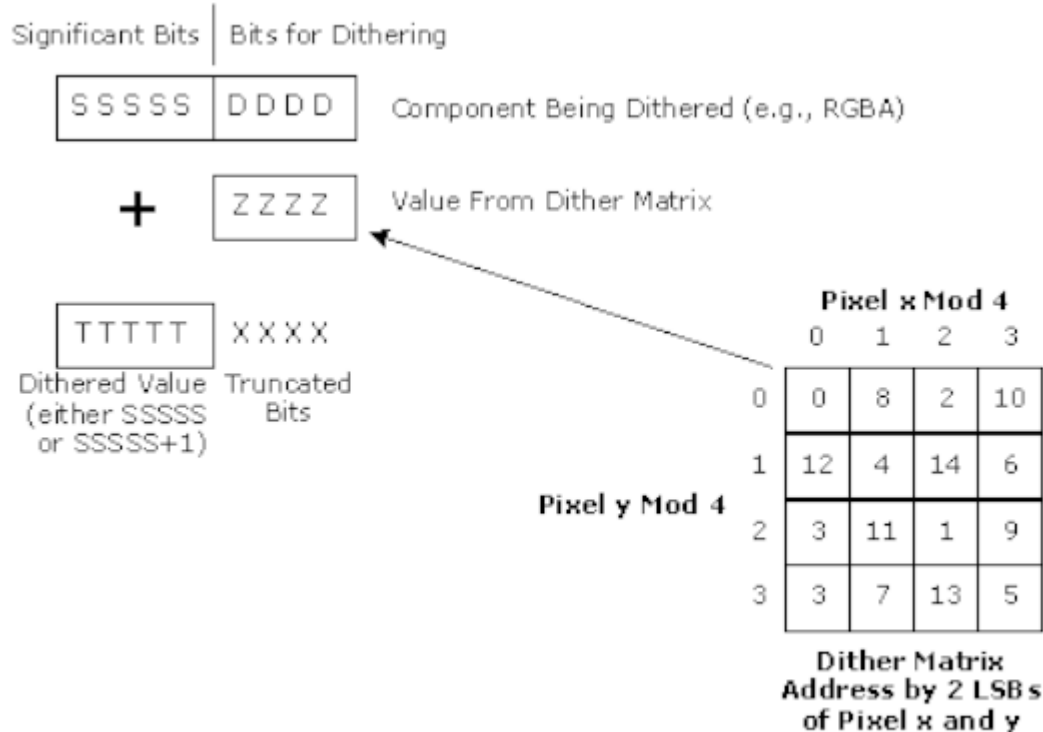
Regardless of the setting of **Post-Blend Color Clamp Enable**, when Blending is enabled color components will be automatically clamped to (at least) the rendertarget surface format range at this stage of the pipeline.

12.1.9 Dithering

Dithering is used to give the illusion of a higher resolution when using low-bpp channels in color buffers (e.g., with 16bpp color buffer). By carefully choosing an arrangement of lower resolution colors, colors otherwise not representable can be approximated, especially when seen at a distance where the viewer's eyes will average adjacent pixel colors. Color dithering tends to diffuse the sharp color bands seen on smooth-shaded objects.

A four-bit dither value is obtained from a 4x4 Dither Constant matrix depending on the pixel's X and Y screen coordinate. The pixel's X and Y screen coordinates are first offset by the **Dither Offset X** and **Dither Offset Y** state variables (these offsets are used to provide window-relative dithering). Then the two LSBs of the pixel's screen X coordinate are used to address a column in the dither matrix, and the two LSBs of the pixel's screen Y coordinate are used to address a row. This way, the matrix repeats every four pixels in both directions.

The value obtained is appropriately shifted to align with (what would be otherwise) truncated bits of the component being dithered. It is then added with the component and the result is truncated to the bit depth of the component given the color buffer format.



B.6852-01

Dithering Process (5-Bit Example)

12.1.10 Logic Ops

The Logic Ops function is used to combine the incoming "source" pixel color/alpha values with the corresponding "destination" color/alpha contained in the ColorBuffer, using a logic function.

The Logic Op function is enabled by the **LogicOp Enable** state variable. If DISABLED, this function is ignored and the incoming pixel values are passed through unchanged.



Programming Notes:

- Color Buffer Blending and Logic Ops must not be enabled simultaneously, or behavior is UNDEFINED.
- Logic Ops are only supported on *_UNORM surfaces, otherwise Logic Ops must be DISABLED.

The following table lists the supported logic ops. The logic op is selected using the **Logic Op Function** field in COLOR_CALC_STATE.

Logic Ops

LogicOp Function	Definition (S=Source, D=Destination)
LOGICOP_CLEAR	all 0's
LOGICOP_NOR	NOT (S OR D)
LOGICOP_AND_INVERTED	(NOT S) AND D
LOGICOP_COPY_INVERTED	NOT S
LOGICOP_AND_REVERSE	S AND NOT D
LOGICOP_INVERT	NOT D
LOGICOP_XOR	S XOR D
LOGICOP_NAND	NOT (S AND D)
LOGICOP_AND	S AND D
LOGICOP_EQUIV	NOT (S XOR D)
LOGICOP_NOOP	D
LOGICOP_OR_INVERTED	(NOT S) OR D
LOGICOP_COPY	S
LOGICOP_OR_REVERSE	S OR NOT D
LOGICOP_OR	S OR D
LOGICOP_SET	all 1's

12.1.11 Buffer Update

The Buffer Update function is responsible for updating the pixel's Stencil, Depth and Color Buffer contents based upon the results of the Stencil and Depth Test functions. Note that Kill Pixel and/or Alpha Test functions may have already discarded the pixel by this point.

12.1.11.1 Stencil Buffer Updates

If and only if stencil testing is enabled, the Stencil Buffer is updated according to the **Stencil Fail Op**, **Stencil Pass Depth Fail Op**, and **Stencil Pass Depth Pass Op** state (or their backface counterparts if **Double Sided Stencil Enable** is ENABLED and the pixel is from a back-facing object) and the results of the Stencil Test and Depth Test functions.

Stencil Fail Op and **Backface Stencil Fail Op** specify how/if the stencil buffer is modified if the stencil test fails. **Stencil Pass Depth Fail Op** and **Backface Stencil Pass Depth Fail Op** specify how/if the stencil buffer is modified if the stencil test passes but the depth test fails. **Stencil Pass Depth Pass Op** and **Backface Stencil Pass Depth Pass Op** specify how/if the stencil buffer is modified if both the stencil and depth tests pass. The operations (on the stencil buffer) that are to be performed under one of these (mutually exclusive) conditions is summarized in the following table.

Stencil Buffer Operations

Stencil Operation	Description
STENCILOP_KEEP	Do not modify the stencil buffer



Stencil Operation	Description
STENCILOP_ZERO	Store a 0
STENCILOP_REPLACE	Store the <i>StencilTestReference</i> reference value
STENCILOP_INCRSAT	Saturating increment (clamp to max value)
STENCILOP_DECRSAT	Saturating decrement (clamp to 0)
STENCILOP_INCR	Increment (possible wrap around to 0)
STENCILOP_DECR	Decrement (possible wrap to max value)
STENCILOP_INVERT	Logically invert the stencil value

Any and all writes to the stencil portion of the depth buffer are enabled by the **Stencil Buffer Write Enable** state variable.

When writes are enabled, the **Stencil Buffer Write Mask** and **Backface Stencil Buffer Write Mask** state variables provide an 8-bit mask that selects which bits of the stencil write value are modified. Masked-off bits (i.e., mask bit == 0) are left unmodified in the Stencil Buffer.

Programming Notes:

- The Stencil Buffer can be written even if depth buffer writes are disabled via **Depth Buffer Write Enable**.

12.1.11.2 Depth Buffer Updates

Any and all writes to the Depth Buffer are enabled by the **Depth Buffer Write Enable** state variable. If there is no Depth Buffer, writes must be explicitly disabled with this state variable, or operation is UNDEFINED.

If depth testing is disabled or the depth test passed, the incoming pixel's depth value is written to the Depth Buffer. If depth testing is enabled and the depth test failed, the pixel is discarded – with no modification to the Depth or Color Buffers (though the Stencil Buffer may have been modified).

12.1.11.3 Color Gamma Correction

Computed RGB (not A) channels can be gamma-corrected prior to update of the Color Buffer.

This function is automatically invoked whenever the destination surface (render target) has an SRGB format (see surface formats in *Sampling Engine*). For these surfaces, the computed RGB values are converted from gamma=1.0 space to gamma=2.4 space by applying a $^{(2.4)}$ exponential function.

12.1.11.4 Color Buffer Updates

Finally, if the pixel has not been discarded by this point, the incoming pixel color is written into the Color Buffer. The **Surface Format** of the color buffer indicates which channel(s) are written (e.g., R8G8_UNORM are written with the Red and Green channels only). The **Color Buffer Component Write Disables** from the Color Buffer's SURFACE_STATE provide an independent write disable for each channel of the Color Buffer.



12.2 Pixel Pipeline State Summary

12.2.1 COLOR_CALC_STATE

COLOR_CALC_STATE													
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000													
This register is pointed to by a field in 3DSTATE_CC_STATE_POINTERS, and stored at a 64-byte aligned boundary.													
DWord	Bit Description												
0	31:24 Stencil Reference Value												
	Format: U8.0												
	This field specifies the stencil reference value to compare against in the (front face) StencilTest function.												
	23:16 BackFace Stencil Reference Value												
	Format: U8.0 This field specifies the stencil reference value to compare against in the StencilTest function.												
15	Round Disable Function Disable												
	Project: All Format: U8.0 Disables the round-disable function of the color calculator. If this bit is zero, dithering is cancelled based on the data used by blend to avoid drift. If this bit is one, this is not done.												
14:1	Reserved												
	Project: All Format: MBZ												
0	Alpha Test Format												
	Project: All												
	This field selects the format for Alpha Reference Value and the format in which Alpha Test is performed.												
	<table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> <th>Description</th> <th>Project</th> </tr> </thead> <tbody> <tr> <td>0h</td> <td>ALPHATEST_UNORM8</td> <td>UNorm8</td> <td>All</td> </tr> <tr> <td>1h</td> <td>ALPHATEST_FLOAT32</td> <td>Float32</td> <td>All</td> </tr> </tbody> </table>	Value	Name	Description	Project	0h	ALPHATEST_UNORM8	UNorm8	All	1h	ALPHATEST_FLOAT32	Float32	All
	Value	Name	Description	Project									
0h	ALPHATEST_UNORM8	UNorm8	All										
1h	ALPHATEST_FLOAT32	Float32	All										
Programming Notes													
Alpha-test format is independent of RT format. When PS outputs UNIT/SINT alpha-value, it will be treated as IEEE 32bit float number for the purpose of alpha-test.													
1	31:0 Alpha Reference Value												
	Project: All												
	Exists If: Alpha Test Format == ALPHATEST_FLOAT32												
	Format: IEEE_Float This field specifies the alpha reference value to compare against in the Alpha Test function.												
31:0	Alpha Reference Value												



COLOR_CALC_STATE								
		<table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Exists If:</td> <td>Alpha Test Format == ALPHATEST_UNORM8</td> </tr> <tr> <td>Format:</td> <td>UNORM8 Upper 24 bits MBZ</td> </tr> </table> <p>This field specifies the alpha reference value to compare against in the Alpha Test function.</p>	Project:	All	Exists If:	Alpha Test Format == ALPHATEST_UNORM8	Format:	UNORM8 Upper 24 bits MBZ
Project:	All							
Exists If:	Alpha Test Format == ALPHATEST_UNORM8							
Format:	UNORM8 Upper 24 bits MBZ							
2	31:0	<table border="1"> <tr> <td colspan="2">Blend Constant Color Red</td> </tr> <tr> <td>Format:</td> <td>IEEE_Float</td> </tr> </table> <p>This field specifies the Red channel of the Constant Color used in Color Buffer Blending.</p>	Blend Constant Color Red		Format:	IEEE_Float		
Blend Constant Color Red								
Format:	IEEE_Float							
3	31:0	<table border="1"> <tr> <td colspan="2">Blend Constant Color Green</td> </tr> <tr> <td>Format:</td> <td>IEEE_Float</td> </tr> </table> <p>This field specifies the Green channel of the Constant Color used in Color Buffer Blending.</p>	Blend Constant Color Green		Format:	IEEE_Float		
Blend Constant Color Green								
Format:	IEEE_Float							
4	31:0	<table border="1"> <tr> <td colspan="2">Blend Constant Color Blue</td> </tr> <tr> <td>Format:</td> <td>IEEE_Float</td> </tr> </table> <p>This field specifies the Blue channel of the Constant Color used in Color Buffer Blending.</p>	Blend Constant Color Blue		Format:	IEEE_Float		
Blend Constant Color Blue								
Format:	IEEE_Float							
5	31:0	<table border="1"> <tr> <td colspan="2">Blend Constant Color Alpha</td> </tr> <tr> <td>Format:</td> <td>IEEE_Float</td> </tr> </table> <p>This field specifies the Alpha channel of the Constant Color used in Color Buffer Blending.</p>	Blend Constant Color Alpha		Format:	IEEE_Float		
Blend Constant Color Alpha								
Format:	IEEE_Float							

12.2.2 DEPTH_STENCIL_STATE

DEPTH_STENCIL_STATE																													
Default Value: 0x00000000, 0x00000000, 0x00000000																													
The DEPTH_STENCIL_STATE is pointed to by a field in 3DSTATE_CC_STATE_POINTERS. It is stored at a 64-byte aligned boundary.																													
DWord	Bit	Description																											
0	31	<table border="1"> <tr> <td colspan="3">Stencil Test Enable</td> </tr> <tr> <td>Project:</td> <td colspan="2">All</td> </tr> <tr> <td>Format:</td> <td colspan="2">Enable</td> </tr> <tr> <td colspan="3">Enables StencilTest function of the Pixel Processing pipeline.</td> </tr> <tr> <td colspan="3" style="text-align: center;">Programming Notes</td> </tr> <tr> <td colspan="3">If any of the render targets are YUV format, this field must be disabled.</td> </tr> </table>	Stencil Test Enable			Project:	All		Format:	Enable		Enables StencilTest function of the Pixel Processing pipeline.			Programming Notes			If any of the render targets are YUV format, this field must be disabled.											
Stencil Test Enable																													
Project:	All																												
Format:	Enable																												
Enables StencilTest function of the Pixel Processing pipeline.																													
Programming Notes																													
If any of the render targets are YUV format, this field must be disabled.																													
	30:28	<table border="1"> <tr> <td colspan="3">Stencil Test Function</td> </tr> <tr> <td>Project:</td> <td colspan="2">All</td> </tr> <tr> <td>Format:</td> <td colspan="2">3D_CompareFunction</td> </tr> <tr> <td colspan="3">This field specifies the comparison function used in the (front face) StencilTest function.</td> </tr> <tr> <td style="text-align: center;">Value</td> <td style="text-align: center;">Name</td> <td style="text-align: center;">Project</td> </tr> <tr> <td>0h</td> <td>COMPAREFUNCTION_ALWAYS</td> <td>All</td> </tr> <tr> <td>1h</td> <td>COMPAREFUNCTION_NEVER</td> <td>All</td> </tr> <tr> <td>2h</td> <td>COMPAREFUNCTION_LESS</td> <td>All</td> </tr> <tr> <td>3h</td> <td>COMPAREFUNCTION_EQUAL</td> <td>All</td> </tr> </table>	Stencil Test Function			Project:	All		Format:	3D_CompareFunction		This field specifies the comparison function used in the (front face) StencilTest function.			Value	Name	Project	0h	COMPAREFUNCTION_ALWAYS	All	1h	COMPAREFUNCTION_NEVER	All	2h	COMPAREFUNCTION_LESS	All	3h	COMPAREFUNCTION_EQUAL	All
Stencil Test Function																													
Project:	All																												
Format:	3D_CompareFunction																												
This field specifies the comparison function used in the (front face) StencilTest function.																													
Value	Name	Project																											
0h	COMPAREFUNCTION_ALWAYS	All																											
1h	COMPAREFUNCTION_NEVER	All																											
2h	COMPAREFUNCTION_LESS	All																											
3h	COMPAREFUNCTION_EQUAL	All																											



DEPTH_STENCIL_STATE

	4h	COMPAREFUNCTION_LEQUAL	All
	5h	COMPAREFUNCTION_GREATER	All
	6h	COMPAREFUNCTION_NOTEQUAL	All
	7h	COMPAREFUNCTION_GEQUAL	All
27:25	Stencil Fail Op		
	Project:	All	
	Format:	3D_StencilOperation	
	This field specifies the operation to perform on the Stencil Buffer when the (front face) stencil test fails. Note: if all three stencil ops (Stencil Fail, Stencil Pass Depth Fail, and Stencil Pass Depth Pass) are KEEP, ZERO, or REPLACE, the stencil buffer is not read.		
	Value	Name	Project
	0	STENCILOP_KEEP	All
	1	STENCILOP_ZERO	All
	2	STENCILOP_REPLACE	All
	3	STENCILOP_INCRSAT	All
	4	STENCILOP_DECRSAT	All
	5	STENCILOP_INCR	All
	6	STENCILOP_DECR	All
	7	STENCILOP_INVERT	All
24:22	Stencil Pass Depth Fail Op		
	Project:	All	
	Format:	3D_StencilOperation see Stencil Fail Op	
	This field specifies the operation to perform on the Stencil Buffer when the (front face) stencil test passes but the depth pass fails.		
21:19	Stencil Pass Depth Pass Op		
	Project:	All	
	Format:	3D_StencilOperation see Stencil Fail Op	
	This field specifies the operation to perform on the Stencil Buffer when the (front face) stencil test passes and the depth pass passes (or is disabled).		
18	Stencil Buffer Write Enable		
	Project:	All	
	Format:	Enable	
	Enables writes to the Stencil Buffer.		
	Programming Notes		
	If this field is enabled, Stencil Test Enable must also be enabled.		
17:16	Reserved		
	Project:	All	
	Format:	MBZ	
15	Double Sided Stencil Enable		
	Project:	All	
	Format:	Enable	
	Enable doubled sided stencil operations.		
	Value	Name	Description
			Project



DEPTH_STENCIL_STATE

1	Enable	Double Sided Stencil Enabled	All
0	Disable	Double Sided Stencil Disabled	All

Programming Notes

Back-facing primitives have a vertex winding order opposite to the currently selected Front Winding state. Culling of primitives is not affected by the double sided stencil state. Back-facing primitives will be rendered, honoring all current device state, as though it were a front-facing primitive with no implicitly overloaded state.

14:12 BackFace Stencil Test Function

Project:	All
Format:	3D_CompareFunction

This field specifies the comparison function used in the StencilTest function.

Value	Name	Project
0h	COMPAREFUNCTION_ALWAYS	All
1h	COMPAREFUNCTION_NEVER	All
2h	COMPAREFUNCTION_LESS	All
3h	COMPAREFUNCTION_EQUAL	All
4h	COMPAREFUNCTION_LEQUAL	All
5h	COMPAREFUNCTION_GREATER	All
6h	COMPAREFUNCTION_NOTEQUAL	All
7h	COMPAREFUNCTION_GEQUAL	All

11:9 Backface Stencil Fail Op

Project:	All
Format:	3D_StencilOperation

This field specifies the operation to perform on the Stencil Buffer when the stencil test fails.

Value	Name	Description	Project
0	STENCILOP_KEEP	STENCILOP_KEEP	All
1	STENCILOP_ZERO	STENCILOP_ZERO	All
2	STENCILOP_REPLACE	STENCILOP_REPLACE	All
3	STENCILOP_INCRSAT	STENCILOP_INCRSAT	All
4	STENCILOP_DECRSAT	STENCILOP_DECRSAT	All
5	STENCILOP_INCR	STENCILOP_INCR	All
6	STENCILOP_DECR	STENCILOP_DECR	All
7	STENCILOP_INVERT	STENCILOP_INVERT	All

8:6 Backface Stencil Pass Depth Fail Op

Project:	All
Format:	3D_StencilOperation see Stencil Fail Op

This field specifies the operation to perform on the Stencil Buffer when the stencil test passes but the depth pass fails.

5:3 Backface Stencil Pass Depth Pass Op

Project:	All
Format:	3D_StencilOperation see Stencil Fail Op

This field specifies the operation to perform on the Stencil Buffer when the stencil test passes and the



DEPTH_STENCIL_STATE																
		depth pass passes (or is disabled).														
	2:0	Reserved														
		Project: All														
		Format: MBZ														
1	31:24	Stencil Test Mask														
		Project: All														
		Format: U8														
		This field specifies a bit mask applied to stencil test values. Both the stencil reference value and value read from the stencil buffer will be logically ANDed with this mask before the stencil comparison test is performed.														
	23:16	Stencil Write Mask														
	Project: All															
	Format: U8															
		This field specifies a bit mask applied to stencil buffer writes. Only those stencil buffer bits corresponding to bits set in this mask will be modified.														
15:8		Backface Stencil Test Mask														
		Project: All														
		Format: U8														
		This field specifies a bit mask applied to backface stencil test values. Both the stencil reference value and value read from the stencil buffer will be logically ANDed with this mask before the stencil comparison test is performed.														
7:0		Backface Stencil Write Mask														
		Project: All														
		Format: U8														
		This field specifies a bit mask applied to backface stencil buffer writes. Only those stencil buffer bits corresponding to bits set in this mask will be modified.														
2	31	Depth Test Enable														
		Project: All														
		Format: Enable														
		Enables the DepthTest function of the Pixel Processing pipeline.														
		Programming Notes														
		If any of the render targets are YUV format, this field must be disabled.														
	30	Reserved														
		Project: All														
		Format: MBZ														
	29:27	Depth Test Function														
	Project: All															
	Format: 3D_DepthTestFunction															
	Specifies the comparison function used in DepthTest function.															
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Name</th> <th style="text-align: center;">Project</th> </tr> </thead> <tbody> <tr> <td>0h</td> <td>COMPAREFUNCTION_ALWAYS</td> <td>All</td> </tr> <tr> <td>1h</td> <td>COMPAREFUNCTION_NEVER</td> <td>All</td> </tr> <tr> <td>2h</td> <td>COMPAREFUNCTION_LESS</td> <td>All</td> </tr> <tr> <td>3h</td> <td>COMPAREFUNCTION_EQUAL</td> <td>All</td> </tr> </tbody> </table>	Value	Name	Project	0h	COMPAREFUNCTION_ALWAYS	All	1h	COMPAREFUNCTION_NEVER	All	2h	COMPAREFUNCTION_LESS	All	3h	COMPAREFUNCTION_EQUAL	All
Value	Name	Project														
0h	COMPAREFUNCTION_ALWAYS	All														
1h	COMPAREFUNCTION_NEVER	All														
2h	COMPAREFUNCTION_LESS	All														
3h	COMPAREFUNCTION_EQUAL	All														



DEPTH_STENCIL_STATE			
	4h	COMPAREFUNCTION_LEQUAL	All
	5h	COMPAREFUNCTION_GREATER	All
	6h	COMPAREFUNCTION_NOTEQUAL	All
	7h	COMPAREFUNCTION_GEQUAL	All
Programming Notes			
if the Depth Test Function is ALWAYS or NEVER, the depth buffer is not read.			
26	Depth Buffer Write Enable		
	Project:		All
	Format:		Enable
Enables writes to the Depth Buffer.			
Programming Notes			
A Depth Buffer must be defined before enabling writes to it, or operation is UNDEFINED.			
25:0	Reserved		
	Project:		All
	Format:		MBZ

12.2.3 BLEND_STATE

BLEND_STATE			
Default Value:		0x00000000, 0x00000000	
<p>The blend state is stored as an array of up to 8 elements, each of which contains the DWords described here. The start of each element is spaced 2 DWords apart. The first element of the blend state array is aligned to a 64-byte boundary, which is pointed to by a field in 3DSTATE_CC_STATE_POINTERS. The 3-bit Render Target Index field in the Render Target Write data port message header is used to select which of the 8 elements from BLEND_STATE that is used on the current message.</p>			
DWord	Bit	Description	
0	31	Color Buffer Blend Enable	
		Project:	All
		Format:	Enable
		Enables the ColorBufferBlending (nee “alpha blending”) function of the Pixel Processing Pipeline for this render target.	
Programming Notes			
Enabling LogicOp and ColorBufferBlending at the same time is UNDEFINED			
30		Independent Alpha Blend Enable	
		Project:	All
		Format:	Enable
		When enabled, the other fields in this instruction control the combination of the alpha components in the Color Buffer Blend stage. When disabled, the alpha components are combined in the same fashion as the color components.	
29		Reserved	
		Project:	All
		Format:	MBZ



BLEND_STATE

28:26	Alpha Blend Function		
	Project:	All	
	Format:	3D_ColorBufferBlendFunction	
	This field specifies the function used to combine the alpha components in the Color Buffer blend stage of the Pixel Pipeline when the IndependentAlphaBlend state is enabled.		
	Value	Name	Project
	0	BLENDFUNCTION_ADD	All
	1	BLENDFUNCTION_SUBTRACT	All
	2	BLENDFUNCTION_REVERSE_SUBTRACT	All
	3	BLENDFUNCTION_MIN	All
	4	BLENDFUNCTION_MAX	All
5 - 7	Reserved	All	
25	Reserved		
	Project:	All	
	Format:	MBZ	
24:20	Source Alpha Blend Factor		
	Project:	All	
	Format:	3D_ColorBufferBlendFactor	
	Controls the “source factor” in alpha Color Buffer Blending stage. Note: For the source/destination alpha blend factors, the encodings indicating “COLOR” are the same as the encodings indicating “ALPHA”, as the alpha component of the color is selected.		
	Value	Name	Project
	00h	Reserved	All
	01h	BLENDFACTOR_ONE	All
	02h	BLENDFACTOR_SRC_COLOR	All
	03h	BLENDFACTOR_SRC_ALPHA	All
	04h	BLENDFACTOR_DST_ALPHA	All
	05h	BLENDFACTOR_DST_COLOR	All
	06h	BLENDFACTOR_SRC_ALPHA_SATURATE	All
	07h	BLENDFACTOR_CONST_COLOR	All
	08h	BLENDFACTOR_CONST_ALPHA	All
	09h	BLENDFACTOR_SRC1_COLOR	All
	0Ah	BLENDFACTOR_SRC1_ALPHA	All
	0Bh-10h	Reserved	All
	11h	BLENDFACTOR_ZERO	All
	12h	BLENDFACTOR_INV_SRC_COLOR	All
	13h	BLENDFACTOR_INV_SRC_ALPHA	All
	14h	BLENDFACTOR_INV_DST_ALPHA	All
	15h	BLENDFACTOR_INV_DST_COLOR	All
	16h	Reserved	All
	17h	BLENDFACTOR_INV_CONST_COLOR	All
	18h	BLENDFACTOR_INV_CONST_ALPHA	All
	19h	BLENDFACTOR_INV_SRC1_COLOR	All
	1Ah	BLENDFACTOR_INV_SRC1_ALPHA	All
19:15	Destination Alpha Blend Factor		
	Project:	All	
	Format:	3D_ColorBufferBlendFactor	
	Controls the “destination factor” in alpha Color Buffer Blending stage. Refer to Source Alpha Blend Factor for encodings.		



BLEND_STATE																								
1	14	<p>Reserved</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Project:	All	Format:	MBZ																		
	Project:	All																						
	Format:	MBZ																						
	13:11	<p>Color Blend Function</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>3D_ColorBufferBlendFunction</td> </tr> </table> <p>This field specifies the function used to combine the color components in the ColorBufferBlending function of the Pixel Processing Pipeline. If Independent Alpha Blend Enable is disabled, this field will also control the blending of the alpha components in the ColorBufferBlending function.</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Name</th> <th>Project</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>BLENDFUNCTION_ADD</td> <td>All</td> </tr> <tr> <td>1</td> <td>BLENDFUNCTION_SUBTRACT</td> <td>All</td> </tr> <tr> <td>2</td> <td>BLENDFUNCTION_REVERSE_SUBTRACT</td> <td>All</td> </tr> <tr> <td>3</td> <td>BLENDFUNCTION_MIN</td> <td>All</td> </tr> <tr> <td>4</td> <td>BLENDFUNCTION_MAX</td> <td>All</td> </tr> </tbody> </table>	Project:	All	Format:	3D_ColorBufferBlendFunction	Value	Name	Project	0	BLENDFUNCTION_ADD	All	1	BLENDFUNCTION_SUBTRACT	All	2	BLENDFUNCTION_REVERSE_SUBTRACT	All	3	BLENDFUNCTION_MIN	All	4	BLENDFUNCTION_MAX	All
	Project:	All																						
	Format:	3D_ColorBufferBlendFunction																						
	Value	Name	Project																					
	0	BLENDFUNCTION_ADD	All																					
	1	BLENDFUNCTION_SUBTRACT	All																					
	2	BLENDFUNCTION_REVERSE_SUBTRACT	All																					
	3	BLENDFUNCTION_MIN	All																					
	4	BLENDFUNCTION_MAX	All																					
	10	<p>Reserved</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>MBZ</td> </tr> </table>	Project:	All	Format:	MBZ																		
	Project:	All																						
Format:	MBZ																							
9:5	<p>Source Blend Factor</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>3D_ColorBufferBlendFactor</td> </tr> </table> <p>Controls the “source factor” in the ColorBufferBlending function. Refer to Source Alpha Blend Factor for encodings.</p>	Project:	All	Format:	3D_ColorBufferBlendFactor																			
Project:	All																							
Format:	3D_ColorBufferBlendFactor																							
4:0	<p>Destination Blend Factor</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>3D_ColorBufferBlendFactor</td> </tr> </table> <p>Controls the “destination factor” in the ColorBufferBlending function. Refer to Source Alpha Blend Factor for encodings.</p>	Project:	All	Format:	3D_ColorBufferBlendFactor																			
Project:	All																							
Format:	3D_ColorBufferBlendFactor																							
31	<p>AlphaToCoverage Enable</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> </table> <p>If set, Source0 Alpha is converted to a temporary 1/2/4-bit coverage mask and the mask bit corresponding to the sample# ANDed with the sample mask bit. If set, sample coverage is computed based on src0 alpha value. Value of 0 disables all samples and value of 1 enables all samples for that pixel. The same coverage needs to apply to all the RTs in MRT case. Further, any value of src0 alpha between 0 and 1 monotonically increases the number of enabled pixels. The same coverage needs to be applied to all the RTs in MRT case.</p>	Project:	All	Format:	Enable																			
Project:	All																							
Format:	Enable																							
30	<p>AlphaToOne Enable</p> <table border="1"> <tr> <td>Project:</td> <td>All</td> </tr> <tr> <td>Format:</td> <td>Enable</td> </tr> </table> <p>If set, Source0 Alpha is set to 1.0f after (possibly) being used to generate the AlphaToCoverage coverage mask. The same coverage needs to be applied to all the RTs in MRT case. If Dual Source Blending is enabled, this bit must be disabled.</p> <table border="1"> <thead> <tr> <th>Errata</th> <th>Description</th> <th>Project</th> </tr> </thead> <tbody> <tr> <td></td> <td>This bit must be disabled.</td> <td></td> </tr> </tbody> </table>	Project:	All	Format:	Enable	Errata	Description	Project		This bit must be disabled.														
Project:	All																							
Format:	Enable																							
Errata	Description	Project																						
	This bit must be disabled.																							

BLEND_STATE

29	AlphaToCoverage Dither Enable		
Project:		All	
Format:		Enable	
<p>If set, sample coverage is computed based on src0 alpha value and it modulates the sample coverage based on screen coordinates. Value of 0 disables all samples and value of 1 enables all samples for that pixel. The same coverage needs to apply to all the RTs in MRT case. Further, any value of src0 alpha between 0 and 1 monotonically increases the number of enabled pixels. The same coverage needs to be applied to all the RTs in MRT case. If AlphaToCoverage is disabled, AlphaToCoverage Dither does not have any impact.</p>			
28	Reserved		
Project:		All	
Format:		MBZ	
27	Write Disable Alpha		
Project:		All	
Format:		Disable	
<p>This field controls the writing of the alpha component into the Render Target.</p>			
Value	Name	Description	Project
0b	Enabled	Alpha component can be overwritten	All
1b	Disabled	Writes to the color buffer will not modify Alpha.	All
Programming Notes			
For YUV surfaces, this field must be set to 0B (enabled).			
Errata	Description		Project
	This field should not be set to 0 if the alpha component is not present in the Render Target surface format.		
26	Write Disable Red		
Project:		All	
Format:		Disable	
<p>This field controls the writing of the red component into the Render Target.</p>			
Value	Name	Description	Project
0b	Enabled	Red component can be overwritten	All
1b	Disabled	Writes to the color buffer will not modify Red.	All
Programming Notes			
For YUV surfaces, this field must be set to 0B (enabled).			
Errata	Description		Project
	This field should not be set to 0 if the Red component is not present in the Render Target surface format.		
25	Write Disable Green		
Project:		All	
Format:		Disable	
<p>This field controls the writing of the green component into the Render Target.</p>			
Value	Name	Description	Project



BLEND_STATE

	0b	Enabled	Green component can be overwritten	All
	1b	Disabled	Writes to the color buffer will not modify Green.	All
Programming Notes				
For YUV surfaces, this field must be set to 0B (enabled).				
Errata	Description			Project
	This field should not be set to 0 if the Green component is not present in the Render Target surface format			
24	Write Disable Blue			
	Project:			All
	Format:			Disable
This field controls the writing of the Blue component into the Render Target.				
	Value	Name	Description	Project
	0b	Enabled	Blue component can be overwritten	All
	1b	Disabled	Writes to the color buffer will not modify Blue.	All
Programming Notes				
For YUV surfaces, this field must be set to 0B (enabled).				
Errata	Description			Project
	This field should not be set to 0 if the Red component is not present in the Render Target surface format			
23	Reserved			
	Project:			All
	Format:			MBZ
22	Logic Op Enable			
	Project:			All
	Format:			Enable
Enables the LogicOp function of the Pixel Processing pipeline.				
Programming Notes				
Enabling LogicOp and Color Buffer Blending at the same time is UNDEFINED				
21:18	Logic Op Function			
	Project:			All
	Format:			3D_LogicOpFunction
This field specifies the function to be performed (when enabled) in the Logic Op stage of the Pixel Processing pipeline. Note that the encoding of this field is one less than the corresponding "R2_" ROP code defined in WINGDI.H, and is a rather contorted mapping of the OpenGL LogicOp encodings. However, this field was defined such that, when the 4 bits are replicated to 8 bits, they coincide with the ROP codes used in the Blter. Note: if the Logic Op Function does not depend on "D", the dest buffer is not read.				
	Value	Name	Description	Project
	0h	LOGICOP_CLEAR	BLACK; all 0's	All
	1h	LOGICOP_NOR	NOTMERGEPEN; NOT (S OR D)	All
	2h	LOGICOP_AND_INVERTED	MASKNOTPEN; (NOT S) AND D	All
	3h	LOGICOP_COPY_INVERTED	NOTCOPYPEN; NOT S	All



BLEND_STATE

	4h	LOGICOP_AND_REVERSE	MASKPENNOT; S AND NOT D	All
	5h	LOGICOP_INVERT	NOT; NOT D	All
	6h	LOGICOP_XOR	XORPEN; S XOR D	All
	7h	LOGICOP_NAND	NOTMASKPEN; NOT (S AND D)	All
	8h	LOGICOP_AND	MASKPEN; S AND D	All
	9h	LOGICOP_EQUIV	NOTXORPEN; NOT (S XOR D)	All
	Ah	LOGICOP_NOOP	NOP; D	All
	Bh	LOGICOP_OR_INVERTED	MERGENOTPEN; (NOT S) OR D	All
	Ch	LOGICOP_COPY	COPYPEN; S	All
	Dh	LOGICOP_OR_REVERSE	MERGEPENNOT; S OR NOT D	All
	Eh	LOGICOP_OR	MERGEPEN; S OR D	All
	Fh	LOGICOP_SET	WHITE; all 1's	All
17	Reserved			
	Project:		All	
	Format:		MBZ	
16	Alpha Test Enable			
	Project:		All	
	Format:		Enable	
	Enables the AlphaTest function of the Pixel Processing pipeline.			
	Programming Notes			Project
	Alpha Test can only be enabled if Pixel Shader outputs a float alpha value. Alpha Test is applied independently on each render target by comparing that render target's alpha value against the alpha reference value. If the alpha test fails, the corresponding pixel write will be suppressed only for that render target. The depth/stencil update will occur if alpha test passes for any render target.			
	When Alpha Test is disabled, Alpha Test Function must be COMPAREFUNCTION_ALWAYS.			
15:13	Alpha Test Function			
	Project:		All	
	Format:		3D_CompareFunction	
	This field specifies the comparison function used in the AlphaTest function			
	Value	Name	Description	Project
	0h	COMPAREFUNCTION_ALWAYS	Always pass	All
	1h	COMPAREFUNCTION_NEVER	Never pass	All
	2h	COMPAREFUNCTION_LESS	Pass if the value is less than the reference	All
	3h	COMPAREFUNCTION_EQUAL	Pass if the value is equal to the reference	All
	4h	COMPAREFUNCTION_LEQUAL	Pass if the value is less than or equal to the reference	All
	5h	COMPAREFUNCTION_GREATER	Pass if the value is greater than the reference	All
	6h	COMPAREFUNCTION_NOTEQUAL	Pass if the value is not equal to the reference	All
	7h	COMPAREFUNCTION_GEQUAL	Pass if the value is greater than or equal to the reference	All
12	Color Dither Enable			
	Project:		All	
	Format:		Enable	
	Enables dithering of colors (including any alpha component) before they are written to the Color Buffer.			
11:10	X Dither Offset			
	Project:		All	



BLEND_STATE

		Format:	U2
		Specifies offset to apply to pixel X coordinate LSBs when accessing dither table.	
9:8	Y Dither Offset	Project:	All
		Format:	U2
		Specifies offset to apply to pixel Y coordinate LSBs when accessing dither table.	
7:4	Reserved	Project:	All
		Format:	MBZ
3:2	Color Clamp Range	Project:	All
		Specifies the clamped range used in Pre-Blend and Post-Blend Color Clamp functions if one or both of those functions are enabled. Note that this range selection is shared between those functions. This field is ignored if both of the Color Clamp Enables are disabled	
		Value	Name
		Description	Project
		0	COLORCLAMP_UNORM
		1	COLORCLAMP_SNORM
		2	COLORCLAMP_RTFORMAT
		3	Reserved
		Clamp Range [0,1]	
		Clamp Range [-1,1]	
		Clamp to the range of the RT surface format (Note: The Alpha component is clamped to FLOAT16 for R11G11B10_FLOAT format).	
		Reserved	
1	Pre-Blend Color Clamp Enable	Project:	All
		Format:	Enable
		This field specifies whether the source, destination and constant color channels are clamped prior to blending, regardless of whether blending is enabled. If DISABLED, no clamping is performed prior to blending. If ENABLED, all inputs to the blend function are clamped prior to the blend to the range specified by Color Clamp Range.	
		Value	Name
		Description	Project
		0	Disabled
		1	Enabled
		No clamping is performed prior to blending.	
		All inputs to the blend function are clamped prior to the blend to the range specified by Color Clamp Range.	
		Programming Notes	
		See table in Pre-Blending Color Clamp subsection for programming restrictions as a function of RT format. This field is ignored (treated as DISABLED) for UINT and SINT RT surface formats. Blending is not supported for those RT surface formats. The device will automatically clamp source color channels to the respective RT surface range.	
0	Post-Blend Color Clamp Enable	Project:	All
		Format:	Enable
		If blending is enabled, this field specifies whether the blending output channels are first clamped to the range specified by Color Clamp Range. Regardless of whether this clamping is enabled, the blending output channels will be clamped to the RT surface format just prior to being written.	
		Programming Notes	
		See table in Pre-Blending Color Clamp subsection for programming restrictions as a function of RT format. This field is ignored (treated as DISABLED) for UINT and SINT RT surface formats. Blending is	



BLEND_STATE

not supported for those RT surface formats. The device will automatically clamp source color channels to the respective RT surface range.

Programming Note: CC Unit also receives 3DSTATE_WM_HZ_OP and 3DSTATE_PS_EXTRA.

Description	AlphaTestEnable
Formula	= BLEND_STATE::AlphaTestEnable && !3DSTATE_WM_HZ_OP::DepthBufferResolveEnable && !3DSTATE_WM_HZ_OP::DepthBufferClear && !3DSTATE_WM_HZ_OP::StencilBufferClear

Description	AlphaToCoverageEnable
Formula	= BLEND_STATE::AlphaToCoverageEnable && !3DSTATE_PS_EXTRA::PixelShaderDisableAlphaToCoverage

12.2.4 CC_VIEWPORT

CC_VIEWPORT

Default Value: 0x00000000, 0x00000000

The viewport state is stored as an array of up to 16 elements, each of which contains the DWords described here. The start of each element is spaced 2 DWords apart. The first element of the viewport state array is aligned to a 32-byte boundary. The Minimum Depth field in CC_Visport state must be greater than or equal to zero on D16_UNORM, D24_UNORM_X8_UINT, or D24_UNORM_S8_UINT depth formats

DWord	Bit	Description
0	31:0	Minimum Depth
		Project: All
		Format: IEEE_Float
		Indicates the minimum depth. The interpolated or computed depth is clamped to this value prior to the depth test.
1	31:0	Maximum Depth
		Project: All
		Format: IEEE_Float
		Indicates the maximum depth. The interpolated or computed depth is clamped to this value prior to the depth test.



12.3 Other Pixel Pipeline Functions

12.3.1 Statistics Gathering

If **Statistics Enable** is set in 3DSTATE_WM, the PS_DEPTH_COUNT register (see Memory Interface Registers in Volume Ia, *GPU*) will be incremented once for each pixel (or sample) that passes the depth, stencil and alpha tests. Note that each of these tests is treated as passing if disabled. This count is accurate regardless of whether **Early Depth Test Enable** is set. In order to obtain the value from this register at a deterministic place in the primitive stream without flushing the pipeline, however, the PIPE_CONTROL command must be used. See the *3D Pipeline* chapter in this volume for details on PIPE_CONTROL.



Revision History

Revision Number	Description	Revision Date
1.0	First 2012 OpenSource edition	May 2012

§§