Product Name: Volari™ V3XE Series
Document Type:
3D Programming Guide

Document No.: V3XE-3D-SPG
Document Ver.: 0.7

XGI Technology, Inc.

Volari™ V3XE 3D Programming Guide

Copyright Notice

Trademarks

XGI™ is a trademarks or a registered trademark of XGI™ Technology Inc.

VESA™ is a registered trademark of Video Electronics Standards Association.

All brand or product names mentioned are trademarks or registered trademarks of their respective holders.

Disclaimer

XGI™ Technology Inc. makes no representations or warranties regarding the contents of this manual. We reserve the right to revise the manual or make changes in the specifications of the product described within it at any time without notice and without obligation to notify any person of such revision or change. The information contained in this manual is provided for the general use by our customers and developers. Our customers and developers should be aware that the personal computer field is the subject of many patents. Our customers and developers should ensure that they take appropriate action so that their use of our products does not infringe upon any patents. It is the policy of to respect the valid patent rights of third parties and not to infringe upon or as XGI™ Technology others to infringe upon such rights.

1. V3XE 3D Engine Overview

1.1. V3XE 3D Pipeline

V3XE supports DirectX9.0b API, the H/W pipeline implementation is illustrated in Figure 1.

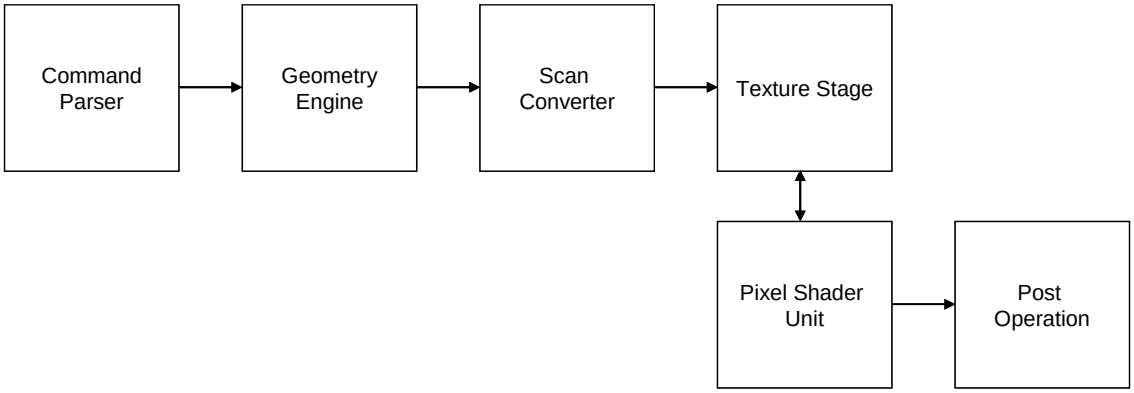


Figure 1-1 : 3D Pipeline

The brief function description of each functional block is as follows

1. **Command Parser** Decode various command types. Translate commands into H/W register setting, and pass the vertex information to succeeding stages.

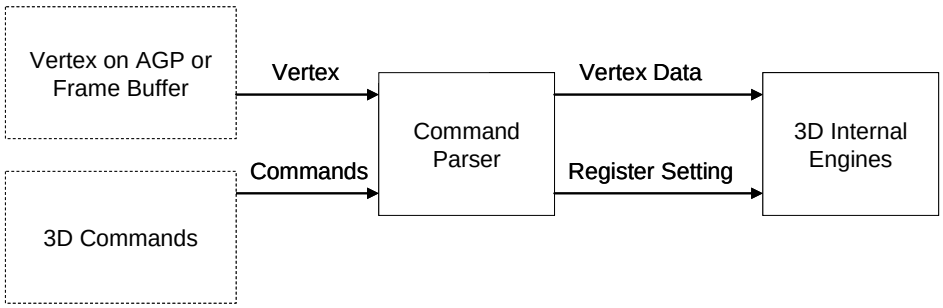


Figure 1-2: Command Parser

2. **Geometry Engine** Process triangle-based operation for triangle setup. (V3XE doesn't support H/W TnL, the TnL functions is performed by S/W TnL in V3XE driver). The Setup functions include shading setup, rasterization, AA line, AA point, jitter AA, polygon offset, culling line stipple, fill mode, flat shading, W normalization, super-sampling AA, stereo, wrap correction, Z scaling, zero testing, trivial reject.

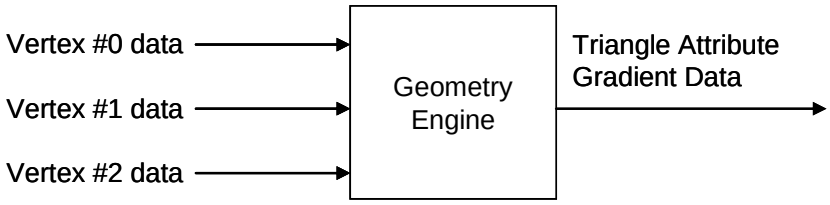


Figure 1-3: Geometry Engine

3. **Scan Converter** Transfer triangles to pixels, generate the 2D coordinate (X,Y) of each pixel.

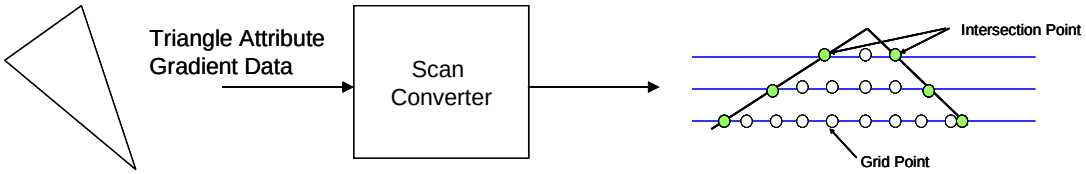


Figure 1-4: Scan Converter

4. **Texture Stage** Generate necessary information for pixel shader unit. The key functions are shading, texture sampler and texture fetch.

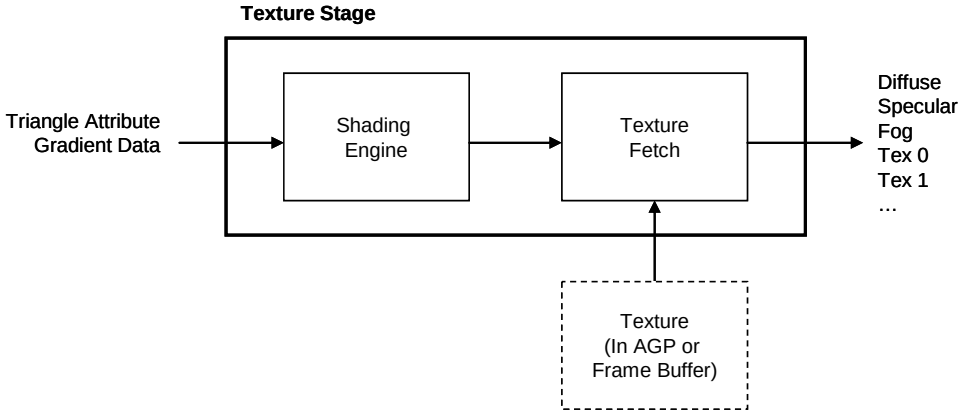


Figure 1-5: Texture Engine

5. **Pixel Shader** Programmable pipeline supports Shader Model 2.0

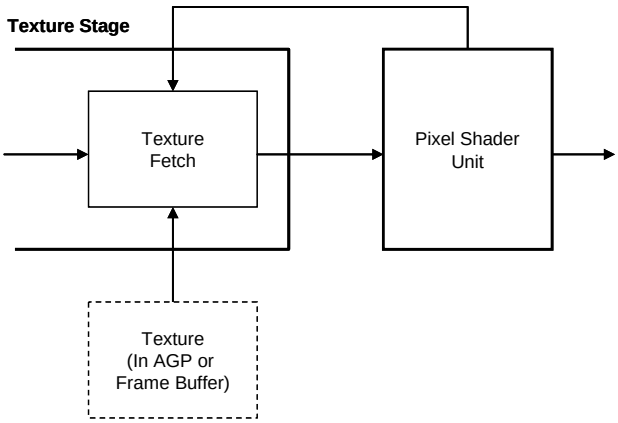


Figure 1-6: Pixel Shader Unit

6. **Post Operation** Final Z test, stencil test, alpha test and alpha blending operation.

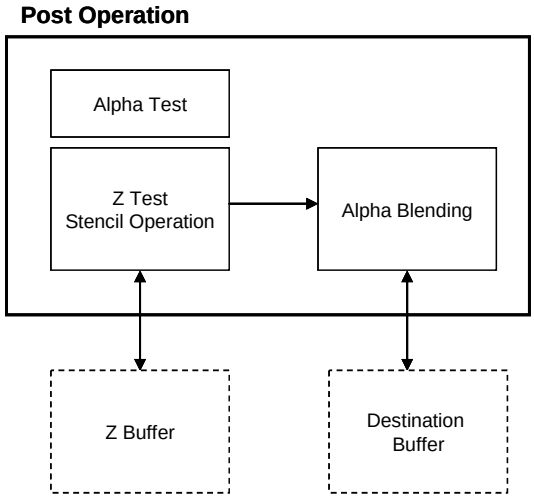


Figure 1-7: Post Engine

1.2. Hardware feature Summary

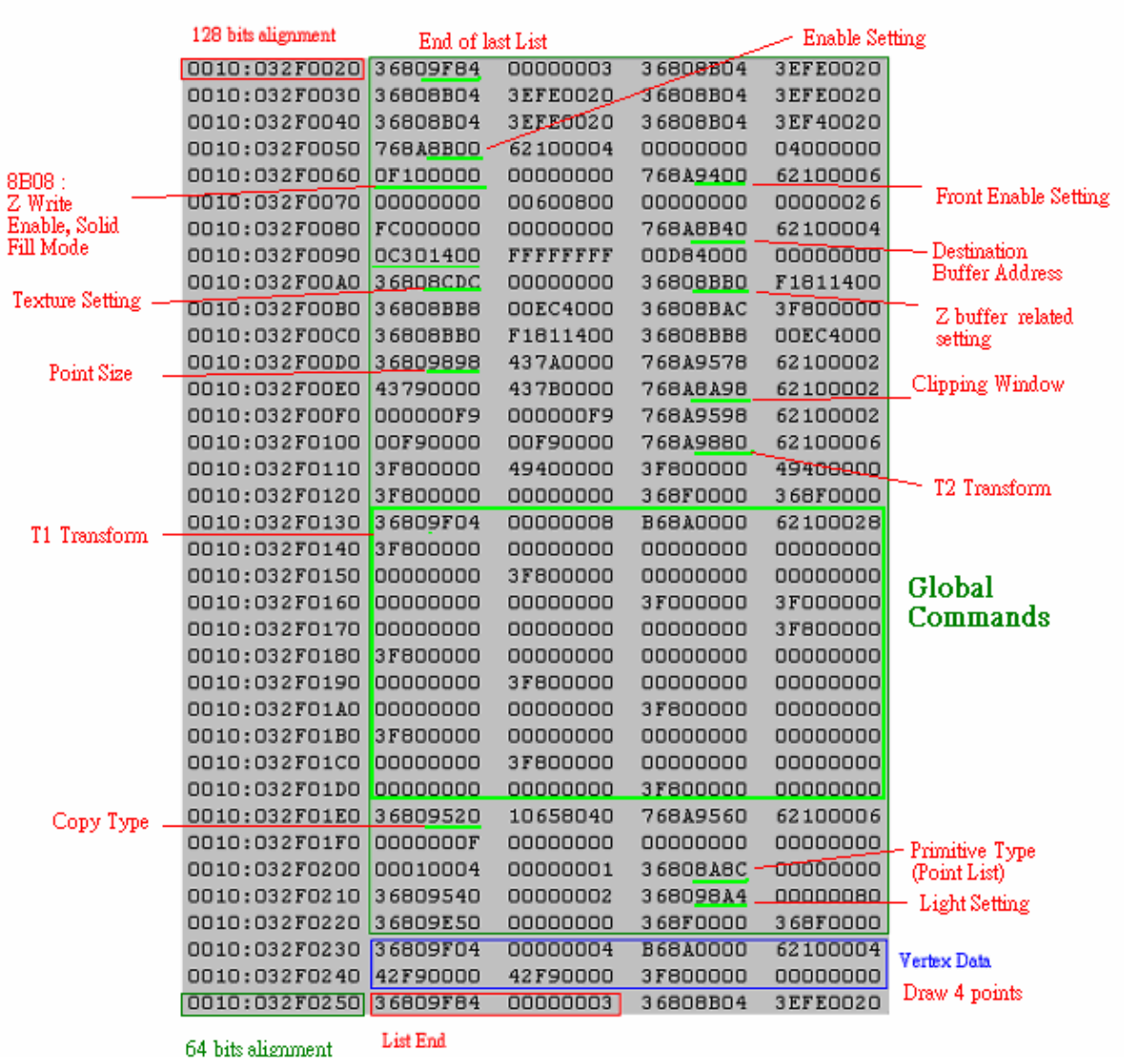
- Built-in a high performance 3D engine
  - Support 16/32 bits wide of index on vertex buffers.
  - Support 16 multi-stream sources.
  - Built-in 32-bit floating point format triangle setup engine
  - Built-in 1 set of Pixel Shader (2.0) HW capable of processing 24 bits floating point. Each set is equipped with 32 multi-threads to avoid data dependent hazard.
  - Built-in 2 pixel rendering pipelines
  - Built-in hardware stereo auto rendering engine
  - Supports AGP 4X 266 MHz for texture fetch
  - Supports AGP 4X 266 MHz for vertex fetch
  - Peak polygon rate: 50M polygon/sec @ 1 pixel/polygon with Gouraud shaded, point-sampled, linear and bilinear texture mapping
  - Peak fill rate: 500 M pixel/sec, 1000M texture/sec @ 10,000 pixel/polygon with Gouraud shaded and two bilinear textured@ 250MHz
  - Support Polygon stipple.
  - Support Line width > 1
  - Support Line Pattern
  - Support Point/Line Anti-alias.
  - Fully compliant Direct3D 9.0 Vertex Shader ver. 2.0. (DirectX 9)
  - Fully compliant Direct3D 9.0 Pixel Shader ver. 2.0. (DirectX 9)
  - Support Bump Mapping, Mipmapped Cubic Mapping and Volume Texture Supports flat and Gouraud shading
  - Support Nearest, Bi-linear, Tri-linear, Anisotropic filter.
  - Support 96 bits floating point of color format in MET/MRT to preserve data precision during multi-pass.
  - Capable of processing Max. to 2 pixels with 2 textures in one clock.
  - Supports high quality dithering function
  - Supports Z-test, stencil test, Alpha-test, and scissors clipping test
  - Supports ROP
  - Supports Z-buffer, stencil buffer and alpha buffer
  - Supports 16/24 bits integer Z buffer format and 16/24/32 bits floating point Z format
  - Supports 16/32/64/128 BPP render buffer format
  - Supports 1/4/8 stencil buffer format
  - Supports 2-side stencil.
  - Supports 1/4/8 stencil buffer format
  - Supports per-pixel texture perspective correction
  - Supports 1-tap, 2-tap, 4-tap and 8-tap texture filtering
  - Supports up to 4096x4096 texture size
  - Supports rectangle structure texture
  - Supports non power of two texture size
  - Supports 8/16/24/32/64/128 bpp texture format.

- Supports ARGB/ABGR, YUY2/AYUV, DXT1/DXT2/DXT3, U8V8/L6V5U5, R16f/R32 texture formats.
- Supports unsigned fixed, signed fixed, floating point of texture format.
- Supports 1/2/4 bpp palletize texture
- Supports 16/24/32 bpp RGB/ARGB texture format
- Supports DTX1, DTX2, DTX3 texture compression formats
- Supports texture transparency, blending, wrapping, mirror, and clamping
- Supports fogging, alpha blending
- Supports vertex fogging and fog table and T/L based vertex fog range
- Supports specular lighting
- Supports 2X/4X full scene anti-aliasing(FSAA)
- Supports back face culling
- Supports auto-stereo rendering

2. Programming guide

2.1. 3D command architecture

An overview of 3D commands for drawing a primitive:



Commands start and end at 128 bit alignment address. First adding the global commands for the engine state setting, then follow the primitive vertex data and cascade a List End.

2.2. Parser setting

Parse decodes various types of commands in the Command Queue (CQ) to 3D inputs (See Figure 22). Commands can contain the globals or vertex data. There are four types of commands: Null Command, Single Command, Burst Command and Packet Command. Null command is just used for memory alignment, Single and Burst commands can only contain the global information, and the Packet Command can issue the globals or the vertex. Different from the MMIO, these commands should be issue by the Parser Fire (0x8AD0[3:0]).

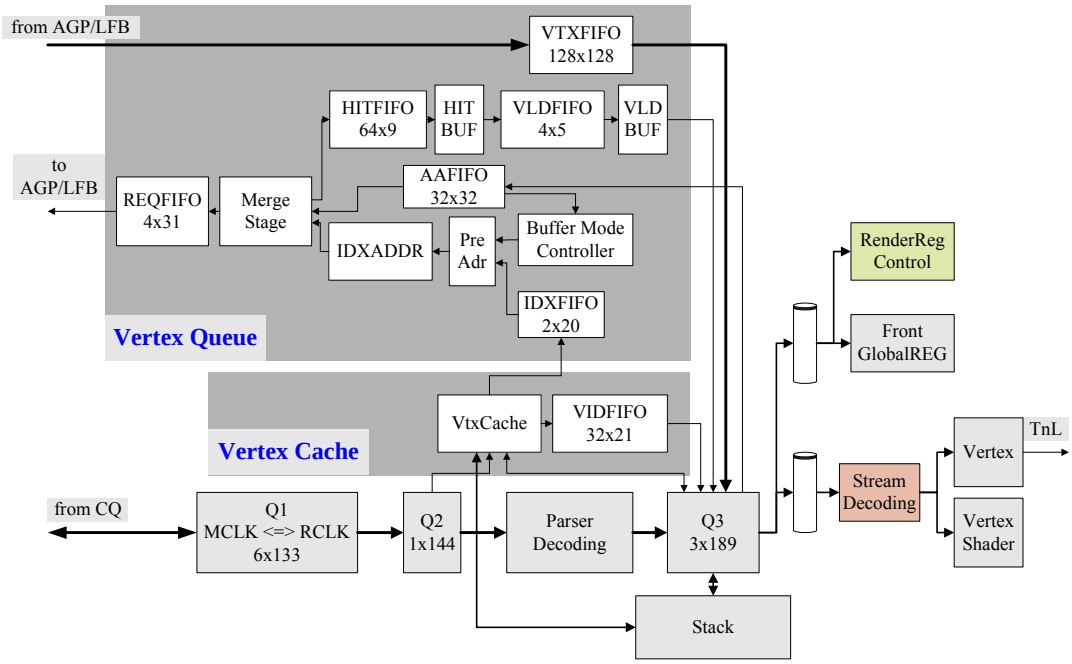
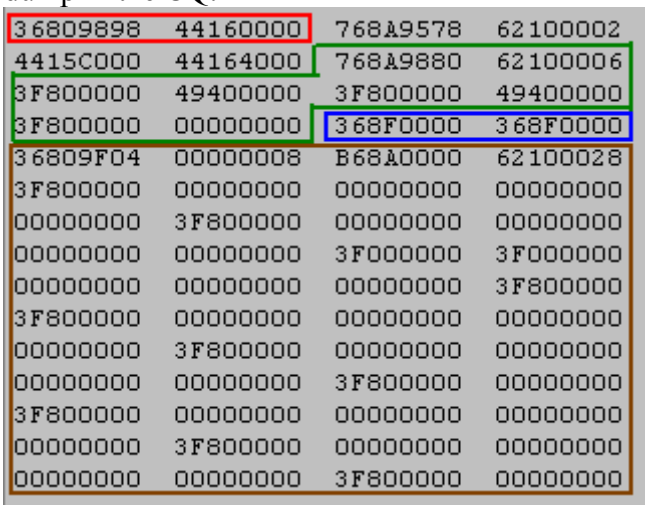


Figure 2-1: Parser architecture

Taking more detail description on the Command Type in the Command Queue (CQ):

Single command	3680(addr)	data		
Burst command	768A(addr)	621(length)		
Packet command	36809F04	(type)(offset)8	B68A0000	621(length)
Null command	368F0000			

All data in CQ must be 64 or 128 boundary, especially for Burst and Packet mode. We use Null command to fill the unused memory for alignment. Let's look some memory dump in the CQ:



The red one is the single command, the green one is the burst command for the continuing six register setting, the blue one is the null command, and the brown one is the packet command.

XG40 classify the globals into the Addressed Globals and the Packet Globals. Addressed Globals are issued by MMIO(PCI), Single and Burst commands(AGP). Packet Globals are issued by Packet Commands(AGP). Related Registers are as the following:

- 0x8AD0: Parser Fire
- 0x8AF8, 0x8AFC: Engine Status
- 0x8B70 - 0x8b7C: DMA Status
- 0x8B80, 0x8B84: Zmax, Zmin
- 0x8D60, 0x8D64: ListEnd and ParserEnd

Using Packet command to issue global, you should:

1. Issue Parser Fire 0x8AD0[3:0] to Packet Globals mode, and setting packet type an packet offset that also at 0x8AD0.
2. Issuing a Packet Header with a serious of Packet Globals.

0x8AD0 format is:

D[31:24] packet type	D[23:16] packet offset		D[3:0] fire mode

The Packet Type use 5 bits currently. Packet type D[4] is 0 represented for Front, and 1 represented for Back. Fire Modes settings are at the following :

- 0001: Index Mode
- 0010: Buffer Mode
- 0100: Packet Vertex
- 1000: Packet Global

For vertex data, you could issue vertex via Packet Vertex Mode, Index Mode or Buffer Mode. Before issue Vertex, you need to set the following globals:

- Primitive Type
- 0x9400 D[31]: EnMultiStream
- 0x9440 - 0x947C: Stream Base
- 0x94C0 - 0x94DC: Stream Stride
- 0x9500 - 0x950C: Stream DWORD
- 0x9520 - 0x953C: Copy Type
- 0x9560, 0x9564: Active Bits
- 0x9570, 0x9574: Vertex Vector Num, DOWRD, Total Vertex

The Copy Type setting tells the parse how to figure out the vertex data. The register format is:

D[13:12] CopyDW	D[11:7] CopyFMT	D[6:5] Modify	D[4:0] Dest

CopyDW[1:0] setting:

- 00 : Copy 1 DW
- 01 : Copy 2 DW
- 10 : Copy 3 DW
- 11 : Copy 4 DW

CopyFMT [4:0] setting:

- D[4]: OpenGL (signed normalization)
- D[3:2]: HW path select
- D[1:0]: HW fix2fp select and normalization select

More detail CopyFMT [4:0] setting:

- x 00 xx : Direct Copy
- x 01 xx : FP to Fix(8)
- x 10 00 : Fix(8) to FP
- x 10 01 : Fix(s15) to FP
- x 10 10 : Fix(16) to FP
- 0 11 00 : Fix(8) to FP Normalized(1/255)
- 0 11 01 : Fix(s15) to FP Normalized(1/32767)
- 0 11 10 : Fix(16) to FP Normalized(1/65535)
- 1 11 00 : 2Fix(s7)+1 to FP Normalized(1/255)
- 1 11 10 : 2Fix(s15)+1 to FP Normalized(1/65535)

Modify [1:0] setting:

- 00 : X 0 0 1
- 01 : X Y 0 1
- 10 : X Y Z 1
- 11 : X Y Z W

In Dest [4:0] setting, D[4] is reserved, and D[3:0] decide which kind of vertex attribute for vertex shader or the fixed function pipeline (decided by other globals). For the fixed pipe line, the combinations are:

00000: XYZW
00001: DiFF2
00010: Spec2
00011: NxNyNz
00100: Point Size
00101: Diff1
00110: Spec1
00111: TX0
01000: TX1
.....
01110: TX7
01111: Fog (for OpenGL)

For Vertex Shader scenario:

000001: Blend Weight
000010: Blend index
others: same as fixed pipeline D[3:0] mapping to V0 - V15

If you enabled vertex shader, you also need to specify the output attribute by setting 0x9800 and 0x9804. 0x9800 D[16:19] represent the output vector num. 0x9804 represent the output attribute(address) for two different vertex shader mode(XG40 Core specific: normal VS or Cascade VS). 0x9804 D[0:15] (D[16:31] the same order)setting:

D[0]: D3DVSDE_POSITION
D[1]: D3DVSDE_DIFFUSE2 (for OpenGL)
D[2]: D3DVSDE_SPECULAR2 (for OpenGL)
D[3]: D3DVSDE_NORMAL
D[4]: D3DVSDE_PSIZE
D[5]: D3DVSDE_DIFFUSE
D[6]: D3DVSDE_SPECULAR
D[7]: D3DVSDE_TEXCOORD0
D[8]: D3DVSDE_TEXCOORD1
D[9]: D3DVSDE_TEXCOORD2
D[10]: D3DVSDE_TEXCOORD3
D[11]: D3DVSDE_TEXCOORD4
D[12]: D3DVSDE_TEXCOORD5
D[13]: D3DVSDE_TEXCOORD6
D[14]: D3DVSDE_TEXCOORD7
D[15]: D3DVSDE_FOG (for OpenGL)

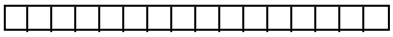
2.3. Setup setting

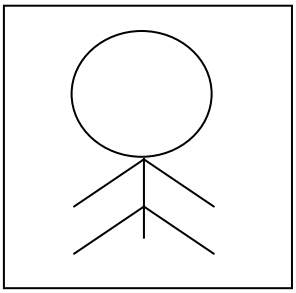
Line Stipple, Polygon Stipple, Line AA by extra texture

Line stipple and polygon stipple are applied by using texture transparency.

Line Stipple
16x1 ARGB1555
0~15 => 0~1

Polygon Stipple
32x32 ARGB1555
0~31 => 0~1





The globals we have to set:

Line Stipple	Polygon Stipple
8b08 D[30] Line stipple enable	8b08 D[31] Polygon stipple enable
8b08 D[29] OGL Line Counter enable	
9600 D[31] Clear Line Counter	
9604 D[31:4] Repeat factor s[8].19	
8b0c D[15:0] texture transparency enable	
8c60 D[28:30] Stipple texture select	
8cdc front texture number++, back texture number ++.	
8e00~8e10 no perspective, wrap, nearest, format, depth, size.	
8e14 transparency high threshold = 0x80808080	
8e1c transparency low threshold = 0x00000000	
9f04 packet command texture base, texture pitch.	

Note1: texture blending (8d20~, 8d60~) is not necessary for texture transparency.
Note2: we have to set 8b04 MSK_En_TexMap_340 and MSK_En_FastText_340

2.4. Texture Stage

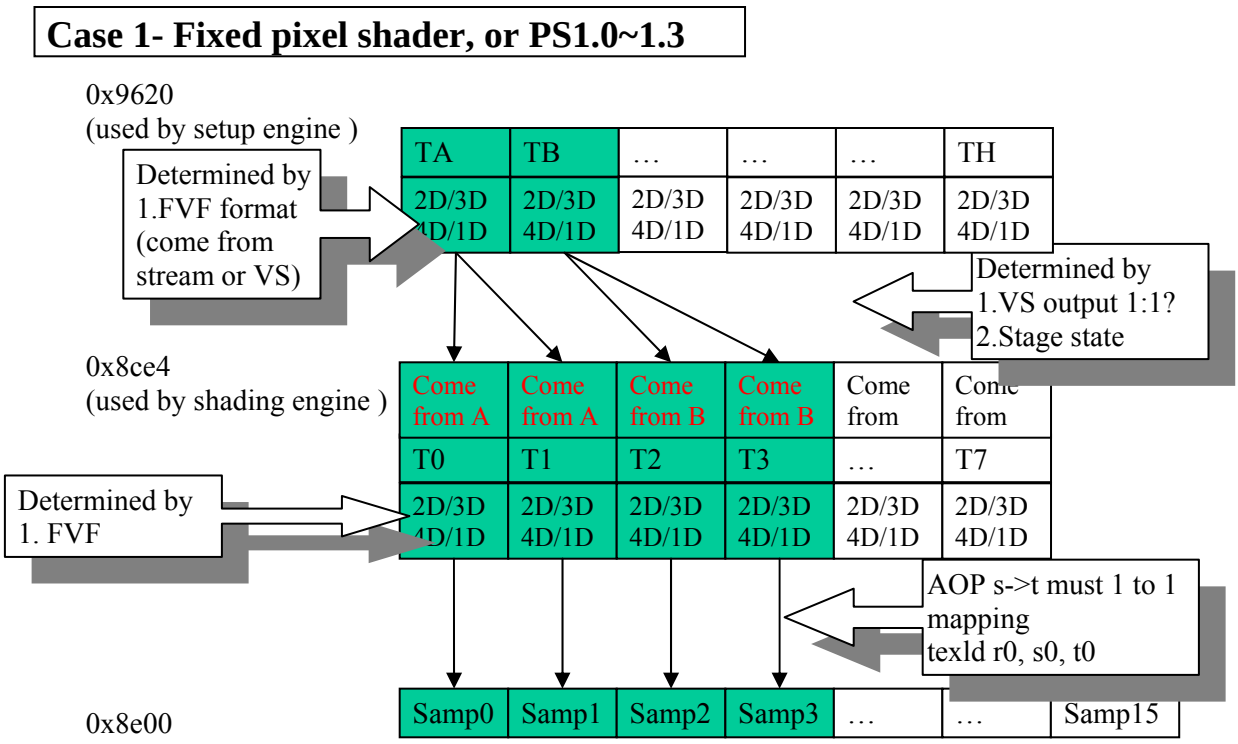
Texture turbo mode

- The setting about Texture Turbo Mode : 0x8CDC[21:20]
  - 4P2T
    - Can handle Tri-Linear

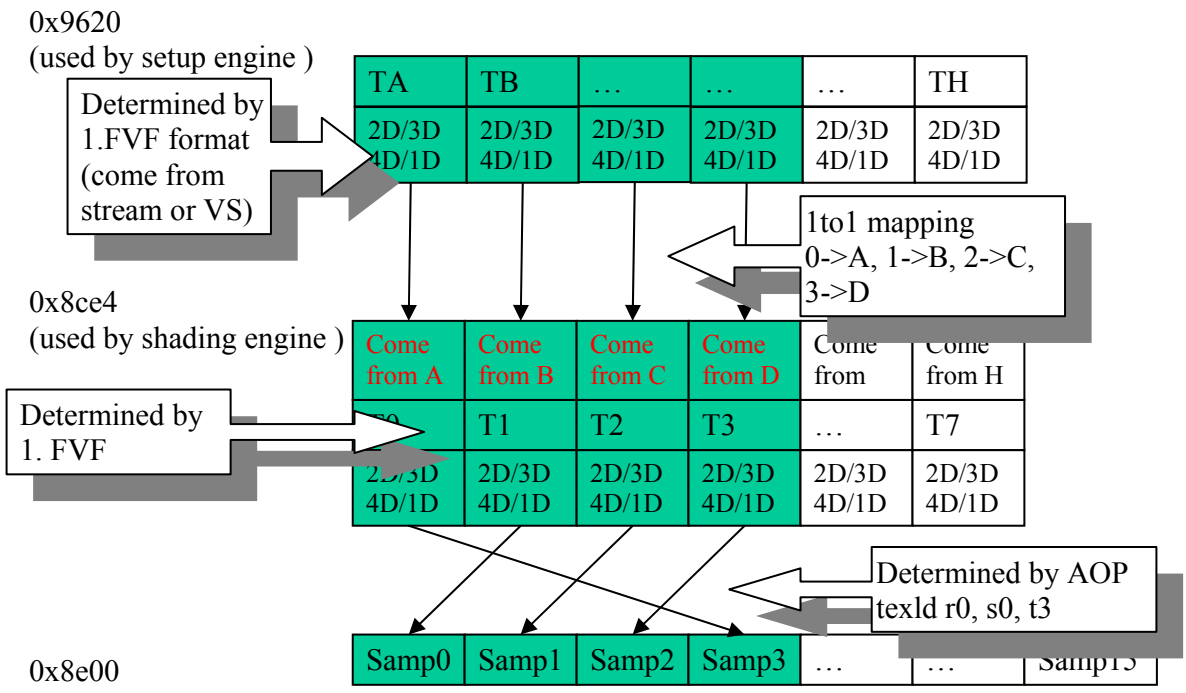
- Can use (s, t, r, q) of one tex. stage for user clip plane

- 4P4T
  - Can handle Tri-Linear
  - Only can use (s, t) for user clip plane
- 4P8T
  - Can handle Bi-Linear
  - Only can use (s, t) for user clip plane

Texture Come From & Coordinate Size



Case 2- PS1.4~2.0



2.5. Supported texture format

3 bit txfm[31:29]

000: reserved

001: 8 bpp

010: 16 bpp

011: 32bpp

100: 64 bpp

101: 128 bpp

110: DXT1 format

111: DXT2/DXT3 format

1 bit txfm[23] degamma not include Alpha,
8b00 D[20] enable degamma correction

1 bit txfm[22] METuseIndex 0: not MET format, use destination to index

1: MET format, use global index to select

1 bit txfm[21] Element bit 0: MET format with 2 channels

1: MET format with 4 channels

format	Extract	filter	Fix Pipe	P.S. Convert
L8	**No**	**Yes**	**Yes**	fix8=>float24

A8	**No**	**Yes**	**Yes**	fix8=>float24
A4L4	**No**	**Yes**	**Yes**	fix8=>float24
A8L8	**No**	**Yes**	**Yes**	fix8=>float24
L16	**No**	**Yes**	**No**	fix16=>float24
D16	**No**	**Yes**	**No**	fix16=>float24
R5G6B5	**No**	**Yes**	**Yes**	fix8=>float24
A4R4G4B4	**No**	**Yes**	**Yes**	fix8=>float24
X4R4G4B4	**No**	**Yes**	**Yes**	fix8=>float24
X1R5G5B5	**No**	**Yes**	**Yes**	fix8=>float24
A1R5G5B5	**No**	**Yes**	**Yes**	fix8=>float24
B5G6R5	**No**	**Yes**	**Yes**	fix8=>float24
A4B4G4R4	**No**	**Yes**	**Yes**	fix8=>float24
X4B4G4R4	**No**	**Yes**	**Yes**	fix8=>float24
X1B5G5R5	**No**	**Yes**	**Yes**	fix8=>float24
A1B5G5R5	**No**	**Yes**	**Yes**	fix8=>float24
V8U8	**No**	**Yes**	**No**	fix8=>float24
L6V5U5	**No**	**Yes**	**No**	fix8=>float24
R16(F)	**No**	**No**	**No**	IEEE_float16=>float24
SiS_R16(F)	**No**	**No**	**No**	SiS_float16=>float24
~~CxV8U8~~	~~**No**~~	~~**No**~~	~~**No**~~	~~fix8=>float24~~
YUYV(MSB to LSB)	**No**	**Yes**	**Yes**	fix8=>float24
UYVY(MSB to LSB)	**No**	**Yes**	**Yes**	fix8=>float24
A8R8G8B8	**No**	**Yes**	**Yes**	fix8=>float24
X8R8G8B8	**No**	**Yes**	**Yes**	fix8=>float24
A8B8G8R8	**No**	**Yes**	**Yes**	fix8=>float24
X8B8G8R8	**No**	**Yes**	**Yes**	fix8=>float24
X8L8V8U8	**No**	**Yes**	**No**	fix8=>float24
Q8W8V8U8	**No**	**Yes**	**No**	
A8X8V8U8	**No**	**Yes**	**No**	fix16=>float24
V16U16	**No**	**Yes**	**(Yes, truncate)**	fix8=>float24
L8X8V8U8	**No**	**Yes**	**No**	
Reserved	Reserved	Reserved	Reserved	Reserved
AYUV	**No**	**Yes**	**Yes**	IEEE_float16=>float24
G16R16(F)	**No**	**No**	**No**	SiS_float16=>float24
SiS_G16R16(F)	**No**	**No**	**No**	fix16=>float24
G16R16	**No**	**Yes**	**No**	**fix24 => float24**
X8D24	**No**	**No**	**No**	IEEE_float32 => float24
R32(F)	**No**	**No**	**No**	None

SiS_R32(F)	**No**	**No**	**No**	SiS_float32=>float24
SiS_Z32(F)	**No**	**No**	**No**	
				fixed=> float 24
Q16W16V16U16	**No**	**No**	**No**	fix8=>float24
SiSCxWC(like R5G5B5)	**Yes**	**Yes**	**Yes**	
				fix8=>float24
DXT1	**Yes**	**Yes**	**Yes**	
				fix8=>float24
DXT2	**Yes**	**Yes**	**Yes**	fix8=>float24
DXT3	**Yes**	**Yes**	**Yes**	

For border color assignment, please following the succeeding rules

1. Nornal case (all ARGB channel length little equal 8)
2. Special case (any one of ARGB channel length exceeds 8 bit, but S3TC and SiSCxCW and MET format excludes) , those texture fomat includes V16U16, L16, D16, G16R16, R16(F),SiS_R16(F),G16R16(F),SiS_G16R16(F),G16R16,X8D24,R32(F),SiS_R32(F)

   <Case 1>: safe mode=0 (Reg(8b04) D[5])

   - V16U16 : assign border color from A,R,G,B unsigned-8 format A_unsign8, R_unsign8, G_unsignu8, B_unsign8) to (Ulow16bit,Uhigh16bit,Vhigh16bit,Vlow16bit) format, where we store Ulow16bit into A channel and Vlow16bit into B channel
   - L16 : assign border color from A,R,G,B unsigned-8 format (A_unsign8,R_unsign8,G_unsignu8,B_unsign8) to (A_unsign8,Llow16bit,Lhigh16bit,B_unsign8,) format, where we store Llow16bit into R channel
   - D16: stuff into border color A,R,G,B channel just like L16 format
   - G16R16 : stuff into border color A,R,G,B channel just like V16U16
   - R16(F) : assign border color from A,R,G,B unsigned-8 format (A_unsign8, R_unsign8, G_unsignu8, B_unsign8) to (A_unsign8, Rhigh16bit, Rlow16bit, B_unsign8,) format
   - SiS_R16(F) : stuff into border color A,R,G,B channel just like R16F, except substitude
   - R16F(s5.10) with SiS float 16 (SiS_R16F) format
   - G16R16(F) : stuff into border color A,R,G,B channel just like V16U16,except substitude fix16 (U16/V16) with float 16 (R16F/G16F s5.10) format
   - SiS_G16R16(F) : stuff into border color A,R,G,B channel just like V16U16,except substitude fix16 (U16/V16) with SiS float 16 (SiS_R16F/SiS_G16F) format
   - X8D24 : assign border color from A,R,G,B unsigned-8 format (A_unsign8, R_unsign8,G_unsignu8,B_unsign8) to (A_unsign8,D24[23:16],D24[15:8],D24[7:0]) format

- R32(F) : assign border color from A,R,G,B unsigned-8 format (A_unsign8, R_unsign8,G_unsignu8,B_unsign8) to (R32F[31:24],R32F[23:16],R32F[15:8],R32F[7:0]) format

- SiS_R32(F) : assign border color from A,R,G,B unsigned-8 format (A_unsign8, R_unsign8,G_unsignu8,B_unsign8) to SiS_R32F[31:24], SiS_R32F[23:16], SiS_R32F[15:8], SiS_R32F[7:0]) format

<Case 2>: safe mode=1 (Reg(8b04) D[5])

- V16U16 : assign border color from A,R,G,B unsigned-8 format (A_unsign8, R_unsign8, G_unsignu8,B_unsign8) to (A_default=0xff,R_unsign8,G_unsign8,B_default=0xff) format

- L16 : assign border color from A,R,G,B unsigned-8 format (A_unsign8, R_unsign8,G_unsignu8,B_unsign8) to (A_default=0xff, R_unsigned8, G_unsigned8, B_unsigned8) format
- D16 : assign border color from A,R,G,B unsigned-8 format (A_unsign8, R_unsign8, G_unsignu8, B_unsign8) to (A_default=0xff,R_default=0x0,G_unsigned8,B_default=0x0) format

- G16R16 : stuff into border color A,R,G,B channel just like V16U16

- R16(F) : assign border color from A,R,G,B unsigned-8 format (A_unsign8, R_unsign8, G_unsignu8, B_unsign8) to (A_default=0xff,R_unsigned8,G_default=0xff,B_default=0xff) format

- SiS_R16(F) : stuff into border color A,R,G,B channel just like R16F

- G16R16(F) : stuff into border color A,R,G,B channel just like V16U16

- SiS_G16R16(F) : stuff into border color A,R,G,B channel just like V16U16
- X8D24 : assign original border color(A_default=0xff, R_default=0x0, G_unsign8, B_default=0x0)
- R32(F) : assign original border color to (A_default=0xff, R_unsign8,G_default=0xff, B_default=0xff)
- SiS_R32(F) : stuff into border color A,R,G,B channel just like R32F

3. For 4pxt and filter mode assignment, some restitutions should be considered
   - Normal texture (exclude V16U16, L16, D16 ,G16R16, Q16W16U16V16, X8D24,MET, or channels contains floating element)
     - 4p2t could support only the subsequent filter modes : nearest, bilinaer, trilinear (bilinear with mipomap), anisotropic w/o mipmap , anisotropic with mipmap, volumn texture w/o mipmap, volumn texture with mipmap

     - 4p4t could support only the subsequent filter modes : nearest, bilinaer, trilinear (bilinear with mipomap)
     - 4p8t could support only the subsequent filter modes : nearest, bilinaer

- 16-bit format texture (V16U16, L16, D16 ,G16R16)
  - 4p2t could support only the sequent filter modes : nearest, bilinaer filter mode
  - 4p4t doesn't support
  - 4p8t deson't support
- Special format (Q16W16U16V16, X8D24,MET, or any channels (A/R/G/B) contain floating element)
  - 4p2t could support only : nearest filter mode
  - 4p4t doesn't support
  - 4p8t deson't support

2.6. Supported texture filtering mode

format	Trii-linear	Remark	format	Tri-linear	Remark
L8	f		V16U16	f	
A8	f		L8X8V8U8	f	
A4L4	f		AYUV	f	
A1	m	1-bit	G16R16(F)	FP16	
A8L8	f		G16R16	f	
L16	f		X8L24		
D16	f		X8D24		
R5G6B5	f		R32(F)		
A4R4G4B4	f		D32F		
X4R4G4B4	f		S8D24		
X1R5G5B5	f		A2B10G10R10	f	
A1R5G5B5	f		A2R10G10B10	f	
B5G6R5	f		A2W10V10U10	f	
A4B4G4R4	f		Q16W16V16U16	f	
X4B4G4R4	f		SiSCxWC(like R5G5B5)	f	CWC
X1B5G5R5	f		A16B16G16R16	f	
A1B5G5R5	f		A16B16G16R16f	FP16	
V8U8	f		A16R16G16B16	f	
L6V5U5	f		A16R16G16B16f	FP16	
R16(F)	FP16		G32R32f		
YUYV(MSB to LSB)	f		A32B32G32R32f		
UYVY(MSB to LSB)	f		A32R32G32B32f		
G8R8_G8B8	f		CWClosslessAlpha	f	CWC
R8G8_B8G8	f		CWClossyAlpha	f	CWC
A8R8G8B8	f		DXT1	f	
X8R8G8B8	f		DXT2	f	
A8B8G8R8	f		DXT3	f	
X8B8G8R8	f		DXT6	f	3Dc: ATI2
X8L8V8U8	f				
Q8W8V8U8	f				
A8X8V8U8	f				

f: fixed-point, **FP16**:16-bit floating-point, **m**: monochrome filtering

Limitation

1. Do not support gamma correction or de-gamma correction for floating-point texture. However, this can be dome by pixel shader.

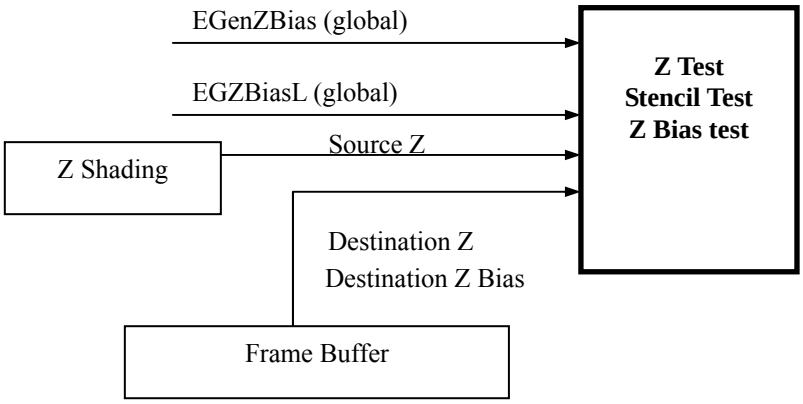
2.7. Post setting

Z Test / Z Bias

- Relative Globals

Z Test		
8B08[19]	EGenZT	Enable Z test
8BB0[18:16]	EGZTMD	Z test mode
Z Bias Test		
8B08[21]	EGenZBias	Enable Z bias test
8BB0[31:28]	EGZBiasL	Z Bias Level

- Hardware Structure



- Formula

•Z test Pass

EGZTMD	Z test Pass (ZTPass)
b000	Z never pass
B001	Pass if Zsrc < Zdst
B010	Pass if Zsrc = Zdst
b011	Pass if Zsrc <= Zdst
b100	Pass if Zsrc > Zdst
b101	Pass if Zsrc != Zdst
b110	Pass if Zsrc >= Zdst
b111	Always ass

•Z Pass

```
if  (EGenZBias = 1)
{
if  (EGZBiasL < ZbiasLdst)
                ZPass = 1;
        else if (EGZBiasL = ZbiasLdst)
                ZPass = ZTPass;
        else
                ZPass = 0;
}
else
        ZPass = ZTPass;
```

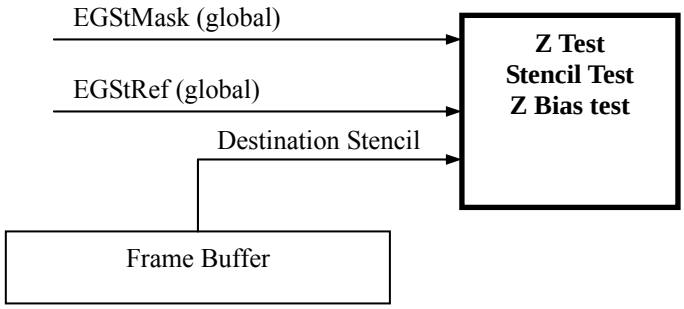
Note: Above the Z Bias define is DX5 Spec. 330 has another z bias function implentation which use polygon offset function at setup stage. This implentation is post transparency.

Stencil Test

Relative Globals

8B00[9]	EGenStT	Enable stencil test
8b00[10]	EGenStTCCW	Enable ccw Stencil Test
9410[20]	RGXEnClipDet	Enable Clipping to caculate Det
8B20[30:28]	EGStTMD	Stencil test mode
8B20 [27:20]	EGStRef	Stencil reference value
8B20 [19:12]	EGStMASK	Stencil mask value
8B20 [10:8]	EGStFAIL	Stencil test fail operation
8B20 [6:4]	EGStZFAIL	Stencil test pass and z test fail operation
8B20 [2:0]	EGStZPASS	Stencil test pass and z test pass operation
8B24[30:28]	EGStTMD	CCW Stencil test mode
8B24 [27:20]	EGStRef	CCW Stencil reference value
8B24 [19:12]	EGStMASK	CCW Stencil mask value
8B24 [10:8]	EGStFAIL	CCW Stencil test fail operation
8B24 [6:4]	EGStZFAIL	CCW Stencil test pass and z test fail operation
8B24 [2:0]	EGStZPASS	CCW Stencil test pass and z test pass operation

Hardware Structure



Formula

•Stencil Pass

EGStTMD	Stencil Pass (StPass)
b000	Stencil never pass
b001	Pass if (EGStRef & EGStMask) < (StBuf & EGStMask)
b010	Pass if (EGStRef & EGStMask) = (StBuf & EGStMask)
b011	Pass if (EGStRef & EGStMask) <= (StBuf & EGStMask)
b100	Pass if (EGStRef & EGStMask) > (StBuf & EGStMask)
b101	Pass if (EGStRef & EGStMask) != (StBuf & EGStMask)
b110	Pass if (EGStRef & EGStMask) >= (StBuf & EGStMask)
b111	Always pass

If (EGenStT = 0) { StPass = 1}

•Stencil Update

```
If ( STPass = 0)
        ST_update_mode = EGStFAIL;
else if (STPass = 1 and ZPass = 0)
        ST_update_mode = EGStZFAIL;
else if (STPass = 1 and ZPass = 1)
        ST_update_mode = EGStZPASS;
```

ST_update_mode	New Stencil value
b000	KEEP (Stnew = StBuf)
b001	ZERO (Stnew = 0)
b010	REPLACE (Stnew = StRef)
b011	INCRSAT(if(Stbuf != Stmax) {Stnew = Stbuf + 1}
b100	DECRSAT(if(Stbuf != Stmin) {Stnew = Stbuf - 1}
b101	INVERT (Stnew = ~Stbuf)
b110	INCR (Stnew = Stbuf + 1)
b111	DECR (Stnew = Stbuf – 1)

Description

When Stencil test turns on, coarse z can not drop any valid point even if the pixel is coarse z test fail. We should update stencil value of every valid point.

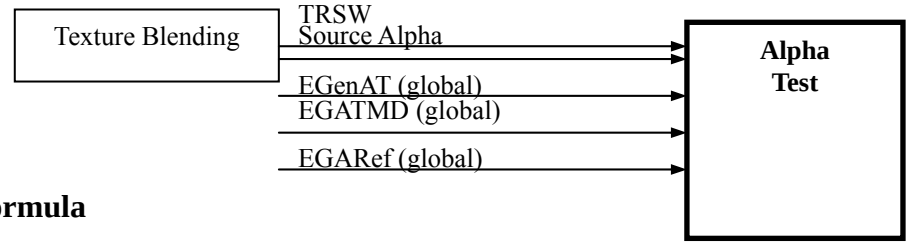
When two-side stencil turns on, 0x9410 D[20] should be enable for HW calculating which side to select.

Alpha test

Relative Globals

8B00[11]	EGenAT	Enable alpha test
8B34[26:24]	EGATMD	Alpha test mode
8B34[23:16]	EGAREF	Alpha reference value

Hardware Structure



Formula

EGATMD	Alpha test Pass (ATPass)
b000	Alpha test never pass
b001	Pass if Asrc < EGARef
b010	Pass if Asrc = EGARef
b011	Pass if Asrc <= EGARef
b100	Pass if Asrc > EGARef
b101	Pass if Asrc != EGARef
b110	Pass if Asrc >= EGARef
b111	Always ass

If (EgenAT = 0 and TRSW !=0) { ATPass = 1}
TRSW is pass form texture for fuction texture transparency and pixel shader command texkill.
When a pix kill by those functions at texture, the TRSW will set to 0.

Z Pixel Write and Color Pixel Write

Z Pixel Write

```
If ( EgenZW = 1)
{
        if ( EGenStT = 1 or Fist touch pixel)
                ZPixWr = PixelValid;
        else
                ZPixWr = ZPass and StPass and PixelValid and ATPass;
}
else
{
        ZPixWr = 0
}
```

Note: Driver won't clear frame buffer z for first touch case. We need to write Z even when z test fail for first touch point.

```
        If (First touch pixel)
    {  If(ZT pass and ATP pass)
                    { write new z value}
     else
                    {write initial Z value}
        }
```

Color Pixel Write

ColorPixWr = ZPass and StPass and PixelValid and ATPass;

Write Mask and Color Equal Not Write

Relative Globals

8B00[2]	EGEnDsEqSrNoCW	Enable No color write when Cdst=Csrc
8B00[1]	EGenCWrMask	Color write mask enable
8B00[1]	EGenZWrMask	Z write mask enable
8BB4	EGZWrMASK	Zwrite mask
8BBC	EGZWrExtMASK	Zwrite mask for Extended case, i.e. stereo or FSAA
8B44	EGD0CWrMASK	Destination 0 color write mask
8B50	EGD1CWrMASK	Destination 1 color write mask
8B5C	EGD2CWrMASK	Destination 2 color write mask
8B68	EGD3CWrMASK	Destination 3 color write mask

Formula

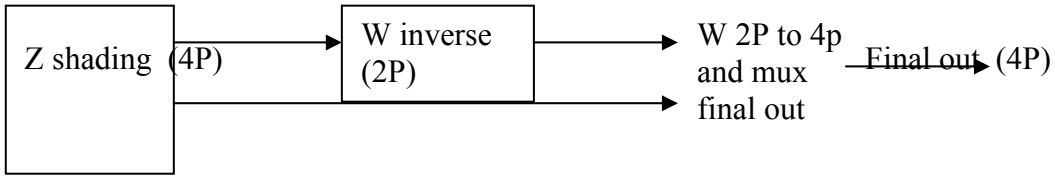
```
If (enmask)
{
        For (i=0;i<format_length;i++)
        {
        if mask[i]
                out[i]=scr[i]
        else
                out[i]=dst[i]
        }
}
if (Egencwmask and cmask=allzero) or (Csrc=Cds)
        Color pixvalid =0
```

W Buffer

Relative Globals

8B08[22]	EGenWbuffer	Enable W buffer as depth comparison

Hardware Structure



Formula

If (EgenWbuffer)
{ Zscr =1/Zshading} Zshading now is RHW shading value

Note:1. For the cost of the hardware, 330 only has two pixel inverse .
2. If Wbuffer is enable. The coarse z can not turn on.
3. W buffer value can large then one. Since only the floating z format can be use when W buffer turn on.
4. When W buffer enable, the w normalize can not turn on. This situation will cause color perspective error at 330 hardware. Since when w buffer turn on, 330 can not tuen on color perspective.

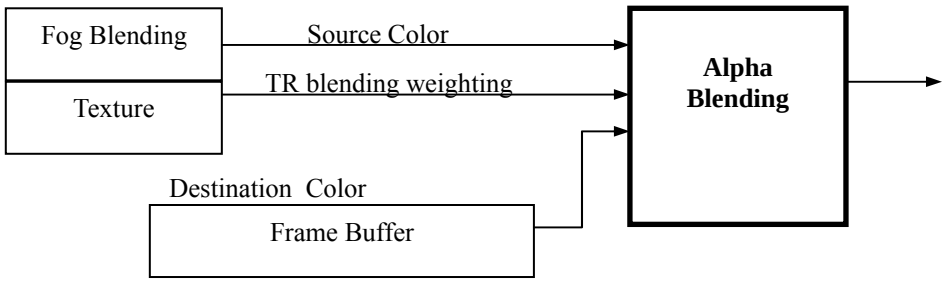
Alpha Blending/Transparency Blending

Relative Globals

Alpha Blending		
8B00[8]	EGenBLEND	Blending enable
8B00[6]	EGenABLSepSet	Enable alpha blending separate setting
8B28[26:24]	EGCBLEqMode	Color blending equation mode
8B28 [8:4]	EGCBLsrc	Color Source blending mode
8B28 [3:0]	EGCBLdst	Coloe destination blending mode
8B30[23:21]	EGABLEqMode	Alpha blending equation mode
8B30 [8:4]	EGABLsrc	Alpha Source blending mode
8B30 [3:0]	EGABLdst	Alpha destination blending mode
Transparency Blending		
8B0C[15]	EGenTX0TR	Texture 0 transparency enable
8B0C[14]	EGenTX1TR	Texture 1 transparency enable
8B0C[13]	EGenTX2TR	Texture 2 transparency enable
8B0C[12]	EGenTX3TR	Texture 3 transparency enable
8B0C[11]	EGenTX4TR	Texture 4 transparency enable
8B0C[10]	EGenTX5TR	Texture 5 transparency enable

8B0C[9]	EGenTX6TR	Texture 6 transparency enable
8B0C[8]	EGenTX7TR	Texture 7 transparency enable
8B0C[7]	EGenTX8TR	Texture 8 transparency enable
8B0C[6]	EGenTX9TR	Texture 9 transparency enable
8B0C[5]	EGenTX10TR	Texture 10 transparency enable
8B0C[4]	EGenTX11TR	Texture 11 transparency enable
8B0C[3]	EGenTX12TR	Texture 12 transparency enable
8B0C[2]	EGenTX13TR	Texture 13 transparency enable
8B0C[1]	EGenTX14TR	Texture 14 transparency enable
8B0C[0]	EGenTX15TR	Texture 15 transparency enable

Hardware Structure



Formula

•Blending Factor

```
DoTR = (EGenTX0TR or EGenTX1TR or EGenTX2TR or EgenTx3TR)
If ( EGenBLEND)
        AFSrc = CFsrc = EGCBLsrc;
        AFdst = CFdst = EGCBLdst;
else if (DoTR)
        AFsrc = 1;
        AFdst = 0;
        CFsrc = TR_Blending_Facror;
        CFdst = (1- TR_Blending_Facror);
else
        AFsrc = 1;
        AFdst = 0;
        CFsrc = 1
        CFdst = 0;
```

•Alpha

```
If (EGenABLSepSet = 1)
        Alpha_eq_mode = EGABLEqMode;
else
        Alpha_eq_mode = EGCBLEqMode;

If (Alpha_eq_mode == 0x0)
        Aout = Asrc * AFsrc + Adst * AFdst
else if  (Alpha_eq_mode == 0x1)
        Aout = Asrc * AFsrc - Adst * AFdst
else if  (Alpha_eq_mode == 0x2)
        Aout = Max(Asrc, Adst)
else if  (Alpha_eq_mode == 0x3)
        Aout = Min(Asrc, Adst)
else if  (Alpha_eq_mode == 0x4)
        Aout = -(Asrc * AFsrc - Adst * AFdst)
```

•Color (RGB)

```
If (EGCBLEqMode == 0x0)
        Cout = Csrc * CFsrc + Cdst * CFdst
else if  (EGCBLEqMode == 0x1)
        Cout = Csrc * CFsrc - Cdst * CFdst
else if  (EGCBLEqMode == 0x2)
        Cout = Max(Csrc, Cdst)
else if  (EGCBLEqMode == 0x3)
        Cout = Min(Csrc, Cdst)
else if  (EGCBLEqMode == 0x4)
        Cout = -(Csrc * CFsrc - Cdst * CFdst)
```

Description

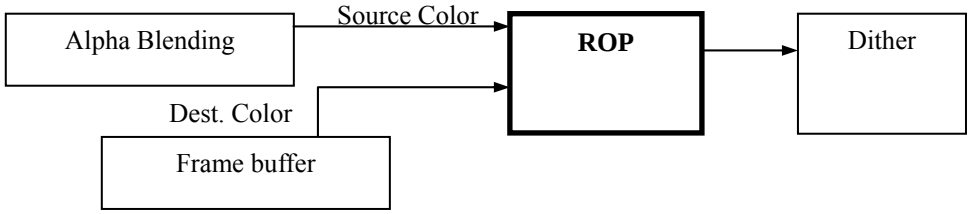
- Cout : Color after alpha blending (8-bit)
- Csrc : Source color, after fog blending.(8-bit)
- Fsrc : Source color blending factor (8-bit)
- Cdst : Destination color read form frame buffer (8-bit).
- Fdst : Destination color blending factor (8-bit)

ROP

Relative Globals

8B40[27:24]	EGDST0ROP	Destination 0[L] raster operation
8B4C[27:24]	EGDST1ROP	Destination 1[R] raster operation
8B00[21]	EGenStereo	Enable automatic stereo

Hardware Structure



Formula

```
If ( EgenStereo )
{
        If ( left_eye_triangle )
                DSTROP          =       EGDST0ROP
        Else
                DSTROP          =       EGDST1ROP
}
Else
{
        DSTROP          =       EGDST1ROP
}

Aout = Asrc
```

Cout = DSTROP(Csrc, Cdst)

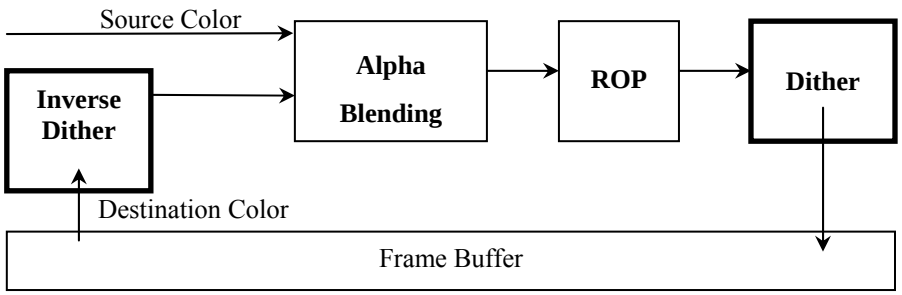
EGDST0ROP(EGDST1ROP)	Function
b0000	Cout = 0
b0001	Cout = ~(Csrc | Cdst)
b0010	Cout = (~Csrc) & Cdst
b0011	Cout = ~Csrc
b0100	Cout = Csrc & (~Cdst)
b0101	Cout = ~Cdst
b0110	Cout = Csrc⊕ Cdst
b0111	Cout = ~(Csrc & Cdst)
b1000	Cout = Csrc & Cdst
b1001	Cout = ~(Csrc ⊕ Cdst)
b1010	Cout = Cdst
b1011	Cout = (~Csrc) | Cdst
b1100	Cout = Csrc
b1101	Cout = Csrc | (~Cdst)
b1110	Cout = Csrc | Cdst
b1111	Cout = 1

Dither and Inverse Dither

Relative Globals

8B00[5]	EGenDith	Enable dither
8B00[4]	EGDithIdxSel	Turbo dither mode
8B00[3]	EGenInvDith	Enable inverse dither

Hardware Structure



Dither

Dither function can be turned on when destination color format is 16-bit mode. There are total 16 tables for dither (DitherTable[16]). These tables are pre-defined and implemented in H/W. The 16 tables are as follows :

Pattern 0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Pattern 1

0	0	0	0
0	0	0	0
0	0	0	0
1	0	0	0

Pattern 2

0	0	0	0
0	0	0	0
0	0	0	0
1	0	1	0

Pattern 3

0	0	0	0
0	0	1	0
0	0	0	0
1	0	1	0

Pattern 4

0	0	0	0
1	0	1	0
0	0	0	0
1	0	1	0

Pattern 5

0	0	0	0
1	0	1	0
0	1	0	0
1	0	1	0

Pattern 6

0	0	0	1
1	0	1	0
0	0	0	0
1	0	1	0

Pattern 7

0	0	0	1
1	0	1	0
0	1	0	1
1	0	1	0

Pattern 8

0	1	0	1
1	0	1	0
0	1	0	1
1	0	1	0

Pattern 9

0	1	0	1
1	0	1	0
0	1	0	1
1	1	1	0

Pattern 10

0	1	0	1
1	0	1	1
0	1	0	1
1	1	1	0

Pattern 11

0	1	0	1
1	0	1	1
0	1	0	1
1	1	1	1

Pattern 12

0	1	0	1
1	1	1	1
0	1	0	1
1	1	1	1

Pattern 13

0	1	0	1
1	1	1	1
1	1	0	1
1	1	1	1

Pattern 14

0	1	1	1
1	1	1	1
1	1	0	1
1	1	1	1

Pattern 15

0	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

How to select an appropriate table for each color channel of a pixel is depending on the destination color format. The selection criterion is as follows

Color Format	Dither Table No	Dither Table No	Dither Table No
RGB444	R&0xF	G&0xF	B&0xF
RGB555	(R&0x7)<<1	(G&0x7)<<1	(B&0x7)<<1
RGB565	(R&0x7)<<1	(G&0x7)<<2	(B&0x7)<<1

For example, if color format is RGB555 and RGB is (0x33, 0x44, 0x55), the corresponding dither table number for each channel is

Dither Table No of R = ((0x33)&0x7) <<1 = 0x6;

Dither Table No of G = ((0x44)&0x7) <<1 = 0x8;

Dither Table No of B = ((0x55)&0x7) <<1 = 0xA;

Once a table is chosen, the next step is to decide if each channel needs to be dithered. We can see that there are 16 elements in each dither table. If we choose a element '1', it indicates this channel needs dither. The question is , how to choose ? Suppose we have chosen a dither table as follows.

Pattern 9

j\ i	00	01	10	11
00	0	1	0	1

01	1	0	1	1
10	0	1	0	1
11	1	1	1	0

In this pattern, each element is indexed by index i and j, the selection criterion of i/j will give various dither effect. The criterion we use is relative to the coordinate of each pixel. If the coordinate of a pixel is (x, y), the index i and j are given as following formula:

Index i = (x[1], y[2] xor x[0])
Index j = (y[1], x[2] xor y[0])

For example, if (x, y) = (0x33, 0x45), according to the formula, the index (i, j) =(b11, b00), and we have an element '1'. This indicates the pixel needs dither.

However, we have another index selection criterion, it's easier but loses image quality. This mode is turned on by the global signal EGDithIdxSel. The index selection is re-defined as:

```
If (EGDithIdxSel)
{
        Index i = (x[1], x[0])
        Index j = (0, 0)
}
```

The final step is to dither RGB channel of each pixel. We must do dither according to different color formats. Three cases need to be considered:

1. 8-bit to 6-bit : Cdith = Cold + 0x04;
2. 8-bit to 5-bit : Cdith = Cold + 0x08;
3. 8-bit to 4-bit : Cdith = Cold + 0x10;

For example, suppose the color of a pixel is (R, G, B) = (0x77, 0x99, 0xF8), and the destination is RGB565, the dither is performed as follows :

R' = R + 0x08 = b**0111_0**111 + b**0000_1**000 = b**0111_1**111

G' = G + 0x04 = b**1001_10**01 + b**0000_01**00 = b**1001_11**01

B' = B + 0x08 = b**1111_1000** + b**0000_1**000 = b**1111_1**000 (clamped)

After dither, the color of pixel becomes (0x7F, 0x9D, 0xF8) and the final color write the DRAM is RGB565 = b**(0111_1)(100_111)(1_1111)** = 0xFCFF.

Summary of the description above, the dither steps are:

1. Select an appropriate pattern
2. Select a element from the pattern

3. Do dither on pixel if necessary, and , do not forget to clamp the color to the maximum range.

Inverse Dither

Inverse dither is mainly to solve the issue of color deviation when multiple blending in a scene. The steps of inverse dither are similar to steps of dither and this function is controlled by the global signal **EGenInvDith**.

The first step is select an appropriate inverse dither pattern according to the color format of each channel, total three cases:

1. 4-Bits Inv-Dither to 8-bits

   Pattern 4to8

j\ i	00	01	10	11
00	+7	-1	+5	-3
01	-5	+3	-7	+1
10	+4	-4	+6	-2
11	-8	0	-6	+2

2. 5-Bits Inv-Dither to 8-bits

   Pattern 5to8

j\ i	00	01	10	11
00	+3	-1	+2	-2
01	-3	+1	-4	+0
10	+2	-2	+3	-1
11	-4	0	-3	+1

3. 6-Bits Inv-Dither to 8-bits

   Pattern 6to8

j\ i	00	01	10	11
00	+1	-1	+1	-1
01	-2	0	-2	0
10	+1	-1	+1	-1
11	-2	0	-2	0

For example, for a color format RGB565, you must perform inverse dither on R and B channel according to pattern 5to8 and G channel according to pattern 6to8.

The second step is to select an element from the chosen pattern. The index selection is the same as we do in dither:

Index i = (x[1], y[2] xor x[0])
Index j = (y[1], x[2] xor y[0])

The final step is to do inverse dither on each channel according to the selected elements. Suppose the elements we select for each channel are (InvDithR, InvDithG, InvDithB),

the destination color is 0xFCFC and the coordinate is (0x33, 0x45) the inverse dither is performed as:

Color Format	Original Color (R, G, B)	(InvDithR, InvDithG, InvDithB)	Color after inverse dither
RGB444	(0xC0, 0xF0, 0xC0)	(-3, -3, -3)	(0xBD, 0xFD, 0xBD)
RGB555	(0xF8, 0xC8, 0xE0)	(-2, -2, -2)	(0xF6, 0xC6, 0xDE)
RGB565	(0xF8, 0x9C, 0xE0)	(-2, -1, -2)	(0xF6, 0x9B, 0xDE)

Summary the steps of inverse dither:

1. Select an appropriate inverse dither pattern.
2. Select an element from pattern.
3. Do the inverse dither and do not forget to clamp the final color between the maximum and minimum color range.