Product Name: Volari™ XP10 Series
Document Type:
3D Software Programming Guide

Document No.: XP10-3D-SPG
Document Ver.: 1.0

XGI Technology, Inc.
December 19, 2005

Volari™ XP10 3D Software 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.

Revision History

Date	Revision	Modification
2005-12-19	1.0	• Initial Release • Based on V8300 3D SPG version 1.2

Contents

1. Overview of XP10 Graphics Engine

1.1. Working on XP10 with Microsoft DirectX

Figure 1-1 illustrates the software environment of PC's display subsystem:

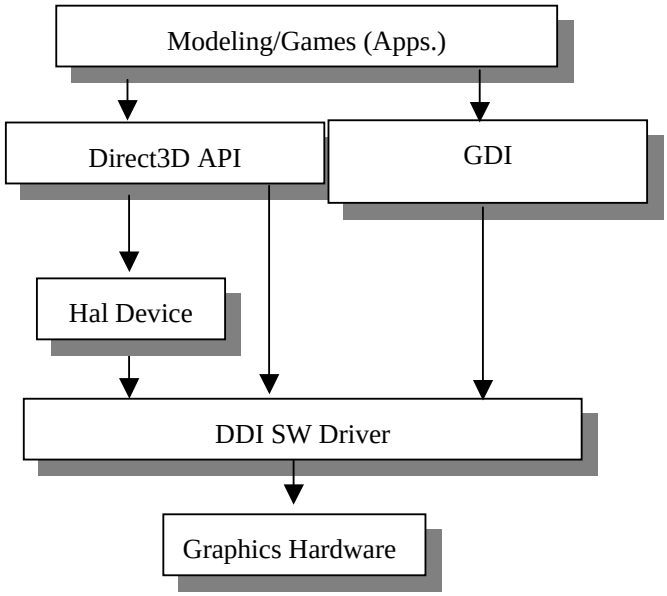


Figure 1-1 Graphics Software Architecture

There are three stages for completely rendering a scene of 3D geometries

1) Application:
   a) to handle user input and form data to device-associated format;
   b) to instruct driver and hardware execute render commands;
2) Driver / DDI:
   a) to transform data from application to hardware internal format;
   b) to communicate with hardware;
3) Hardware:
   a) to do real rendering jobs;
   b) to help CPU do geometric operations.

There are two main part of graphics chips

1) Geometry Processor
   a) to transform vertices form 3D modeling space to 2D screen space;
   b) to assemble primitives out of vertex stream;

2) Rendering Engine
   a) to rasterize 2D primitives to pixels;
   b) to sample and filter textures to get color;

c) to do per-pixel shading;
d) to blend different color components to form a single pixel color;
e) to write pixel color in the frame buffer.

1.2. Bump Loop Control Architecture

Microsoft DirectX graphics pipeline requirement is as below:

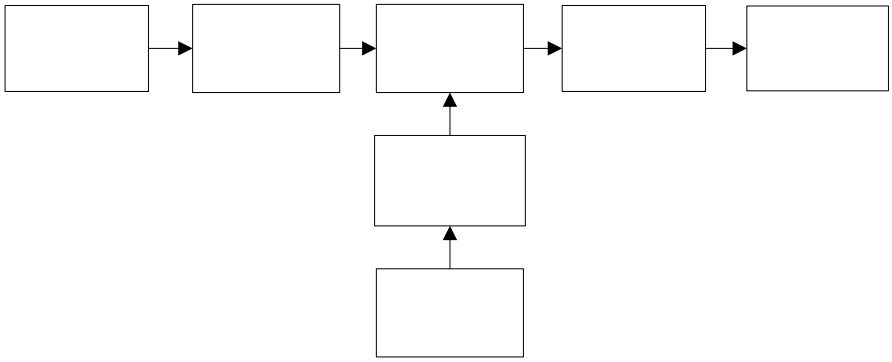


Figure 1-2 Microsoft DirectX graphics pipeline requirement

To cover the long latency of texture sampling instruction, and also to save gates, XP10 designs the bump loop control architecture, shares texture addressing and color operation. Then architecture becomes one below:

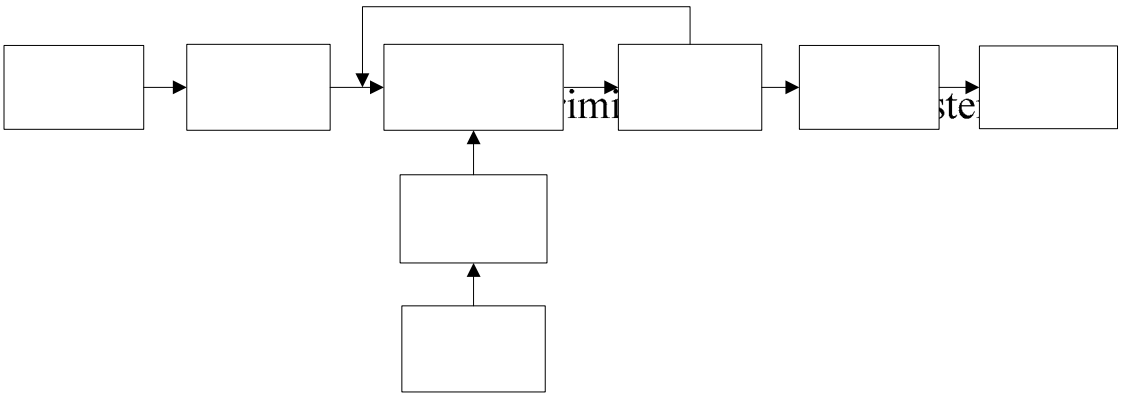


Figure 1-3 XP10 Bump Loop Control Architecture

2. Programming Concepts

This chapter gives you some concepts and very rough perception on how to program XP10 graphics engine. For detail of each function please see next chapters. First we introduce a command/register structure. Then give you simple (non-executable) examples to help you understand programming methods.

The commands in XP10 graphics engine have two types.

- Render State: set up registers according to the situations, such as base address of destination buffer, texture magnification filter, etc. Any attribute setting should be finished in this step.
- Render Primitive: transmit vertex parameters to draw the primitives. These primitive data includes x, y, z, w, diffuse color, specular color, (normal vectors,) texture coordinates, etc.

In general, render states will affect the rendering of following primitives.

2.1. Register/Command Structure

A command consists of an 8-bit CMD_ID and 24-bit Header with optionally 32-bit aligned data. The format of a command is as following:

```
COMMAND ::=
{
    BITS_8      CMD_ID;
    BITS_24     Header;
    [BITS_32    data;]*             //optionally repeated;
}
```

The smallest command contains only CMD_ID and Header without data. Therefore the smallest command is 32-bit. This implies all commands are 32-bit aligned. For register setting command such as rendering state register setting, the CMD_ID is actually used for deriving the register address. In this category of commands, the register address is defined as (CMD_ID<<2). Therefore, when S/W driver issues register setting commands through PCI, it just uses CMD_ID as register address (= (CMD_ID<<2)). This implementation eliminates the need for extra register address FIFO and unifies the command list format for both PCI and PCI-E.

The 24 to 31 bits of each register are register ID(divided by 4), which is used to identify register. The Graphics engine knows which register the command belongs to in this way. This scheme reduces the number of commands, improving performance. Next chapter defines 3D registers and it won't mention D[31:24] because they are all register ID. The content of D[31:24] is register ID divided by 4. For instance, D[31:24] of register 0x020 must be 0000 1000(0x08*4=0x20).

To transmit a single DWORD register, just program:

```
WRITE_REG(REG_ID + 24bit data);  // REG_ID = CMD_ID.
```

WRITE_REG is to add a command (32bit) to command list in PCI-E system memory. Here "+" means combination of bits or bitwise "OR" instead of actual addition operation. For example, (RED_ID + 24bit data) = (REG_ID<<24)|(24bit data). So do others below.

If a command is going through PCI, use a function SEND_REG to replace each WRITE_REG below. SEND_REG has two parameters: register address and register content. Register address can be (REG_ID<<2)+register base address or any address in graphics register address range. Register content is the same as parameter in WRITE_REG.
SEND_REG(register address, register content).

If a command needs a sequence of data, for instance, load instruction, primitive data, etc. The transmission falls into two steps:

```
WRITE_REG(REG_ID + length + flag);    // 24bit data = length + flag
WRITE_REG(32bit data);
WRITE_REG(32bit data);
      :
```

Commands in GP and RE are all 128bit aligned. Driver needs to send enough data + some garbage for alignment. HW needs to throw garbage away. For example, driver load 7 DWORDs but in command buffer 8 DWORDs will be put. HW needs to check if writing data is over than the count or data array size. If yes, it will be automatically thrown away.

A 128bit command is defined by 4 DWORDs: D0 (lowest), D1, D2 and D3 (Highest).

Each command / register has the following definition form:
- Overall function description
- Register address (access attribute)command name
- Bit range function description, // comments
- General comments.

2.2. Command List

A command list consists of a series of commands.

```
COMMAND ::=
{
    BITS_8      CMD_ID;
    BITS_24     Header;
    [BITS_32    data;]*         //optionally repeated;
}

Command List::=
{
    [BEGIN_2D/3D]
    [Global Environment Setting]
    [Primitives]*
    [EOP]
    [Flush]
    [Flip Command]
}

Global Environment Setting Command::=
{
    [Render States]*
    [Pixel Processing Instructions & Constants]*
```

```
    [Texture Sampler States & Instructions]*
    [Vertex Processing Instructions]*
}

Primitives::=
{
    Primitive type and count
    [indices of vertices]*          // for indexed VB.
    [stream of vertex data]*        // for immediate primitives.
}
```

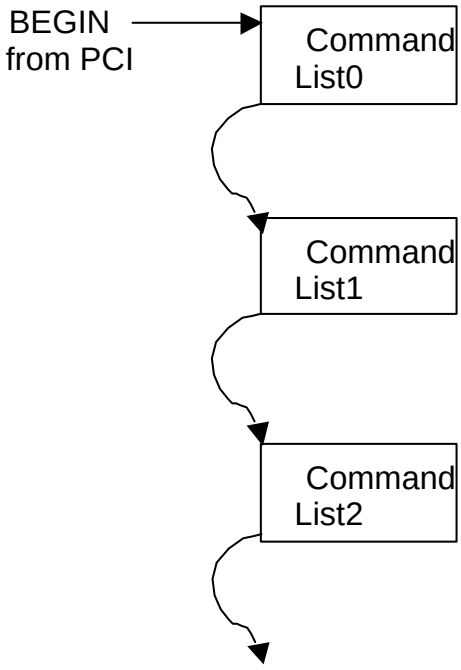
NULL primitive is End-Of-Primitive (EOP): end flag of a batch of primitives before updating new environment.

See Master Engine Commands for details of BEGIN.

2.3. Command Buffer

A command buffer consists of a series of command lists.

When all embedded command lists are fetched, switch to PCI mode.



3. Master Engine Commands

Command ID for Master Engine.

- Pipeline Control Commands, 0x000 to 0x00C.
- Command Buffer Fetch, 0x010 to 0x070.
- Display Flip Address, 0x080 to 0x098.
- 2D Command Overflow Buffer, 0x0C0 to 0x0C4.
- CPU Read/Write Non-linear Buffer, 0x0D0

Programming Notes & Rules:

1. All the command from PCI should have strict address:
   (1) 2D command: keep 2D command address;
   (2) Master command: master command is 32bit aligned, each command has a in-dependent address = 2800 + cmd_id << 2;
   e.g. 2804_01000004 (reset 2D engine)
   (3) Flip command: flip command is 32bit aligned, each command has a address as master command;
   (4) 3D command: 3D command is 128bit aligned, each four command has a group of address range. The address of 1st Dword is 2800 + cmd_id << 2, the followed three Dword with automatically adding 4 on previous address.
   e.g. 2920_48000000 (1st Dword)
   2924_00000000 (2nd Dword)
   2928_00000000 (3rd Dword)
   292c_00000000 (4th Dword)
   (5) Load command: it may have many Dwords in one load. The load length must be divided exactly by four Dwords. And, each four Dwords has the same group of address as following:
   GP load: 2900_xxxxxxxx
   2904_xxxxxxxx
   2908_xxxxxxxx
   290c_xxxxxxxx
   RE load: 2a00_xxxxxxxx
   2a04_xxxxxxxx
   2a08_xxxxxxxx
   2a0c_xxxxxxxx

2. About BEGIN command:
   The 1st BEGIN, i.e. the BEGIN command from PCI must be VALID, as well as no HOLD flag and STOP flag.

3.1. Pipeline Control Commands

3.1.1. Flush

Flush specific engine(s). S/W Driver can set flush command to flush a part of graphics engine including rendering and geometry processor. For example, a flush command to flush depth engine is a must before S/W Driver read correct depth engine status. Flush command may be used

occasionally to adjust synchronization timing to get correct images. XP10 provides flexibility to flush a combination of several engines.

Register
0x000 (W): 0x00 FLUSH_ENGINE // default setting is all 0.
D0 [23:22] DELAY_CYCLES, Enumerate.
{
0: 32_clocks, // default
1: 64_clocks,
2: 96_clocks,
3: 128_clocks,
} DelayClockCount
// The number of delay cycles to be applied in FLUSH. Unit = 32 clocks.
// Receive idle then wait for the delay cycles.
// During waiting there is no busy signal happening.

D0 [21:21] Flush_Flip_Engine, Boolean.
D0 [20:20] Flush_2D_Engine, Boolean.
D0 [19:0] Flush_3D_Engine, Integer.
D0 [19:19] Flush_GE_MI, Boolean.
D0 [18:18] Flush_OUTPUT_ENGINE, Boolean.
D0 [17:17] Flush_Pixel_Blender, Boolean.
D0 [16:16] Flush_Pixel_Shader, Boolean.
D0 [15:15] Flush_Texture_Engine, Boolean.
D0 [14:14] Flush_Filter_Loop, Boolean.
D0 [13:13] Flush_LOD_Engine, Boolean.
D0 [12:12] Flush_Depth_Engine, Boolean.
D0 [11:11] Flush_Tiling_Engine, Boolean.
D0 [10:10] Flush_Bump_Shader_Evaluator, Boolean.
D0 [9:9] Flush_Rasterization_Engine, Boolean.
D0 [8:8] Flush_Setup_Engine, Boolean.
D0 [7:7] Flush_Overflow_Buffer, Boolean.
D0 [6:6] Flush_Viewport_Engine, Boolean.
D0 [5:5] Flush_Clip_Cull_3DClip, Boolean.
D0 [4:4] Flush_Primitive_Assembly, Boolean.
D0 [3:3] Flush_Vertex_Shader, Boolean.
D0 [2:2] Flush_PUT_Processor, Boolean.
D0 [1:1] Flush_Vertex_Fetcher, Boolean.
D0 [0:0] Flush_GP_Decoder, Boolean.

// Different engines can be flushed at the same time. For instance, command 0x00600 will flush Setup Engine and Rasterizer together.

Driver must add a Flush_3D_Engine when switch from 3D command list to 2D command list.

3.1.2. Reset

Some engine block or status or register can be reset, for debugging and Conditional-End (will be described later). For example, a RESET to reset rendering status should be done before a primitive of bounding box is drawn and tested, its test status will be read by a conditional end command later. Before a RESET, a FLUSH sometimes is necessary. For exception handling like

deadlock, a hardwired RESET decoded outsides graphics engine must be used. The command is purposively used by S/W Driver.

Register
0x004 (W): 0x01 RESET // default setting is all 0.
D0 [10:10] RESET_Status2, Boolean. // 1: reset the status#2, 0x1FC;
D0 [9:9] RESET_Status1, Boolean. // 1: reset the status#1, 0x1F8;
D0 [8:8] RESET_Status0, Boolean. // 1: reset the status#0, 0x004;
D0 [4:4] RESET_3DENG, Boolean. // 1: reset the whole 3D engine;
D0 [2:2] RESET_2DENG, Boolean. // 1: reset the 2D engine;

3.1.3. Timer Threshold

To set a threshold to tell how long command engine waits then rendering engine / geometry processor will be reset automatically.

Register
0x008 (W): 0x02 MTIME
D0 [23] Enable_Timer, Boolean.
{
0: Disable timer check. // default.
1: Enable.
}
D0 [22:0] Timer_threshold, Integer. // (unit: 1024 clocks). Default = 0x7fffff

If a command could not be executed because the engine is busy, A **watchdog** automatically checks if (time lapse >= TimingThreshold). If yes, reset rendering engine and discard all of coming commands.

For those machines not using Little-End expression, CommandSwap register setting is provided.

Register
0x00C (R/W): 0x03 CommandSwap // Only go through PCI.
D0 [1:0] CommandSwapMode, Enumerate.
{
0: no_swap: // unchanged. default.
1: half_swap: // bit[7:0] <--> bit[15:08] and bit[31:24] <--> bit[23:16].
2: word_swap: // bit[7:0] <--> bit[23:16] and bit[15:08] <--> bit[31:24].
3: full_swap: // bit[7:0] <--> bit[31:24] and bit[15:08] <--> bit[23:16].
} SurfaceSwapMode

3.2. Command Buffer Fetch

3.2.1. Command List Linking Mode

A special command is used.

Register
0x010 (W): 0x04 Auto_Link_Setting // Only go through PCI.

D0 [11] Clear_Timer_Interrupt, Boolean. // Timer Reset.
D0 [10:10] Clear_Interrupt_3, Boolean. // Stop & Store.
D0 [9:9] Clear_Interrupt_2, Boolean. // GE Idle.
D0 [8:8] Clear_Interrupt_0, Boolean. // Invalid BEGIN.
D0 [4:4] Clear_Counters, Boolean. // for method#1 defined in D[1].
D0 [2:2] GE_Interrupt_Enable, Boolean. // (GE_INT_EN)
// S/W pre-arrange an INT routine and enable this bit.
// When command engine finishes all command lists and flushed whole GE,
// Then send out a signal = GE_DONE = GE_FLUSH_ALL_DONE & GE_INT_EN to
// AGP/REG block and combine with AGDONE2 (DONE2 = AGDONE2 | GE_DONE).
// When DONE2 == TRUE, automatically trigger the INT routine to tell S/W that whole GE is idle.
// and S/W can send next jobs to GE. Thus S/W does not need to port GE idle.

D0 [1:1] PCI_Trigger_Mode, Boolean. // Driver sends trigger register through PCI.
D0 [0:0] Invalid_List_Auto_Interrupt, Boolean. // HW sends an interrupt to SW when it gets
// an invalid BEGIN.

Register
0x010 (R): 0x04 Auto_Link_Status // Only go through PCI.
D0 [11] Active_Timer_Interrupt, Boolean. // Timer Reset.
D0 [10] Active_Interrupt_3, Boolean. // Stop/Store.
D0 [9] Active_Interrupt_2, Boolean. // GE Idle.
D0 [8] Active_Interrupt_0, Boolean. // Invalid BEGIN
D0 [2] GE_Interrupt_Enabled, Boolean. // (GE_INT_EN)
D0 [1] PCI_Triggerred_Mode, Boolean. // Driver sends trigger register through PCI.
D0 [0] Invalid_List_Auto_Interrupted_mode, Boolean.
// HW sends an interrupt to SW when it gets an invalid BEGIN.
// The lowest three bits can not be enabled at the same time.
// Only one bit can be set at a time.
// This register will cause Master Engine be flushed automatically before it is activated.

Register
0x014 (R/W): 0x05 PCI_Trigger_Register
D0 [23:0] SW_Trigger_Register, Integer.
// no pre-defined content. When D[1] of 0x010 == 1, it acts a trigger register.
// otherwise, it is a scratch register.

PCI Interrupt: Driver sends trigger register through PCI. There are following three cases of PCI interrupt:

1. HW runs faster than Driver. When Driver receives the request from HW but still not prepare the next job for HW, Driver just put a fake head (BEGIN) and set **Invalid flag**. When the next job is ready, Driver will replace the fake head with right head, and set it **Valid**. At the same time, SW sends the PCI trigger register. If HW receives valid begin, it goes on; if HW receives invalid begin, it continue to run only when trigger counter >= valid counter.
2. HW sends interrupt to SW when it gets an invalid BEGIN. SW gets which command list is not ready from corresponding register setting, and resend right command list.
3. HW is Idle, so sends Interruption to SW. If SW has some resources ready, send down PCI command list and let HW run new items.

The efficiency is 1 >2 >3.

3.2.2. BEGIN

To start to fetch a command list and link two command lists together. This command will tell Master engine to fetch a command list from PCI-E with end option and length of the command list. Several command lists may be linked by embedded Begin commands. The triggering Begin command from PCI is needed only once in this case.

0x020 (W/R): 0x08 BEGIN_MASTER
0x030 (W): 0x0C BEGIN_2D // HW still puts the BEGIN flag to a readable register 0x020.
0x040 (W): 0x10 BEGIN_3D // HW still puts the BEGIN flag to a readable register 0x020.
0x050 (W): 0x14 BEGIN_Flip // HW still puts the BEGIN flag to a readable register 0x020.

Instruction Begin
D0 [23] Drop_Primitive_Enable, Boolean.
D0 [22] Stop_Store_Current_Pointer, Boolean.
D0 [21] Hold, Boolean.
D0 [20] Valid, Boolean.
D0 [19:0] BEGIN_Identification, Integer.
// set by driver. HW puts the value to a readable register 0x020. Driver will read it back to check
// which BEGIN is interrupted.

D1 [31] Link_Enable, Boolean.

D1 [21:0] Command_List_Length, Integer.
//Unsigned integer length (unit = 32bit), including commands defined by endOption.
// Length == 0 means fetch nothing.

D2 [27:0] Command_List_Start_Address, Integer. // 28bit start address (128bit alignment).
D3 [31:00] Reserved.

// 0x020 (W/R) BEGIN_MASTER
// 0x030 (W) BEGIN_2D. // HW still puts the BEGIN flag to a readable register 0x020.
// 0x040 (W) BEGIN_3D. // HW still puts the BEGIN flag to a readable register 0x020.
// 0x050 (W) BEGIN_Flip. // HW still puts the BEGIN flag to a readable register 0x020.

3.3. Display Flip Address

Register
0x080 (W): 0x20 Surface0_Control_mode
D0 [0] Surface0_HW_SWAP_Enable, Boolean.
D0 [1] Surface0_WAIT_Display, Boolean.
D0 [2] Surface0_FrameBuffer_Over_Draw, Boolean.
D0 [3] Surface0_Check_Display, Boolean.
D0 [23:4] Reserved.

Register
0x084 (W): 0x21 Surface0_Base
D0 [22:0] Surface0_Base_Address, Integer. // (unit: 4byte)
D0 [23] Surface0_Flip_End_Of_Frame, Boolean.

Register
0x088 (W): 0x22 Surface1_Control_mode
D0 [0] Surface1_HW_SWAP_Enable, Boolean.
D0 [1] Surface1_WAIT_Display, Boolean.
D0 [2] Surface1_FrameBuffer_Over_Draw, Boolean.
D0 [3] Surface1_Check_Display, Boolean.
D0 [23:4] Reserved.

Register
0x08C (W): 0x23 Surface1_Base
D0 [22:0] Surface1_Base_Address, Integer. // (unit: 4byte)
D0 [23] Surface1_Flip_End_Of_Frame, Boolean.

Register
0x090 (W): 0x24 Surface2_Control_mode
D0 [0] Surface2_HW_SWAP_Enable, Boolean.
D0 [1] Surface2_WAIT_Display, Boolean.
D0 [2] Surface2_FrameBuffer_Over_Draw, Boolean.
D0 [3] Surface2_Check_Display, Boolean.
D0 [23:4] Reserved.

Register
0x094 (W): 0x25 Surface2_Base
D0 [22:0] Surface2_Base_Address, Integer. // (unit: 4byte)
D0 [23] Surface2_Flip_End_Of_Frame, Boolean.

Register
0x098 (W): 0x26 Extended_Base_Address
D0 [3:0] Surface0_High_Base_Address, Integer.
D0 [7] Surface0_High_Base_Enable, Boolean.
D0 [11:8] Surface1_High_Base_Address, Integer.
D0 [15] Surface1_High_Base_Enable, Boolean.
D0 [19:16] Surface2_High_Base_Address, Integer.
D0 [23] Surface2_High_Base_Enable, Boolean.

Note: If a buffer flip address is in a command list, each command list only can have one flip address. Each flip address should be put just in the end of a list and just before a BEGIN indicated by an EndOption.

- For two consecutive begin commands, if there's flipping command in the first one, please insert one flush command between two begin commands.
- Please make sure the address of "0Cxxxxxx" command and its following base address command are the same.

Programming Sequence

- First Batch:
  Set surface control mode register, (Please note set this register will reset Flipping Engine):
  HW_SWAP_Enable = 1,
  Control code in WAIT_Display, FrameBuffer_Over_Draw, Check_Display;
  Set surface base address register:
  Flip_End_Of_Frame = 0,
  Base_Address = Next drawing frame buffer base address;

……
Drawing commands on the previous programmed FBBAS(Frame Buffer Base Address);

- Next Batch:
  Set surface base address register:
  Flip_End_Of_Frame = 1,
  FBBAS = Next drawing frame buffer base address;
  ……
  Drawing commands on the previous programmed FBBAS;
- ……
- End Batch:
  Set surface base address register:
  Flip_End_Of_Frame = 1,
  FBBAS = Not valid;
  Wait for Fliiping engine idle;
  // Read control mode register, check bit[18], [12],[8] all equal one
  Set surface control mode register:
  HW_SWAP_Enable = 0;

3.4. 2D Command Overflow Buffer

S/W should force the address of the following data in 2D Begin command, i.e. base address & port, equal to the address of 2D Begin command (0x2a18). Thus, HW can make the judgment of 2D special registers simple.

Register
0x0C0 (W): 0x30 2D_Overflow_Buffer_Base
D0 [23:0] Overflow_Buffer_Base_Address, Integer. //(32bytes alignment)

Register
0x0C4 (W): 0x31 2D_Overflow_Buffer_Offset
D0 [23] Overflow_Enable, Boolean.
1: Enable Overflow Buffer.
0: Disable Overflow Buffer.
D0 [22] Debug_Mode, Boolean.
1: Enable commands always go through Overflow buffer.
0: Disable debug mode.
D0 [21:0] Overflow_Buffer_Offset, Integer. // unit = 256bits.

3.5. CPU Read/Write Non-linear buffer

Register
0x0D0 (R/W): 0x34 MultiSurfaces_Load_Command
// only this command and first surface is readable.

// CPU read / write table
D0 [23] Surface_Table_Enabled, Boolean.
D0 [21:16] Total_Entry, Integer. // first one is valid. 0~47.

```
D0 [15]         Element_Size, Enumerate.
      {
            0x0: 16BPP.
            0x1: 32BPP.
      } ElementColorSize
D0 [14:12]      MultiSurfaces_Banded_Tiled_Mode, Enumerate.
      {
            0:    Linear.
            1:    Tiled.            // 4x4 pixel tiled.
            2:    Band64.           // 64*16 pixel banded.
            3:    Band32_Tiled.     // 64*64 pixel banded + 2x2 32 pixel block+ 4*4 pixel tiled.
            4:    Band64_InterleavedZ.
            5:    BandedCompresson.      //Band32_Tiled_InterleavedZ.
            6:    Band32_Zbuffer_Only.
            7:    Two_Line_Linear.  // color compression
      } MultiSurfacesBandedTiledMode
D0 [11:10]      Total_Elements, Integer.
// The number of Multiple Render Targets or Multiple Elements Texture target.
// 0: means one element; 3 means 4 elements.
D0 [9:0]        Buffer_Width, Integer.   // divided by 4.       0x000 means 4096.

D1 [26]         Enable_Tiled, Boolean. // valid only for Band_Tile_mode = 1, 3, 5
D1 [25:24]      Buffer_Swap_Mode, Enumerate.
      {
            0:    no_swap:      // unchanged.    default.
            1:    half_swap:    // bit[7:0] ←→bit[15:08] and bit[31:24] ←→ bit[23:16].
            2:    word_swap:    // bit[7:0] ←→bit[23:16] and bit[15:08] ←→ bit[31:24].
            3:    full_swap:    // bit[7:0] ←→bit[31:24] and bit[15:08] ←→ bit[23:16].
      } BufferSwapMode
D1 [23:0]       ONE_OVER_WIDTH,   IFF8.16.
// default = 0.0, for any buffer width from 1~4K.
D2 [31:16]      Buffer_Base_Address, Integer.
// 8Kbytes alignment. Up to 512MB. Default = 0x000000.
D2 [15:0]       Buffer_End_Address,    Integer.   // default = 0.0, 8Kbytes aligned
```

If buffer is linear, all attributes except for address range and swap mode are meaningless. In this case, the address range may be for a group of surfaces.

Other surfaces are followed in the order of address increasing, sorted by Driver, as data of the command.

This command with all surfaces can be put in BEGIN_MA command list or through PCI. Master engine decodes it and pass them to CPU read/write block.

If Driver locks Z buffer, SW Driver first does conversion blit to color buffer format, then allow CPU read/write. If Driver color buffer, Driver needs to finish color buffer repairing (completion) first, then allow CPU access. If buffer is special format such as YUV etc, when it is locked, issue a conversion through texture engine to convert it to ARGB mode.

Note: Every entry has three DWORDs, all entries will be packed tightly without 128bit aligned, i.e., four entries take 3x128bits. If table is invalid, still send three DWORDs, but no write to CPU read/write block. The same rule for both PCI and PCI-E command list.

3.6. Engine Status

Statuses are used only for debugging. It is reset by RESET. Some addresses are overridden with write only registers / commands. Every register is reset by RESET or an outer reset signal decoded outside graphics engine.

3.6.1. Engine Busy Status

Register
0x000 (R): 0x00 GRAPHICS_ENGINE_BUSY_STATUS: // override register with FLUSH.
D0 [31] Whole_Graphics_Engine_Busy. // including 2D, 3D, Master and Flip engine
D0 [30] Whole_2D_Engine_Busy.
D0 [29] Whole_3D_Engine_Busy.
D0 [28] Master_Engine_Busy.
D0 [27] Flip_Engine_Busy.
D0 [26:21] Reserved.
D0 [20] OEII_PCI-E_Busy.
D0 [19] Output_EngineII_Busy.
D0 [18] Output_EngineI_Busy.
D0 [17] Pixel_Blender_Busy.
D0 [16] Pixel_Shader_Busy.
D0 [15] Texture_Engine_Busy.
D0 [14] Filter_Loop_Busy.
D0 [13] LOD_Engine_Busy.
D0 [12] Depth_Engine_Busy.
D0 [11] Tiling_Engine_Busy.
D0 [10] Bump_Evaluator_Busy.
D0 [9] Rasterization_Engine_Busy.
D0 [8] Setup_Engine_Busy.
D0 [7] Overflow_Buffer_Busy.
D0 [6] Viewport_Engine_Busy.
D0 [5] Clip_Cull_3Dclipper_Busy.
D0 [4] Primitive_Assembly_Busy.
D0 [3] Vertex_Shader_Busy.
D0 [2] PUT_Processor_Busy.
D0 [1] Vertex_Fetch_Busy.
D0 [0] GP_Decoder_Busy.

4. Geometry Processor Commands

Command ID for Geometry Processor.

- LOAD_GP, 0x100.
- Primitive Type, 0x120.
- Geometry Environment, 0x130 to 0x18C
- Viewport, 0x190 to 0x1AC

4.1. Load Command

To load fetch, put instruction and user clip planes. LOAD command can load a part of data array with offset and can multiple loads to a data array.

Register
0x100 (W) : 0x40 LOAD_GP

```
// Load stream data following load command is sent to an indicated RAM/array.
D0 [23]         IEEE_To_IFF, Boolean. // Can be used for VS constants & user planes.
{
      0: disable. // Load data as AS_IS.
      1: enable. // On the way of loading to Ram, Convert IEEE float to IFF.
}
D0 [22:20]      GP_Load_code, Enumerate.
{
      0x0:      NULL                        // default
      0x1:      Load_Fetch_Instructions     // to FETCH Array. See chapter 4. Start from D2
      0x2:      Load_Put_Instructions       // to PUT Array. See chapter 4. Start from D1
      0x3:      Reserved
      0x4:      Reserved
      0x5:      Load_User_Clip_Planes       // Each plane has 4 floats, up to 8 planes.
} GPLoadCodes
D0 [18:10]      GPLoad_Start_Offset, Integer.    // instruction or 4D vector unit.
                                                 // Unit: 32bit for PUT, 64bit for FETCH, 128bit for others.
D0 [8:0]        GPLoad_Length, Integer.    // (unit: 128bit). Not include this 128bit.
          // If put/fetch and length is zero, only data within the LOAD instruction dwords is loaded.
D1 [31:0]       Put0.              // if loading PUT, grab first instruction from here,
                                   // set to zero for all other load cases.
D2 [31:0]     Put1_Fetch0_D0. // 2nd put instruction or first DWORD of first fetch instruction
D3 [31:0]     Put2_Fetch0_D1. // 3rd put instruction or second DWORD of first fetch instruction
```

The Load Length D0 [8:0] defines how many data this load command is followed by, **unit: 128bit**, not including the 128bit of this load command.

Load GP has different instruction UNIT: **32bit for PUT, 64bit for FETCH, and 128bit for others**. If Load_Put_Instructions, PUT0, PUT1, PUT2 is placed at D1, D2, D3 of this 128bit; if Load_fetch_Instructions, one fetch instruction can be placed at D2, D3 of this 128bit since fetch instruction unit is 64bit.

4.1.1. User Clip Planes Loading

Every user clip plane has 4 float coefficients to become a 4D vector. Its data order is ABCD = XYZW from low to high, that is, A is lowest dword and D is highest dword in 128 bit 4D vector. Here every float is XP10 internal float format. XP10 supports maximum 8 user clipping planes. First user plane is in first 4D vector and eighth plane in eighth 4D vector. Use LOAD command to load these coefficients. The sequence is plane0.A, plane0.B,…, planei.A, planei.B,…, plane7.A, plane7.B, plane7.C and plane7.D. How many coefficients are necessary is dependent on how many user planes enabled. Defined in:

0x130 D1 [6:3] Number_Of_User_Planes, Integer. // Zero means no user planes defined.

4.2. Primitive & Vertex Type

Primitive type & count
To define a primitive type and the number of primitives contained.

```
Register
0x120 (R/W) : 0x48      Primitive_Command
D0 [23:21]      Shading_Mode, Enumerate.
{
        0x0             Gouraud_Shading,    // nothing changes
        0x1             Flat_Shading_V0,    // copy color of v0 to v1 and v2
        0x2             Flat_Shading_V1,    // copy color of v1 to v0 and v2
        0x3             Flat_Shading_V2,    // copy color 0f v2 to v0 and v1
        0x4             Flat_Shading_Odd_V1_Even_V2, //for odd triangle copy v1 to v0 and v2;
                                                     //for even triangle copy v2 to v0 and v1.
        0x5             Flat_Shading_Odd_V2_Even_V1, //for odd triangle copy v2 to v0 and v1;
                                                     //for even triangle copy v1 to v0 and v2.
} FlatShadingMode

D0 [20:16]      Primitive_Type, Enumerate. // The defined values aren't ordered in order to be
                                           // conformant with D3D constant definition. For detailed
                                           // explanation of these constants, please refer to
                                           // Microsoft Direct 3D document.
{
        0x0:    NULL,                      // as a last one.
        0x01:   IndexedPointList,          // Indexed point list
        0x02:   IndexedLineList,
        0x03:   IndexedLineStrip,
        0x04:   IndexedTriangleList,
        0x05:   IndexedTriangleStrip,
        0x06:   IndexedTriangleFan,
        0x07:   IndexedLineloop,
        0x08:   IndexedQuadList,
        0x09:   IndexedRectangleList,      // list of rectangles.

        0x10:   ImmediateEOP,
        0x11:   PointListImmediateMode,        // Immediate mode points.
        0x12:   LineListImmediateMode,         // Immediate mode list
        0x13:   LineStripImmediateMode,
        0x14:   TriangleListImmediateMode,
```

```
        0x15:   TrianglestripImmediateMode,
        0x16:   TrianglefanImmediateMode,
        0x17:   LineloopImmediateMode,
        0x18:   QuadListImmediateMode,
        0x19:   RectangleImmediateMode,
        0x1a:   XP_PrimitiveImmediateMode      // partial/full triangles with flags
} M2PrimitiveTypes

// No limit such as two short2 for rectangle, it can be any vertex like other primitives.
// primitive count = edge flag (3bit) [6:4] + triangle/line (1bit) [3:3] +
// full/partial (1bit) [2:2] + replacement rule (2bit) [1:0].

D0 [6:4]        Edge_Flag, Integer.           //for XP style only
D0 [3:3]        Triangle_Or_Line, Integer.    //for XP style only
D0 [2:2]        Full_Or_Partial, Integer.     //for XP style only
D0 [1:0]        Replacement_Rule, Integer.    //for XP style only

D0 [15:0]       Primitive_Count, Integer.     // initial value = 0x0000, means no data

D1 [31:10]      Total_Vertex_Data, Integer. // 128bit unit, easy for IMM decoder.
D1 [9:8]        Vertex_Index_Type, Enumerate.
{
        0:    Index16                       // two indices are packed to a 32bit. default.
        1:    Index24_EdgeFlag3             // bit24 = edge01, bit25 = edge12, bit26 = edge20.
        3:    Index24                       // default.
} VertexIndexType

D1 [5:0]        Vertex_Length, Integer.       // per input vertex before vertex cache. Unit: 64bits
                                              // default = 1. Zero means one 64bit data.
        // for indexed primitives, Vertex_Length = Sum((FetchSizeInFetch[I]+1+3)>>2)*2 – 1;
        // for immediate primitive, Vertex_Length = (number of 64 bits per vertex) –1;

D2 [31:30]      GP_Mode, Integer.             // Reserved for late.
D2 [23:0]       Index_Range_Low, Integer.
D3 [31]         Visibility_Bit_Stream_Followed, Boolean.  // 128bit, 1bit/box.
D3 [29:28]      Box_Size, Integer.
                // Real box size = (16<<box_size) triangles = (48<<box_size) indices.
                // Only for indexed triangle list.
D3 [23:0]       Index_Range_High, Integer.
```

All information in command needs to pass down to Vertex Processor without flush pipeline.
Note: Primitive count is the number of basic primitives defined by this command. It is not a vertex count. How many vertices should be followed depends on primitive type and the count. See the detail of primitive parser in architecture specification.
Vertex cache is configurable. Set by S/W Driver. Immediate mode only uses D1[17:6] PUT array offset and vertex length.

Two types of EOPs

For easy HW control switch, HW needs two types of EOPs (= end of primitive):
If a batch ends in index primitive mode, you need send Indexed EOP: Primitive type = 0x0;
If a batch ends in immediate mode, you need send Immediate EOP: Primitive type = 0x10;
When switch from IMM to Index mode or vice verse, it requires an EOP & a FLUSH of GP decoder and vertex fetch.

Programming Nodes:

- Total_Vertex_Data should be identical with the actual vertex data length in IMM.
  Pseudo code:

```
{
    PrimitiveType = GetValue("Primitive_Type");
    if(PrimitiveType <= 0x10) //index primitive
        exit(0);
    TotalVertData = GetValue("Total_Vertex_Data");
    PrimitiveCount = GetValue("Primitive_Count");
    VertexCount = CalculateNumberOfVerticesPerPrimitive(PrimitiveType, PrimitiveCount);
    VertexStrideLength = GetValue("Vertex_Length");
    actual_length = VertexCount * (VertexStrideLength + 1);
    actual_length = (actual_length + 1) & ~(1);
    assert(TotalVertData * 2 == actual_length);
}
```

4.3. Geometry Environment

Register
0x130 (R/W) : 0x4C GEOMETRY_ENVIRONMENT

D0 [23] Viewport_XForm_Disable, ReversedBoolean.
// Set Viewport scale (0x190) .
// D1[23:0] X_Scale_Factor = 1.0
// D2[23:0] Y_Scale_Factor = 1.0.
// D3[23:0] Z_Scale_Factor = 1.0.
// 0x1A0(W) : 0x68 Viewport_Offset
// D1[31:0] X_Offset = 0.0
// D2[31:0] Y_Offset = 0.0
// D3[31:0] Z_Offset = 0.0.

D0 [22] Frustum_GuardBand_ClipCode_Disable, ReversedBoolean.
// CC ignore computed clipcode, and set all accept.

D0 [20] VertexShaderInRom, Boolean.

D0 [19] Vertex_Cache_Clear, Boolean. // HW will only check this bit to do clear.
// No implicit clearing when some other setting changes.
// set it on when: vertex buffer changes; switch from IMM to index vertex mode;
// switch from bypass to non-bypass.

D0 [18] Vertex_Cache_Bypass, ReversedBoolean.

D0 [17:12] PUT_Instruction_Offset, Integer. // RAM (0~47) and ROM (48~55).
// default = 0. See chapter 4.

D0 [11:8] Fetch_Instruction_Offset, Integer.
D0 [7:0] Vertex_Shader_InstructionPair_Offset, Integer. // which instruction begins the VS.
D1 [31:16] VS_Boolean_Constants, Integer. // booleans for flow control in the VS

D1 [15:12] Vertex_Size_In_VS, Integer. // in Vertex Shader output RAM, minimum size = 5.
// Output Vertex RAM block will send data (N) and clipcode to Primitive Assembly based on the number.

```
D1 [11]         Vertex_Cache_Configuration_In_VS, Enumerate. // QW = 128bit.
      {
            0:    Vertex32 // 32entry*8QW/vertex. default
            1:    Vertex16 // 16entry*16QW/vertex.
      } CacheEntrySizeInVS

D1 [10]         Rectangle_Mode, Enumerate.
{
            0:    Horizontal_Mode.  // default
            1:    Vertical_Mode.          // rotate rendering and vertical gradient fill
} RectangleMode;

D1 [9]          Disable_PA_Fast_Rejection, ReversedBoolean.
D1 [8]          Point_Mode, Enumerate.
// valid when (primitive type == POINT_LIST || primitive type == TRIANGLE_ POINT).
{
            0:    Simple_Point_List.
            1:    MS_Point_Sprite_List. // It will be used by VS, PA and CC.
                  // VS will pack first pair of UVs to second 3x128bits if this bit is not set.
                  // PA will unpack it also if it is not set.
                  // CC decides if it enters point sprite mode.
} PointSpriteType

D1 [7:7]        Use_Special_Texture_Coordinate, Enumerate.       // only for point sprite.
{
            0:    Use_Texture_Coordinate,         // defined in vertex.
            1:    Use_Special_Constant_Coordinates.
} PointSpriteTexgenMode

D1 [6:3]        Number_Of_User_Planes, Integer. // Zero means no user planes defined.
D1 [2:2]        User_Clip_Code_Valid, Integer.  // user clip code is computed by VS.
                                                // Invalid for sprite/Qsplat.
D1 [1:1]        W_Clipping_Enable, Boolean.     // valid only when 3D clipping is enabled.
{
      0: disable; // default.
      1: enable;
}
D1 [0:0]        3D_Clipping_Enable, Boolean.
{
      0: disable; // default.
      1: enable;
}

D2 [31:24]      W_Exponent_For_Normalization, Integer.  // signed 2's integer NormWExp.
      // default value = 0. For rhW buffering, use it to do normalization for every source rhW.

D2 [22]         Two_Sided_Lighting_Enable, Boolean.
D2 [21]         Front_Face_Type, Enumerate.
{
      0:    Counter_Clockwise;
      1:    Clockwise;
} FrontFaceType

D2 [19]         Fog_Valid_InGP, Boolean.         // Pass down fog to SE only when it is valid.
D2 [18]         Specular_Valid_InGP, Boolean.  // Pass down to SE only when it is valid.
```

D2 [17:16] Diffuse_Color_Clamp_Mode, Enumerate.
{
0: No_Clamp,
1: Clamp_0_1, // to [0.0, 1.0].
2: Clamp_1_1, // to [-1.0, 1.0].
} ColorClampMode

D2 [15:14] Specular_Color_Clamp_Mode, Enumerate. ColorClampMode.
D2 [13:12] Fog_Clamp_Mode, Enumerate. ColorClampMode.
D2 [11:11] Perspective_Correction, Boolean.
D2 [10:10] W_Buffer_Enable, Boolean.

D2 [9:8] Culling_Mode, Enumerate.
{
0x1: No_Culling, // default
0x2: Clockwise_Culling,
0x3: Counter_Clockwise_Culling,
} CullingMode

D2 [7:6] Back_Fill_Mode, Enumerate.
{
0x1: Point, // for triangle.
0x2: Wireframe, // for triangle only.
0x3: Solid, // (default).
} FillMode
D2 [5:4] Front_Fill_Mode, Enumerate, FillMode.

D2 [3:0] GP_Total_Texture_Coordinate_Pairs, Integer.

D3 [31:0] Texture_Coordinate_Wrap_Enables, Integer.
// 1bit per dimension of texture coordinate 4 bits / texture pair. Bit0 for X dimension, ..., bit3 for W dimension.

Programming Notes:

- Has to reset vertex cache when change vertex setting: length, point sprite on/off, texture pairs, vertex buffer address in FETCH instruction etc. To reset vertex cache, set D0 [19] = 1.
- The UV pair number should be consistent with "GP_Total_Texture_Coordinate_Pairs". For example, if GP_Total_Texture_Coordinate_Pairs = = 2, the vertex data should include UV0~3.

  If it is Point_Mode = 0x1 (MS_Point_Sprite_List),
  Vertex_Size_In_VS = GP_Total_Texture_Coordinate_Pairs + 6;
  Else, it is Point_Mode = 0 (Simple_Point_List)
  Vertex_Size_In_VS = GP_Total_Texture_Coordinate_Pairs + 5;

  Relate register definition

0x130	D2 [3:0]	GP_Total_Texture_Coordinate_Pairs, Integer.
0x130	D1 [15:12]	Vertex_Size_In_VS, Integer.
0x130	D1 [8]	Point_Mode, Enumerate.
0x130	D0 [17:12]	PUT_Instruction_Offset, Integer.

  Pseudo code:

```
{
    vtxSizeInVS = GetValue("Vertex_Size_In_VS");
    gpUVPairs = GetValue("GP_Total_Texture_Coordinate_Pairs");
```

```
        pointSpriteMode = GetValue("Point_Mode");
        iGPTotalTexturePairs = vtxSizeInVS - (pointSpriteMode? 6: 5);
        assert(iGPTotalTexturePairs = = gpUVPairs);
    }
```

Cull Threshold

Register
0x140 (W) : 0x50 Cull_Threshold

D0 [23:16]	Clip_Space_Determinant_Threshold, Integer.	
	// to select (XY, XW, WY) for 3D clipper. Default is -64 = 0xC0.	
D0 [15:0]	Use_Special_Per_Texture_Coordinate, Integer.	
	// Each bit per 2d texture, 0 bit for texture 0. 2 bits for a texture pair.	
	// Each bit is a enumerate type: PointSpriteTexgenMode	
D1 [31:0]	W_Threshold, IFF1.8.23.	// default = 0.0.
D2 [31:0]	Zero_Threshold, IFF1.8.23.	// default = 0.0. in screen space.
D3 [31:0]	Normal_Threshold, IFF1.8.23.	// default = 0.0. in screen space.

Viewport Clip // before viewport scale & offset.

Register
0x150 (W) : 0x54 Viewport_Clip

D0 [23:0]	Viewport_Clip_Left,	IFF1.8.15.	// default = 0.
D1 [23:0]	Viewport_Clip_Right,	IFF1.8.15.	// default = 0.
D2 [23:0]	Viewport_Clip_Top,	IFF1.8.15.	// default = 0.
D3 [23:0]	Viewport_Clip_Bottom,	IFF1.8.15.	// default = 0.

Register
0x160 (W) : 0x58 Far_Near_Clip

D0 [23:0]	Viewport_Clip_Near,	IFF1.8.15.	// default = 0.
D1 [23:0]	Viewport_Clip_Far,	IFF1.8.15.	// default = 0.
D2 [23:0]	Guardband_Clip_Near,	IFF1.8.15.	// default = 0.
D3 [23:0]	Guardband_Clip_Far,	IFF1.8.15.	// default = 0.

Register
0x170 (W) : 0x5C Guard_Band_Clip

D0 [23:0]	Guardband_Clip_Left,	IFF1.8.15.	// default = 0.
D1 [23:0]	Guardband_Clip_Right,	IFF1.8.15.	// default = 0.
D2 [23:0]	Guardband_Clip_Top,	IFF1.8.15.	// default = 0.
D3 [23:0]	Guardband_Clip_Bottom,	IFF1.8.15.	// default = 0.

Register
0x180 (W) : 0x60 Guard_Band_Scale

D0 [15:0]	X_Scale_Guardband_Left,	IFF1.8.7.	// default = 1.0.
D1 [15:0]	X_Scale_Guardband_Right,	IFF1.8.7.	// default = 1.0.
D2 [31:16]	Y_Scale_Guardband_Top,	IFF1.8.7.	// default = 1.0.
D2 [15:0]	Y_Scale_Guardband_Bottom,	IFF1.8.7.	// default = 1.0.
D3 [31:16]	Z_Minimum_Scale_Guardband,	IFF1.8.7.	// default = 1.0.
D3 [15:0]	Z_Maximum_Scale_Guardband,	IFF1.8.7.	// default = 1.0.

// Can refer to section "Programming of Clipping" for more reference.

4.4. Viewport Engine Setting & Others

ScaleX, ScaleY and ScaleZ

XYZ coordinate scaling to screen size.

Register
0x190 (W) : 0x64 Viewport_Scale
D0[23:0] Reserved.
D1 [23:0] Viewport_X_Scale_Factor, IFF1.8.15. // default = 1.0
D2 [23:0] Viewport_Y_Scale_Factor, IFF1.8.15. // default = 1.0.
D3 [23:0] Viewport_Z_Scale_Factor, IFF1.8.15. // default = 1.0.

OffsetX, OffsetY & OffsetZ

XYZ coordinate adjustment in screen coordinate system.

Register
0x1A0 (W) : 0x68 Viewport_Offset
D0[23:0] Reserved.
D1 [31:0] Viewport_X_Offset, IFF1.8.23. // default = 0.
D2 [31:0] Viewport_Y_Offset, IFF1.8.23. // default = 0.
D3 [31:0] Viewport_Z_Offset, IFF1.8.23. // default = 0.

Window Clip / Scissor Test

Register
0x1B0 (W) : 0x6C GP_Window_Clip // for viewport
D0 [15:0] Window_Clip_Left, Unsigned Fix point, 12.4. // default = 0.
D1 [15:0] Window_Clip_Right, Unsigned Fix point, 12.4. // default = 0.
D2 [15:0] Window_Clip_Top, Unsigned Fix point, 12.4. // default = 0.
D3 [15:0] Window_Clip_Bottom, Unsigned Fix point, 12.4. // default = 0.

It is usually the same as one in Rendering engine. The two rectangle settings are made because GP and RE has different decoder stage. The second reason is to fix a bug in XP4: Point will be adjusted by drawing rule and then reject/accept in Raster engine, but hardware driver will not consider the drawing rule and drop more. So Rectangle in GP should consider drawing rule by adding more fraction part.

SW Scratch

Register
0x1F0 (W/R) : 0x7C SW_Scratch_Register
D0 [23:0] Scratch1, Integer. // No pre-defined content.
// The content is written and read by S/W driver.
D1 [23:0] Scratch2, Integer.

The register is not used for any HW block. It stays in command engine. It can be used for read back. Its usage is, for example, in the end of every command list, put this register with count or special flags. After a while, driver reads the register to check its content. Then driver knows hardware is running which command list/task etc.

5. Vertex Processing

Bits 20 to 22 of Register 0x40 (LOAD_GP) notify graphics engine that the following commands will be some arrays or tables. The following sections describe how to program these arrays as well as tables. In this way, we can save the number of registers and make the usage more flexible.

5.1. FETCH Instruction

The FETCH instructions tell Command Engine how to fetch vertex data from outside memory to vertex cache. FETCH instruction RAM size = 43bit*16entries.

Instruction Fetch

```
D0 [12:12]     Memory_Port_Flag, Enumerate, MemoryPortSelection.
{
     0:   PCI-E
     1:   FB
}
D0 [11:7]      Vertex_Size, Integer. // (in 4 bytes) of the source vertex data. Zero means one.
D0 [6:1]       Vertex_Stride, Integer.  // (in 4 bytes) index * stride = offset to vertex, (m-1)
D0 [0]         Last_Command, Boolean.
D1 [29:0]      Vertex_Buffer_Base_Address. // 32bit alignment
```

Note that both Vertex Size and Vertex Stride in fetch instruction are “Zero means one”.

One fetch instruction can fetch at most 32 DWORDS. Last fetch instruction need one “last fetch” flag.

Based on the vertex fetch, one miss from vertex cache may issue several read requests, each one is for one FETCH, to different memory (PCI-E/FB), and even different vertex data arrays if they are stored separately.

Requests are issued in the order of FETCH commands so that read back data have the same order for writing to the cache. It needs a 4 bit FIFO with enough depth to store request tag (one bit for PCI-E / FB0 requests, 3bit for alignment). One vertex data should be 32bit alignment in one vertex buffer area. IMM data for a vertex has to be aligned to 64bit by S/W Driver. For IMM, Vertex_Length (in VERTEX_INDEX) is used to count parameters too.

The following examples will demonstrate how to program FETCH Array instructions.

```
40100000 //    === Load Fetch Instruction ===
               //    [23:23] IEEE_TO_IFF : 0x0
               //    [22:20] GP_LOAD_CODE : 0x1, tell graphics engine it will load FETCH
               //    [18:10] GPLOAD_START_OFFSET : 0x0
               //    [ 8: 0] GPLOAD_LENGTH : 0x0, it will be followed by 0 DWORDs.
```

```
00000000
        //  [31: 0] PUT0 : 0x0
00001597
        //  [31: 0] PUT1_FETCH0_D0 : 0x1597
000f6800
        //  [31: 0] PUT2_FETCH0_D1 : 0xf6800
        //  Stream Stride: 12 DWORD, Stream Size: 12 DWORD, Last
        //  Port: FB, Address: 0003da00 (128bit alignment), Offset: 0
```

Programming Notes:

If multi-fetch, some fetch instruction only need 1 or 2 Dwords, in this case, the high 64bit of one 128bit read-back data is useless. But the Vertex_length set in 0x48 should count these useless Dwords.

e.g. there is one FETCH with 3 multi-instructions:

1st instruction need fetch 6 Dwords (2 * 128bit, H64 of 2nd data is useless)
2nd instruction need fetch 2 Dwords (1* 128bit, H64 of it is useless)
3rd instruction need fetch 2 Dwords (1 * 128bit, H64 of it is useless)

Vertex length should count all the useless H64 in. So the vertex length is:

4 + 2 + 2 - 1 = 7

5.2. PUT Instruction

The PUT instructions tell Graphics Engine how to put vertex data from vertex cache to Input Vertex Array of Vertex Shader. Each command will put 64 bits data. PUT instruction RAM size = 30bit*48entries.

```
Instruction Put     // one 32 bit in command stream; one PUT per 64bit data.
D0 [31:30]      SwapModeHigh, Enumerate, //do swap before format conversion.
      {
            0:    no_swap:      // unchanged.   default.
            1:    half_swap:    // bit[7:0]      bit[15:08] and bit[31:24]      bit[23:16].
            2:    word_swap:    // bit[7:0]      bit[23:16] and bit[15:08]      bit[31:24].
            3:    full_swap:    // bit[7:0]      bit[31:24] and bit[15:08]      bit[23:16].
      } SurfaceSwapMode
            // Note: any data can be swapped in any mode.
D0 [29:24]      High_Offset, Integer.// For higher part, destination bytes to where you want to put in.
D0 [23:22]      SwapModeLow, Enumerate, SurfaceSwapMode.
D0 [21:16]      Low_Offset, Integer. // For lower part, destination bytes to where you want to put in.
D0 [15:14]      High_Default_Code, Enumerate.
      {
            0x0:      Convert                 // default
            0x1:      Passthrough.            // pass source directly to Special Data Register;
            0x2:      ConvertSetConstants     // conversion and set (0.0, 0.0, 0.0, 1.0) to components
                                              // after High/Low_Offset.
                                              // Only for first eight types of conversions.
            0x3:      SetConstantOnly     // (0.0, 0.0, 0.0, 1.0) to components from High/Low_Offset.
      } PutDefaultCode
D0 [13:12]      Low_Default_Code, Enumerate, PutDefaultCode.
D0 [11:8]       High_ConversionID, Enumerate.
                // Do what type of conversions while it is moved(for higher 32bit).
```

```
    {
        0:  IEEE32_IFF,
        1:  IEEE16_IFF,
        2:  Int32_IFF,
        3:  DWORD_IFF,
        4:  Short_Normalize_IFF,
        5:  Short_IFF,
        6:  WORD_Normalize_IFF,
        7:  WORD_IFF,
        8:  UBYTE_Normalize_IFF,
        9:  UBYTE_Normalize_IFF_Swap_XZ,
        10: UBYTE_IFF,
        11: UBYTE_IFF_Swap_XZ,
        12: Signed10_Normalize_IFF,
        13: Signed10_Normalize_IFF_Swap_XZ,
        14: Unsigned10_IFF,
        15: Unsigned10_IFF_Swap_XZ,
    } PutConversionCodes
D0 [7:4]    Low_ConversionID, Enumerate. PutConversionCodes
D0 [3]      Hold, Boolean.      // Hold data for next use
    {
        0:      Retire it after use.            // default.
        1:      Hold it for next PUT use.       // one data can be used multiple times.
    }
D0 [2]      High_Valid, Boolean.
    {
        0:      Invalid, don't pass it.
        1:      Valid, pass it.
    }
D0 [1]      Low_Valid, Boolean.
    {
        0:      Invalid, don't pass it.
        1:      Valid, pass it.
    }
D0 [0]      Last_Command, Boolean.
    {
        0:      Not the last command.
        1:      The instruction is the last one for one vertex PUT.
    }
```

Conversion ID in PUT instruction is updated as follows to DX9 vertex stream types.
Conversion_ID has the following meanings:

PutConversionCodes Bit[3:0] ==

- 0000: IEEE float32 → IFF;
- 0001: 2 float16 (=1.5.10) → 2xIFF;
- 0010: signed int32 to IFF;
- 0011: unsigned int32 to IFF;
- 0100: signed short2 normalized to 2xIFF: [-1.0, 1.0] = fvalue / (215 – 1).
- 0101: signed short2 to 2xIFF: fvalue
- 0110: unsigned short2 normalized to 2xIFF: [0.0, 1.0] = fvalue / (216 – 1).
- 0111: unsigned short2 to 2xIFF: fvalue.
- 1000: uByte4 → 4xIFF in [0.0, 1.0];
  D[7:0] = X, D[15:8] = Y, D[23:16] = Z, D[31:24] = W.
- 1001: uByte4 → 4xIFF in [0.0, 1.0]; X ←→ Z.

1010: uByte4 → 4xIFF in [0.0, 255.0];
1011: uByte4 → 4xIFF in [0.0, 255.0]; X ←→ Z.
1100: signed fixed point 10-10-10 format to IFF in [-1.0, 1.0] = fvalue/(29 – 1) in order of (X, Y, Z, 1.0) and set 1.0 to W. D[9:0] = X, D[29:20] = Z.
1101: signed fixed point 10-10-10 format to IFF in [-1.0, 1.0] = fvalue/(29 – 1) in order of (Z, Y, X, 1.0) and set 1.0 to W.
1110: unsigned fixed point 10-10-10 format to IFF = fvalue in order of (X, Y, Z, 1.0) and set 1.0 to W.
1111: unsigned fixed point 10-10-10 format to IFF = fvalue in order of (Z, Y, X, 1.0) and set 1.0 to W. // X ←→ Z.

Redefine default value code as follows to save put constant instructions:

00: Do as conversionIDL/H. // default
01: Pass source directly to Special Data Register;

It means you need to put a data to a readable 32bit register without any format conversion. The special readable 32bit register has no offset or ID which does not occupy a space in vertex shader input ram. That register sits in PUT block and will be read by driver through CE. No other guy/engine will use it.

10: Do as conversionIDL/H & Set (0.0, 0.0, 0.0, 1.0) to components after High/Low_Offset.
11: Directly set (0.0, 0.0, 0.0, 1.0) to components from High/Low_Offset.

For instance, original PUT a float to destOffset, it is required to PUT as follows:
(destOffset&3) == 0: (value, 0.0, 0.0, 1.0).
(destOffset&3) == 1: (xx, value, 0.0, 1.0).
(destOffset&3) == 2: (xx, xx, value, 1.0).
For instance, original PUT a short2 to destOffset, it is required to PUT as follows:
(destOffset&3) == 0: (value0, value1, 0.0, 1.0).

Default code = 1~3 is only for first eight types of conversions.

For design simplification, if one conversionID is 0x8 ~0xF, the other 32bit source should be set invalid by SW and the last two bits of offset should be zero. Use hold flag for next instruction if the other 32bit is still needed. For conversionID = 4~7, the last bit of offset should be zero (even number).

Put parameters as order below:

0: Xscreen_position
1: Yscreen_position
2: Zscreen_position
3: Wscreen_position
4: Xclip_position
5: Yclip_position
6: Zclip_position
7: Wclip_position
8: Rdiffuse
9: Gdiffuse
a: Bdiffuse
b: Adiffuse

c: Rspecular
d: Gspecular
e: Bspecular
f: ASpecular
10: Fog
14: Psizexs
15: Psizeys
16: Psizexc
17: Psizeyc
18: U0
19: V0
1a: U1
1b: V1
1c: U2
1d: V2
1e: U3
1f: V3
20: U4
21: V4
22: U5
23: V5
24: U6
25: V6
26: U7
27: V7
…

In XP10, at least 9 data should be sent to PUT, there are XYZWs, XYZWc and diffuse. If the data are less than 9 sets, HW will get some unknown data, which will cause HW to hang.

Just Note: has to set global to reset vertex cache when change vertex setting: length, point sprite on/off, texture pairs, vertex buffer address in FETCH instruction etc.

The laws of the put instruction

- There are up to two 32-bit data chunks operated on per instruction. These are designated high and low.
- Each 32 bit data chunk, can be valid or invalid.
- If a single 32 bit data chunk is treated as a packed byte, only ONE can be converted per instruction.
- The put instruction can't move both data chunks to the same 'column' of memory. This is laid out in the offsets, so that the lower two bits of each destination cannot be equal. For example, the put processor can't have Y (1&3=1) and DiffuseGreen(5&3=1) as destinations in the same instruction. This also applies to the generic vector GP destinations, not just the RE.
- 64 bits of data are retrieved from the source at the end of an instruction, unless it's told not too. It can be 'held' and re-used in the next instruction.
- Data fetched is ALWAYS on a 64 bit boundary. Fetch instructions can provide the put processor with 32 bit aligned data however (via the vertex cache).
- With multiple streams (ie, multiple fetch instructions), data fetched is automatically packed to the next 64 bit boundary in HW. If the stride is 1, then one garbage dword will be packed

together, for a stride of 3 DWORDs a garbage dword is actually packed to 4th dword, so your put instructions must maintain proper alignment.

- A vertex is done when all put instructions are done. There is a "Last instruction" bit to set on the last instruction.

Put processor controls re-formating and re-ordering of parameters. If position in vertex buffer is ZYX, it should be re-ordered to XYZ by destOffset in PUT instructions. The similar process is done for other vertex data. Vertex data for IMM mode will bypass Cache. PUT processor puts all IMM, valid/invalid, hit/miss vertices.

A ROM (30bit*4) is available to store typical PUT instructions for default and 2D functions. 6bit to select first entry of PUT instructions in ROM (48~55) and RAM (0~47). When a data is converted to internal format, if it is overflow, clamp it to the largest one, zero for underflow and set a flag to status register. **No PUT for hit vertices.**

Put ROM instructions have to be changed in XP10. The original instructions are

```
0x0000205a,
0x20008505,
0x01000006,
0x03020006,
0x0004009a,
0x20000004,
0x00212003,
0x00000000;
```

The modified instructions are

```
0x0000205a,/*signed short2 ==>2xIFF, screen*/
0x0004205a,/*signed short2 ==>2xIFF, position*/
0x0008300a,/*directly set (0.0, 0.0, 0.0, 1.0), diffuse*/
0x000c300a,/*directly set (0.0, 0.0, 0.0, 1.0), specular*/
0x0010300a,/*directly set (0.0, 0.0, 0.0, 1.0), fog*/
0x18008505,/*signed short2 ==>2xIFF, tex0*/
0x0100000e,/*screen*/
0x05040006,/*position*/
0x0302000e,/*screen*/
0x07060006,/*position*/
0x0008009a,/*uByte4 ==> 4xIFF, diffuse*/
0x000c300a,/*directly set (0.0, 0.0, 0.0, 1.0), specular*/
0x0010300a,/*directly set (0.0, 0.0, 0.0, 1.0), fog*/
0x18000004,/*tex0.x*/
0x00192003,/*tex0.ywz*/
0x00000000
```

The following examples will demonstrate how to program PUT Array instructions.

Example 1:
Suppose the primitive data in vertex cache are ordered like Xs, Ys, Zs, Ws, Xc, Yc, Zc, Wc, Diffuse, Specular, U0, V0 and they are floating point number (but Diffuse and Specular are fixed point number). The following instructions will put these data in Command Engine.

```
40200001 //   === Load Put Instruction ===
         //   [23:23] IEEE_TO_IFF : 0x0
         //   [22:20] GP_LOAD_CODE : 0x2
         //   [18:10] GPLOAD_START_OFFSET : 0x0
         //   [ 8: 0] GPLOAD_LENGTH : 0x1
01000006
         //   [31: 0] PUT0 : 0x1000006
         //          Valid    Special ID    Dst Offset      Value Code      Conversion
         //   High:    1          0              1          Conversion      IEEE 32    ->IFF
         //   Low :    1          0              0          Conversion      IEEE 32    ->IFF
         //
03020006
         //   [31: 0] PUT1_FETCH0_D0 : 0x3020006
         //          Valid    Special ID    Dst Offset      Value Code      Conversion
         //   High:    1          0              3          Conversion      IEEE 32    ->IFF
         //   Low :    1          0              2          Conversion      IEEE 32    ->IFF
         //
05040006
         //   [31: 0] PUT2_FETCH0_D1 : 0x5040006
         //          Valid    Special ID    Dst Offset      Value Code      Conversion
         //   High:    1          0              5          Conversion      IEEE 32    ->IFF
         //   Low :    1          0              4          Conversion      IEEE 32    ->IFF
         //
07060006
         //          Valid    Special ID    Dst Offset      Value Code      Conversion
         //   High:    1          0              7          Conversion      IEEE 32    ->IFF
         //   Low :    1          0              6          Conversion      IEEE 32    ->IFF
         //
0008009a
         //          Valid    Special ID    Dst Offset      Value Code      Conversion
         //   High:    0          0              0          Conversion      IEEE 32    ->IFF
         //   Low :    1          0              8          Conversion      UByte4     ->ABGR 1.0
         //              <Hold>
0c000904
         //          Valid    Special ID    Dst Offset      Value Code      Conversion
         //   High:    1          0             12          Conversion      UByte4     ->ABGR 1.0
         //   Low :    0          0              0          Conversion      IEEE 32    ->IFF
         //
19180007
         //          Valid    Special ID    Dst Offset      Value Code      Conversion
         //   High:    1          0             25          Conversion      IEEE 32    ->IFF
         //   Low :    1          0             24          Conversion      IEEE 32    ->IFF
         //   <Last>
```

Example 2:

Suppose the primitive data in vertex cache are ordered like Z, Y, X, Ca, Cr, Cg, Cb, Sc, U, V(Ca, Cr, Cg, Cb are floating point ARGB value) and they are all floating point number. The following instructions will put these data in Command Engine.

```
01 02 00 06   // Put Z, Y to the 2, 1st dword
07 00 00 06   // Put X, A to the 0, 7th dword
05 06 00 06   // Put R, G to the 6, 5th dword
```

```
00 04 00 0A   // Put B to the 4th dword
08 00 09 04   // Put Sc to the 8 9 10 11th dwords
21 20 00 07   // Put U, V to the 0x20, 0x21th dword, and convert them to internal float format.
```

5.3. Uniform Fog Solution

XP10 can support all fog type: Z based, W based, OpenGL fog distance etc.

If Z based, your SW VS output Z to fog channel.
If W based, your SW VS output W to fog channel.
If OpenGL fog distance, VS computed fog distance to fog channel.

For Z based fog or any fog type that does not need perspective correction,
you need to set fog perspective correction disable (new register in XP10) .

Based on different fog type, you set proper other fog parameters: fogStart, fogEnd and fogDensity etc.

6. Rendering Engine Commands

Command ID for Rendering Engine:

- LOAD_RE, 0x200.
- Scissor Test Rectange, 0x210.
- Rasterization + Fog function, 0x220.
- Depth Buffering, 0x230 to 0x25C.
- Stencil Buffering Register, 0x260.
- Alpha Function Register, 0x270
- Color Operation Register, 0x280 to 0x2AC.
- Output Operation Register, 0x2B0 to 0x2CC.
- Texture Function Register, 0x300 to 0x36C.
- Pixel Shader Setting, 0x380.

6.1. Load Command

To load bump shader, pattern table, texture base address and PS etc.

Register
0x200 (W) : 0x80 LOAD_RE

```
// Load stream data following load command is sent to an indicated RAM/array.
D0 [23:20]    RE_Load_Code, Enumerate.
{
      0x0:    NULL,          // default
      0x1:    Load_Bump_Shader_Instructions,
               // RAM size = 64 x 128bit
               // start offset is 128bit aligned.
      0x2:    Load_Texture_Sampler_States.       // Size = 16x256bit.
               // 16x256 in load data array.
               // start offset is 128bit aligned.
      0x4:    Load_DeltaUV_Weights.              // array for 4x4 filtering.
              // RAM Size = 64x128bit.
              // start offset is 128bit aligned.
              // Dword[9:0]: deltaU, s.4.5 fix point.
              // Dword[19:10]: deltaV, s.4.5 fix point.
              // Dword[28:20]: weight, s.8 fix point.
      0x5:    Texture_Base_Address.      // size = 208 x 30bit.
              // start offset is 128bit aligned.
      0x6:    Load_PixelShader20_Constants.      // RAM size = 32 x 128 bits.
              // IEEE 32bit float -> 32bit IFF, 128bits in LOAD.
              // start offset is 128bit aligned.
              // PS2 Constant in normal mode RE Decoder send data
              //      data128 = {R32, G32, B32, A32}
              // for PP mode, RE covert 32 bit to 16 bit, then duplicate to fill 32 bit
              //      data128 =   {R16,R16, G16,G16, B16,B16, A16,A16}
      0x7:    Load_PixelShader20_Instructions.     // Ram size = 128 x 128bit.
              // each 90bit per 128bit.
              // start offset is 128bit aligned.
      0x8:    Reserved
```

```
        0x9:    Reserved
        0xA:    Load_Gamma_Tables.   // Ram size = 32 x 64 bits
                // 2 combined entry per 128bit, 16 for sub table
                //128bit[N]:{ ColorSubDoGamma [2N+1], ColorDoGamma [2N+1],
                              ColorUndoGamma [2N+1], TexUndoGamma [2N+1],
                              ColorSubDoGamma[2N], ColorDoGamma[2N],
                              ColorUndoGamma[2N], TexUndoGamma[2N]}
                // each entry is 16bit. 8*16 = 128 bits
                // UndoGamma means undo gamma table, DoGamma means gamma table.
                // SubSoGamma means sub table of gamma table.
                // start offset is 128bit aligned.
        0xC:    Load_Texture_LoadTable.
                // max size = 64x5bits used in PS2.0 working only mode.
                // Every 6 entries of texture load table are tightly packed into one 32 bits, first
                // in LSB and 2 MSB bits are unused. Total max 3x128 in this load command.
        0xD:    Load_ROP3_Stipple_Pattern.
                // start offset is 128bit aligned.
} RELoadCodes
D0 [18:10]      RELoad_Start_Offset, Integer.    // 32bit alignment.
D0 [9:0]        RELoad_Length, Integer.     // (unit: 32bit). Default = 0: no data to be loaded.
            // This is a positive integer number. Driver needs to pack garbage for 128bit
            // alignment. Individual engine needs to take care of data read/write for unaligned
            // instructions / constants.
D1 [31:0]       Reserved.
D2 [31:0]       Reserved.
D3 [31:0]       Reserved.
```

Note: RE Load length **unit: 32bit**. This is a positive integer number. Driver needs to pack garbage for 128bit alignment.

OP	Description	Size per entry	Max Entry
1	Bump Shader	4 instructions per 128bit	64
2	TextureSamplerState	1 set per 256bit bits	16
4	DeltaUV_Weight	4 sets per 128bits	64
5	Texture Base Address	4 addresses per 128bits	208
6	PixelShader20 Const	1 const (4D) per 128bits	32 (x128bits)
7	PixelShader20 Instr	1 instruction per 128bits	128 (x90bit)
8	Reserved		
9	Reserved		
10	Gamma Table	2 combined entry / 128bits	32 (16 for sub table) + 1 register
12	Texture Load Table	Tightly packed 128bit * N	64 (x5bit)
13	ROP3_Stipple pattern	Tightly packed 128bit * N	64 (x32bit)

Note: Always load at least 128 bits, or the element size, whichever is greater.

Programming Notes:

- It is required to set PP mode first then load constant. RE decoder need PPMODE enable signal to do different conversion operation.

As general solution: **please always put all RE load command after RE Setting and before draw primitive command.**

6.1.1. Texture Base Address List

Base Address format:
D[28]: Port Number. // default = PCI-E.
0: PCI-e.
1: FB.
D[27:0]: Base address (unit: 16 bytes). // default = 0.
One Base Address RAM is 208*30bit.

6.1.2. Pattern

Pattern for ROP3 is 8*8 pixels. Here pixel may be 1/8/16/32 bpp color.

For advanced 2D (using 3D engine), 1/8/16 bpp ROP3 pattern should be converted to 32bpp pattern, then it be loaded to graphics engine. Pattern array is shared by basic 2D (Ferrari 2D) engine and 3D engine. For basic 2D, load ROP3 pattern as usual.

6.1.3. Gamma Table

All gamma table should be in 1023 base instead of 255 base. That means your table entry[index] = 1023*pow(index*32/1023, gamma factor) + [offset].

Programming Notes:

- Limitiation for gamma table

Load_Gamma_Tables, 2 combined entry/ 128bits.
128bit[N] = { ColorSubDo[2N+1], ColorDo[2N+1],
ColorUndo[2N+1], TexUndo[2N+1],
ColorSubDo[2N], ColorDo[2N],
ColorUndo[2N], TexUndo[2N] };
ColorSubDo only has 16 entries, others have 33 entries. Each entry is 16bit.

In general case, the last entry of gamma table should be the following:

```
{ ColorUndo[32], TexUndo[32] }          //DWORD0
{ 0x0000,        ColorDo[32] }          //DWORD1
{ 0x0000,        0x0000      }          //DWORD2
{ 0x0000,        0x0000      }          //DWORD3
```

For an easy implementation in HD, driver should patch the last entry of the gamma table like the following:
// copy 64bits of the first entry to the second entry

```
{ ColorUndo[32], TexUndo[32] }          //DWORD0
{ 0x0000,        ColorDo[32] }          //DWORD1
```

{ ColorUndo[32], TexUndo[32] } //DWORD2
{ 0x0000, ColorDo[32] } //DWORD3

- Gamma Table Load should NOT be the last valid command of one batch of RE command."

6.2. Clip Ranges

These are used in rendering engine only.

Register
0x210 (R/W) : 0x84 RE_Window_Clip
// for Rasterization, may be different from one in HW Driver.
D0 [23:0] Z_Scale_Factor, IFF1.8.15. // default = -1.0. used in Setup.
D1 [29:28] Parameter_Bank_Configuration, Enumerate.
{
0: 32_Primitives,// 2 colors + 2 Tex + fog. (0~2 tex)
1: 24_Primitives,// 2 colors + 4 Tex + fog. (3~4 tex)
2: 14_Primitives,// 2 colors + 8 Tex + fog. (5~8 tex)
} ParameterBankConfig

D1 [27:16] X_Clip_Right, Integer. // default = 0.
D1 [11:0] X_Clip_Left, Integer. // default = 0, maximum is 4K

D2 [31:28] Page_Max_Prim_Threshold, Integer.
// for page sorting ram to trigger flush. Real value = double of the setting.
D2 [27:16] Y_Clip_Bottom, Integer. // default = 0.
D2 [15:12] Page_Min_Prim_Threshold, Integer.
// for page sorting to stop flush. Real value = double of the setting.
D2 [11:0] Y_Clip_Top, Integer. // default = 0.

D3 [31] Line_Stipple_Enable, Boolean.
D3 [30:27] Max_Page_Threshold, Integer.
// Reach Max page in sort Ram then output. Real value = double of the setting.

D3 [26] Line_Stipple_Mode, Enumerate.
{
0: Line_Wireframe_List.
// Reset stipple counts before every single line/wireframe is drawn
1: Line_Strip_Loop.
// Reset once every EOP. Every strip/loop needs an EOP.
} LineStippleMode
D3 [25:16] Line_Stipple_Repeat_Factor, Integer.
// Should be scaled by 4 for uniform rasterization then minus one.
D3 [15:0] Line_Stipple_Pattern, Integer.

Programming Notes:
- Parameter Bank Configuration is determined by UVPair number,
0~2 tex, Parameter_Bank_Configuration = 0 (32 primitives)
3~4 tex, Parameter_Bank_Configuration = 1 (24 primitives)
5~8 tex, Parameter_Bank_Configuration = 2 (14 primitives)
Register definition:

```
0x210   D1 [29:28]   Parameter_Bank_Configuration, Enumerate.
{
    0:    32_Primitives,   // 2 colors + 2 Tex + fog. (0~2 tex)
    1:    24_Primitives,   // 2 colors + 4 Tex + fog. (3~4 tex)
    2:    14_Primitives,   // 2 colors + 8 Tex + fog. (5~8 tex)
} ParameterBankConfig
```

Pseudo code:

```
{
    gpUVPairs = GetValue("GP_Total_Texture_Coordinate_Pairs");
    if(gpUVPairs >= 0 && gpUVPairs <= 2)
        paramBankConf = 0;
    else if(gpUVPairs >=3 && gpUVPairs <=4)
        paramBankConf = 1;
    else if(gpUVPairs >=5 && gpUVPairs <=8)
        paramBankConf = 2;
    SetValue("Parameter_Bank_Configuration", paramBankConf);
}
```

- There are three easy confusable global registers:
  Page Max Primitive Threshold, for page sorting ram to trigger flush, Real value=setting+1
  Page Min Primitive Threshold, for page sorting to stop flush, Real value = setting;
  Max Page Threshold, reach max page in sort ram then output, Real value is like following:
  Max_Page_Threshold = 0, Real value = 96 (max)
  Max_Page_Threshold = 1 ~ 12, Real value = 8 * Max_Page_Threshold
  Max_Page_Threshold > 12, Real value = 96 (max)
  And, Page Max Primitive Threshold is restricted by ParameterBankConfig, Page Min Primitive Threshold is restricted by Page Max Primitive Threshold.
- Page_Max_Prim_Threshold restricted by ParameterBankConfig
  Parameter_Bank_Configuration = 0 => Page_Max_Prim_Threshold = 16 (32 primitives)
  Parameter_Bank_Configuration = 1 => Page_Max_Prim_Threshold = 12 (24 primitives)
  Parameter_Bank_Configuration = 2 => Page_Max_Prim_Threshold = 7 (14 primitives)
- If Page_Min_Prim_Threshold = 0, Page_Max_Prim_Threshold should >= 0; else if Page_Min_Prim_Threshold > 0, Page_Max_Prim_Threshold > Page_Min_Prim_Threshold.
  Pseudo code:

```
{
    pageBasedRendering = GetValue("PageBased_Rendering_Enable");
    maxPrimThreshold = GetValue("Page_Max_Prim_Threshold");
    minPrimThreshold = GetValue("Page_Min_Prim_Threshold");
    maxPrimThreshold <<= 1;
    minPrimThreshold <<= 1;
    if(minPrimThreshold == 0)
        assert(maxPrimThreshold >= minPrimThreshold);
    else
        assert(maxPrimThreshold > minPrimThreshold);
}
```

6.3. Rasterization Setting

It is to set rasterization modes and used for rendering engine only. (band clipping is always on), antialiasing modes and fog function etc.

Register
0x220 (R/W) : 0X88 RASTER_MODE
D0 [23] RE_Advanced_2D_Shortcut_Register_Enable, Boolean.

```
    {
        0:  disable,   // default.
        1:  enable.   // Force a set of global registers / instructions to default.
    }

D0 [22]     PageBased_Rendering_Enable, Boolean.
    {
        0:  Disable, // Bypass.
        1:  Enable, // do page raster and page sorting etc.
    }
D0 [21]     End_Of_Page_At_End_Of_Triangle, Boolean.   // just use in Page Bypass Mode
D0 [18]     Flush_NoSorting_Enable, Boolean.    // no sorting when flush

D0 [17]     Constant_Color_Render_Enable, Boolean.
    {
        0:  Disable.
        1:  Enable.   // will disable Bump Shader to Pixel Blender.
    }
D0 [16]     Last_Pixel_On_Line, ReversedBoolean.
    {
        0:  No_Last_Pixel.      // default.
        1:  Draw_Last_Pixel.
    }
D0 [15]     Force_Always_Flush_Page, Boolean.
D0 [14]     Always_FullPage_Enable, Boolean.   // OE always pre-fetch full page

D0 [13:8]   Half_Line_Width, FP4.2.
D0 [7]      Disable_Multi_Triangles_For_BumpLoop, ReversedBoolean.
    {
        0:  Enable. Default.
        1:  Disable.
    }
D0 [6]      Fog_Valid_InRE, Boolean.   // Checked by color evaluation unit.
            // If invalid, it will not do evaluation to improve performance.
D0 [5]      Fog_PerspectiveDisable, ReversedBoolean.
D0 [4]      DX9C_Depth_Clip_Mode, Integer.          // for depth engine. Default = disable.
D0 [3]      Diffuse_Color_Clamp_InRE, Enumerate.   // for color tiling use only.
    {
        0:  No_Clamp.               // to the range of s.6.15 IFF range
        1:  Clamp_0_1.              // to [0, 1.0].
    } REColorClampMode
D0 [2]      Specular_Color_Clamp_InRE, Enumerate, REColorClampMode.
        // for color tiling use only.
D0 [1]      Specular_Valid_InRE, Boolean.   // Checked by color evaluation unit.
        // If invalid, it will not do evaluation to improve performance.
D0 [0]      Depth_Clip_Enable, Boolean.          // for depth engine. Default = disable.
D1 [23:22]  Fog_Table_Mode, Enumerate.
    {
        0x0:    Local_vertex_Fog,           // Disable fog table. Default.
        0x1:    Exponential_Fog.
        0x2:    Squared_Exponential_Fog.
        0x3:    Linear_Fog.
    } GlobalFogMode
    // Fog table mode can be set independently from original fog valid in vertex data.
    // Lookup result will replace its original fog source.
```

// For exponential or squared exponential fog table:
D1 [21:0] Fog_density, IFF8.14. // default = 0.0

// For linear fog table: one over fog range.
D1 [21:0] One_Over_Fog_Range, IFF8.14. // (1/(fogEnd - fogStart)), default = 1.0.

D2 [21:0] Fog_End, IFF8.14. // default = 1.0.

D3 [23:16] Fog_Color_Red, Integer.
D3 [15:8] Fog_Color_Green, Integer.
D3 [7:0] Fog_Color_Blue, Integer.

Programming Notes:

- If PageBased_Rendering_Enable is disabled, bypass page raster and page sorting etc. Usually this register can be on, but if Line_Stipple_Enable is on and drawing line or wire-frame, disable PageBased Rendering.
  Pseudo code:

```
{
    lineStippleEnable = GetValue("Line_Stipple_Enable");
    frontFillMode = GetValue("Front_Fill_Mode");
    backFillMode = GetValue("Back_Fill_Mode");
    bLineDrawed = (Primitive_Type = 0x02:  IndexedLineList
                                  or 0x03:  IndexedLineStrip
                                  or 0x07:  IndexedLineloop
                                  or 0x12:  LineListImmediateMode
                                  or 0x13:  LineStripImmediateMode
                                  or 0x17:  LineloopImmediateMode);
    if (lineStippleEnable && (frontFillMode == 0x2)) // Wireframe
        pageBaseRendering = 0;
    if (lineStippleEnable && (backFillMode == 0x2) // Wireframe
        pageBaseRendering = 0;
    if (lineStippleEnable && bLineDrawed)
        pageBaseRendering = 0;

    SetValue("PageBased_Rendering_Enable", pageBaseRendering);
}
```

- If fill mode = Wireframe or Point, and PageBased_Rendering_Enable (0x220 D0[22]) = true, have to set Flush_NoSorting_Enable (0x220 D0[18]) = true.
  Pseudo code:

```
{
    frontFillMode = GetValue("Front_Fill_Mode");
    backFillMode = GetValue("Back_Fill_Mode");
    pageBased = GetValue("PageBased_Rendering_Enable");
    if((2 == frontFillMode || 2 == backFillMode || 1 == frontFillMode || 1 == backFillMode)
        && 0 != pageBased)
    {
        flushNoSorting = 1;
    }
    SetValue("Flush_NoSorting_Enable", flushNoSorting);
}
```

- If PageBased_Rendering_Enable is disabled, please set "Always_FullPage_Prefetch = 0" and "DE_Invisible_Tile_Drop = 0"
  Pseudo code:

```
{
```

```
        pageBasedRenderingEnable = GetValue("PageBased_Rendering_Enable");
        if(0 == pageBasedRenderingEnable)
        {
            allPagePrefetch = 0;
            deInvTileDrop = 0;
        }
        SetValue("Always_FullPage_Enable", allPagePrefetch);
        SetValue("DE_Invisible_Tile_Drop", deInvTileDrop);
    }
```

6.4. Depth Buffering

6.4.1. Polygon Offset & Initial Depth

To define OpenGL polygon offset: polygon offset = factor * max(|DZx|, |DZy|) + r*unit. Two parameters are set by S/W Driver: factor and offset (= r*unit done by S/W Driver). DZx and DZy are gradients computed by setup engine. When factor = 0, offset is z bias in Microsoft D3D.

Register
0x230 (W) : 0x8C Polygon_Offset_Clip_Range
D0 [23:0] Polygon_Offset_Factor, IFF1.8.15. // default = 0.0
D1 [31:0] Polygon_Offset, IFF1.8.23. // default = 1.0.
D2 [31:0] Z_Biased_Clip_Far, IFF1.8.23. // value is (0, 1). Default = 1.0.
D3 [31:0] Z_Biased_Clip_Near, IFF1.8.23. // default = 0.0.

Biased clip range in Z direction. S/W have to reverse when rhW buffering. They are used for 1-Z and depth clipping after polygon offset. See technical reports for 1-Z buffer purpose.

Register
0x240 (W) : 0x90 Depth_Constant_Values
D0 [23:16] Reference_Stencil_CCW, Integer.
D0 [15:8] Stencil_Write_Enables_CCW, Integer.
  // default = 0x00, bit[I] = 0 means no update of that bit.
D0 [7:0] Stencil_Format_Mask_CCW, Integer. // default = 0xff.
D1 [23:0] Constant_Depth_Value, IFF5.19.// if pixel is outside of Z buffer rectangle.
  // in final depth format identical to register setting: 0x250 D0[15:14].
D2 [27:16] Z_Stencil_Buffer_Start_OffsetY, Integer. // Inclusive.
D2 [11:0] Z_Stencil_Buffer_Start_OffsetX, Integer. // Inclusive.
D3 [27:16] Z_Stencil_Buffer_End_OffsetY, Integer. // Inclusive.
D3 [11:0] Z_Stencil_Buffer_End_OffsetX, Integer. // Inclusive.

6.4.2. Z function & Z write enable

Register
0x250 (R/W) : 0X94 Depth_Buffer_Settings. Only D0 is readable.
D0 [23] Sync_Mode, Enumerate.

```
    {
        0:  HiZ_Mask_Sync.
        1:  Cov_Mask_Sync.
    } TileSyncMode
D0 [22]     Depth_Source, Enumerate.
{
    0:  Source_Z.               // default.
    1:  PixelShader_Depth.
} DepthSourceMode
    // PS2 depth for visibility test and use coverage mask to do sync.
D0 [21:19]  Depth_Test_Function, Enumerate.
// S/W driver has to reverse it due to 1-Z buffer. See a paper “Optimal Depth Buffer” published on
// the proceedings of ACM/EuroGraphics99 Hardware Rendering Workshop.
{
    0x0:    NEVER.
    0x1:    LESS.
    0x2:    EQUAL.
    0x3:    LESSEQUAL.
    0x4:    GREATER.
    0x5:    NOTEQUAL.
    0x6:    GREATEREQUAL.
    0x7:    ALWAYS.                 // default.
} VisibilityTestFunctionCodes
D0 [18]     Stencil_Test_Enable, Boolean.
{
    0:      Disable stencil test    // default.
    1:      Enable stencil test.
}
D0 [17:16]  Depth_Buffer_Configuration, Enumerate.
{
    0:  Band32_OneTile_Z_Only.  // 2x2x32x32Banded + One Tile interleaved.
    1:  Band16_OneTile_Z_Only.  // 2x2x16x16Banded + One Tile interleaved.
    2:  Band64_OneTile_ZColor.  // (1x2x64x16) Banded + One Tile interleaved.
    3:  Band32_OneTile_ZColor.  // (2x2x32x16) Banded + One Tile interleaved.
} DepthBufferFormat
    // 128bit HiZ0 + 128bit HiZ1 + 128bit stencil0 + 128bit stencil1 +
    // 256bit MiLoZ for tile0 + 256bit MiLoZ for tile1.
    // Output engine needs to pack two tiles together and
    // unpack them for depth engine. There is no difference between stencil on/off.
    // If (stencil is disabled), still leave stencil space. Not packing tightly.
    // If (16bit Z), still leave LoZ space. Not packing tightly.

D0 [15:14]  Depth_Exponent_Bits, Enumerate.
{
    0x0:    0_Bit.
    0x1:    4_Bit.                  // default.
    0x2:    5_Bit.
    0x3:    7_Bit.
} DepthExponentBits
D0 [13]     Enable_Depth_Write, Boolean.
{
    0:      Disable.                // default.
    1:      Enable.
}
D0 [12:0]   Depth_Buffer_Width, Integer.
```

```
      // divided by 4. Shared by HiZ, Mi/LoZ and Stencil.
      // default = 0.
      // for band mode, width = 64*M.
D1 [27]       DE_Invisible_Tile_Drop, Boolean.

D1 [26]       Depth_Read_Disable, ReversedBoolean.

D1 [25]       ZOverlay_Control_Disable, ReversedBoolean.
{
      0:   Enable. Default.
      1:   Disable.  // for the case wo alpha-blending and z write(or always).
}
D1 [24]       Depth_Buffer_Port, Enumerate.
{
      0:   Buffer_In_FB,
      1:   Buffer_In_PCI_e,
} MemoryPortConfiguration
D1 [23:0]     Depth_Buffer_Base_Address, Integer.
      // 32bytes alignment. Up to 256MB. Default = 0x000000
```

Programming Notes:

- If PageBased_Rendering_Enable is disabled, please set "Always_FullPage_Prefetch = 0" and "DE_Invisible_Tile_Drop = 0"
  Pseudo code:

```
{
    pageBasedRenderingEnable = GetValue("PageBased_Rendering_Enable");
    if(0 == pageBasedRenderingEnable)
    {
        allPagePrefetch = 0;
        deInvTileDrop = 0;
    }
    SetValue("Always_FullPage_Enable", allPagePrefetch);
    SetValue("DE_Invisible_Tile_Drop", deInvTileDrop);
}
```

- depth_read_disable should **not be set** when either of the following holds:
  a) pp mode is on (bumpshader needs eomerge);
  b) bump is on (bumpshader needs eomerge);
  c) depth_test_function is not ALWAYS or NEVER
  d) stencil_test_enable
  e) bound_test_enable
  f) Invisible_PixelPair_Drop_Enable = 1

6.5. Stencil Buffering

Stencil is an advanced feature applied for shadow generation, transparency and reflection etc. See OpenGL for detail.

D2, D3 of 0x250 (W) Stencil_FUNCTION

```
D2 [31]       Enable_HiZ_Prefetch_Threshold, Boolean.
```

D2 [29:24] HiZ_Prefetch_Threshold, Integer.
D2 [23:16] Reference_Stencil, Integer. // default = 0x00
D2 [15] Depth_Bound_Test_Enable, Boolean.
D2 [14] Stencil_Read_Disable, ReversedBoolean.
{ // if disable, use reference stencil as destination stencil for test & operation.
1: no read stencil. Use Reference stencil as destination stencil (stencilBuf) and do stencil operation. When stencil clear, stencil test = ALWAYS or NEVER and stencil operation does not relate to destination stencil.
0: read stencil. When stencil test is enabled.
}
D2 [13] Use_Stencil_TestMask_ForBumpSync_Enable, Boolean.
{
0: disable.
1: enable.
// When stencil operation == KEEP or writemask are all masked out,
// the stencil buffer will not finally be updated. This bit can be enabled.
// It can not work with StencilReadDisable.
// (for stencil clear: stencil operation will mostly not be KEEP)
}
D2 [12] Two_Sided_Stencil_Enable, Boolean.
// See counter clockwise stencil setting in D3 of 0x260.
D2 [11:9] Stencil_Operation_SPass_ZPass, Enumerate.
{
0x0: KEEP: // StencilRet = StencilBuf. default.
0x1: ZERO: // StencilRet = 0.
0x2: REPLACE: // StencilRet = StencilRef.
0x3: INCRSAT: // StencilRet = (StencilBuf == 0xff) ? 0xff : (StencilBuf++).
0x4: DECRSAT: // StencilRet = (StencilBuf == 0x00) ? 0x00 : (StencilBuf--).
0x5: INVERT: // StencilRet = ~StencilBuf.
0x6: INCR: // StencilRet = StencilBuf++.
0x7: DECR: // StencilRet = StencilBuf--.
} StencilOperationCode
D2 [8:6] Stencil_Operation_SPass_ZFail, Enumerate, StencilOperationCode
D2 [5:3] Stencil_Operation_SFail, Enumerate, StencilOperationCode
D2 [2:0] Stencil_Test_Function, Enumerate, VisibilityTestFunctionCodes

D3 [31] Z_16Bit_Comparison_Enable, Boolean.
D3 [28] Depth_Bound_Test_Enable_PDE, Boolean.
D3 [27] Stencil_Test_Enable_PDE, Boolean.
D3 [26:24] Depth_Test_Function_PDE, Enumerate, VisibilityTestFunctionCodes.
D3 [23:16] Constant_Stencil, Integer. // if pixel is outside of stencil buffer rectangle.
D3 [15:8] Stencil_Write_Enables, Integer.
// default = 0x00, bit[I] = 0 means no update of that bit.
D3 [7:0] Stencil_Format_Mask, Integer. // default = 0xff.

6.6. Alpha Function

Register
0x260 (R/W) : 0x98 Fragment_Operation Only D0 is readable.

D0 [23:16] Reference_Alpha, Integer. // default = 0xff.
D0 [15:13] Alpha_Blend_OpCode, Enumerate. // clamp is always enabled.

```
{
      1:   ADD.                //   SrcFactor*Src + DestFactor*Dest.
      2:   Subtract.           //   SrcFactor*Src - DestFactor*Dest.
      3:   InverseSubtract.    //   DestFactor*Dest - SrcFactor*Src.
      4:   MIN.                // Min(Src,   Dest).
      5 :  MAX.                // Max(Src,   Dest).
} AlphaBlendingOperationCodes
D0 [12:10]     Alpha_Test_Function, Enumerate, VisibilityTestFunctionCodes
D0 [9:6]       Source_RGB_Blending_Factor, Enumerate.
{
      0x0      Inverse_Temp_Alpha.
      0x1      Zero.
               // SrcColorFactor.R = SrcColorFactor.G = SrcColorFactor.B = 0.0.
      0x2      One,      // default
               // SrcColorFactor.R = SrcColorFactor.G = SrcColorFactor.B = 1.0.
      0x3      SourceColor,
               // SrcColorFactor = SrcColor. for RGB
      0x4      InverseSourceColor,
               // SrcColorFactor = 1 - SrcColor.  for RGB
      0x5      SourceAlpha,
               // SrcColorFactor.R = SrcColorFactor.G = SrcColorFactor.B = SrcColor.A.
      0x6      InverseSourceAlpha,
               // SrcColorFactor.R = SrcColorFactor.G = SrcColorFactor.B = 1- SrcColor.A.
      0x7      DestinationAlpha,
               // SrcColorFactor.R = SrcColorFactor.G = SrcColorFactor.B = DestColor.A.
      0x8      InverseDestinationAlpha,
               // SrcColorFactor.R = SrcColorFactor.G = SrcColorFactor.B = 1 - DestColor.A.
      0x9      DestinationColor,
               // SrcColorFactor = DestColor. for RGB
      0xA      InverseDestinationColor,
               // SrcColorFactor = 1 - DestColor. for RGB
      0xB      SourceAlphaSaturate,
               // SrcColorFactor.R = SrcColorFactor.G =
               // SrcColorFactor.B = MIN(SrcColor.A, 1 - DestColor.A).
      0xC      ConstantColor,
               // SrcColorFactor = ConstantColor. for RGB
      0xD      InverseConstantColor,
               // SrcColorFactor = 1 - ConstantColor. for RGB
      0xE      ConstantAlpha,
               // SrcColorFactor = Constant_Alpha;
      0xF      InverseConstantAlpha,
               // SrcColorFactor = 1- Constant_Alpha;
} RGBBlendingFactorCodes
D0 [5:2]       Destination_RGB_Blending_Factor, Enumerate, RGBBlendingFactorCodes
D0 [1:0]       Premultiply_Source_Alpha_Mode, Enumerate.
{
      0:       Disable.          // default.
      1:       Line_Alpha_Mask,
               // Src.alpha = (alphaMask<<4) | alphaMask;.
      2:       Modulate_LineAlphaMask,
               // Src.alpha = (Src.alpha * alphaMask) >> 4;.
      3:       Modulate_ConstantAlpha,
               // Temp.alpha = Src.alpha * Constant_Alpha. for third alpha blit case.
} SourceAlphaPremultiplyMode
```

6.7. Pixel Operation & Color Buffer Registers

To define pixel blending operations for advanced 2D and 3D.

6.7.1. Pixel Operation

D1 of 0x260 (W) PIXEL_Operation

```
D1 [31:28]    Source_Alpha_Blending_Factor, Enumerate.
{
      0x0     Inverse_Temp_Alpha,
      0x1     Zero,
              // SrcColorFactor.A = 0.0.
      0x2     One,
              // default. SrcColorFactor.A = 1.0.
      0x5     SourceAlpha,
              // SrcColorFactor.A = SrcColor.A.
      0x6     InverseSourceAlpha,
              // SrcColorFactor.A = 1- SrcColor.A.
      0x7     DestinationAlpha,
              // SrcColorFactor.A = DestColor.A.
      0x8     InverseDestinationAlpha,
              // SrcColorFactor.A = 1 - DestColor.A.
      0xE     ConstantAlpha,
              // SrcColorFactor.A = Constant_Alpha;
      0xF     InverseConstantAlpha,
              // SrcColorFactor.A = 1- Constant_Alpha;
} AlphaBlendingFactorCodes
D1 [27:24]    Destination_Alpha_Blending_Factor, Enumerate, AlphaBlendingFactorCodes
D1 [23]       Polygon_Stipple_Enable, Boolean.
{
      0:  disable.
      1:  enable. Load 32x32bit pattern.
}

D1 [22]       Color_Dither_Enable, Boolean.
{
      0:  disable.            // default.
      1:  enable.
}
D1 [21]       Destination_Color_Key_Polarity, Enumerate.    // for 2D operation.
{
      0:  Normal.             // default.
      1:  Invert.             // (pixel color == key color) will write the pixel out.
} ColorKeyPolarityMode
D1 [20]       Destination_Color_Key_Enable, Boolean.  // for 2D operation.
{
      0:  Disable.            // default.
      1:  Enable.
}
D1 [19:12]    ROP3_Code, Integer.           // default = 0x00. For 2D and 3D.
```

```
D1 [11]         Mono_Pattern_Expansion, Boolean.   // for 2D operation.
{
      0:        Disable.        // for color pattern (Default).
      1:        Enable.         // 1: foreground color;    0: background color.
}
D1 [10]         Destination_Color_Buffer_Read_Enable, Boolean.
{
      0:        Disable.              // default.
      1:        Enable.
}
// When alpha blending, ROP3 and bit mask operation do not need destination color,
// then do not set it. Otherwise set it. HW will read destination color based on this bit
// and tile's visibility flag. If this bit is zero, HW will not read destination color and
// just use 0x00000000 for all operations requiring destination color.

D1 [9]          Enable_Bit_Mask, Boolean.       // affect MET and MRT too.
{
      0:        Disable.                  // default.
      1:        Enable.
}
D1 [8]          ROP3_Enable, Boolean.
{
      0:        Disable.                  // default.
      1:        Enable.
}
D1 [7]          Alpha_Blending_Enable, Boolean.
{
      0:        Disable alpha blending.       // default.
      1:        Enable alpha blending.
}
D1 [6]          Alpha_Test_Enable, Boolean.
{
      0:        Disable alpha test.       // default.
      1:        Enable alpha test.
}
D1 [5]          Fog_Blending_Enable, Boolean.
{
      0:        Disable.                  // default.
      1:        Enable.
}
D1 [4]          Specular_Blending_Enable, Boolean.
{
      0:        Disable, ignore Specular Tile.      // default.
      1:        Enable.
}
D1 [3:2]        Alpha_Post_Blender, Enumerate.
{
      0x0:      Constant.       // Result.A =    a constant alpha defined in 0x0A4.
      0x1:      Diffuse.        // Result.A = Diffuse.A.  default.
      0x2:      Texture.        // Result.A = Texture.A.
      0x3:      Modulate.       // Result.A = Diffuse.A * Texture.A/255.
} PostBlenderCodes
// for RGB Post Blender: Diffuse * Texture.
D1 [1:0]        RGB_Post_Blender, Enumerate, PostBlenderCodes
      // RGB channels instead of alpha
```

// Final texel key mask is stored together with Result. Use it to combine with alpha test mask.

Programming Notes:

- Polygon_Stipple_Enable = 1, global register 0x260 must be set prior to ROP3 stipple pattern loading.

D2 of 0x260 (W) background

D2 [31:24] Background_Color_Alpha, Integer.
D2 [23:16] Background_Color_Red, Integer.
D2 [15:8] Background_Color_Green, Integer.
D2 [7:0] Background_Color_Blue, Integer.

D3 of 0x260 (W) foreground

D3 [31:24] Foreground_Color_Alpha, Integer. // default = 0xff.
D3 [23:16] Foreground_Color_Red, Integer.
D3 [15:8] Foreground_Color_Green, Integer.
D3 [7:0] Foreground_Color_Blue, Integer.

Register
0x270 (W) : 0x9C Misc_Operation

D0 [23:20] ReadDepth_GRANTPARK_THRESHOLD, Integer.
// request arbitrator parked in this agent even though THERE is no request
// FROM this agent to wait for discontinued request that may come very soon.
// after the request fifo served to empty,
// still wait for new ones until (the threshold<<2) cycles. Then serve next client
D0 [19:16] ReadDepth_REQWAIT_THRESHOLD, Integer.
// requests will not send out until wait for (the threshold<<3)
// cycles (or it reaches 8 levels).

D0 [15:12] ReadColor_GRANTPARK_THRESHOLD, Integer.
// request arbitrator parked in this agent even though THERE is no request
// FROM this agent to wait for discontinued request that may come very soon.
// after the request fifo served to empty,
// still wait for new ones until (the threshold<<2) cycles. Then serve next client
D0 [11:8] ReadColor_REQWAIT_THRESHOLD, Integer.
// requests will not send out until wait for (the threshold<<3)
// cycles (or it reaches 8 levels).

D0 [7:4] ReadTexture_GRANTPARK_THRESHOLD, Integer.
// request arbitrator parked in this agent even though THERE is no request
// FROM this agent to wait for discontinued request that may come very soon.
// after the request fifo served to empty,
// still wait for new ones until (the threshold<<2) cycles. Then serve next client
D0 [3] Page_Cache_Mode, Enumerate.
{
0: OnePagePerCache,
1: TwoPagePerCache,
} PageCacheMode
D0 [2] Color_Write_Disable, Boolean.
D0 [1] Cache_Forward_Enable, Boolean.
D0 [0] Page_Cache_Enable, Boolean.

```
D1 [31]      Constant_Alpha_Replace_Enable3, Boolean.   // for target3 / element3
D1 [30]      Constant_Alpha_Replace_Enable2, Boolean.   // for target2 / element2
D1 [29]      Constant_Alpha_Replace_Enable1, Boolean.   // for target1 / element1
D1 [28]      Constant_Alpha_Replace_Enable0, Boolean.   // for target0 / element0
             // when enable, for color buffer read, replace alpha by 1.0 (=255),
             // for write, replace alpha by 0.0 (=0)
D1 [27]      Color_Prefetch_Enable, Boolean.
D1 [26:24]   Alpha_Blend_OpCode_Alpha, Enumerate, AlphaBlendingOperationCodes.
D1 [23]      Depth_Prefetch_Enable, Boolean.
D1 [22]      Force_EndOfMerging, Boolean.  // When_LastCache_Read,
D1 [21]      Stop_Waiting_When_EndOfMerging, Boolean. // in bump shader
D1 [20]      Force_ZMIN_Or_ZMAX, Boolean.
             // Z-Cache always return min (if z-function is >=) or max (if z-function is <=) value
             // at Z-Cache pre-fetch
D1 [19:18]   Page_Size_Mode, Enumerate.
     {
          0:   16x16_Pixels,// single target
          1:   16x8_Pixels,  // two render targets or just small page
          2:   16x4_Pixels,  // three and four render targets or just small page
     } PageSizeSelection

D1 [17]      Bypass_Ram_Control_On, Boolean.
D1 [16]      PreFetch_Flush_On, Boolean.    // Prefetch flush can be delayed.
D1 [15]      Color_Compression_Enable, Boolean
D1 [14]      Stencil_Write_Enable_OE, Boolean.

D1 [13:12]   Color_ReadBack_Count_Limit, Integer;
             // allows (20+(Count_Limit<<2)) levels for read back color.

D1 [11:8]    L2_Request_Flush_Threshold, Integer.
             // hold (n+1) requests in FIFO, then Flush.
D1 [7]       L2_Request_Reorder_Enable, Boolean.
D1 [6:4]     L2_Reorder_Time_Threshold, Integer;          // 16t*(n+1);

D1 [3:0]     Round_Factor_For12_10, Integer.

D2 [31:30]   Reserved.
D2 [29]      Conversion_Blit_Enable, Boolean.
D2 [28]      Alpha_Blending_Rounding_Mode, Enumerate.
     {
          0:   Add_Round_Clamp.    // default.
          1:   Round_Add_Clamp.
     } AlphaBlendingRoundingMode

D2 [27]      Bypass_Dither, Boolean.
{
     0:   not bypass.
     1:   Bypass. No operation.
}
// Dithering on matrix if dither is on,
// Dithering on constant if dither is off.

D2 [26]      Destination_Color_Buffer_Read_Enable3, Boolean.
{
     0:        Disable.          // default.
```

```
      1:          Enable.
}
// When alpha blending and bit mask operation do not need destination color,
// then do not set it. Otherwise set it. HW will read destination color based on this bit
// and tile's visibility flag. If this bit is zero, HW will not read destination color and
// just use 0x00000000 for all operations requiring destination color.

D2 [25]         Destination_Color_Buffer_Read_Enable2, Boolean.
{
      0:          Disable.              // default.
      1:          Enable.
}
D2 [24]         Destination_Color_Buffer_Read_Enable1, Boolean.
{
      0:          Disable.              // default.
      1:          Enable.
}

D2 [23:20]      Component_Mask, Integer.  // 1 means no write out.
                // D[23]Alpha, D[22] Red, D[21] Green, D[20] Blue.
D2 [19:18]      Conversion_Blt_Code, Enumerate.
      {
            0x0  CompressColorBuffer,       // from decompressed to compressed
            0x1  DeCompressColorBuffer,     // from compressed to decompressed
            0x2  DepthToColor,              // use Z buffer base for destination color buffer
                                            // with the same other settings as source color except
                                            // for decompression.
            0x3  ColorToDepth,              // Source buffer will consider compression but
                                            // destination is only uncompressed.
      } ConversionBlitCode

D2 [17]         Time_Stamp_On, Boolean.         // for realtime performance analysis.
{
      0:          Off.        // default.
      1:          On.         // output time stamps instead of color. Keep alpha unchanged.
}
D2 [16]         Request_Combine, Boolean.       // for color requests from OE#1 to OE#2 (GEMI).
{
      0:          disable combining.
      1:          enable combining.
}
D2 [15:13]      PCI-E _Arbitration_Threshold, Integer.                  // for OE#2 == GEMI.
                // 0:  No Threshold, served until FIFO empty.
// TH>=1:Threshold = 4*TH requests.
D2 [12:10]      Zport_Arbitration_Threshold, Integer.
// 0:  No Threshold, served until FIFO empty.
// TH>=1:Threshold = 4*TH requests.
D2 [9:7]   ColorPort_Arbitration_Threshold, Integer.
// 0:  No Threshold, served until FIFO empty.
// TH>=1:Threshold = 4*TH requests.
D2 [3]          Alpha_Blending_Need_Destination_Color, Boolean.
{
0:    does not require destination color. Default.
      1:    requires destination color.   0x260 D1[10] has to be 1.
}
```

D2 [2] EOMerge_Generation_Mode, Enumerate.
{
0: Necessarily, // Conditionally send EOMerge
1: Always, // for always send EOMerge
} EOMergeGenerationMode
D2 [1] Color_Sync_Depth_Disable, ReversedBoolean.
{
0: Enable;
1: Disable;
}

D2 [0] Color_Buffer_Port, Enumerate, MemoryPortConfiguration. // for all MRTs.

D3 [31:28] Write_DepthColor_GTANTPARK_Threshold, Integer.
// request arbitrator parked in this agent even though THERE is no request
// FROM this agent to wait for discontinued request that may come very soon.
// after the request fifo served to empty,
// still wait for new ones until (the threshold<<2) cycles. Then serve next client
D3 [27:24] WriteDepth_REQWAIT_Threshold, Integer.
// requests will not send out until wait for (the threshold<<3)
// cycles (or it reaches 8 levels).

D3 [22:18] Polygon_Stipple_Xoffset, Integer.
D3 [16:12] Polygon_Stipple_Yoffset, Integer.

Counter Clockwise Stencil setting for Two Sided Stencil function.

D3 [11:9] Stencil_Operation_SPass_ZPass_CCW, Enumerate, StencilOperationCode.
// for stencil test pass & depth test pass.
D3 [8:6] Stencil_Operation_SPass_ZFail_CCW, Enumerate, StencilOperationCode.
// for stencil test pass & depth test fail.
D3 [5:3] Stencil_Operation_SFail_CCW, Enumerate, StencilOperationCode.
// for stencil test fail.
D3 [2:0] Stencil_Test_Function_CCW, Enumerate, VisibilityTestFunctionCodes.

We have constant color rendering mode to accelerate Z/stencil only and their clear with color. In general, each target/element is cleared by one batch of commands. 4 targets/elements needs four batches of commands.

Even if depth buffer is not “discardable”, if the depth buffer is cleared every frame, it still can be double buffered. The only issue is waste memory.

If the depth buffer is persistent and without clear every frame. We still can use interleave mode: conversion blit last frame’s Z to a reserved buffer, then blit back to new target. I think this is rare case. (there is no direct conversion blit from last frame Z to new frame Z).

Programming Notes:

- Page Size is defined by global register “Page_Size_Mode” (0x270 D1 [19 : 18]) , and this register setting is restricted by MRT targets.
  MRT = 0, Page_Size_Mode = 0 (16x16)
  MRT = 1, Page_Size_Mode = 1 (16x8)
  MRT = 2, Page_Size_Mode = 2 (16x4)

MRT = 3, Page_Size_Mode = 2 (16x4)
Pseudo code:

```
{
    totalTargets = GetValue("Total_Targets_Elements");
    if(totalTargets == 0)
        pageSizeMode = 0;
    else if (totalTargets == 1)
        pageSizeMode = 1;
    else if (totalTargets >= 2)
        pageSizeMode = 2;
    SetValue("Page_Size_Mode", pageSizeMode);
}
```

- If Stencil Write and texture engine is changing write mask, i.e.
  1) Alpha_Test_Enable
  2) Polygon_Stipple_Enable
  3) Dest_Color_Key_Enable
  4) Texture_Kill_Enable(SamplingState = 6 (Coord_Kill_NoClamp) or 7 (Coord_Kill_Clamp))

  Set Color_Sync_Depth_Disable to 0.

- About Page_Cache_Enable and Page_Cache_Mode setting:
  In normal case, set

```
Page_Cache_Enable = 0x0;      // Cache hold data as much as it can;
// GE will ignore Page_Cache_Mode setting when Page_Cache_Enable = 0x0;
```

  In a special case: PointSprite && Alpha Blending enable, set

```
Page_Cache_Enable = 0x1;      //one cache hold at most one page or two page.
Page_Cache_Mode = 0x0;        //One page one cache;
```

6.7.2. Color depth, buffer width

Register
0x280 (R/W) : 0xA0 COLOR_BUFFER

```
D0 [23:22]    Destination_Selection, Enumerate.
{
    0:  Destination_Buffer;
    1:  Diffuse;
    2:  Specular;
    3:  Background_Global;
} DestinationSelector
D0 [21:20]    Fog_Color_Selection, Enumerate.
{
    0:  Fog_Color_Global;
    1:  Diffuse;
    2:  Specular;
    3:  Texture;
} FogColorSelector

D0 [19]       MRT_Mode, Enumerate.
{
```

```
      0:    MET_mode.   // default.
      1:    MRT_mode.   // D0[23:22] must be non-zero.
} MultipleRenderTargetMode
D0 [18:16]      Main_Target_Color_Format, Enumerate.
// Color+Alpha BPP For main render target #0.
{
      0x0: RGB565.                  // default. 16bpp
      0x1: ARGB1555.                // 16 bpp.
      0x2: ARGB4444.                // 16 bpp.
      0x4: ARGB8888.                // 32 bpp.
      0x5: ARGB2_10_10_10.          // 32 bpp.
} RenderTargetFormat
D0 [15:14]      Total_Targets_Elements, Integer.
      // The number of Multiple Render Targets or Multiple Elements Texture target.
      //    0: means one target or one element; 3 means 4 targets or elements.
      //    If (it is multiple element target)
      //          HW will only need main render target setting.
D0 [13:12]      Banded_Tiled_Mode, Enumerate.
{
      0:    Linear.         // when color compression on, it also means 2 scanlines linear.
      1:    Tiled.          // 4x4 pixel tiled.
      2:    Banded64.       // 1x4x (64*16) pixel banded. when color compression on
                            // (0x270_D1_15), it means banndedcompression mode
      3:    Banded32_Tiled.   // 2x2x (32*32) pixel banded + 4*4 pixel tiled.
} BandTileMode
D0 [11]         Color_Interleave_With_Zbuffer, Boolean.
{
      0:    no interleave with Z.           // Z buffer is always 2x2x32x32 banded_twoTiles
      1:    interleaved with Z buffer.      // For Band64 and Band32_Tiled
}
D0 [10:0]       Color_Destination_Buffer_Width, Integer.  // divided by 4.       default = 0x000.
      // This is a width of alpha destination buffer as well.
      // for linear mode, width = 16*N. for band mode, width = 64*M.

D1 [31:26]      MRT_Read_Order, Integer.   // 2bit=MRT_NO/read.
      // [31:30] for Read3, [27:26] for Read1. MRT0 is always read first.
      Enumerate
{
      1:    MRT1.
      2:    MRT2.
      3:    MRT3.
} MRTReadOrder
D1 [25]         Tile2X2X4X4_Enable, Boolean.  // for BandRileMode = 1 or 3 case only.
D1 [24]         Gamma_Correction_Enable, Boolean.
      // for both read & write. For render target#0 only.

D1 [23:0]       Color_Buffer_Base_Address0, Integer.
        // 32bytes alignment. Up to 256MB. default = 0x000000.

D2 [31:0]       Destination_Color_Key_CONST_COLOR, Integer.
      // for destination color key and constant rendering color.
      // A = D[31:24], R = D[23:16], G = D[15:8], B = D[7:0].

D3 [31:0]       Color_Write_Bit_Mask0, Integer.             // default = 0xffffffff.
      // For each bit of 0 to 31:
```

```
//    0:        Don't write out this bit.
//    1:        Write out this bit.
```

Note:

- Pixel Shader outputs key mask for every MRT. For correct write mask sync, MRT0 has to be first MRT from PS. Driver has to make it.
- Color buffer in low address should be shifted from depth buffer by one bank (4KB) to keep better memory efficiency.

6.7.3. Multiple Render Target Buffer Configuration

Register
0x290 (W) : 0xA4 Render_Targets
D0 [2:0] Target1_Color_Format, Enumerate, RenderTargetFormat.
D0 [3] Bit_Masking_Enable1, Boolean.
D0 [7:4] Component_Mask1, Integer.
D0 [10:8] Target2_Color_Format, Enumerate, RenderTargetFormat.
D0 [11] Bit_Masking_Enable2, Boolean.
D0 [15:12] Component_Mask2, Integer.
D0 [18:16] Target3_Color_Format, Enumerate, RenderTargetFormat.
D0 [19] Bit_Masking_Enable3, Boolean.
D0 [23:20] Component_Mask3, Integer.

D1 [23:0] Color_Buffer_Base_Address1, Integer.
// 32bytes alignment. Up to 256MB. default = 0x000000.

D2 [31:0] Bit_Mask1, Integer;

Register
0x2A0 (R/W) : 0xA8 Render_Target23
D0 [23:0] Color_Buffer_Base_Address2, Integer.
// 32bytes alignment. Up to 256MB. default = 0x000000.
D1 [23:0] Color_Buffer_Base_Address3, Integer.
// 32bytes alignment. Up to 256MB. default = 0x000000.

D2 [31:0] Bit_Mask2, Integer;
D3 [31:0] Bit_Mask3, Integer;

Register
0x2C0 (W) : 0xB0 Conversion_Blit_Rectangle
D0 [23:12] Source_Ystart, Integer. // unsigned 12bit integer = pixel based. Default = 0x000
D0 [11:0] Source_Xstart, Integer. // pixel based. default = 0x000.

D1 [31] Blit_Fill_Mode_Enable, Boolean.
D1 [23:12] RECT_Height, Integer.
// unsigned 12bit integer = (pixels in height - 1). Default = 0x000
D1 [11:0] RECT_Width, Integer.
// unsigned 12bit integer = (pixels in width - 1). Default = 0x000.

D2 [23:12] Dest_Ystart, Integer. // (pixels based). Lowest 2bit = Source_Ystart[1:0]
D2 [11:0] Dest_Xstart, Integer. // (pixels based). Lowest 2bit = Source_Xstart[1:0]

7. Bump Shader & Texture Samplers

7.1. Bump Shader & Setting

Maximum texture (Bump Shader) instructions per loop is 32.

7.1.1. Bump Shader Instruction

Instruction BumpShader, 32bit/instruction

```
D0 [3:0]        LeftTCI, Integer.
D0 [7:4]        RightTCI, Integer.
D0 [11:8]       LeftSamplerID, Integer.
D0 [15:12]      RightSamplerID, Integer.
D0 [18:16]      SamplingState, Enumerate.
{
      0:    Invalid_Pair;
      1:    Valid_Invalid; // left valid, right invalid
      2:    Invalid_Valid; // left invalid, right valid
      3:    Valid_Valid;   // both valid
      4:    Coord_NoKill_NoClamp;
      5:    Coord_NoKill_Clamp;
      6:    Coord_Kill_NoClamp;
      7:    Coord_Kill_Clamp;
} SampleState
D0 [19]         RepeatUForVEnable, Boolean.    // only for right pipe, V1 = U1
D0 [21:20]      LeftBumpAttribute, Enumerate.
{
      0:    No_Bump_Required;
      1:    2D_Bump_Required;
      2:    4D_Bump_Required;    // force it no perspective correction.
} BumpAttribute
D0 [23:22]      RightBumpAttribute, Enumerate, BumpAttribute.
D0 [26:24]      TextureType, Enumerate.
{
      0:    2D_Texture;
      1:    Depth_Texture;
      2:    Cube_Map;
      3:    3D_Texture;
      4:    Projected_2D_Texture;
      5:    Projected_Depth_Texture;
      6:    Projected_TexCoord;          // SamplingAttribute should = 4~7
      7:    Projected_3D_Texture;
} TextureType
D0 [27]         Transpose_Even_Odd, Boolean.
{
```

0: even;
1: odd.
}
D0 [28] Transpose_Enable, Boolean.
D0 [29:29] End_Of_Texture, Boolean.
D0 [30:30] End_Of_Loop, Boolean.
D0 [31:31] Replacement_Enable, Boolean.
// texture coordinates replace diffuse and /or specular in a loop.
// U0 to R, V0 to G, U1 to B and V1 to A for a texture pair to replace a color.

TCI (4bit) = Texture Coordinate Index, what texture coordinate will be used for current texture pair in current loop. SamplerID (4bit) indicates which texture sampler states will be used for the texture in current loop.

SamplingAttribute (3bit):
0: invalid texture pair; // no this case.
1: valid_left_lookup, invalid_right;
2: invalid_left, valid_right_lookup;
3: valid_left_right_lookup;
4: valid, texture coordinate mode, no TexKill, no clamp;
5: valid, texture coordinate mode, no TexKill; clamp;
6: valid, texture coordinate mode, TexKill, no clamp;
7: valid, texture coordinate mode, TexKill; clamp;
Pass clamp bit down to color pipe if replacement is enabled, otherwise pass it down to texture filtering.

BumpAttribute (2bit):
00: default, no bump required;
01: 2D bump required;
10: 3D/4D bump required; force it no perspective correction.

TextureType (3bit):
000: 2D Texture;
001: Depth Texture;
010: Cube Map;
011: 3D Texture;
100: Projected 2D Texture;
101: Projected Depth Texture;
110: Projected Texcoord; // SamplingAttribute should = 4~7
111: Projected 3D Texture;

If a texture type is 3D/4D or texture coordinate mode, RightType, RightSamplingState and RightBumpAttribute should be set to the same as Left Ones.

Texture Coordinates Replace Diffuse and /or Specular

- Every loop has 2bits global to enable replace diffuse and specular respectively.
- Bump shader instruction bit31 to enable the replacement.
- If global say to replace diffuse, do not send diffuse, wait for bump shader instruction execution result. The same for specular.
- If global enable the replacement, the corresponding bump shader instructions are always put the last one or two instructions with bit31 enable.

- Global bits should be consistent with enable in bump shader instruction. (driver make sure)
- If one of replacements is enabled, only last instruction can be enabled too. The instruction is for the one which global enabled.
- If both replacements are enabled, last two bump shader instructions need to be enabled. and always last second instruction for diffuse, last one for specular.
- If global enable replacement, color clamp is done based on enable bit in Bump Shader instruction for the loop. Otherwise clamp is done based on color clamp global register.
- If specular is invalid in global setting, but specular replacement is enabled, the specular will be valid after replacement in current loop.

Vector Transpose for Texm3x3*

We add two bits in bump shader instruction: D[28] = transpose_enable, D[27] = Even/Odd component. The operation is defined as below:

```
If transpose_enable == FALSE,
{
    keep the current operation. read a row.
    source read pointer = (TCI<<1) for U.
    source read pointer ++ for V.
}
else
{
    // read a column as a pair of textures to PS;
    source read pointer = (TCI<<1) + Even (0) / Odd (1) for U.
    source read pointer += 4 for V.
}
```

It will automatically transpose matrix for tex3x3* like instructions in PS and save PS instructions without SW workaround by sacrifying 12 VS instructions.

For example, texm3x3spec or texm3x3tex, code BS as following:

	Left	Right	
BS0:	tci0, sampler0.		
BS1:	tci2, transpose, even,	tci6, transpose, even.	
BS2:	tci2, transpose, odd,	tci6, transpose, odd.	
BS3:	tci3, transpose, even,	tci7, transpose, even.	
[BS4:	tci3, transpose, odd,	tci7, transpose, odd.	// for texm3x3vspec only.]

Programming Notes:

- Bump Shader Instruction used texture TCI is restricted by the texture pairs from GP.
  Maximum BumpShader_Instruction::TCI < GP_Total_Texture_Coordinate_Pairs * 2
  For example, if GP_TOTAL_TEXTURE_COORDINATE_PAIRS = 2, BS can only use TCI from 0 ~ 3.
  Pseudo code:

```
{
    gpTwoSideLighting = GetValue("Two_Sided_Lighting_Enable");
    gpUVPairs = GetValue("GP_Total_Texture_Coordinate_Pairs");
    if(gpTwoSideLighting)
        gpUVPairs -= 2;
    maxTCI = GetMaximumTCIInBumpShader(curBatchData);
    assert(maxTCI < gpUVPairs*2);
```

```
}
```

7.1.2. Bump Shader Setting:

```
Register
0x300 (R/W) : 0xC0     Bump_Shader_Setting
D0 [23:16]      Bump_Sync_Enables, Integer.
      // 1bit/loop. D0[16]: loop0, D0[23]: loop7.
      // Zero = Don't keep key pixels, One = Keep key pixels for LOD
      // 0:  use depth_mask to sync texture tile. Default. Don't keep key pixels
      // 1:  use (depth_mask&0xA0A0) to sync texture tile. Keep key pixels for LOD
D0 [23]         Bump_Sync_Enable_Loop7, Boolean.
D0 [22]         Bump_Sync_Enable_Loop6, Boolean.
D0 [21]         Bump_Sync_Enable_Loop5, Boolean.
D0 [20]         Bump_Sync_Enable_Loop4, Boolean.
D0 [19]         Bump_Sync_Enable_Loop3, Boolean.
D0 [18]         Bump_Sync_Enable_Loop2, Boolean.
D0 [17]         Bump_Sync_Enable_Loop1, Boolean.
D0 [16]         Bump_Sync_Enable_Loop0, Boolean.

D0 [15]         Invisible_PixelPair_Drop_Enable, Boolean.
      // It can be enabled only when there is no 2D/4D bump loop case and
      // shadow/working mode in PS setting D0[6] = 0.
D0 [14:12]      Bump_Shader_Total_Loops, Integer. // Zero means (1) color loop only.
D0 [11]         Diffuse_Only, Boolean.                  // no texture case.
D0 [10:9]       Bump_Shader_Mode, Enumerate.
      {
            0:    One_Loop_Only         // no bump loop, consistent to total bump loop = 0.
            1:    2D_Bump_Loop
            2:    4D_Bump_Loop
            3:    2D_4D_Bump_Mixed
      } BumpShaderMode

D0 [8:4]        Bump_Shader_Maximum_Tiles_For_Loop, Integer.        // (N-1).   default = 0.
      // It controls bump map loop. See detail of control algorithm in Architecture specification.

D0 [3]          Bypass_L2_Cache, Boolean.
D0 [2]          Bypass_L1_Cache, Boolean.
D0 [1]          Clear_L2_Cache, Boolean.
{
      0:        No Clear.             // default.
      1:        Clear tag ram.        // Reset cache Tags.
}
D0 [0]          Clear_L1_Cache, Boolean.
{
      0:        No Clear.             // default.
      1:        Clear tag ram.        // Reset cache Tags.
}
// In general, Flush L2 will not be frequent and do it only when texture swap occurs. L1 cache
// should be flushed when textures are updated. Well designated clear caches may
// decrease / increase efficiency sometimes and remove incorrect images.
```

D1 [05:0] Loop0_Bump_Instruction_Entry, Integer.
D1 [7] Anisotropic_Filtering_Enabled, Boolean.
D1 [13:8] Loop1_Bump_Instruction_Entry, Integer.
D1 [21:16] Loop2_Bump_Instruction_Entry, Integer.
D1 [29:24] Loop3_Bump_Instruction_Entry, Integer.
D2 [05:0] Loop4_Bump_Instruction_Entry, Integer.
D2 [13:8] Loop5_Bump_Instruction_Entry, Integer.
D2 [21:16] Loop6_Bump_Instruction_Entry, Integer.
D2 [29:24] Loop7_Bump_Instruction_Entry, Integer.

D3 [3:0] LeftL1_TextureID_Threshold, Integer.
D3 [4] LeftL1_TexIDThreshold_Enable, Boolean.
D3 [5] RightL1_TexIDThreshold_Enable, Boolean.
D3 [11:8] RightL1_TextureID_Threshold, Integer.
D3 [12] Float_TexCoord_Enable, Boolean.
// enable only for PS2.0 and no clamp for texture coordinate mode.

D3 [31:16] Color_Replace_Enable, Enumerate.
// 2bits /loop. D3[17:16] for loop0, D3[31:30] for loop7.
{
0: No_Replacement.
1: Replace_Diffuse.
2: Replace_Specular.
3: Replace_Diffuse_Specular.
} ColorReplacementMode

7.1.3. PS2.0 Pixel Based LOD Bias

HW will have a global register: D1 and D2 of 0x310, 64bit

Register
0x310 (W) : 0xC4 PixelBased_LodBias_TextureLoad
D1 [31:0] Pixel_LodBias_Enables32, Integer.
// one bit per bump shader instruction. bit0 for BS0, ..., bit31 for BS31.
// D1 [0] BiasLoadEnableForBumpShader0, Boolean.
// D1 [1] BiasLoadEnableForBumpShader1, Boolean.
// …
// D1 [31] BiasLoadEnableForBumpShader31, Boolean.

D2 [31:0] Pixel_LodBias_Enables64, Integer.
// one bit per bump shader instruction. bit0 for BS32, ..., bit31 for BS63.
// D2 [0] BiasLoadEnableForBumpShader32, Boolean.
// D2 [1] BiasLoadEnableForBumpShader33, Boolean.
// …
// D2 [31] BiasLoadEnableForBumpShader63, Boolean.

It equivalently adds one bit to each bump shader instruction. Use bump shader read pointer to select correspondent bit in bump shader execution block and pass this bit to LOD engine. Each texture pair needs to attach only one bit.

For any texture: 2D/cube/projected/depth texture etc,
First bump shader loads the texture coordinate with W = pixel lod bias.
Set **BiasLoadEnable == 1 for the BS instruction.**
Second bump shader loads the texture with sampler as usual
but set BiasLoadEnable == 1 for the BS instruction.

If a texture is 4D bump required, the SW driver should program previous loop PS shader to output 4D bump two times. And set 4D bump required for both pairs: texture coordinate mode and original texture.

7.1.4. Texture Instruction with Swizzle RGAA

Bump shader already support swizzle code = RGBB for texture instruction but missed RGAA. The simple implementation/fixing is add bits to D3 of register 0x310, as below.

D3 of 0x310 (W) RGAA_Enables
D3 [31:0] RGAA_Enables, Integer.
D3 [0] RGAAEnableForBumpShader0, Boolean.
D3 [1] RGAAEnableForBumpShader1, Boolean.
…
D3 [31] RGAAEnableForBumpShader31, Boolean.

It also equivalently adds on bit to each bump shader instruction. Use bump shader read pointer to select correspondent bit in bump shader execution block and do operation as followings:

```
if (Transpose is off)
{
    If (RGAAEnable)      // for right pipe only (in HW is left pipe)
    {
        // similar to Repeat U for V.
        right texture coordinate U read pointer = (right texture coordinate index<<1)+1;
        copy the U to V;
    }
    else if (RGBBEnable)  // == RepeatUForV
    {
        right texture coordinate U read pointer = (right texture coordinate index<<1);
        copy the U to V;
    }
    else
    {
        right texture coordinate U read pointer = (right texture coordinate index<<1);
        right texture coordinate V read pointer = (right texture coordinate index<<1)+1;
    }
}
else
{
    right texture coordinate U read pointer =
```

(right texture coordinate index<<1) + Even (0) / Odd (1);
if (RGAAEnable | RGBBEnable)
Copy the U to V;
Else
right texture coordinate V read pointer = right texture coordinate U read pointer +4;
}

7.2. Texture Sampler State Array [N]

Instruction TextureSampleState

D0 [31] Texture_Prefetch_Enable, Boolean.
D0 [30] Hold_ForPack_ForcedTexture, Boolean.
D0 [29] Force_Texture_ToRightPipe, Boolean.

D0 [28] DXTn_Fix4X4_Enable, Boolean.
// only set for cube/3D volume texture with DXTn format.
D0 [27] YUV2RGB_Conversion_Enable, Boolean.
// YUV texture may disable this bit, non YUV texture may enable this bit.
D0 [26] YUV_DCT_Enable, Boolean.
D0 [25] Paired_Texture_Modulate_Alpha_Enable, Boolean.
D0 [24] Paired_Texture_Modulate_RGB_Enable, Boolean.
// For (1) Two bilinear filtered 2D texture modulation,
// still pass second pipe's texture down to PS and
// (2) weight multiplication of precise high order filtering.
D0 [23] Mipmap_Linear_Filter_Enable, Boolean.
// MIPMAP is always enabled. Let Max LOD = 0 for disable.
{
0: Point_Sampling. // default
1: Linear_Filtering.
}

D0 [22:21] Magnification_Filter_Mode, Enumerate.
{
1: Point_Sampling.
2: Linear_Filtering.
3: Anisotropic_Filtering.
} TextureFilterMode

D0 [20:19] Minification_Filter_Mode, Enumerate, TextureFilterMode.

D0 [18:16] Max_Anisotropic_Ratio_Exponent, Enumerate.
{
0: OneToOne // 1:1
1: TwoToOne // 2:1
2: FourToOne // 4:1
3: EightToOne, // 8:1
4: SixteenToOne // 16:1
0x5 to 0x7 Reserved.
} AnisotropicRatio

```
D0 [15:12]      Maximum_LOD, Integer.      // default = 0
                // This is a positive integer number range from 0 to 12.
D0 [09:0]       Lod_Bias, SignedFixPoint 1.4.5.  // default = 0.

D1 [31]         Fast_Anisotropic_Enable, Boolean.

D1 [30:16]      Texture_Height, IFF4.11
D1 [30:27]      Texture_Height_Exponent, Integer.    // default = 0
D1 [26:16]      Texture_Height_Mantissa, Integer.    // default = 0
D1 [14:0]       Texture_Width, IFF4.11
D1 [14:11]      Texture_Width_Exponent, Integer.     // default = 0
D1 [10:0]       Texture_Width_Mantissa, Integer.     // default = 0
      // This is a positive integer number ranging from 0 to 4096.

D2 [31]         TEX_KILL_Enable, Boolean.
D2 [30:28]      W_Addressing_Mode, Enumerate.           // texel based.
      {
            0:          None.          // Always W = W&0xfff;
            1:          Wrap.          // default
            2:          Mirror.
            3:          MirrorOnce.
            4:          Clamp.
            5:          Clamp_To_Edge.
            6:          Clamp_To_Border.
            7:          Reserved.
      } TexelAddressMode
D2 [27:22]      TextureFilteringEntry, Integer.
      // for high order filtering deltaUV or UV offset for deltaUV;

D2 [21:16]      TextureFilteringLength, Integer.        // Actual size = TextureFilteringLength+1;
D2 [14:11]      Texture_Depth_Exponent, Integer.     // default = 0.  third dimension size.
D2 [10:0]       Texture_Depth_Mantissa, Integer.     // default = 0.

D3 [31:24]      Texture_Base_Index, Integer.     // to base address array. default = 0.

// Kernel size & scale factor for high order filtering
// Ku = KernelSizeU<<(5-Su).
// TextureFilteringLength+1 = KernelSizeU*KernelSizeV.
D3 [23:21]      Vertical_Scale_Factor, Integer.              // Sv;
D3 [20:12]      Vertical_Scaled_Kernel_Size, Integer.       // Kv;
D3 [11:9]       Horizontal_Scale_Factor, Integer.            // Su;
D3 [8:0]        Horizontal_Scaled_Kernel_Size, Integer.   // Ku;

D4 [31]         OpenGL_Addressing_Enable, Boolean.
D4 [30]         Border_Mode, Enumerate.   // default = 0.
      {
            0:    Border_Color.// use constant border color.
            1:    Border_Texel.// use border texel in texture map. The border width = 1.
      } BorderMode
D4 [29:27]      V_Addressing_Mode, Enumerate, TexelAddressMode
D4 [26:24]      U_Addressing_Mode, Enumerate, TexelAddressMode.
// For DX7 MS_Border mode, please use Clamp_To_Border + border color.
// For DX8~ MS_Border mode, please set texture format = NULL and set proper border color.
// For MS_Clamp, please use Clamp_To_Edge.
```

```
D4 [23:22]      Texture_Swap_Mode, Enumerate, SurfaceSwapMode.   // (byte revert in 32bit)
      {
            0:    no_swap:      // unchanged.   default.
            1:    half_swap:    // bit[7:0] <--> bit[15:08] and bit[31:24] <--> bit[23:16].
            2:    word_swap:    // bit[7:0] <--> bit[23:16] and bit[15:08] <--> bit[31:24].
            3:    full_swap:    // bit[7:0] <--> bit[31:24] and bit[15:08] <--> bit[23:16].
      } SurfaceSwapMode
D4 [21:20]      TotalElements, Integer.
      // Range = 0~3, (N-1). Maximum 4 elements.    0:    single element.

D4 [19]         Enable_2x2xTile_Mode, Boolean.      // only valid for D4[17:16] = 1, 3.
D4 [18]         Enable_CompressedColorBuffer, Boolean.// surpass D4[17:16] & D4[19].
D4 [17:16]      Texture_Band_Tile_Mode, Enumerate, BandTileMode.
      {
            0:    Linear.         // default, when color compression on, it also means 2 scanlines linear.
            1:    Tiled.          // 4x4 pixel tiled.
            2:    Banded64.   // 64*16 pixel banded. when color compression on (0x270_D1_15),
                              // it means banndedcompression mode
            3:    Banded32_Tiled.   // 32*32 pixel banded + 4*4 pixel tiled.
      } BandTileMode

D4 [15:11]      Texture_Format, Enumerate. // see format conversion in architecture specification.
{
      0:          NULL          // default. No map available, Return Border color.
      1:          Reserved1.
      2:          Reserved2.
      3:          XRGB8888.     // 32bit, may be tiled.
      4:          RGB565.       // may be tiled.
      5:          L16.          // may be tiled
      6:          G16R16.       // may be tiled.
      7:          Reserved7.
      8:          XRGB1555.     // may be tiled.
      9:          G16R16F.      // may be tiled.
      10:         ARGB4444.     // may be tiled.
      11:         L8.           // luminance map
      12:         A8L8.         // may be tiled.
      13:         U8V8.         // for bump mapping; may be tiled.
      14:         U5V5L6.       // may be tiled.
      15:         V16U16.       // may be tiled.
      16:         UYVY.         // for video texture. for CYUV, do 1-v for every vertex by S/W Driver.
      17:         YUY2:
      18:         DXT1.         // for texture compression
      19:         DXT2.
      20:         DXT3.
      21:         DXT4.
      22:         DXT5.
      23:         R16F.         // may be tiled.
      24:         DXT7.
      25:         RGB332.
      26:         R32F.         // may be tiled.
      27:         A2R10G10B10
      28:         A2W10V10U10
      29:         W11V11U10
      30:         XLVU8888.     // 32bit, may be tiled.
      31:         QWVU8888.     // 32bit, may be tiled.
```

```
} M2TextureFormat
D4 [10:0]       Texture_Pitch, Integer. //   (unit: 128bit).     default = 0.

D5 [31:29]      Vertical_Slices_Exponent, Integer.          // N for texture slice/face group.
D5 [28:26]      Horizontal_Slices_Exponent, Integer.        // M for texture slice/face group.
      // M and N set for (2^M)*(2^N) maps grouping if it is a 3D texture / Cube map.

D5 [25:24]      High_Order_Filtering_Mode, Enumerate.
      {
            0:    Disable
            1:    UVOffset
            2:    StagedFiltering      // load weight table directly.
            3:    PreciseFiltering     // Use weight map in the same pair.
      } HighOrderFilterMode
      // by default, last two modes will enable UV offset too. Please set proper offset.
      // please set left & right pipe same setting for mode 3.

D5 [23]         Texture_Pair_Shared, Boolean.
D5 [22:20]      Depth_Texture_Test_Function,   Enumerate, VisibilityTestFunctionCodes.
D5 [19:18]      UndoGamma_Correction_Enable, Enumerate.
      {
            0:          disable.
            2:          Enable_For_RGB.
            3:          Enable_For_Blue_Only.
      } TextureUndoGammaMode
D5 [17]         Texel_Clamp_Enable, Boolean.  // clamp after filtering.
{
      0:    Disable.              // default.
      1:    Enable.
}

D5 [16]         Swap_Red_Blue, Boolean.         // Red <--> Blue after filtering for ABGR textures.
{
      0:    Keep unchanged.       // default.
      1:    Swap.
}
D5 [15:08]      Constant_Texture_Alpha, Integer.      // for textures without alpha. default = 0xff.
D5 [7]          YUV420_Enable, Boolean.
D5 [6]          Texel_Key_Polarity, Enumerate.
{
      0:          Normal.               // default.
      1:          Invert.
} TexelKeyPolarity

D5 [05:04]      Texel_Key_Color_Selection, Enumerate.
{
      0:          Disable.              // default.
      1:          TexelColorKey0.
      2:          TexelColorKey1.
      3:          TexelColorKey2.
} TexelColorKeySelection

D5 [03:02]      Texel_Key_Mode, Enumerate.
{
      0:          Microsoft_Standard.     // default
```

```
      1:          Nearest_Key.
      2:          Read_Destination_Key.       // second Microsoft key method.
} TexelKeyMode

D5 [1]          Constant_Alpha_Replace, Boolean.
{
      0:          disable.          // default.
      1:          enable to replace alpha in texel by a constant alpha.
}
D5 [0]          Texel_Bits_Swap, Boolean. // in a byte.
{
      0:    disable.          // default.
      1:    enable.           // Do it for each byte if the flag is set
}
      // for 1bpp:
      //     bit7 <--> bit0, bit6 <--> bit1, bit5 <--> bit2, bit4 <--> bit3
      // for 2bpp:
      //     bit[7:6] <--> bit[1:0], it means bit7 <--> bit1, bit6 <--> bit0, the same as below.
      //     bit[5:4] <--> bit[3:2].
      // for 4bpp:
      // bit[7:4] <--> bit[3:0].
      // (bit7 <--> bit3, bit6 <--> bit2, bit5 <--> bit1, bit4 <--> bit0).

D6 [31:16]    DeltaV, FP9.7.            // for UV coordinate adjustment in filter loop.
D6 [15:0]     DeltaU, FP9.7.            // no adjustment for third dimension.
D7 [31:24]    Border_Color_Alpha, Integer.
D7 [23:16]    Border_Color_Red, Integer.
D7 [15:08]    Border_Color_Green, Integer.
D7 [07:00]    Border_Color_Blue, Integer.
```

Texture Base Address Array = 208 x 30bit.

Note for programming

For a regular texture, it is illegal to use Addressing_None. Addressing_None should be only applied to A2D and Texcoord (in XP4). Because A2D has UV pre-multiplied by size before going to graphics engine and texture size is set to one. In this case texture coordinate is well adjusted so valid pixel will not access to garbage texel.

Just for validating logic, you may extend your map area to some predefined values, so "garbage" will be your defined values to see if the Addressing_None works properly even not for A2D.

In XP10, Texcoord mode will not do any addressing mode (even Addressing_None).

When you set Border_Texel mode, your texture should be added border texel in related dimension. For example, in your case, if UBorderMode = Border_Texel, your real texture should be 10x8 instead of 8x8, but you still set texture size = 8x8, left one texel and right one texel are border texel you added. The base address is pointed to first texel of 10x8 map. So after border mode, all texels will be accessed in valid texel range instead of garbage. If both U and V are Border_Texel, the size will be 10x10.

7.3. Palette Table Support

Register
0x330 (W) : 0xCC PALETTE_LOAD
D0 [8] Activate_Palette_Table, Boolean. // default = inactive.
D0 [7:6] PortNo, Enumerate. // default = PCI-E.
{
0: PCI-E.
1: Z_Port.
2: C_Port.
} MemoryPortSelection
D0 [5:0] Palette_Table_Size, Integer. // (unit: 128bit). default = 0.
D1 [27:0] Memory_Base_Address, Integer. // (28bit, unit: 128bit) even if active0 = 0.

7.4. Texel key color

Register
0x340 (W) : 0xD0 TEXEL_KEY_01 // #0 Texel Key High: default = 0x000000
D0 [23:16] Texel_Key_High0_Red, Integer.
D0 [15:08] Texel_Key_High0_Green, Integer.
D0 [07:00] Texel_Key_High0_Blue, Integer.

D1 [23:16] Texel_Key_Low0_Red, Integer.
D1 [15:08] Texel_Key_Low0_Green, Integer.
D1 [07:00] Texel_Key_Low0_Blue, Integer.

D2 [23:16] Texel_Key_High1_Red, Integer.
D2 [15:08] Texel_Key_High1_Green, Integer.
D2 [07:00] Texel_Key_High1_Blue, Integer.

D3 [23:16] Texel_Key_Low1_Red, Integer.
D3 [15:08] Texel_Key_Low1_Green, Integer.
D3 [07:00] Texel_Key_Low1_Blue, Integer.

Register
0x350 (W) : 0xD4 TEXEL_KEY_2 // #2 Texel Key Hi: default = 0x000000
D0 [23:16] Texel_Key_High2_Red, Integer.
D0 [15:08] Texel_Key_High2_Green, Integer.
D0 [07:00] Texel_Key_High2_Blue, Integer.

D1 [23:16] Texel_Key_Low2_Red, Integer.
D1 [15:08] Texel_Key_Low2_Green, Integer.
D1 [07:00] Texel_Key_Low2_Blue, Integer.

8. Pixel Shader

8.1. Pixel Shader Setting

Register
0x380 (R/W) : 0xE0 PixelShader_SETTING

```
D0 [23:08]      Texel_Key_Mask_Operation_Code, Integer.     // 1bit/texture,
// 1 means final key mask |= the texture key mask.
//            Initial final key mask for each tile = 0;
//            Pass the final key mask together with pixel shader's final result.
//            Do it only in last (base / color) loop.
//            D[08] for texture0.
//            D[09] for texture1.
//            ...
//            D[23] for texture15.
//            Here texture number is just input texture order Number in last loop.

D0 [7]          Pixel_Shader_Version, Enumerate.
      {
            0:    Undefined. // not used in XP10
            1:    PixelShader20.
      } PixelShaderVersion
D0 [6]          Shadow_Working_Disable, ReversedBoolean.
      {
            0:    Enable.   // 8 or 16 input samplings mode, default
            1:    Disable.   // 32 input samplings mode. Low performance
                              // please do not set Invisible pixel drop enable in 0x300 D0[15].
                              // and only one tile is allowed for bump loop.
      }
D0 [5:4]        Shadow_Working_Mode, Enumerate.
      {
            0:    8_Texture_Input_Mode.
                  // 8 texture inputs mode, can set invisible pixel pair drop for single loop.
            1:    16_Texture_Input_Mode.
                  // 16 texture input mode, can not set invisible pixel pair drop any way.
            2:    32_Buffer_Working_Only.
                  // 32 buffer working only, can not set invisible pixel pair drop.
                  // Will require texture load table.
      } PixelShader2Mode

D0 [3]          Partial_Precision_PS2_Enable, Boolean.

D0 [2:0]        Float_Color_Buffer_Format, Enumerate.
      {
            0:    ASIS.          // no float buffer
            1:    L16.           // 16bits integer buffer
            2:    G16R16.        // 32bits integer buffer
            3:    R16F.
```

4: G16R16F.
5: R32F.
6: V16U16.
} FloatcolorBufferformat

D1 [06:00] Loop0_PSInstruction_Entry, Integer.
D1 [14:08] Loop1_PSInstruction_Entry, Integer.
D1 [22:16] Loop2_PSInstruction_Entry, Integer.
D1 [30:24] Loop3_PSInstruction_Entry, Integer.
D2 [06:00] Loop4_PSInstruction_Entry, Integer.
D2 [14:08] Loop5_PSInstruction_Entry, Integer.
D2 [22:16] Loop6_PSInstruction_Entry, Integer.
D2 [30:24] Loop7_PSInstruction_Entry, Integer.

// For performance tuning and bypass source selection,
// only affect in last loop.
D3 [02:00] Diffuse_RGB_Resource, Enumerate. // to bypass to Post Blender.
{
0x0: Diffuse_RGB.
0x1: Specular_RGB.
0x4: Texture0_RGB.
0x5: Texture1_RGB.
0x6: Texture2_RGB.
0x7: Texture3_RGB.
} DiffuseRGBSource
D3 [06:04] Diffuse_Alpha_Resource, Enumerate.// to bypass to Post Blender.
{
0x0: Diffuse_Alpha.
0x1: Specular_Alpha.
0x4: Texture0_Alpha.
0x5: Texture1_Alpha.
0x6: Texture2_Alpha.
0x7: Texture3_Alpha.
} DiffuseAlphaSource
D3 [11:08] Specular_RGB_Resource, Enumerate. // to bypass to Post Blender.
{
0x0: Diffuse_RGB.
0x1: Specular_RGB.
0x2: PixelShader_Output.
0x4: Texture0_RGB.
0x5: Texture1_RGB.
0x6: Texture2_RGB.
0x7: Texture3_RGB.
} SpecularRGBSource
D3 [12] Fog_Resource, Enumerate. // bypass to Post Blender.
{
0: Fog_Bypass.
1: PixelShader_Output_Specular_Alpha.
} FogSource
D3 [31:13] Reserved.

Note
PS will split the output to Post Blender to 3 channels:
- Result MRT_ID EOTile:

- Specular: could be PS result or bypass, diffuse and tile Information go with it.
- Fog: could be PS result or bypass

PBE will do sync of the 3 outputs, It will hold one tile of specular and fog for all 4 result.

So there must be some limitation

- There must be **one and only one** write of specular or fog in the last loop. This means if the specular is not used in PBE, Global register must set specular and fog are bypassed. Otherwise we consider this is an error pattern.
- If the specular and fog are from PS output, It must be write **before or with** result. If not so, PBE have to hold 4 results to avoid block PS, the size is too big.
- Since Result 0 is always written first in the last loop, so if the specular or fog is bypassed, it will write bypassed data out when Result 0 is written, as rule 1 required.
- If there are multiple targets, do not enable invisible pixel pair drop. Otherwise PS needs 4x FIFO to do reorder.
- Finish one target's all components then next target.
- If Specular(RGB) and Fog(A) are from PS output and is not written in the same instruction, set release flag at the last write instruction, do not set twice, otherwise hardware will be think as two tile released.

8.2. PS2.0 Programming Note

For performance, each instruction will do scale and clamp before writing its result out to save instructions. Also we will have few FPUs like CND/CMP/LRP etc for PS only.

Performance and functionality coverage

PS2 has 32 Tile buffer, support 3 mode

- Mode 1 (simple case): 8 texture Shadow/Working mode with invisible pair drop. In this mode, **there is no bump loop**, 3x8 texture buffer used as texture buffer and **8 buffers as temp buffers**. 8 working texture inputs are used as temps too.
- Mode 2 (main case used in games): 16 texture Shadow/Working mode, **suppose be used in bump loop**. Cannot have invisible pair drop. All texture sampling and coordinate mode are mapped to temp registers in driver.
- Mode 3 (extreme DCT case): 32 texture working only mode, no invisible pair drops. **Need a texture load table to keep persistent information, which also means only one tile in each loop**. Driver will map first 16 temps to texture inputs (re-writeable) and last 16 temps as temps in instruction. So we do not need to change instruction definition. The load table is defined as below: 64 entries * 5bits, each entry is used for one texture input. Use texture input sequence number (**accumulated across** loops, reset for each xy tile) as index to table to get texture input (loading) destination (to one of 16 texture inputs (0~15) and 16 temps (16~31)). Input buffer[index] = texture input[j], index = LoadTable[j]. Texture Load Table is loaded in by RE_LOAD command. No PP mode.
- If the two textures are paired together & both are all valid, with auto counter texture ID, they are put in successive buffer. If we use LoadTable, we also add limitation that table data follow the same behavior: two textures loaded to the successive buffer.

- Texture Load support texkill instruction, in this case, **not texture load in texture buffer but save the mask in mask buffer**. In addition, all key masks from texture filtering are accumulated into mask buffer. So all texture buffer are fully used to meet DCT requirement.
- No texture buffer, texture buffer are temp buffer. (if mode 1 is not easy to implement, add 4 extra temp buffer to support 12 temp in PS2)., this will save 16 Tile buffer(100k)
- Remove persistent temp mechanism.
- Add a Mask buffer for all tiles in the loop. (now 32x16 bit) to accumulate texkill mask. (only for PS2.0)
- PP mode, one1.8.23 full precision FPU and one 1.5.10 partial FPU. Truncate input data to 16 bit and accumulate 2 tile if possible to run PS instructions. Expand to full precision and two pixel throughput when output. Input texture coordinate, output to 4D bump & Depth are all full precision.
- Keep color input 1.6.15 precision. Expanded to 1.8.23 for full precision FPU.
- PS2.0 constants = 1.8.23 IFF precision.
- PS2.0 do not check global 0x380 D0 [23:08] Texel_Key_Mask_Operation_Code and always accumulate key/kill mask from texture filter.

Bypass Implementation

Full precision case, do bypass for all three modes:

last loop: texture inputs --> 1.8.23 IFF --> texture input ram or temp ram.
after last instruction, bypass related textures from input ram or
temp ram to output FIFO with 1.8.23 IFF to fix 10.2 conversion.

PP mode, do bypass in the same way as full precision for all three modes:

last loop: texture inputs --> 1.5.10 IFF --> texture input ram or temp ram.
after last instruction, bypass related textures from input ram or
temp ram to output FIFO with 1.5.10 IFF to fix 10.2 conversion.

only 1 bit precision issue competing with post blending 12bit precision in some case.
it is not an issue because it is for performance and will not cause any artifacts.

Driver can make sure not overwrite original texture inputs for correct result.

Programming examples

Example#0, if bump loop tiles is 16~32 and shader is simple, every loop has few input textures and temps used. Zero or one temp can be looped up and down to be used for next loop. All 8 input textures and 8 temps are working temps for multiple loops.

Example#1, if bump loop tiles is 16~32 and shader is a little big, every loop has 9~16 input textures and temps used. Zero or one temp can be looped up and down to be used for next loop. All 16 temps are working temps for multiple loops, 16 inputs for shadowing temps. No persistent temps.

Example#2, if it is still hard to be programmed as example#1, driver can set functionality mode to use all 16 input textures as working with very low performance.

- Bump tiles = 1.
- 16 samples & 8 texture coordinates into 24 temps loaded by loading table.
- All temps will be persistent until it is overloaded.

In working only mode (last example), HW only resets input texture count = 0 for first loop. So input texture ID will be increased loop by loop. While shadow / working mode, input texture ID will be reset for every loop for the same XY tile.

8.3. PS2.0 Instruction

An instruction is defined as below // 90bit

- QW[04:00] ---- OpCode = Type (5bit)
- QW[05:05] ---- Last instruction (1bit)
- QW[07:06] ---- Input Source Component Valid (1bit for Z, 1bit for w, 0 means use Zero)
- QW[11:08] ---- Exponent of scale factor, signed 4bit from -8 ~ +7.
- QW[12:12] ---- Clamp to [0, 1.0] after scaling, 0 means no clamp.
- QW[13:13] ---- Source0 Negative flag (1bit)
- QW[14:14] ---- Source1 Negative flag (1bit)
- QW[15:15] ---- Source2 Negative flag (1bit)
- QW[31:16] ---- Source0 definition
- QW[47:32] ---- Source1 definition
- QW[63:48] ---- Source2 definition
- QW[73:64] ---- Destination0 (10bit)
- QW[83:74] ---- Destination1 (10bit), for second MOVE only.
- QW[84:84] ---- Second MOVE enable.
- DW[85:85] ---- Clamp for destination1 (second). (also apply for second MOV)
- DW[89:86] ---- Scale factor for destination1. (also apply for second MOV)

Two destinations ID may be the same, use write enables to merge to one write. If write enables are overlapped also, destination1 will overwrite destination0. Destination1 can be to any destination tiles except for input textures.

The MOVE in PS2.0 is from source#2 to destination#1 parallel with FPU of less than or equal to two operands. ZW channels are always valid to the MOVE.

The definition of DDP4 (= 0x1C), DMUL(=0x1D), DADD(=0x1E) is as following:

Operation: dest0 = DP4(src0, src1), dest1 = DP4(src0, src2).

The same for DMUL and DADD. Second DP4 and MUL can share second MUL in LERP. Apply input source component valid (DW[7:6]) for source2 also except for second MOV.

Operation Code is defined as follows

```
// zero operand.
MTL_NOP         = 0x0       // = D3DSIO_NOP.

// one operand.
MTL_MOV         = 0x01      // = D3DSIO_MOV
MTL_RCP         = 0x06      // = D3DSIO_RCP
MTL_RSQ         = 0x07      // = D3DSIO_RSQ
MTL_EXP         = 0x0E      //.
MTL_LOG         = 0x0F      //.
MTL_FRC         = 0x13      // = D3DSIO_FRC
```

```
MTL_SGN         = 0x16
MTL_SIN         = 0x18
MTL_COS         = 0x19

// two operands
MTL_ADD         = 0x02      // = D3DSIO_ADD
MTL_MUL         = 0x05      // = D3DSIO_MUL
MTL_DP4         = 0x09      // = D3DSIO_DP4
MTL_MIN         = 0x0A      // = D3DSIO_MIN
MTL_MAX         = 0x0B      // = D3DSIO_MAX
MTL_SLT         = 0x0C      // = D3DSIO_SLT, return TRUE/FALSE to local controller too.
MTL_SGE         = 0x0D      // = D3DSIO_SGE, return TRUE/FALSE to local controller too
MTL_DST         = 0x11
MTL_FRCMUL      = 0x03
MTL_EXPMUL      = 0x14
MTL_LOGMUL      = 0x15
MTL_RCPMUL      = 0x10
MTL_RSQMUL      = 0x17

// three operands
MTL_MAD         = 0x04
MTL_LERP        = 0x12
// src0*src1 + (1-src0)*src2, but HW do: (1-src0)*src1 + src0*src2.
// So driver needs to swap src1 and src2.
MTL_CND         = 0x1A
MTL_CMP         = 0x1B
MTL_REFV        = 0x08
MTL_DDP4        = 0x1C
MTL_DMUL        = 0x1D
MTL_DADD        = 0x1E

Following are done by few atomic instructions.
MTL_DP3         // = MTL_DP4 with component valid & write mask out.
MTL_POW         // = MTL_LOGMUL + MTL_EXP
MTL_CRS         // = MTL_MUL + MTL_MAD
MTL_NRM         // = MTL_DP3 + MTL_RSQMUL.
```

Source is defined as follows

- WORD[06:00] ---- Source ID (7bit)
- WORD[07:07] ---- ABS flag (1bit)
- WORD[15:08] ---- Swizzle code (8bit, 2bit/component), from High to low: ABGR

Destination is defined as follows

- QW[68:64] ---- Destination ID (5bit)
- QW[69:69] ---- Ready to release result tile for next block (1bit)
- QW[73:70] ---- Write Enable (4bit, 1bit/component), from High to low: RGBA

PS2.0 Input source ID:

- Texture: 0x0 ~ 0xF,
  // only first 8 input textures are re-writeable as more temporary result.
- Temp: 0x10 ~ 0x1F.
- Constant: 0x20 ~ 0x3F.
- Constant#0: 0x40: (0.0, 0.5, 1.0, 2.0)
- Constant#1: 0x41: (1/3.0, 3.0, 0.25, 4.0)

- Constant#2: 0x42: (0.2, 5.0, 1/6.0, 6.0)
- Constant#3: 0x43: (1/7.0, 7.0, 0.125, 8.0).
- Diffuse: 0x44.
- Specular: 0x45.

PS2.0 Output Destination ID:

- Result: 0x4 ~ 0x7
  // for four render target.
  // lowest 2bit ID should be carried as tile tag to post blender.
- 2D Bump: 0x0.
- 4D Bump: 0x1.
- Depth: 0x2.
- Specular: 0x3.
- Texture: 0x8 ~ 0xF. // first 8 input textures are re-writeable as more temporary result
- Temp: 0x10 ~ 0x1F.

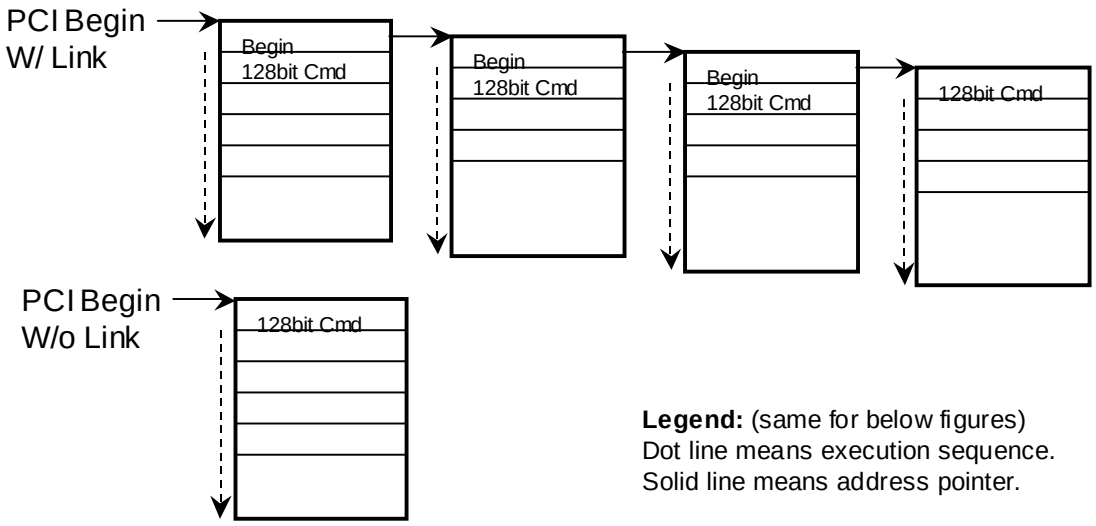
The multiple render targets will be output in the order: 0~3. The DX spec also has the constraint to output MRTs components in not random way: Finish one target's all components then next target.

9. Programming of Command List

We have four types of BEGINs: Begin2D, Begin3D, BeginFlip and BeginMaster. First Begin which can be any one of four Begins is sent via PCI bus by CPU directly and always valid and no STOP. Use Begin ID to distribute commands in the command list to ID indicated Engine: 2D, 3D, Flip or Master.

9.1. Three Command List Fetch Commands

First BEGIN goes through PCI bus. It may enable to link next list (BEGIN). The embedded BEGIN may also enable link next list continuously. See demonstrations in Figures.



If a Begin disables link next list, there is no embedded Begin at the beginning of the next list, for instance, last list in above figure.

If a Begin is invalid, the list pointed by the Begin is not ready yet.

If a Begin has a Hold set, it means after the current list fetching is finished, needs to hold / wait until the list is back and executed or gone out from Master engine to 3D/2D/Flip etc. Then starts fetch the list pointed by the embedded Begin. PCI Begin should not have a Hold set.

9.2. PCI Trigger Mode to Link Next List

There are three auto-link modes to link command lists together. Here we describe one of them (so called PCI trigger mode) first.
Due to that S/W processes drawing batches ahead of H/W execution for a while and S/W is hard to know where H/W is currently executing, it causes miscommunication between S/W and H/W or overhead of checking H/W status. One solution to this issue is:

S/W Side:

- S/W prepares first list (List#0) in PCI-E, put an invalid embedded Begin (Begin #1) always at the beginning of List#0. Send a PCI Begin (Begin#0) after the list preparation is finished.
- S/W continues to prepare second list (List#1) with another invalid embedded Begin (Begin #2) first. After List#1 preparation is finished, update correct info in Begin #1 and set it back to valid. Send a PCI trigger command (Trigger#1) via PCI bus.
- S/W continues to prepare second list (List#2) with another invalid embedded Begin (Begin #3) first. After List#2 preparation is finished, update correct info in Begin #2 and set it back to valid. Send a PCI trigger command (Trigger#2) via PCI bus.
- Repeat last step for continuous command lists (Begin #4, List#3, Trigger#3), …, (Begin [N], List[N-1], Trigger[N-1]).
- If List[N-1] is last list, S/W should not put embedded Begin [N] into the beginning of List[N-1] and disable link in Begin [N-1].

H/W Side:

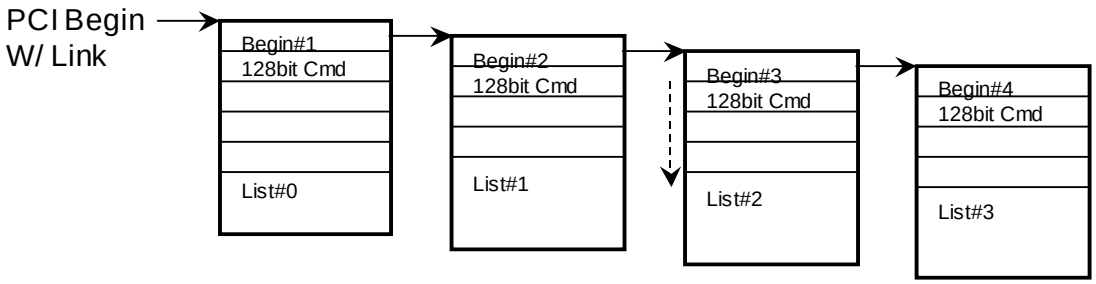
- There are two 16bit counters: PCI Trigger counter and PCI-E execute counter, initialize both to zero.
- In PCI command decoder side, if a PCI trigger command comes, PCI trigger counter increases by one.
- In PCI-E command list fetcher side, an embedded Begin is decoded, PCI-E execute counter increases by one too.
- If an embedded Begin [M] is decoded, Begin [M] is valid, directly execute it after List[M-1] fetching is finished.
- Else
  {
  - If Begin [M] is invalid, check if (PCI Trigger counter >= PCI-E Executor counter);
  - If No, Hold PCI-E fetching until (PCI Trigger counter >= PCI-E Executor counter)
  - Re-fetch Begin [M] from PCI-E;
  - Wait it back and decode & execute it; // not re-count PCI-E executor counter.

  }
- If Begin [N-1] disables link, that means no Begin[N], after List[N-1] fetching is finished, H/W goes back to wait/decode PCI commands.

To avoid counters overflow, H/W does following protections:

- When two counters are equal, immediately set both to zero.
- When PCI Trigger counter > PCI-E executor counter, do subtraction when no new PCI trigger command comes, PCI trigger counter -= PCI-E executor counter and set PCI-E executor = 0.
- S/W reset two counters when H/W runs out of order.

Example#1 Begin#1 is invalid and Begin#2~M are valid when H/W receives/decodes them.



At HW decoding Begin#1
Begin#1 is mostly invalid because HW receives PCI Begin and fetch List#0. At the moment of fetching request comes to Memory side, S/W had not finished List#1 yet. But when Begin#1 is back and being decoded, S/W may be preparing List#2 because List#2 ~ 3 are very simple/short. The PCI trigger counter = 1. So H/W will re-fetch Begin#1 immediately after List#0 fetching is finished. After that, H/W receives valid Begin#2, …, Begin[M].

But Somehow, Begin[M+1] may be invalid and PCI trigger counter = M-1 at decoding Begin[M+1] because S/W took long time for List[M+1]. after List[M] fetching is finished, H/W will wait until PCI trigger counter = M+1. Then re-fech Begin[M+1] and wait it back and decode/execute it.

Example#2
PCI Begin#0 takes long time to come HW decoder. When Begin#1 is fetched back and decoded, Begin#1 is valid. S/W is preparing List#3 and two PCI trigger commands arrived and PCI trigger counter = 2. No matter what the counter is equal to, HW directly execute Begin#1.

Example#3
Begin[X] is invalid. At its decoding time, S/W is preparing List[X+4] and already sent X+3 PCI trigger commands. But PCI trigger counter still = X-2 due to PCI bus busy or some else reason. H/W will wait also until PCI trigger count >= X.

9.3. Two Interrupt Modes to Link Command List

9.3.1. Invalid List Interrupt Mode

Similar to PCI trigger mode, S/W driver will prepare a series of command lists except for sending PCI trigger command per list finishing. Set invalid Begin [N+1] at the beginning of List[N], when List[N+1] is finished, re-write Begin [N+1] to valid.

HW side, if receiving an invalid Begin, after the current list fetching is finished and data is back, Send a interrupt signal to S/W. After S/W receives an Interrupt, Driver read Interrupt register and command list identification to identify which invalid list HW got. Then resend the Begin via PCI.

9.3.2. Engine Idle Interrupt Mode

It is the same as XP4/5 Engine Idle Interrupt mode. S/W prepares a command list with link enable (= embedded Begin). At the end of the list, put a Engine Idle Interrupt enable (At the beginning, it is disabled). Then HW finishes the fetching the list and decode the Interrupt enable, when the engine idle, HW will send an interrupt signal to S/W. S/W get the signal then sends next PCI Begin for next list.

But in XP10, the Engine Idle Interrupt Enable/Disable register belongs to Master engine group. So the Enable/disable should be in a Master Engine's Begin. To do so, Create a BeginMA and a short list to disable Engine Idle Interrupt, link to your "real" command list that links to another BeginMA to Enable the Engine Idle Interrupt. When all the three lists are finished, send a PCI BeginMA (first one).

9.4. Command List Interrupt

For multi-thread OS or apps, a graphics HW is time-shared. OS will send Interrupt to driver and let driver stop HW and switch to next thread or app. The programming for this issue in XP10 is as below:

S/W driver prepares APP#1's list, List#0, List1, …, ListM in one of auto-link mode. For example, S/W is in the middle of List#4, receives an OS interrupt. At that time, HW is running List#0 and Begin #1 is pushed into fetch Fifo. S/W will do the followings:

- Read HW status to see where HW is running to. The return position is in the middle of List0.
- Set Stop flag into Begin #1 ~ 4.
- Save condition & situation for APP#1.
- Do jobs for APP#2.

H/W will continue running until List#1 is finished. Begin #2 has a Stop flag, so HW will stop fetch List#2 and after List#1 is fetched, send a "Stop" interrupt to S/W.

When S/W gets "Stop" interrupt from HW, S/W may be doing List#2 for APP#2. S/W handles the interrupt as below:

- Read status to identify which Begin is stopped.
- Save situation for APP#1.
- Remove "Stop" flags from Begin #1 ~ 4.
- If APP#2's List#0 is ready, send a PCI Begin for List#0 of APP#2.
- Continue prepare Lists for APP#2.

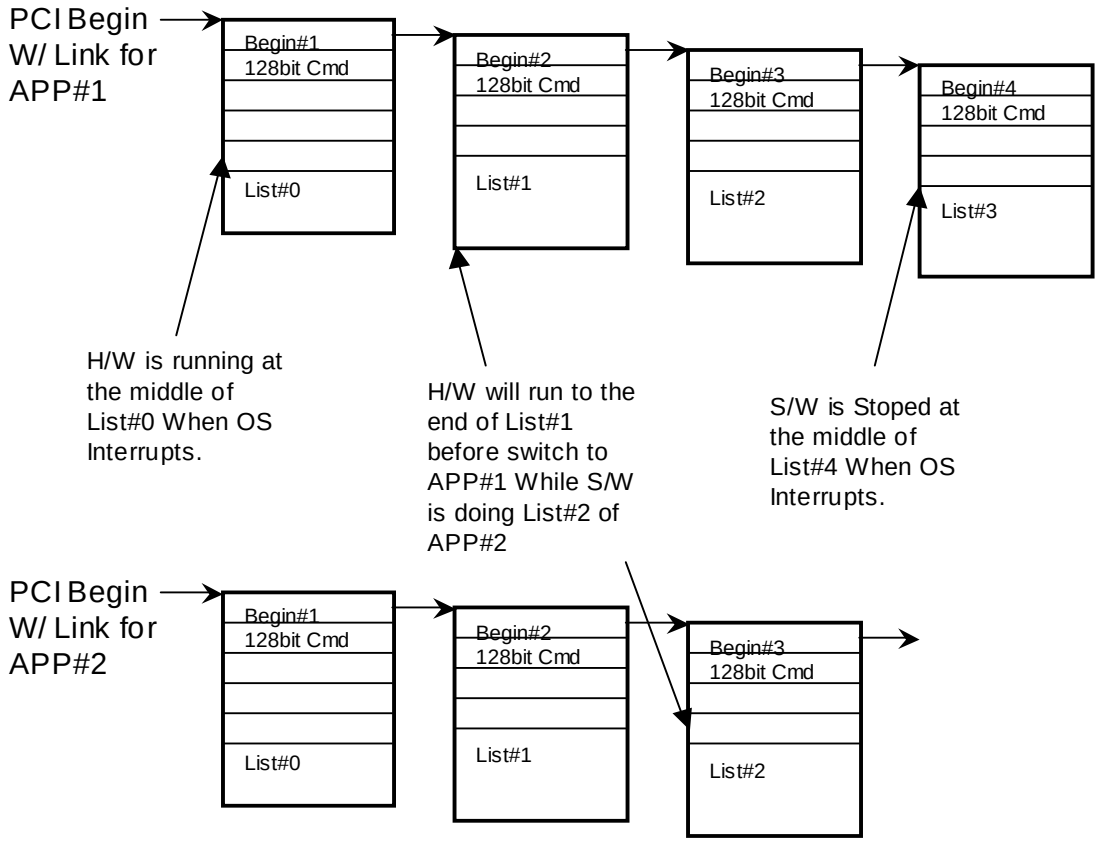
….

Sometime later, OS sends an restoring APP#1 interrupt to driver. S/W will do the followings:

- Save condition & situation for APP#N.
- Retrieve info from saved memory to know APP#1 is stopped at Begin #2. Due to some global settings that List#2 ~ need may be in List0~1, so Driver can set "Drop/Ignore" flag into Begin #0~1.
- Prepares List#4 ~.

- After HW “Stop” interrupt comes, saves situations for APP#N.
- Send a PCI Begin#0.
- Continue prepare Lists for APP#1.

….



10.Programming of Clipping

10.1. Geometry Clipping Window Setting

Geometry Clipping is done in CullClip Engine, including W Clipping and 3D Clipping.

- **W Clipping**: if three positions are all out of **w-plane**, reject; if part of them are out but some of them are in, do w-clipping; if all of them are in w-plane, accept.

- **3D Clipping**: if all three positions are out of **view-port**, reject; if part of them are out of **guard-band** but some of them are in guard-band, do 3D clipping; otherwise, accept.

- W-plane is defined by global register setting W_Clipping_Enable and W_Threshold, W_Clipping_Enable is valid only when 3D clipping is enabled.

- 3D Clipping decision is made base on View-port Clip and Guard-band Clip setting, but the 3D Clipping is done base on Guard-band Scale setting.

- If 3D_Clipping_Enable = 1, set Frustum_Guardband_ClipCode_Disable = 0.
  Pseudo code:
  ```
  {
      GP3DClipEnable = GetValue("3D_Clipping_Enable");
      if(GP3DClipEnable)
          frustumClipDisable = 0;
      SetValue("Frustum_GuardBand_ClipCode_Disable", frustumClipDisable);
  }
  ```

- Set Disable_PA_Fast_Rejection (0x130, D1[9]) to 1 when you set 3D_CLIPPING_ENABLE to 0 (0x130 D1[0]); If Point_mode = 1, set Disable_PA_Fast_Rejection = 0.
  Pseudo code:
  ```
  {
      GP3DClipEnable = GetValue("3D_Clipping_Enable");
      pointMode = GetValue("Point_Mode");
      if(GP3DClipEnable == 0)
          FastPARejectionDisable = 1;
      if(pointMode == 1)
          FastPARejectionDisable = 0;
      SetValue("Disable_PA_Fast_Rejection", FastPARejectionDisable);
  }
  ```

- View-port clip setting:
  Viewport _CLIP_LEFT,
  Viewport _CLIP_RIGHT,
  Viewport _CLIP_TOP,
  Viewport _CLIP_BOTTOM,
  Viewport _CLIP_NEAR,
  Viewport _CLIP_FAR.

- Guard-band clip setting:
  Guardband _CLIP_LEFT,
  Guardband _CLIP_RIGHT,
  Guardband _CLIP_TOP,
  Guardband _CLIP_BOTTOM,
  Guardband _CLIP_NEAR,
  Guardband _CLIP_FAR.

- Guard-band scale setting:
  X_Scale_Guardband_Left,
  X_Scale_Guardband_Right,
  Y_Scale_Guardband_Top,
  Y_Scale_Guardband_Bottom,
  Zmin_Scale_Guardband,
  Zmax_Scale_Guardband.

- If W_Clipping_Enable = 0, should set W_Threshold = 0x40000000 (0.0f).
  Pseudo code:

```
{
    wClip = GetValue("W_Clipping_Enable");
    wThreshold = GetValue("W_Threshold");
    if(!wClip)
        ASSERT(wThreshold == 0x40000000);
}
```

10.2. Screen Clipping Window Setting

If triangle is out of clip window,

1) TopY >= Window_Clip_Bottom
2) BottomY <= Window_Clip_Top
3) LeftX >= Window_Clip_Right
4) RightX <= Window_Clip_Left

it will be dropped in Viewport stage.
Register definitions:

0x1b0	[15 : 0]	M2REG_WINDOW_CLIP_LEFT
0x1b4	[15 : 0]	M2REG_WINDOW_CLIP_RIGHT
0x1b8	[15 : 0]	M2REG_WINDOW_CLIP_TOP
0x1bc	[15 : 0]	M2REG_WINDOW_CLIP_BOTTOM

Another group of clip ranges is used in RasterEngine, and it will affect the scan line's start position and end position. All visible primitives are rendered in this clip range. Register definitions:

0x214	[27 : 16]	M2REG_X_CLIP_RIGHT
0x214	[11 : 0]	M2REG_X_CLIP_LEFT
0x218	[27 : 16]	M2REG_Y_CLIP_BOTTOM
0x218	[11 : 0]	M2REG_Y_CLIP_TOP

- When debugging, it is found that, if the clip window is changed only in RE, GP may send down some triangles which are totally outside of the clip window and don't follow the RE's design rule, which means RE can't handle these triangles according to our design rule. So when you need to set clip window to generate debug patterns, please make sure you set the RE's clip window & VP's window the same value.

10.3. Depth Clipping Window Setting

Depth Clipping is done in Depth Engine if depth clip is enabled, if one of following is meet, the pixel will be masked:

1) z > Z_BIASED_CLIP_FAR
2) z < Z_BIASED_CLIP_NEAR

Register definitions:

0x220	[0 : 0]	M2REG_DEPTH_CLIP_ENABLE
0x238	[31 : 0]	M2REG_Z_BIASED_CLIP_FAR
0x23c	[31 : 0]	M2REG_Z_BIASED_CLIP_NEAR

11.Programming of Page-base Rendering

- Page Size is defined by global register “Page_Size_Mode” (0x270 D1 [19 : 18]) , and this register setting is restricted by MRT targets.
  MRT = 0, Page_Size_Mode = 0 (16x16)
  MRT = 1, Page_Size_Mode = 1 (16x8)
  MRT = 2, Page_Size_Mode = 2 (16x4)
  MRT = 3, Page_Size_Mode = 2 (16x4)
  Pseudo code:

```
{
    totalTargets = GetValue(“Total_Targets_Elements”);
    if(totalTargets == 0)
        pageSizeMode = 0;
    else if (totalTargets == 1)
        pageSizeMode = 1;
    else if (totalTargets >= 2)
        pageSizeMode = 2;
    SetValue(“Page_Size_Mode”, pageSizeMode);
}
```

- Parameter Bank Configuration is determined by UVPair number,
  0~2 tex, Parameter_Bank_Configuration = 0 (32 primitives)
  3~4 tex, Parameter_Bank_Configuration = 1 (24 primitives)
  5~8 tex, Parameter_Bank_Configuration = 2 (14 primitives)
  Register definition:

```
    0x210   D1 [29:28]   Parameter_Bank_Configuration, Enumerate.
    {
        0:    32_Primitives,   // 2 colors + 2 Tex + fog. (0~2 tex)
        1:    24_Primitives,   // 2 colors + 4 Tex + fog. (3~4 tex)
        2:    14_Primitives,   // 2 colors + 8 Tex + fog. (5~8 tex)
    } ParameterBankConfig
```

  Pseudo code:

```
{
    gpUVPairs = GetValue(“GP_Total_Texture_Coordinate_Pairs”);
    if(gpUVPairs >= 0 && gpUVPairs <= 2)
        paramBankConf = 0;
    else if(gpUVPairs >=3 && gpUVPairs <=4)
        paramBankConf = 1;
    else if(gpUVPairs >=5 && gpUVPairs <=8)
        paramBankConf = 2;
    SetValue(“Parameter_Bank_Configuration”, paramBankConf);
}
```

- There are three easy confusable global registers:
  Page Max Primitive Threshold, for page sorting ram to trigger flush, Real value=setting+1
  Page Min Primitive Threshold, for page sorting to stop flush, Real value = setting;
  Max Page Threshold, reach max page in sort ram then output, Real value is like following:
  Max_Page_Threshold = 0, Real value = 96 (max)

Max_Page_Threshold = 1 ~ 12, Real value = 8 * Max_Page_Threshold
Max_Page_Threshold > 12, Real value = 96 (max)
And, Page Max Primitive Threshold is restricted by ParameterBankConfig, Page Min Primitive Threshold is restricted by Page Max Primitive Threshold.

- Page_Max_Prim_Threshold restricted by ParameterBankConfig
  Parameter_Bank_Configuration = 0 => Page_Max_Prim_Threshold = 16 (32 primitives)
  Parameter_Bank_Configuration = 1 => Page_Max_Prim_Threshold = 12 (24 primitives)
  Parameter_Bank_Configuration = 2 => Page_Max_Prim_Threshold = 7 (14 primitives)

- If Page_Min_Prim_Threshold = 0, Page_Max_Prim_Threshold should >= 0; else if Page_Min_Prim_Threshold > 0, Page_Max_Prim_Threshold > Page_Min_Prim_Threshold.
  Pseudo code:

```
{
    pageBasedRendering = GetValue("PageBased_Rendering_Enable");
    maxPrimThreshold = GetValue("Page_Max_Prim_Threshold");
    minPrimThreshold = GetValue("Page_Min_Prim_Threshold");
    maxPrimThreshold <<= 1;
    minPrimThreshold <<= 1;
    if(minPrimThreshold == 0)
        assert(maxPrimThreshold >= minPrimThreshold);
    else
        assert(maxPrimThreshold > minPrimThreshold);
}
```

- If PageBased_Rendering_Enable is disabled, bypass page raster and page sorting etc. Usually this register can be on, but if Line_Stipple_Enable is on and drawing line or wire-frame, disable PageBased Rendering. Register definition:
  0x220 [22 : 22] M2REG_PAGEBASED_RENDERING_ENABLE
  Pseudo code:

```
{
    lineStippleEnable = GetValue("Line_Stipple_Enable");
    frontFillMode = GetValue("Front_Fill_Mode");
    backFillMode = GetValue("Back_Fill_Mode");
    bLineDrawed = (Primitive_Type = 0x02:  IndexedLineList
                                    or 0x03:  IndexedLineStrip
                                    or 0x07:  IndexedLineloop
                                or 0x12:  LineListImmediateMode
                                    or 0x13:  LineStripImmediateMode
                                    or 0x17:  LineloopImmediateMode);
    if (lineStippleEnable && (frontFillMode == 0x2)) // Wireframe
        pageBaseRendering = 0;
    if (lineStippleEnable && (backFillMode == 0x2) // Wireframe
        pageBaseRendering = 0;
    if (lineStippleEnable && bLineDrawed)
        pageBaseRendering = 0;

    SetValue("PageBased_Rendering_Enable", pageBaseRendering);
}
```

- If PageBased_Rendering_Enable is disabled, please set "Always_FullPage_Prefetch = 0" and "DE_Invisible_Tile_Drop = 0"
  Pseudo code:

```
{
    pageBasedRenderingEnable = GetValue("PageBased_Rendering_Enable");
```

```
        if(0 == pageBasedRenderingEnable)
        {
            allPagePrefetch = 0;
            deInvTileDrop = 0;
        }
        SetValue("Always_FullPage_Enable", allPagePrefetch);
        SetValue("DE_Invisible_Tile_Drop", deInvTileDrop);
    }
```

- If fill mode = Wireframe or Point, and PageBased_Rendering_Enable (0x220 D0[22]) = true, have to set Flush_NoSorting_Enable (0x220 D0[18]) = true.
  Pseudo code:

```
    {
        frontFillMode = GetValue("Front_Fill_Mode");
        backFillMode = GetValue("Back_Fill_Mode");
        pageBased = GetValue("PageBased_Rendering_Enable");
        if((2 == frontFillMode || 2 == backFillMode || 1 == frontFillMode || 1 == backFillMode)
            && 0 != pageBased)
        {
            flushNoSorting = 1;
        }
        SetValue("Flush_NoSorting_Enable", flushNoSorting);
    }
```

12.Programming of Bump Shader

- If loop output to 4D Bump, the corresponding sync enable bit of the loop should be set. In a bump loop, if level N sets BUMP_SYNC_EN, then all levels below N also have to set BUMP_SYNC_EN. The signal is come from Bump_Sync_Enabels global register.
  Pseudo code:

```
{
    bsMode = GetValue("Bump_Shader_Mode");
    if(bsMode == 2 || bsMode == 3)
    {
        bsTotalLoops = GetValue("Bump_Shader_Total_Loops");
        for(int n = 0; n < bsTotalLoops; ++n)
        {
            if (loopn of PS has output to 4D Bump)
            {
                for(int i = 0; i <= n; ++i)
                    bsSyncEnable[i] = 1;
            }
        }
        for( i = 0; i < bsTotalLoops; ++i )
            SetValue("Bump_Sync_Enable_Loopi", bsSyncEnable[i]);
    }
}
```

- When Constant_Rendering_Enable, discard the status of Invisible_PixelPair_Drop.
- If Pixel Shader setting: Shadow_Working_Mode = 0, that mode is only for single loop, set Bump_Shader_Total_Loops = 0.
- For no texture case, should set Diffuse_Only.
- Bump_Shader_Mode:
  1) One_Loop_Only (0): no bump loop, consistent to total bump loop = 0
  2) 2D_Bump_Loop (1): bump loop, with 2D bump
  3) 4D_Bump_Loop (2): bump loop, with 4D bump
  4) 2D_4D_Bump_Mixed (3): bump loop, with 2D and 4D bump

  Bump Shader Mode One_Loop_Only should consistent to total bump loop.
  Pseudo code:

```
{
    bsTotalLoops = GetValue("Bump_Shader_Total_Loops");
    if(bsTotalLoops == 0)
        bsMode = 0;
    SetValue("Bump_Shader_Mode", bsMode);
}
```

- If there are multiple Bump data out in one loop, the last is 2DBump, all the other is 4DBump. The reason is as below. **Thus, mixed 2D and 4D bump in the same loop should be avoided.** Suggest put all 2Dbump in the first loop and other 4D bump in next loops.

  For HW, the PS output sequence in HW need be adjusted.
  PP mode can deal with 4 pixels per clock, for example, p1 and p2 will go to 16bit ALU, p3 and p4 will go to 32bit ALU, the right output should be "p1->p2->p3->p4", but HW's output before adjusting is "p1->p3->p2->p4".

According to HW's latest update, XP10 PS output will deal with 2D Bump and 4D Bump at the same form. For PP mode, it needs one signal to identify the last bump output in current loop. So we suggest if there are multiple Bump data out in one loop, the last is 2DBump, all the other is 4DBump.

- Bump_Shader_Maximum_Tiles_For_Loop:

  1) If PS2.0 and using mode2 (32 texture working only mode), can only set one tile in each loop.

  2) If source z come from PS and bump loop is enabled, tiles per loop should be limited.

  Pseudo code:

```
{
    srcZ = GetValue("Depth_Source");
    bumpLoop = GetValue("Bump_Shader_Total_Loops");
    if(srcZ == 1(PixelShader_Depth) && bumpLoop > 0)
    {
        tilePerLoop = 0;
        ppMode = 0;
        invPixPairDrop = 0;
    }
    SetValue("Bump_Shader_Maximum_Tiles_For_Loop", tilePerLoop);
    SetValue("Partial_Precision_PS2_Enable", ppMode);
    SetValue("Invisible_PixelPair_Drop_Enable", invPixPairDrop);
}
```

- If there is Bump Shader Instruction of Texture Coordinate, set register floating coordinate enable.

  Pseudo code:

```
{
    if(IsTexCoordUsedInBumpShader(currentBumpShader))
    {// SamplingState   = 4:    Coord_NoKill_NoClamp
     //                 or 5: Coord_NoKill_Clamp
        floatCoord = 1;
        SetValue("Float_TexCoord_Enable", floatCoord);
    }
}
```

- If floating coordinate enable, disable Clamp of Bump Shader Instruction.

  Pseudo code:

```
{
    floatCoord = GetValue("Float_TexCoord_Enable");
    bTexClamp = IsTexCoordClamInBumpShader(currentBumpShader);
    // SamplingState   = 5:    Coord_NoKill_Clamp
    //                 or 7:    Coord_Kill_Clamp
    if(floatCoord)
        assert(bTexClamp == 0);
}
```

How to compute total tiles for bump loop?

Assume we have total loops = L.

After PS optimizer and write into binary shader code,

```
totalTilesForBumpLoops = 32;
For (i = 0; i < L-1; i++)       // exclude last loop
{
    totalTilesForCurrentLoop = floor(BumpFIFOLevels
/TotalOutBumpsFromPSInCurrentLoop) +
                                        Min(PipeTiles[i], PipeTiles[i+1]) + ConstTiles;
```

```
        if (totalTilesForBumpLoops > totalTilesForCurrentLoop)
            totalTilesForBumpLoops = totalTilesForCurrentLoop;
}
```

Set (totalTilesForBumpLoops-1) into register.

For PP mode, ConstTiles = 3 ;
BumpFIFOLevels = 20 ;
For full precision mode, ConstTiles = 1;
BumpFIFOLevels = 18 ;

// Please change FilteringPipeLevels = **16** for both 2D bump and 4D bump cases,
// because 24 may be too aggressive & large for some heavy texture miss cases.

For 4Dbump only:
ShadingPipeLevels = 4;
FilteringPipeLevels = 16; // original value is 24

For 2DBump only:
BumpFIFOLevels += 48;
ShadingPipeLevels = 0;
FilteringPipeLevels = 16; // original value is 24

PipeTiles[i] = floor(ShadingPipeLevels/TotalBSIntoTexturePipe[i]) +
floor(FilteringPipeLevels/TotalFilteringPairs[i]);

TotalBSIntoTexturePipe[i] = TotalBSInstructions - (TexCoordReplaceDiffuseEnable +
TexCoordReplaceSpecularEnable);
TotalFilteringPairs[i] = TotalBSIntoTexturePipe[i] - TotalHoldBitsInBSInCurrentLoop +
TotalExpandedFilteringPairs;

TotalExpandedFilteringPairs is computed as follows:
ExpandedFilteringPairs = 0;
For a texture pair of two valid 2D textures and at least one of two has trilinear filtering enable,
ExpandedFilteringPairs++;
For a 2D/cube texture with anisotropic filtering, ExpandedFilteringPairs +=
2^(MaxAnisoRatio-1+trilinearFilteringEnable) - 1;
For a 2D texture with higher order filtering, ExpandedFilteringPairs +=
HorizontalKernelSize*VerticalKernelSize * (1+trilinearFilteringEnable)/2 - 1;
For a 3D texture, ExpandedFilteringPairs += trilinearFilteringEnable;
If YUV_DCT enable, ExpandedFilteringPairs += ExpandedFilteringPairs + 1;
If YUV_420 enable, ExpandedFilteringPairs = ExpandedFilteringPairs*3 + 2;

Then TotalExpandedFilteringPairs = SUM(ExpandedFilteringPairs);
// use info in current loop i only to compute all values above,

13.Programming of Texture Sampler

- Texture pairing: only 2D textures can be paired up, either in one bump shader instruction, or in 2 continuous BS instructions with Hold/Force flags.
  Hold and Force-to-right rule:
  1. Hold and Force must be paired
  2. It works in cases:
     - ✓ 2D / Cube / projected_2D / projected_Cube texture type
     - ✓ with point or bilinear sampling mode
     - ✓ High Order Filtering Mode is 00(disable), or 01(UV offset enable)
  3. It doesn't work in following case:
     - ✓ With trilinear, or anisotropic sampling mode
     - ✓ High Order Filtering Mode is 10(staged kernel filtering), or 11(precise kernel filtering)
     - ✓ 3D texture
     - ✓ Depth Texture
     - ✓ Texture Coordinate Mode

- About High Order Filter Mode and Bump Shader Instruction Sampler Attribute
  In Instruction TextureSampleState, register

```
D5 [25:24]      High_Order_Filtering_Mode, Enumerate.
{
        0:  Disable
        1:  UVOffset
        2:  StagedFiltering     // load weight table directly.
        3:  PreciseFiltering    // Use weight map in the same pair.
} HighOrderFilterMode
```

  HOF mode 2 requires Left_Lookup_Only BumpShader instruction; HOF mode 3 requires Left_Right_Lookup BumpShader instruction.

  In XP10, there are 2 cases of HOF:
  1) one texture and Staged Kernel Filtering: use the weight being loaded.
  2) two texture and Precised Kernel Filtering: use the right texture as weight.

  Pseudo code:

```
{
    bsTotalLoops = GetValue("Bump_Shader_Total_Loops");
    for(int i = 0; i <= bsTotalLoops; ++i)
    {
        curLoopEntry = GetValue("Loopi_Bump_Instruction_Entry");
        curBSInst = bumpShaderPtr + 4 + curLoopEntry;
        bool lastInst = false;
        do{
            lastInst = (0 != (*curBSInst & 0x40000000));
            LeftSamplerID = (*curBSInst >> 8) & 0x0f;
            RightSamplerID = (*curBSInst >> 12) & 0x0f;
            SamplingAttrib = (*curBSInst >> 16) & 0x07;

            HOFMode = (samplerState[LeftSamplerID * 8 + 5] >> 24) & 0x03;
            if(HOFMode == 2)
```

```
                    assert(SamplingAttrib == 1); //Left Only
                if(HOFMode == 3)
                    assert(SamplingAttrib == 3); //Left Right

                curBSInst++;
            }while(!lastInst);
        }
    }
```

- Banded tile mode setting:
  1) For MET mode, both tile and band32_tile can be used.
  2) UYVY and YUV2 are treated as 16bpp, they can be all formats. DXTn is already tiled, treat them as 4bpp and 8bpp, and they can also be band64 or band32 etc. In general, formats are independent from buffer mode: band or tile or linear.
  3) both cube and volume map can be band mode up to 1x1 size, when the map size is less than 64, driver size should take some special code: put one slice or face into a part of band and may cross band and to be tightly placed each other slice by slice or face by face.

- For cube-map, U size and V size should be same!

- High order vector, if High_Order_Filtering_Mode D5[25 : 24] = 2, staged kernel filtering, load weight table directly, so only left channel valid here; if High_Order_Filtering_Mode = 3, precise kernel filtering. Use weight map in the same pair. So must left and right both valid.

- If there is any anisotropic texture filtering, please enable “Anisotropic_Filtering_Enabled” register. If no anisotropic filter is used, set Anisotropic_Filtering_Enabled = 0.

- Border Mode Setting:
  Border_Mode:
  0: Border_Color. use constant border color.
  1: Border_Texel. use border texel in texture map. The border width = 1.
  If Bordre_Mode = Border_Texel, texture
  U_Size = U_Size + 2
  V_Size = V_Size + 2
  That is to say, the texture data in buffer should include the border texel, but the corresponding texture width and height in Texture Sampler State should not include the border width. For example, if a 64x64 texture is used with Border_Texel, the texWidth = 64, texHeight = 64, but in buffer it should contain 66x66 texel data.

- Guide for high order filtering
  HOF related register definition:

```
Texture Sampler State Array Register
D5 [25:24]  High_Order_Filtering_Mode, Enumerate.
{
        0:  Disable
        1:  UVOffset
        2:  StagedFiltering      // load weight table directly.
        3:  PreciseFiltering     // Use weight map in the same pair.
} HighOrderFilterMode
```

```
D2 [27:22]  TextureFilteringEntry, Integer.     // for high order filtering deltaUV
D2 [21:16]  TextureFilteringLength, Integer.  // Actual size = TextureFilteringLength+1;

// Kernel size & scale factor for high order filtering
// Ku = KernelSizeU<<(5-Su). TextureFilteringLength+1 = KernelSizeU*KernelSizeV.
D3 [23:21]        Vertical_Scale_Factor, Integer.              // Sv;
D3 [20:12]        Vertical_Scaled_Kernel_Size, Integer.      // Kv;
D3 [11:9]         Horizontal_Scale_Factor, Integer.            // Su;
D3 [8:0]          Horizontal_Scaled_Kernel_Size, Integer.  // Ku;

D6 [31:16]  DeltaV, FP9.7.      // for UV coordinate adjustment in filter loop.
D6 [15:0]   DeltaU, FP9.7.      // no adjustment for third dimension.
```

Pseudo code for HOF filtering:

```
{
    if HighOrderFilterMode == UVOffset
    {
        //Only do SW deltaUV, no High_Order_Filtering.
        U += DeltaU    (TextureSamplerStates: D6[31:16, DeltaU]
        V += DeltaV    (TextureSamplerStates: D6[31:16, DeltaV]
    }
    else if HighOrderFilterMode == StagedFiltering || HighOrderFilterMode ==
    PreciseFiltering
    {
        //do SW deltaUV, and High_Order_Filtering for left texture

        //for each Pixel, we will sample n texel (n = TextureFilteringLength),
        //(TextureSamplerStates: D2 [21:16] TextureFilteringLength)
        //then do average with weight to get the final texel.
        hofCurIndex = 0;
        while ( hofCurIndex <= TextureFilteringLength )
        {
            //(TextureSamplerStates: D2 [27:22]   TextureFilteringEntry)
            hofCurEntry = hofCurIndex + TextureFilteringEntry;

            //HOFDeltaUVWeight RAM is build by the RE load command
            //Each entry is 29 bits:
            //                  D[9,0]    : DeltaUVTale.U, s.4.5 fix point
            //                  D[19,10]: DeltaUVTale.V, s.4.5 fix point
            //                  D[28,20]: DeltaUVTale.W, s.8     fix point
            Read HOFDeltaUVWeight RAM data at hofCurEntry

            Adjust UV value
                (DeltaU, DeltaV;
                 Vertical_Scale_Factor, Horizontal_Scale_Factor;
                 Vertical_Scaled_Kernel_Size, Horizontal_Scaled_Kernel_Size)
            {
                U += DeltaU    +    DeltaUVWeightTable[hofCurIndex].U
                V += DeltaV    +    DeltaUVWeightTable[hofCurIndex].V

                if HighOrderFilterMode == PreciseFiltering
                {
                    // Process the left texture as the case of StagedFiltering,
                    // But weight will be sample from Right Texture,
                     // not from the DeltaUVWeightTable
```

```
                    For the right texture:
                    Uright += deltaU of right +
                            DeltaUVWeightTable[hofCurIndex].U * GL_U_ScaleFactor
                            + GL_U_KernalSize * (ULeft + deltaU of Left).Fraction
                    Vright += deltaV of right +
                            DeltaUVWeightTable[hofCurIndex].V * GL_V_ScaleFactor
                            + GL_V_KernalSize * (VLeft + deltaV of Left).Fraction
                }
            }

            Filter texture

            //For each Pixel, we will sample n texel (n = TextureFilteringLengths),
            //then do average with weight to get the final texel.
            // in general case, Sum of all the weight set into the weight table should be 1.
            if HighOrderFilterMode == StagedFiltering,
            {
                multiply the texel with nWeight
            }
            else if HighOrderFilterMode == PreciseFiltering,
            {
                Right texture used as weight
                Convert right texel to weight format
                multiple the texel with this weight
            }
            Accumulate current texel to HOF result
            hofCurIndex++; // for next texel
        }
    }
}
```

14.Programming of Pixel Shader

- NOTE: Texture coordinate mode does not occupy tile buffer in PS20, so it is not counted into the Texture Number which can affect the pixel shader Shadow_Working_Mode setting.

- When should we set Partial Precision mode enable?
  If one loop's texture loader number < PS ALU instruction number, PP mode can improve the GE performance. Pseudo code is like below:

```
{
    ppModeEnable = false;
    for ( i = 0; i < totalLoops; i ++ )
    {
        if ( loopTexLoaders[i] < loopPSAlus[i] )
            ppModeEnable = true;
    }
     SetValue("Partial_Precision_PS2_Enable", ppModeEnable);
}
```

The modifier "_pp" could be ignored completely in PS compiler & optimizer because if we check "_pp" exists, then we turn on PP mode, we still can not make sure no precision issue since other instructions may strongly require full precision to get correct result.

- PS 1.x is expired, no PS 1.x instruction is required. All PS 1.x will be converted into PS20 in PS Compiler. So in command list, Pixel_Shader_Version != 0 (PS 1.x).

- If Diffuse or Specular source is from texture, it can not over the maximum texture number. That is to say,

  Diffuse_RGB_Resource = 0x4 (Tex0) => maximum texture >= 1
  0x5 (Tex1) => maximum texture >= 2
  0x6 (Tex2) => maximum texture >= 3
  0x7 (Tex3) => maximum texture >= 4

  Diffuse_Alpha_Resource = 0x4 (Tex0) => maximum texture >= 1
  0x5 (Tex1) => maximum texture >= 2
  0x6 (Tex2) => maximum texture >= 3
  0x7 (Tex3) => maximum texture >= 4

  Specular_RGB_Resource = 0x4 (Tex0) => maximum texture >= 1
  0x5 (Tex1) => maximum texture >= 2
  0x6 (Tex2) => maximum texture >= 3
  0x7 (Tex3) => maximum texture >= 4

- If Partial_Precision_PS2_Enable = 1, global register 0x380 must be set prior to PS 2.0 Constant loading.

- PS 1.x is expired, no PS 1.x instruction is required. All PS 1.x will be converted into PS20. Pixel_Shader_Version != 0 (PS 1.x)

- PS20 has 32 tile buffer, support 3 mode:
  1) Mode0: 8 texture shadow/working mode
  2) Mode1: 16 texture shadow/working mode
  3) Mode2: 32 texture working only mode

- Pixel Shader Mode0
  1) no bump loop
  2) 8 texture inputs mode, texture and temp register can not more 8
  3) Can set invisible pixel pair drop for single loop

- Pixel Shader Mode1
  1) can not set invisible pixel pair drop any way
  2) all texture sampling and coordinate mode are mapped to temp registers
  3) 16 texture input mode

- Pixel Shader Mode2
  1) require texture load table
  2) can not set invisible pixel pair drop
  3) 32 buffer working only
  4) no pp mode
  5) only one tile in each loop

- If PS2.0 and using texture coordinate, we should set floating coordinate enable.

- If Pixel Shader output Z to depth, invisible pixel pair drop should be disabled.

- It is illegal to output 2 results(MRT) in one PS instruction.

- Co-issue rules
  There are at most 3 sources and 2 destinations in Pixel Shader 2.0 instruction bit, if one instruction with less than 3 sources and 2 destinations, it can co-issue with one MOV instruction to reduce the total instruction number.

  The MOVE in PS2.0 is from source#2 to destination#1 parallel with FPU of less than or equal to two operands. ZW channels are always valid to the MOVE.

- The definition of DDP4 (= 0x1C), DMUL (=0x1D), DADD (=0x1E) is as following:
  Operation: dest0 = DP4 (src0, src1), dest1 = DP4 (src0, src2).
  The same is for DMUL and DADD.

- Second Move Enable rule
  If pixel sahder instruction's OP code >= 0x1c (two destinations instruction: DDP4, DADD, DMUL), [84:84]--Second MOVE enable bit should not be set. This bit is only set when another MOVE is co-issued with this instruction.

- For performance, if specular is not used in PS & PBE, set Specular_Valid_InGP = 0 and Specular_Valid_InRE = 0, Put Instruction do not put specular.
  Pseudo code:
  {

```
specularUsed = IsSpecularUsedInRE(curBatchData);
//we need to check ReplaceMent in BumpShader Instruction and
// Color_ReplaceMent_Enable Setting.
colorReplace = GetValue("Color_Replace_Enable");
/*
    0x300   D3 [31:16]   Color_Replace_Enable, Enumerate.
    {
        0:    No_Replacement.
        1:    Replace_Diffuse.
        2:    Replace_Specular.
        3:    Replace_Diffuse_Specular.
    } ColorReplacementMode
    // 2bits /loop. D3[17:16] for loop0, D3[31:30] for loop7.
*/
for(int i = 0; i < 8; ++i)
{
    replacedSpecular |=   (colorReplace & (1 << (2 * i + 1)));
}

if(!specularUsed)
{
    specularInGP = 0;
    specularInRE = 0;
    putSpecular = 0;
}
else
{
    specularInRE = 1;
    if(replacedSpecular &&(isReplacementInBS))
    {
        specularInGP = 0;
        putSpecular = 0;
    }
    else
    {
        specularInGP = 1;
        putSpecular = 1;
    }
}

SetValue("Specular_Valid_InRE", specularInRE);
SetValue("Specular_Valid_InGP", specularInGP);
Assert(IsPutSpecular() == putSpecular);
}
```

- A HD limitation on register 0x300 and 0x380 setting

```
0x300 (R/W)        Bump_Shader_Setting
D1 [05:0]       Loop0_Bump_Instruction_Entry, Integer.
D1 [13:8]       Loop1_Bump_Instruction_Entry, Integer.
D1 [21:16]      Loop2_Bump_Instruction_Entry, Integer.
D1 [29:24]      Loop3_Bump_Instruction_Entry, Integer.
D2 [05:0]       Loop4_Bump_Instruction_Entry, Integer.
D2 [13:8]       Loop5_Bump_Instruction_Entry, Integer.
D2 [21:16]      Loop6_Bump_Instruction_Entry, Integer.
D2 [29:24]      Loop7_Bump_Instruction_Entry, Integer.

0x380 (R/W)        PixelShader_SETTING
D1 [06:00]      Loop0_PSInstruction_Entry, Integer.
D1 [14:08]      Loop1_PSInstruction_Entry, Integer.
D1 [22:16]      Loop2_PSInstruction_Entry, Integer.
D1 [30:24]      Loop3_PSInstruction_Entry, Integer.
```

D2 [06:00] Loop4_PSInstruction_Entry, Integer.
D2 [14:08] Loop5_PSInstruction_Entry, Integer.
D2 [22:16] Loop6_PSInstruction_Entry, Integer.
D2 [30:24] Loop7_PSInstruction_Entry, Integer.

For those bits, if they are not used, please always reset them to ZERO.
Take the following as an example. There are totally two Loops.
For Bump Shader, set

```
   Loop0_Bump_Instruction_Entry    =   n1;
   Loop1_Bump_Instruction_Entry    =   n2;
//all other bits are set to ZERO.
   Loop2_Bump_Instruction_Entry   = 0x0
   Loop3_Bump_Instruction_Entry   = 0x0
   Loop4_Bump_Instruction_Entry   = 0x0
   Loop5_Bump_Instruction_Entry   = 0x0
   Loop6_Bump_Instruction_Entry   = 0x0
   Loop7_Bump_Instruction_Entry   = 0x0
```

For Pixel Shader, set

```
   Loop0_PSIntruction_Entry    =   m1;
   Loop1_PSIntruction_Entry    =   m2;
//all other bits are set to ZERO.
   Loop2_PSIntruction_Entry   = 0x0
   Loop3_PSIntruction_Entry   = 0x0
   Loop4_PSIntruction_Entry   = 0x0
   Loop5_PSIntruction_Entry   = 0x0
   Loop6_PSIntruction_Entry   = 0x0
   Loop7_PSIntruction_Entry   = 0x0
```

15.Programming of Render Target

Render Target includes depth/stencil and color buffer.

15.1. Optimal Depth Buffer Setting

XP10 use (1-Z) optimal depth buffer for better precision & saving read depth bandwidth. To do so, driver needs to set the following registers:

- Set 0x210 D0[23:0] = -1.0 in IFF. // -Z
- Set 0x230 D1[31:0] = 1.0 – polygon_offset. // usually polygon_offset = 0.
  // The two above registers perform (1-Z) automatically.
- Set 0x250 D0[15:14] = exponent of float depth value. // usually it is 5, always 24bits.
- Set 0x250 D0[21:19] = opposite depth test function to DDI setting.

15.2. Target Buffer Format & Settings

Set the following registers for rendering target format:

- 0x240 for relative depth/stencil buffer rectangle setting.
- 0x250 D0[17:16] depth buffer format.
- 0x280 D0[19] MRT_Mode,
- 0x280 D0[18:16] Main_Target_Color_Format, Enumerate.
- 0x280 D0[15:14] Total_Targets_Elements, Integer.
  // The number of Multiple Render Targets or Multiple Elements Texture target.
  0: means one target or one element; 3 means 4 targets or elements.
  If (it is multiple element target)
  HW will only need main render target setting.
- 0x280 D0[13:12] Banded_Tiled_Mode, Enumerate. // for all MRTs.
  {
  0: Linear.
  1: Tiled4x4. // 4x4 pixel tiled.
  2: Banded64. // 1x4x(64*16) pixel banded.
  3: Banded32_Tiled. // 2x2x(32*32) pixel banded + 4*4 pixel tiled.
  } BandTileMode
- 0x270 D2 [23:20] = Component_Mask, Integer. // 1 means no write out.
- 0x250 D0 [10:0] = Depth_Buffer_Width, Integer. // divided by 4.
- 0x250 D1 [23:0] = Depth_Buffer_Base_Address, Integer.
- 0x280 D0 [10:0] = Color_Destination_Buffer_Width, Integer. // divided by 4.
- 0x280 D1 [23:0] = Color_Buffer_Base_Address0, Integer.
  // 32bytes alignment. Up to 256MB. default = 0x000000.
- 0x250 D2 [14] = **StencilReadEnable**, Boolean.
  0: no read stencil. Use Reference stencil as destination stencil (stencilBuf) and do stencil operation.
  1: read stencil. When stencil test is enabled.
- 0x260 D1 [10] = Destination_Color_Buffer_Read_Enable, Boolean. For main target
  When alpha blending, ROP3 and bit mask operation do not need destination color, then do not set it. Otherwise set it. HW will read destination color based on this bit and tile's visibility flag. If this bit is zero, HW will not read destination color and

just use 0x00000000 for all operations requiring destination color.

- 0x270 D2 [26] = Destination_Color_Buffer_Read_Enable3, Boolean.
- 0x270 D2 [25] = Destination_Color_Buffer_Read_Enable2, Boolean.
- 0x270 D2 [24] = Destination_Color_Buffer_Read_Enable1, Boolean.

When it really needs destination color/stencil for operation, set read enable. Otherwise set disable for performance. When color read disabled, HW will use constant background color for destination operation.

15.3. Compressed Color Buffer Allocation and Setting

- Driver will allocate an original size of color buffer, for safe (not overwrite) when decompression, driver may allocate two more lines.
- Driver sets compressed color buffer or texture base address starting the middle line of the allocated buffer.
- Driver sets start and end address of the allocated whole buffer for CPU read/write table.

15.4. Buffer Clearing ---- Constant Color Render Mode

XP10 has a constant color rendering mode to speed up rendering when there are no texture, PS operation, pixel blending and object light. This mode can be used for the following cases:

- Depth/color buffer clearing.
- Depth / stencil buffering only pass.
- Constant color fill only.

Register setting as follows:

- 0x220 D0 [17] = Constant_Color_Render_Enable.
- 0x280 D2 [31:0] = CONST_COLOR , Integer.
- Set vertex Z to a constant depth and set test = always for depth clear, stencil test = always and operation = ZERO for stencil clear.
- Use original vertex Z and test function for depth/stencil buffering only pass.
- Set corresponding depth/stencil write enable and color component mask.

15.5. Pixel Operation & Color Buffer Setting

- Always enable bitmask feature for all available render targets, set correct mask to ensure correct result.

  Register definitions:

0x264	[9 : 9]	M2REG_ENABLE_BIT_MASK
0x28c	[31 : 0]	M2REG_COLOR_WRITE_BIT_MASK0
0x290	[3 : 3]	M2REG_BIT_MASKING_ENABLE1
0x290	[11 : 11]	M2REG_BIT_MASKING_ENABLE2
0x290	[19 : 19]	M2REG_BIT_MASKING_ENABLE3
0x298	[31 : 0]	M2REG_BIT_MASK1
0x2a8	[31 : 0]	M2REG_BIT_MASK2
0x2ac	[31 : 0]	M2REG_BIT_MASK3

- Color Read bits always enable for all available render targets.
  Register definitions:

0x264	[10 : 10]	M2REG_DESTINATION_COLOR_BUFFER_READ_ENABLE
0x278	[26 : 26]	M2REG_DESTINATION_COLOR_BUFFER_READ_ENABLE3
0x278	[25 : 25]	M2REG_DESTINATION_COLOR_BUFFER_READ_ENABLE2
0x278	[24 : 24]	M2REG_DESTINATION_COLOR_BUFFER_READ_ENABLE1

  Pseudo code:

```
{
    for MRT0 =>
        Enable_Bit_Mask = 1
        Destination_Color_Buffer_Read_Enable = 1
    for MRTi(i = 1 ~ 3) =>
        Bit_Masking_Enablei = 1
        Destination_Color_Buffer_Read_Enablei = 1
}
```

- Enable color-prefetch when real frame buffer read happens. "Real frame buffer read" means that at least one of following features is enable:
  a) Bitbask enable &&
  i) Targeti_Color_Format = 0x4 (ARGB8888) && Bitmask != 0xffffffff
  or
  ii) Targeti_Color_Format != 0x4 (ARGB8888) && Bitmask != 0xffffffff or 0x0
  (exception: if render target is X888 format and bitmask = 0x00ffffff, it has no use to set color-prefetch, this case could be distinguished manually)
  b) If Alpha_Blending_Enable && Destination color selection is from Destination_Buffer, and one of (Source_RGB_Blending_factor, Source_Alpha_Blending_factor, Destination_RGB_Blending_factor, Destination_Alpha_Blending_factor) is from destination.
  i) Alphablending_Enable = true
  ii) Destination_Selection = 0 - from color buffer
  iii) one of the 4 factors has a value of 0x7, 0x8, 0x9 or 0xa
  c) If rop3 enable and rop3 code indicates it will use destination.
  i) ROP3_Enable = true
  ii) ROP3_Code != (0x00, 0x03, 0x0c, 0x0f, 0x30, 0x33, 0x3c or 0x3f)
  d) destination color key enable.
  Register definitions:

0x274	[27 : 27]	M2REG_COLOR_PREFETCH_ENABLE

  Pseudo code:

```
{
    totalTarget = GetValue("Total_Targets_Elements") + 1;
    alphaBlendingEnable = GetValue("Alpha_Blending_Enable");
    destSelectFromColorBuf = GetValue("Destination_Selection");
    factors[0] = GetValue("Source_RGB_Blending_Factor");
    factors[1] = GetValue("Source_Alpha_Blending_Factor");
    factors[2] = GetValue("Destination_RGB_Blending_Factor");
    factors[3] = GetValue("Destination_Alpha_Blending_Factor");
    ROP3Enable = GetValue("ROP3_Enable");
    ROP3Code = GetValue("ROP3_Code");
    ColorKeyEnable = GetValue("Destination_Color_Key_Enable");
    bitMaskEnable[0] = GetValue("Enable_Bit_Mask");
    bitMask[0] = GetValue("Color_Write_Bit_Mask0");
    targetFormat[0] = GetValue("Main_Target_Color_Format");
    for (i= 1; i < totalTarget; i++)
    {
```

```
            bitMaskEnable[i] = GetValue("Bit_Masking_Enablei");
            bitMask[i] = GetValue("Bit_Maski");
            targetFormat[i] = GetValue("Targeti_Color_Format");
        }

        // condition a)
        for (n = 0; n < totalTarget; n++)
        {
            needPrefetch = bitMaskEnable[n] && ( 0xffffffff != bitMask[n]);
            if (targetFormat[n] != 0x4)
                needPrefetch &= (0 != bitMask[n]);
            if (needPrefetch)
                break;
        }
        if (needPrefetch)
        {
            SetValue("Color_Prefetch_Enable", needPrefetch);
            Exit(0);
        }

        // condition b)
        if(alphaBlendingEnable && 0 == destSelectFromColorBuf)
        {
            for(int i = 0; i < 4; ++i)
            {
                needPrefetch |= (factors[i] >= 0x7 && factors[i] <= 0xa);
            }
        }
        if (needPrefetch)
        {
            SetValue("Color_Prefetch_Enable", needPrefetch);
            Exit(0);
        }

        // condition c)
        if(ROP3Enable)
        {
            needPrefetch = (ROP3Code!= (0x00, 0x03, 0x0c, 0x0f, 0x30, 0x33, 0x3c or 0x3f))
        }
        if (needPrefetch)
        {
            SetValue("Color_Prefetch_Enable", needPrefetch);
            Exit(0);
        }

        // condition d)
        if(colorKeyEnable)
            needPrefetch = 1;
        SetValue("Color_Prefetch_Enable", needPrefetch);
    }
```

- If render_target_color_format != ARGB8888 && bit_mask = 0, disable color_prefetch.

- When Conversion_Blit_Enable = 1,
  if Conversion_Blt_Code == 2 (DepthToColor)
  Color_Prefetch_Enable = 0
  else if Conversion_Blt_Code == 3 (ColorToDepth)
  Depth_Prefetch_Enable = 0
  Register definitions:

```
0x278       [ 29 :  29]     M2REG_CONVERSION_BLIT_ENABLE
0x278       [ 19 :  18]     M2REG_CONVERSION_BLT_CODE
Conversion_Blt_Code, Enumerate.
  {
      0x0  CompressColorBuffer,          // from decompressed to compressed
      0x1  DeCompressColorBuffer,            // from compressed to decompressed
      0x2  DepthToColor,
      0x3  ColorToDepth,
  } ConversionBlitCode
```

Pseudo code:

```
{
    blit = GetValue("Conversion_Blit_Enable");
    blitCode = GetValue("Conversion_Blt_Code");
    if ( blit && (2 == blitCode))
        colorPrefetch = 0;
    if ( blit && (3 == blitCode))
        depthPrefetch = 0;
    SetValue("Color_Prefetch_Enable", colorPrefetch);
    SetValue("Depth_Prefetch_Enable", depthPrefetch);
}
```

- When Constant_Color_Render_Enable = 1, set Color_Prefetch_Enable = 0.
  Pseudo code:

```
{
    constRender = GetValue("Constant_Color_Render_Enable");
    if (constRender)
        colorPrefetch = 0;
    SetValue("Color_Prefetch_Enable", colorPrefetch);
}
```

- About format type A8R8G8B8 & XRGB8888,
  For D3DFMT_A8R8G8B8, set ColorBufferFormat as RTF_ARGB888 (color buffer format define in XP10); for D3DFMT_XRGB8888, set ColorBufferFormat as RTF_ARGB888 too, but:
  Enable CONSTANT_ALPHA_REPLACE_ENABLE0 (for MRT=0)
  Enable CONSTANT_ALPHA_REPLACE_ENABLE1 (for MRT=1)
  Enable CONSTANT_ALPHA_REPLACE_ENABLE2 (for MRT=2)
  Enable CONSTANT_ALPHA_REPLACE_ENABLE3 (for MRT=3)
  No need to set DESTINATION_COLOR_KEY_CONSTANT_COLOR.

- DESTINATION_COLOR_KEY_CONSTANT_COLOR is used for 2D destination color key, constant alpha blend factor and constant color fill.

- In XP10, color buffer and depth buffer base address are 256 bit alignment.

- If the buffer format is floating, disable all post-ps operations. That is to say, if Float_Color_Buffer_Format != 0 (floating color buffer), please disable the following PBE processes:
  1) Alpha_Blending_Enable = 0
  2) Fog_Blending_Enable = 0
  3) Specular_Blending_Enable = 0
  4) ROP3_Enable = 0
  5) Color_Dither_Enable = 0
  6) Destination_Color_Key_Enable = 0

7) Gamma_Correction_Enable = 0

Pseudo code:

```
{
    floatRT = GetValue("Float_Color_Buffer_Format");
    if(floatRT)
    {
        alphaBlend = 0;
        fogBlend = 0;
        specularBlend = 0;
        rop3 = 0;
        dither = 0;
        colorKey = 0;
        gammaCorrect = 0;
    }
    SetValue("Alpha_Blending_Enable", alphaBlend);
    SetValue("Fog_Blending_Enable", fogBlend);
    SetValue("Specular_Blending_Enable", specularBlend);
    SetValue("ROP3_Enable", rop3);
    SetValue("Color_Dither_Enable", dither);
    SetValue("Destination_Color_Key_Enable", colorKey);
    SetValue("Gamma_Correction_Enable", gammaCorrect);
}
```

- Color_Interleave_With_Zbuffer is expired. Color_Interleave_With_Zbuffer must be set 0.

  Pseudo code:

```
{
    ColorInterleaveZ = GetValue("Color_Interleave_With_Zbuffer");
    Assert(ColorInterleave == 0);
}
```

- If No Color R/W, please set following register as below:
  1) Constant_Color_Render_Enable = 1
  2) Depth_Test_Function = 7 (ALWAYS)
  3) Depth_Bound_Test_Enable = 0
  4) Stencil_Test_Enable = 0
  5) Depth_Read_Disable = 1
  6) Sync_Mode = 0 (HiZ_Mask_Sync)

  Pseudo code:

```
{
    // color write
    totalTarget = GetValue("Total_Targets_Elements");
    For each renderTarget
        if(componentMask != 0x0f)
            isColorWrite = true;

    // color read, please refer to above color-prefetch setting in this section
    isColorRead    = RealFrameBufferRread();

    if(!isColorWrite && !isColorRead)
    {
        constantRender = 1;
        depthTest = 7;
        depthBound = 0;
        stencilTest = 0;
        depthReadDisable = 1;
        syncMode = 0;
    }
```

```
        SetValue(“Constant_Color_Render_Enable”, constantRender);
        SetValue(“Depth_Test_Function”, depthTest);
        SetValue(“Depth_Bound_Test_Enable”, depthBound);
        SetValue(“Stencil_Test_Enable”, stencilTest);
        SetValue(“Depth_Read_Disable”, depthReadDisable);
        SetValue(“Sync_Mode”, syncMode);
    }
```

- If Color_Write_Disable = 1, set Cache_Forward_Enable = 0
  Pseudo code:

```
{
    colorWriteDisable = GetValue("Color_Write_Disable");
    if(colorWriteDisable)
        cacheForwardEnable = 0;
    SetValue ("Cache_Forward_Enable", cacheForwardEnable);
}
```

- Blit batch patch method: after every Blit batches, there needs to be a patch batch to clear cache. This patch batch should be:
  1) Constant_Color_Render_Enable = 1
  2) Page_Cache_Mode = 0
  3) No Color R/W
  4) No Depth R/W
  5) No Stencil R/W
  6) the primitive should touch 4 tiles that belongs to 4 different pages, such as a rectangle with corner (12, 12)-(19, 19)

- How to know Color Read?
  Color Buffer Read is occurred when:
  1) Enable_Bit_Mask = 1 and Bit_Mask0 != 0xffffffff (MRT = 0)
     or
     Bit_Masking_Enablei = 1 and Bit_Maski != 0xffffffff (MRT > 0)
  2) Alpha blending enable && Destnation color is from color buffer && one of the 4 factors need dest color or alpha.
     The factor enums need dest:
     0x7 - Dest.Alpha
     0x8 - (1 - Dest.Alpha)
     0x9 - Dest.RGB
     0xa - (1 - Dest.RGB)
  3) ROP3 enable & ROP3 Code need dest color.
     ROP3 Codes that do not need dest: 0x00, 0x03, 0x0c, 0x0f, 0x30, 0x33, 0x3c, 0x3f
  4) Color Key Enable

- How to know Color Write?
  1) Color_Write_Disable = 0;
  2) bitMaskEnable = 0 or bitMask != 0xffffffff
  Take the Main Target as example, the pseudo code is like below:

```
{
    isColorWrite = true;
    colorWriteDisable = GetValue("Color_Write_Disable");
    if(colorWriteDisable)
    {
        isColorWrite = false;
```

```
        }
        bitMaskEnable = GetValue("Enable_Bit_Mask");
        bitMask = GetValue("Color_Write_Bit_Mask0");
        if(bitMaskEnable && bitMask == 0xffffffff)
        {
            isColorWrite = false;
        }
    }
```

15.6. Z & Stencil Buffer Setting

- Two-side-stencil setting in XP10 is inverted against DirectX Application setting, that is to say, the setting of CCW in XP10 takes the value of CW setting from App. The root cause is, in XP10, the ccw flag is computed in Viewport Engine, which already inverse the Y axis direction, so that the ccw flag here indicates the CW state of App.

- Set register Stencil_write_Enable_OE to be "!(Stencil_Write_Enables == 0 && Stencil_Write_Enables_CCW == 0)". The pseudo code like below:

```
{
    pdeStencilWriteEnables = GetValue("Stencil_Write_Enables");
    pdeStencilWriteEnablesCCW = GetValue("Stencil_Write_Enables_CCW");
    oeStencilWriteEnable = !((0 == pdeStencilWriteEnables ) &&
                             (0 == pdeStencilWriteEnablesCCW)));
    SetValue("Stencil_Write_Enable_OE", oeStencilWriteEnable);
}
```

- If Stencil_Test_Enable_PDE is true, and not Stencil Write, set Stencil_Read_Disable to 0. The pseudo code like below:

```
{
    stencilTestPDE = GetValue("Stencil_Test_Enable_PDE");
    stencilWrite = GetValue("Stencil_Write_Enables");
    twoSideStencil = GetValue("Two_Sided_Stencil_Enable");
    stencilWriteCCW = GetValue("Stencil_Write_Enables_CCW");
    bool isStencilWrite = (twoSideStencil == 0) ?
                          (0 != stencilWrite) :
                          ((0 != stencilWrite) && (0 != stencilWriteCCW));
    if(stencilTestPDE && !isStencilWrite)
    {
        stencilReadDisable = 0;
        SetValue("Stencil_Read_Disable", stencilReadDisable);
    }
}
```

- Enable Depth-Prefetch when one of following features is ok:
  1) Depth_Test_Func and Depth_Test_Func_PDE not NEVER or ALWAYS
  2) Bound_Test_Enable or Bound_Test_Enable_PDE

  Pseudo code:

```
{
    depthBoundTestEnable = GetValue("Depth_Bound_Test_Enable");
    depthBoundTestEnablePDE = GetValue("Depth_Bound_Test_Enable_PDE");
    depthTestFunction = GetValue("Depth_Test_Function");
    depthTestFunctionPDE = GetValue("Depth_Test_Function_PDE");

    if(depthBoundTestEnable || depthBoundTestEnablePDE)
        depthPrefetchEnable = 1;
```

```
        if( (depthTestFunction != 0x0) && (depthTestFunction != 0x7) )
            depthPrefetchEnable = 1;

        if( (depthTestFunctionPDE != 0x0) && (depthTestFunctionPDE != 0x7) )
            depthPrefetchEnable = 1;

        SetValue("Depth_Prefetch_Enable", depthPrefetchEnable);
    }
```

- If depth test function is EQUAL, set Depth_Test_Function_PDE = 0x2 (EQUAL) and Depth_Test_Function = 0x7 (ALWAYS); if depth test function is NOTEQUAL, set Depth_Test_Function_PDE = 0x5 (NOTEQUAL) and Depth_Test_Function = 0x7 (ALWAYS).
  Otherwise, set Depth_Test_Function = Depth_Test_Function_PDE = depth test function.
  Pseudo code:

```
{
    if (depth test function is EQUAL)
    {
        depthTestFunctionPDE = 0x2;
        depthTestFunction = 0x7;
    }
    else if (depth test function is NOTEQUAL)
    {
        depthTestFunctionPDE = 0x5;
        depthTestFunction = 0x7;
    }
    else
    {
        depthTestFunctionPDE = depthTestFunction = depth test function;
    }
    SetValue("Depth_Test_Function_PDE", depthTestFunctionPDE);
    SetValue("Depth_Test_Function", depthTestFunction);
}
```

- PDE_ Depth_Bound_Test = = DE_ Depth_Bound_Test
  Pseudo code:

```
{
    boundTestEnablePDE = GetValue("Depth_Bound_Test_Enable_PDE");
    boundTestEnable = GetValue("Depth_Bound_Test_Enable");
    assert(boundTestEnablePDE = = boundTestEnable);
}
```

- Stencil_Test_Enable = = Stencil_Test_Enable_PDE
  Pseudo code:

```
{
    stencilTestEnablePDE = GetValue("Stencil_Test_Enable_PDE");
    stencilTestEnable = GetValue("Stencil_Test_Enable");
    assert(stencilTestEnablePDE = = stencilTestEnable);
}
```

- depth_read_disable should **not be set** when either of the following holds:
  a) pp mode is on (bumpshader needs eomerge);
  b) bump is on (bumpshader needs eomerge);
  c) depth_test_function is not ALWAYS or NEVER

d) stencil_test_enable
e) bound_test_enable
f) Invisible_PixelPair_Drop_Enable = 1 (=> Depth_Read_Disable = 0)

Register definition:

0x254 [26 : 26] M2REG_DEPTH_READ_DISABLE, ReversedBoolean.

Pseudo code:

```
{
    ppMode = GetValue("Partial_Precision_PS2_Enable");
    bumpLoop = GetValue("Bump_Shader_Total_Loops");
    stencilEnable = GetValue("Stencil_Test_Enable");
    boundTestEnable = GetValue("Depth_Bound_Test_Enable");
    depthTestFuncPDE = GetValue("Depth_Test_Function_PDE");
    invPixelPairDrop = GetValue("Invisible_PixelPair_Drop_Enable");

    if(0 != ppMode)
        depthReadDisable = 0;
    if(0 != bumpLoop) // bumpLoop, zero means (1), color loop only
        depthReadDisable = 0;
    if(0 != stencilEnable)
        depthReadDisable = 0;
    if(0 != boundTestEnable)
        depthReadDisable = 0;
    if(7 != depthTestFuncPDE || 0 != depthTestFuncPDE)
        depthReadDisable = 0;
    if(0 != invPixelPairDrop)
        depthReadDisable = 0;

    SetValue("Depth_Read_Disable", depthReadDisable);
}
```

- **Color Sync Depth Disable**

  Register definition:

  0x278 [1 : 1] M2REG_COLOR_SYNC_DEPTH_DISABLE

  We might need a new register in OE to control sync between color_write_mask (from PBE) and stencil_write_mask (from PDE).

  in software side, this register is called M2REG_COLOR_SYNC_DEPTH_DISABLE (0x9e[1]).

  This bit's function is as following:

  ```
  if (0x9e[1] == 0)
      stencil_write_mask |= color_write_mask;
  ```

  This bit should be set to 1 when
  stencil_write_enable && sync_mode == hiz_mask_sync

  Otherwise, those pixels that fail z test can't write stencil -- which is not what we want.

  It's worth pointing out that when both alpha_test_enable and stencil_write_enable are on, sync_mode must be cov_mask_sync.

- If Stencil Write and texture engine is changing write mask, i.e.
  1) Alpha_Test_Enable
  2) Polygon_Stipple_Enable

3) Dest_Color_Key_Enable
4) Texture_Kill_Enable (need to check bump shader instruction)

Set COLOR_SYNC_DEPTH_DISABLE to 0.

Pseudo code:

```
{
    // is stencil test enable
    stencilTestEnable = GetValue("Stencil_Test_Enable");
    stencilTestEnablePDE = GetValue("Stencil_Test_Enable_PDE");
    if(stencilTestEnable == 0 && 0 == stencilTestEnablePDE)
        exit(0); // no stencil test, must no stencil write

    // is stencil write keep stencil buffer, on CW setting
    isCWStencilKEEP = 0;
    stencilFunc = GetValue("Stencil_Test_Function");
    sFail = GetValue("Stencil_Operation_SFail");
    sPzP = GetValue("Stencil_Operation_SPass_ZPass");
    sPzF = GetValue("Stencil_Operation_SPass_ZFail");
    sWMask = GetValue("Stencil_Write_Enables");
    if(0 == sWMask)
        isCWStencilKEEP = 1;
    else
    {
        if(0 == stencilFunc)//NEVER
        {
            isCWStencilKEEP = (0 == sFail);
        }
        else if(7 == stencilFunc)//ALWAYS
        {
            isCWStencilKEEP = (0 == sPzP) && (0 == sPzF);
        }
        else
        {
            isCWStencilKEEP = (0 == sFail) && (0 == sPzP) && (0 == sPzF);
        }
    }

    // is stencil write keep stencil buffer, on CCW setting
    isCCWStencilKEEP = 0;
    stencilFuncCCW = GetValue("Stencil_Test_Function_CCW");
    sFailCCW = GetValue("Stencil_Operation_SFail_CCW");
    sPzPCCW = GetValue("Stencil_Operation_SPass_ZPass_CCW");
    sPzFCCW = GetValue("Stencil_Operation_SPass_ZFail_CCW");
    sWMaskCCW = GetValue("Stencil_Write_Enables_CCW");
    if(0 == sWMaskCCW)
        isCCWStencilKEEP = 1;
    else
    {
        if(0 == stencilFuncCCW)//NEVER
        {
            isCCWStencilKEEP &= (0 == sFailCCW);
        }
        else if(7 == stencilFuncCCW)//ALWAYS
        {
            isCCWStencilKEEP &= (0 == sPzPCCW) && (0 == sPzFCCW);
        }
        else
        {
            isCCWStencilKEEP &= (0 == sFailCCW) && (0 == sPzPCCW) && (0 == sPzFCCW);
        }
```

```
        }

        // check the stencil setting base on cullMode
        isStencilAlwaysKEEP = 0;
        twoSidedStencil = GetValue("Two_Sided_Stencil_Enable");
        cullMode = GetValue("Culling_Mode");
        if(0 == twoSidedStencil || 2 == cullMode)
        // 0x2: Clockwise_Culling, pass down CCW triangles to RE, use CW stencil setting
        {
              isStencilAlwaysKEEP = isCWStencilKEEP;
        }
        else if(3 == cullMode)
        // 0x3: Counter_Clockwise_Culling, pass down CW triangles to RE, use CCW setting
        {
              isStencilAlwaysKEEP = isCCWStencilKEEP;
        }
        else if(1 == cullMode)
        // 0x1: No_Culling, default, need both CW and CCW test
        {
              isStencilAlwaysKEEP = isCWStencilKEEP && isCCWStencilKEEP;
        }

        // stencil write
        isStencilWrite = ! isStencilAlwaysKEEP;

        // is write mask changed?
        alphaTestEnable = GetValue("Alpha_Test_Enable");
        polygonStippleEnable = GetValue("Polygon_Stipple_Enable");
        destColorKeyEnable = GetValue("Destination_Color_Key_Enable");
        isTexKillEnable = IsTexCoordKillInBumpShader(); // any kill bump shader instruction?
        isTexEngineChangingWriteMask = alphaTestEnable || polygonStippleEnable ||
                                       destColorKeyEnable || isTexKillEnable;

        // color sync depth disable
        if( isTexEngineChangingWriteMask && isStencilWrite )
              colorSyncDepthDisable = 0;

        SetValue("Color_Sync_Depth_Disable", colorSyncDepthDisable);
  }
```

- About Sync_Mode (0x250_D0_23) Setting.
  We have to always set Sync_Mode to HiZ_Mask_Sync(0).
  For cases that still use Cov_Mask_Sync(1), please change to HiZ_Mask_Mode by disabling z comparion in Depth Engine. But please note **one exception, if pixel shader is writing z, only cov_mask_sync can be used.**

  Disabling z comparion in Depth Engine:
  a) Set Depth function to always. Depth_Test_Function (0x250_D0_19_32) = Always(7)
  b) Disable Depth read. Depth_Read_Disable (0x250_D1_26) = True.

- About Z_Biased_Clip_Far and Z_Biased_Clip_Near Setting.
  In register definition:
  Z_Bias_Clip_Near: IFF, 1.8.23
  Z_Bias_Clip_Far: IFF, 1.8.23

  But you can not set it as a negative data for the following HD issue:
  If newZ >= 0 and Z_Bias_Clip_Near >=0,

All right.
If newZ < 0 and Z_Bias_Clip_Near >=0,
All right
If newZ >= 0 and Z_Bias_Clip_Near <0,
All right
If newZ < 0 and Z_Bias_Clip_Near <0,
The result is wrong for HD issue.

So we can not set Z_Bias_Clip_Near as a small negative data to avoid the Depth_Clip_Range issue, and we need to patch it like the following:
a) For Z Buffer case (Depth_Exponent_Bits = DEB_0_Bit, 24bit Z format), HD define the Clip_Range as [0,1), ZBiasNear <= NewZ < ZBiasFar, do the following patch to get result of clip_range [0,1],
Z_Bias_Clip_Near = 0x0;
Z_Bias_Clip_Far = 0x1;
newZ = newZ * 0.9999..
b) For 1-Z Buffer case (Depth_Exponent_Bits = DEB_5_Bit, 5.19 Z format), HD define the Clip_Range as (0,1], ZBiasNear < NewZ <= ZBiasFar, do the following patch to get result of clip_range [0,1],
Z_Bias_Clip_Near = 0x0;
Z_Bias_Clip_Far = 0x1;
newZ = newZ * 0.9999 + 0.0001.

- How to know Depth Read?
Pseudo code:

```
{
    isDepthRead = true;
    deFunction = GetValue("Depth_Test_Function");
    pdeFunction = GetValue("Depth_Test_Function_PDE");
    if( 0 == deFunction ||
       (7 == deFunction && (0 == pdeFunction || 7 == pdeFunction)) )
    {
        isDepthRead = false;
    }
}
```

- How to know Depth Write?
Pseudo code:

```
{
    isDepthWrite = true;
    depthWriteEnable = GetValue("Enable_Depth_Write");
    if(depthWriteEnable == 0)
        isDepthWrite = false;
    deFunction = GetValue("Depth_Test_Function");
    pdeFunction = GetValue("Depth_Test_Function_PDE");
    if(0 == deFunction)
    {
        isDepthWrite = false;
    }
    if(7 == deFunction && 0 == pdeFunction)
    {
        isDepthWrite = false;
    }
}
```

- How to know Stencil Read?
  Pseudo code:

```
{
    stencilEnable = GetValue("Stencil_Test_Enable");
    stencilEnablePDE = GetValue("Stencil_Test_Enable_PDE");
    if(0 == stencilEnable && 0 == stencilEnablePDE)
    {
        isStencilRead = false;
    }

    bool cwRead = false;
    bool ccwRead = false;
    cullMode = GetValue("Culling_Mode");
    if(cullMode == 3) //0x3: Counter_Clockwise_Culling
    {
        cwRead = false;
    }
    else
    {
        stencilTestFunction = GetValue("Stencil_Test_Function");
        if(7 != stencilTestFunction && 0 != stencilTestFunction)
        {
            cwRead = true;
        }
        else
        {
            stencilOpSPZP = GetValue("Stencil_Operation_SPass_ZPass");
            stencilOpSPZF = GetValue("Stencil_Operation_SPass_ZFail");
            stencilOpSF = GetValue("Stencil_Operation_SFail");
            if(7 == stencilTestFunction) // ALWAYS
            {
                if(stencilOpSPZP > 2 || stencilOpSPZF > 2)
                {
                    cwRead = true;
                }
            }
            else // NEVER
            {
                if(stencilOpSF > 2)
                {
                    cwRead = true;
                }
            }
        }
    }

    twoSidedStencil = GetValue("Two_Sided_Stencil_Enable");
    if(0 == twoSidedStencil || 2 == cullMode)
    {
        ccwRead = false;
    }
    else
    {
        stencilTestFunction = GetValue("Stencil_Test_Function_CCW");
        if(7 != stencilTestFunction && 0 != stencilTestFunction)
        {
            ccwRead = true;
        }
        else
```

```
            {
                stencilOpSPZP = GetValue("Stencil_Operation_SPass_ZPass_CCW");
                stencilOpSPZF = GetValue("Stencil_Operation_SPass_ZFail_CCW");
                stencilOpSF = GetValue("Stencil_Operation_SFail_CCW");
                if(7 == stencilTestFunction) // ALWAYS
                {
                    if(stencilOpSPZP > 2 || stencilOpSPZF > 2)
                    {
                        ccwRead = true;
                    }
                }
                else // NEVER
                {
                    if(stencilOpSF > 2)
                    {
                        ccwRead = true;
                    }
                }
            }
        }

        isStencilRead = (cwRead || ccwRead);
    }
```

- How to know Stencil Write?
  Pseudo code:

```
{
    stencilWrite = GetValue("Stencil_Write_Enables");
    twoSideStencil = GetValue("Two_Sided_Stencil_Enable");
    stencilWriteCCW = GetValue("Stencil_Write_Enables_CCW");
    isStencilWrite = (twoSideStencil == 0) ? (0 != stencilWrite) :
                                             ((0 != stencilWrite) && (0 != stencilWriteCCW));
    if(isStencilWrite)
        exit(0);

    // advanced checking
    // is CW stencil keep
    isCWStencilKeep = 0;
    stencilFunc = GetValue("Stencil_Test_Function");
    stencilTestEnable = GetValue("Stencil_Test_Enable");
    sFail = GetValue("Stencil_Operation_SFail");
    sPzP  = GetValue("Stencil_Operation_SPass_ZPass");
    sPzF  = GetValue("Stencil_Operation_SPass_ZFail");
    sWMask= GetValue("Stencil_Write_Enables");

    if(0 == stencilTestEnable)
        isCWStencilKeep = 1;
    if(0 == sWMask)
        isCWStencilKeep = 1;
    if(0 == stencilFunc)//NEVER
    {
        isCWStencilKeep = (0 == sFail);
    }
    else if(7 == stencilFunc)//ALWAYS
    {
        isCWStencilKeep = (0 == sPzP) && (0 == sPzF);
    }
    else
    {
        isCWStencilKeep = (0 == sFail) && (0 == sPzP) && (0 == sPzF);
```

```
        }

        // is CCW stencil keep
        isCCWStencilKeep = 0;
        stencilFuncCCW = GetValue("Stencil_Test_Function_CCW");
        stencilTestEnable = GetValue("Stencil_Test_Enable");
        twoSidedStencilEnable = GetValue("Two_Sided_Stencil_Enable");
        sFailCCW = GetValue("Stencil_Operation_SFail_CCW");
        sPzPCCW    = GetValue("Stencil_Operation_SPass_ZPass_CCW");
        sPzFCCW    = GetValue("Stencil_Operation_SPass_ZFail_CCW");
        sWMaskCCW= GetValue("Stencil_Write_Enables_CCW");

        if(0 == stencilTestEnable)
              isCCWStencilKeep = 1;
        if(0 == twoSidedStencilEnable)
              isCCWStencilKeep = 1;
        if(0 == sWMaskCCW)
              isCCWStencilKeep = 1;
        if(0 == stencilFuncCCW)//NEVER
        {
              isCCWStencilKeep &= (0 == sFailCCW);
        }
        else if(7 == stencilFuncCCW)//ALWAYS
        {
              isCCWStencilKeep &= (0 == sPzPCCW) && (0 == sPzFCCW);
        }
        else
        {
              isCCWStencilKeep &= (0 == sFailCCW) && (0 == sPzPCCW) && (0 == sPzFCCW);
        }

        // is stencil keep
        isStencilAlwaysKEEP = 0;
        cullMode = GetValue("Culling_Mode");
        if(0 == twoSidedStencil)
        {
              isStencilAlwaysKEEP = isCWStencilKeep;
        }
        switch(cullMode)
        {
             case 2: //CW
              isStencilAlwaysKEEP = isCWStencilKeep;
             case 3: //CCW
              isStencilAlwaysKEEP = isCCWStencilKeep;
             case 1:
             default:
              isStencilAlwaysKEEP = isCWStencilKeep && isCCWStencilKeep;
        }

        isStencilWrite = !isStencilAlwaysKEEP;
    }
```

15.7. MRT Setting

- Page_Size_Mode restricted by MRT targets.
  Total_Targets_Elements = 0 => Page_Size_Mode = 0 (16x16)
  Total_Targets_Elements = 1 => Page_Size_Mode = 1 (16x8)
  Total_Targets_Elements = 2 => Page_Size_Mode = 2 (16x4)

Total_Targets_Elements = 3 => Page_Size_Mode = 2 (16x4)

- No PP mode for MRT/MET.
  Pseudo code:

```
{
    MRT = GetValue("Total_Targets_Elements");
    if (MRT)
        PPmode= 0;
    SetValue("Partial_Precision_PS2_Enable", PPmode);
}
```

- No color compression when MRT/MET.
  Pseudo code:

```
{
    MRT = GetValue("Total_Targets_Elements");
    if (MRT)
        ColorCompression = 0;
    SetValue("Color_Compression_Enable", ColorCompression);
}
```

- Always enable bitmask feature for all available render targets.

- Color Read bits always enable for all available render targets.

- If MRT, no invisible pixel pair drop is enabled.
  Pseudo code:

```
{
    MRT = GetValue("Total_Targets_Elements");
    If(MRT)
        invPixPairDrop = 0;
    SetValue("Invisible_PixelPair_Drop_Enable", invPixPairDrop);
}
```

16.Programming of Query

HW always counts some statistical numbers such as visible pixel count when run the frame. If provide proper register setting, HW can export these statistic according to the SW query.

- Query register

```
Register
0x2E0 (W)       Query_Statistic_Write
D0 [23:8]  Extra_Query_Type, Integer.
D0 [7:4]    Query_Type, Integer.
// {
//      0:     Other_Group.                  //No query
//      1:     Pixels_Count.
//      2:     Engine_Clocks.
//      3:     Color_Read_Request.      //unit 128
//      4:     Color_Write_Request.     //unit 128
//      5:     Z_Read_Request.                 //unit 128
//      6:     Z_Write_Request.           //unit 128
//      7:     Texture_Read_Request.   //unit 256
// } Query_Type
D0 [3]              Clear_Counter_After_Query,    Boolean.
D0 [2]              Clear_All_Counters,    Boolean.
D0 [1:0]    Query_Command_Type, Integer.
// {
//      0:     Source_From_Query
//      1:     Source_From_D2
//      2:     Source_From_D2D3
//      3:     Reserced
// } Query_Command_Type
D1 [31]             Dest_Memory_Port, Enumerate.
{
        0:     Local_FB. // Frame Buffer 1 (Local Frame Buffer)
        1:     PCI_E.              // Frame Buffer 0 or PCI_E
} DestMemoryPort
D1 [29:0]  Dest_Memory_Address, Integer. // 32 bits alignment
D2 [31:0]  Query_Source0, Integer.
D3 [31:0]  Query_Source1, Integer.
```

- Write Query Batch

There are three kinds of Query Batch:
. Clear Counter
. Query Write
. Stop Write

If a batch is a Query Batch, following registers should be set in the batch command list:
. Register 0x270, Misc_Operation

```
        D2 [29]             Conversion_Blit_Enable, Boolean.
        Set 0x270 :: D2 [29:29] = 0x1;                  // blit batch
```

. Register 0x2C0, Conversion_Blit_Rectangle

```
        D1 [31]             Blit_Fill_Mode_Enable, Boolean.
        {
                = 0 : do Blit.
                = 1 : do query
        }
```

Set 0x2c0 :: D1[31:31] = 0x1; // do query

- To Clear Counter

  Set 0x270 :: D2 [29:29] = 0x1; // blit batch
  Set 0x2c0 :: D1[31:31] = 0x1; // do query
  Set 0x2e0 :: D0 [7:4] = the query type you want to clear
  Set 0x2e0 :: D0 [3] = 0x1; // will clear the query type set in D0[7:4]
  Set 0x2e0 :: D0 [2] = 0x1; // will clear all query types
  Set 0x2e0 :: D0 [1:0] = 0x1 or 0x2; // will init the output data with “query_source0” or “query_source0 and query_source1”
  Set 0x2e0 :: D1 [31] = dest_memory_port
  Set 0x2e0 :: D1 [29:0] = dest_memory_address // usually this is the same value where next Query Write batch use
  Set 0x2e0 :: D2 [31:0] = the value you want to init the output data
  Set 0x2e0 :: D3 [31:0] = the value you want to init the output data if source come from this DWORD

- To Query Write

  Set 0x2e0 :: D0 [7:4] = the query type you want to write
  Set 0x2e0 :: D0 [3] = 0x0;
  Set 0x2e0 :: D0 [2] = 0x0;
  Set 0x2e0 :: D0 [1:0] = 0x0; // the output data is come from Query
  Set 0x2e0 :: D1 [31] = dest_memory_port
  Set 0x2e0 :: D1 [29:0] = dest_memory_address
  Set 0x2e0 :: D2 [31:0] = 0x0
  Set 0x2e0 :: D3 [31:0] = 0x0

- To Stop Write

  Set 0x2e0 :: D0 [7:4] = 0x0; // No query
  Set 0x2e0 :: D0 [3] = 0x0;
  Set 0x2e0 :: D0 [2] = 0x0;
  Set 0x2e0 :: D0 [1:0] = 0x0;
  Set 0x2e0 :: D1 [31] = dest_memory_port
  Set 0x2e0 :: D1 [29:0] = dest_memory_address
  Set 0x2e0 :: D2 [31:0] = 0x0
  Set 0x2e0 :: D3 [31:0] = 0x0

17.Important Settings for Performance

17.1. Invisible Tile / Pixel Pair Drop

In general, it is possible to drop invisible tiles always before texturing and PS operation to speed up rendering because on average it will have 30~50% invisible tiles. Set it to HiZ_Mask_Sync mode to do so.

0x250 D0 [23]Sync_Mode = 0: HiZ_Mask_Sync.

But when PS depth case and depth/stencil test == ALWAYS, please set Sync_Mode = 1: Cov_Mask_Sync to save pipeline latency.

In addition, we have a register 0x300 D0[15] = Invisible_Pixel_Pair_Drop_Enable to speed up when heavy texturing or PS operations.

17.2. Disable Stencil Read

When stencil test is ALWAYS or NEVER and stencil operation is KEEP, ZERO or REPLACE etc, driver can set 0x250 D2[14] = StencilReadDisable to save bandwidth of stencil buffer read.

17.3. Band+Tile Texture Buffer

XP10 texture engine is optimized to the following texture buffer formats:

- Band32+Tiled4x4
- Tiled4x4
- Band64

So driver needs to transfer any linear buffer to Band32+Tiled4x4. For this purpose, XP10 3D color buffer and 2D blit support (Band32+Tiled4x4) also but DXTn and YUV textures. CPU needs to transfer DXTn (already tiled4x4) to Band32+tiled4x4. Use 2D blit to transfer YUV textures to band32_tiled4x4 by setting color format = 16bpp.

Band64 is XP10 main render target format for display.

For YUV textures, in performance mode, please do not set YUV_DCT enable.

17.4. Speed up Alpha Blending

Sometimes, destination color is not necessary for alpha blending, Driver should disable destination color read.

0x260 D1 [10]= Destination_Color_Buffer_Read_Enable, Boolean. For main target
When alpha blending, ROP3 and bit mask operation do not need destination color, then do not set it. Otherwise set it. HW will read destination color based on this bit and tile's visibility flag. If this bit is zero, HW will not read destination color and just use 0x00000000 for all operations requiring destination color.

0x270 D2 [26] = Destination_Color_Buffer_Read_Enable3, Boolean.
0x270 D2 [25] = Destination_Color_Buffer_Read_Enable2, Boolean.
0x270 D2 [24] = Destination_Color_Buffer_Read_Enable1, Boolean.

Try to use constant color blend for some tinting effects.

17.5. Normal Setting

- 0x130 (R/W) GEOMETRY_ENVIRONMENT

D0 [19] Vertex_Cache_Clear, Boolean = 0 if vertex buffer not changed in multi-batches
D1 [11] Vertex_Cache_Configuration_In_VS, Enumerate = 0 if number of texture pairs <= 2
D1 [9] Disable_PA_Fast_Rejection, ReversedBoolean = 0
D1 [2:2] User_Clip_Code_Valid, Integer = 1

- 0x210 (R/W) RE_Window_Clip

D2 [31:28] Page_Max_Prim_Threshold, Integer = (D1 [29:28] Parameter_Bank_Configuration-4)/2
D2 [15:12] Page_Min_Prim_Threshold, Integer = 2;
D3 [30:27] Max_Page_Threshold, Integer = 0xf (96 pages)

- 0x220 (R/W) RASTER_MODE

D0 [22] PageBased_Rendering_Enable, Boolean
(if 0x210 D3 [31] Line_Stipple_Enable!=1)
D0 [18] Flush_NoSorting_Enable = 0
D0 [15] Force_Always_Flush_Page, Boolean = 0
D0 [14] Always_FullPage_Enable, Boolean = 0 // OE always pre-fetch full page

- 0x240 (W) Depth_Constant_Values

D0 [15:8] Stencil_Write_Enables_CCW, Integer = 0(set by app)

- 0x250 (R/W) Depth_Buffer_Settings

D0 [23] Sync_Mode, Enumerate = HiZ_Mask_Sync //default is cov_mask_sync
// 0x260 D1 [6] Alpha_Test_Enable, Boolean == false
// &&
// Instruction TextureSampleState-> D2 [31] TEX_KILL_Enable ==false
// &&
// 0x260 D1 [20] Destination_Color_Key_Enable == false

D0 [18] Stencil_Test_Enable, Boolean == D3 [27] Stencil_Test_Enable_PDE
D0 [17:16] Depth_Buffer_Configuration, Enumerate = 1(Band16_OneTile_Z_Only)
D0 [13] Enable_Depth_Write, Boolean //when only rendering color
D1 [27] DE_Invisible_Tile_Drop = 0 when
D3 [27] Stencil_Test_Enable_PDE, Boolean = 1
DE_Invisible_Tile_Drop = 1 otherwise

D1 [26] Depth_Read_Disable, ReversedBoolean = 1 //default is 0
//[1] 0x250 D2 [2:0] Stencil_Test_Function != ALWAYS || !=NEVER
// &&
//[2] 0x250 D0 [21:19] Depth_Test_Function!=ALWAYS && !=NEVER) ||
// 0x250 D2 [15] Depth_Bound_Test_Enable, Boolean
// &&
// [3] 0x300 (R/W) D2 [14:12] Bump_Shader_Total_Loops_SC==0

// OR~~~~

// 0x250 D2 [8:6] Stencil_Operation_SPass_ZFail==KEEP
// &&
// 0x250 D2 [5:3] Stencil_Operation_SFail == KEEP
// &&
// 0x270 D3 [11:9] Stencil_Operation_SPass_ZPass_CCW == KEEP
// &&
// 0x270 D3 [8:6] Stencil_Operation_SPass_ZFail_CCW ==KEEP
// 0x270 D3 [2:0] Stencil_Test_Function_CCW == KEEP

D2 [13] Use_Stencil_TestMask_ForBumpSync_Enable = 1
D3 [28] Depth_Bound_Test_Enable_PDE == D2 [15] Depth_Bound_Test_Enable
D3 [26:24] Depth_Test_Function_PDE, Enumerate == D0 [21:19] Depth_Test_Function

- 0x270 (W) Misc_Operation

D0 [1] Cache_Forward_Enable, Boolean = 1
D0 [0] Page_Cache_Enable, Boolean = 0
D1 [27] Color_Prefetch_Enable, Boolean = 0 when don't need read color from destination
D1 [23] Depth_Prefetch_Enable, Boolean = 0 when don't need read Z from destination
// 0x250 D0 [21:19] Depth_Test_Function==ALWAYS || ==NEVER
D1 [21] Stop_Waiting_When_EndOfMerging, Boolean = 0
D2 [2] EOMerge_Generation_Mode, Enumerate = 0 (Necessarily) //default is 1

- 0x280 (R/W) COLOR_BUFFER

D0 [13:12] Banded_Tiled_Mode, Enumerate = 2 // 64*16 pixel banded. when color compression on (0x270_D1_15), it means banndedcompression mode

- 0x300 (R/W) Bump_Shader_Setting

D0 [8:4] Bump_Shader_Maximum_Tiles_For_Loop, Integer =
// (int) (32/max_bumptexture_number in all bumploops) – 1; when PS mode 2, it should be 0.
D0 [15] Invisible_PixelPair_Drop_Enable, Boolean = 1 // when PS MODE ==0
// except: 0x250 D0 [22] Depth_Source, Enumerate = 0x1 (pixel shader)

- 0x380 (R/W) PixelShader_SETTING

D0 [3] Partial_Precision_PS2_Enable = 1(except: 0x280 D0 [15:14] Total_Targets_Elements > 0 , PS mode == 2)

- Instruction TextureSampleState

D4 [17:16] Texture_Band_Tile_Mode, Enumerate, BandTileMode = 0x3

17.6. Clear Screen Setting

Const Rendering

0x220 (R/W) RASTER_MODE D0 [17] Constant_Color_Render_Enable==true
0x270 (W) Misc_Operation
D2 [23:20] Component_Mask, Integer = 0xf

Blit Fill

- 0x2C0 (W) Conversion_Blit_Rectangle

D1 [31] Blit_Fill_Mode_Enable, Boolean == true

D0 [23:12] Source_Ystart, Integer = 0
D0 [11:0] Source_Xstart, Integer = 0
D1 [23:12] RECT_Height, Integer = height
D1 [11:0] RECT_Width, Integer = pitch
D2 [23:12] Dest_Ystart, Integer = 0
D2 [11:0] Dest_Xstart, Integer = 0

- 0x270 (W) Misc_Operation

D2 [29] Conversion_Blit_Enable, Boolean = true
D2 [19:18] Conversion_Blt_Code, Enumerate =0 (CompressColorBuffer)

- 0x220 (R/W) RASTER_MODE

D0 [22] PageBased_Rendering_Enable, Boolean = 0