Chapter 13

Pixel Shaders

"The tree casts its shade upon all,
even upon the woodcutter."

Sanskrit Proverb

"To sit in the shade on a fine day and look upon verdure is
the most perfect refreshment."

Jane Austen: *Mansfield Park*, IX, 1814

13.1 Overview

In the same way that vertex shader programs replace fixed-function vertex processing, pixel shader programs replace fixed-function pixel processing. The multitexture cascade is entirely replaced by a pixel shader. The specular addition, fog and frame buffer processing are unaffected by pixel shaders.

With fixed-function pixel processing, the application sets texture stage state variables for each texture stage and attempts to arrange the per-pixel processing into the restricted dataflow of the multitexture cascade. With a pixel shader, a small program is written that samples the textures from the interpolated texture coordinates and combines those samples to produce a color that is passed on to the frame buffer for further processing.

Direct3D exposes several different pixel shader architecture versions in DirectX 9.0c. We will examine the register model, instruction execution model and instruction set for each of the architecture versions. As with vertex shaders, the pixel shader is defined by an array of tokens, one token per instruction. Most applications will not construct token arrays directly, but use D3DX to assemble text instructions into tokens. We will use the text form of instructions in this chapter. The shader tokens are described in appendix C. We will close the chapter with sample pixel shaders implementing typical effects.

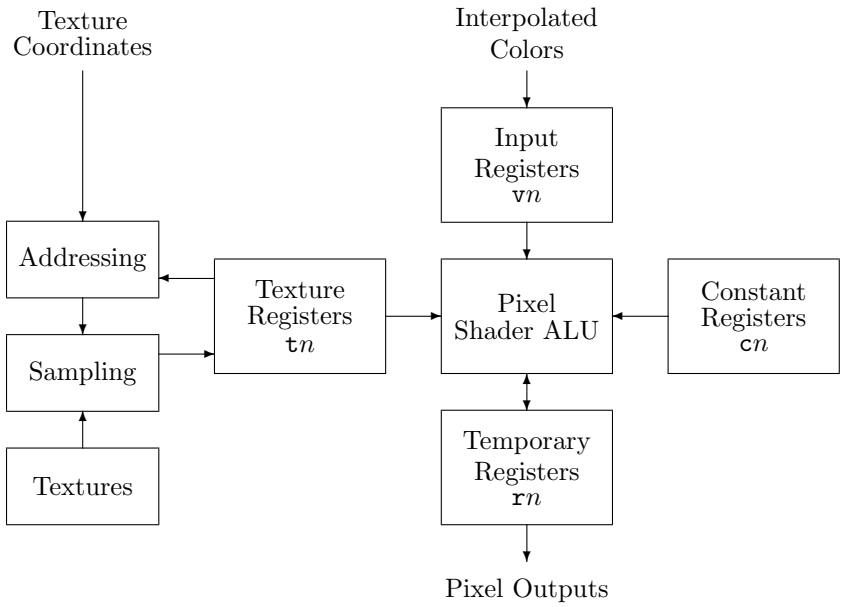

Figure 13.1: Pixel shader architecture overview.

13.2 Pixel Shader Architecture

An overview of the pixel shader architecture is shown in figure 13.1. The general architecture of a pixel shader is similar in all the pixel shader versions exposed in DirectX 9.0c, but differences in the instruction set and register files exist between the different versions. A summary of the differences between pixel shader architecture verions is given in table 13.1. The shader itself consists of a small sequence of instructions which are executed in order. No branching or looping is provided in any version of the instruction set.

A pixel shader must perform two distinct tasks: describe how textures are addressed, or sampled, and combine the samples to produce an output color. Although all pixel shaders perform these tasks, the details are slightly different depending on the pixel shader version.

13.3 Pixel Shader Architecture Versions

13.3.1 Pixel Shader 1.0

Version 1.0 of the pixel shader architecture includes instructions for arithmetic operations and instructions for texture addressing. In general, the arithmetic instructions can use any register for the source operands and any temporary register for the destination operand. The one exception to these general rules

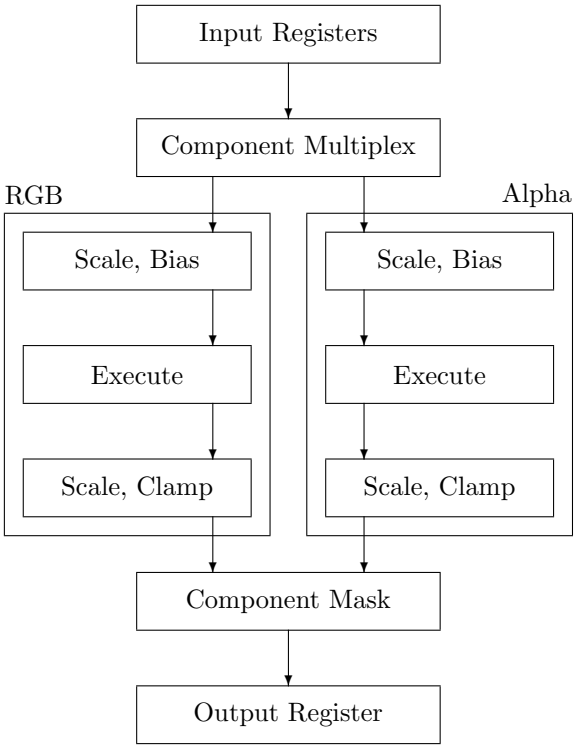

Figure 13.2: Pixel shader instruction execution.

	Pixel Shader Version					
					1.4	
Feature	**1.0**	**1.1**	**1.2**	**1.3**	**Phase 1**	**Phase 2**
arithmetic instructions	8	8	8	8	8	8
texture address instructions	4	4	4	4	6	6
total instructions	8	12	12	12	14	14
constant registers (`c`n)	8	8	8	8	8	
temporary registers (`r`n)	2	2	2	2	6	
texture registers (`t`n)	4	4	4	4	6	
color registers (`v`n)	2	2	2	2	0	2

Table 13.1: Pixel shader architecture versions.

	Availability				
Instruction	**1.0**	**1.1**	**1.2**	**1.3**	**1.4**
`add`	Yes	Yes	Yes	Yes	Yes
`bem`					Yes
`cmp`			Yes	Yes	Yes
`cnd`	Yes	Yes	Yes	Yes	Yes
`def`	Yes	Yes	Yes	Yes	Yes
`dp3`	Yes	Yes	Yes	Yes	Yes
`dp4`			Yes	Yes	Yes
`lrp`	Yes	Yes	Yes	Yes	Yes
`mad`	Yes	Yes	Yes	Yes	Yes
`mov`	Yes	Yes	Yes	Yes	Yes
`mul`	Yes	Yes	Yes	Yes	Yes
`nop`	Yes	Yes	Yes	Yes	Yes
`phase`					Yes
`ps`	Yes	Yes	Yes	Yes	Yes
`sub`	Yes	Yes	Yes	Yes	Yes
`tex`	Yes	Yes	Yes	Yes	
`texbem`	Yes	Yes	Yes	Yes	
`texbeml`	Yes	Yes	Yes	Yes	
`texcoord`	Yes	Yes	Yes	Yes	
`texcrd`					Yes
`texdepth`					Yes
`texdp3`			Yes	Yes	
`texdp3tex`			Yes	Yes	
`texkill`	Yes	Yes	Yes	Yes	Yes
`texld`					Yes
`texm3x2depth`				Yes	
`texm3x2pad`	Yes	Yes	Yes	Yes	
`texm3x2tex`	Yes	Yes	Yes	Yes	
`texm3x3`			Yes	Yes	
`texm3x3pad`	Yes	Yes	Yes	Yes	
`texm3x3spec`	Yes	Yes	Yes	Yes	
`texm3x3tex`	Yes	Yes	Yes	Yes	
`texm3x3vspec`	Yes	Yes	Yes	Yes	
`texreg2ar`	Yes	Yes	Yes	Yes	
`texreg2gb`	Yes	Yes	Yes	Yes	
`texreg2rgb`			Yes	Yes	

Table 13.2: Pixel shader instruction availability.

Instruction	Syntax	Meaning
`add`	d, s_0, s_1	component addition
`bem`	d.`rg`$, s_0, s_1$	bump environment map
`cmp`	d, s_0, s_1, s_2	compare to 0.0
`cnd`	d, s_0, s_1, s_2	compare to 0.5
`def`	d, v_0, v_1, v_2, v_3	constant definition
`dp3`	d, s_0, s_1	dot product
`dp4`	d, s_0, s_1	dot product
`lrp`	d, s_0, s_1, s_2	linear interpolation
`mad`	d, s_0, s_1, s_2	multiply and add
`mov`	d, s	register copy
`mul`	d, s_0, s_1	component multiply
`nop`		no operation
`phase`		instruction phase
`ps`	$.major.minor$	shader version
`sub`	d, s_0, s_1	component subtraction
`tex`	d	sample texture
`texbem`	d, s	bump environment map
`texbeml`	d, s	bumpmap with luminance
`texcoord`	d	texture is coordinate
`texcrd`	d, s	texture is coordinate
`texdepth`	d	compute pixel depth
`texdp3`	d, s	texture dot product
`texdp3tex`	d, s	dot product with lookup
`texkill`	s	kill source pixel
`texld`	d, s	sample from register
`texm3x2depth`	d, s	compute pixel depth
`texm3x2pad`	d, s	partial matrix product
`texm3x2tex`	d, s	final matrix product
`texm3x3`	d, s	final matrix product
`texm3x3pad`	d, s	partial matrix product
`texm3x3spec`	d, s_0, s_1, s_2	reflection vector lookup
`texm3x3tex`	d, s	final matrix product
`texm3x3vspec`	d, s	variable reflection lookup
`texreg2ar`	d, s	dependent texture lookup
`texreg2gb`	d, s	dependent texture lookup
`texreg2rgb`	d, s	dependent texture lookup

Table 13.3: Pixel shader instruction set summary.

Pixel Shader 1.0 Register Restrictions

Instruction	**Operand**	cn	rn	tn	vn
add d, s_0, s_1	d		*		
	$s_0,\ s_1$	*	*	*	*
cnd d, s_0, s_1, s_2	d		*		
	s_0		r0.a		
	$s_1,\ s_2$	*	*	*	*
dp3 d, s_0, s_1	d		*		
	$s_0,\ s_1$	*	*	*	*
lrp d, s_0, s_1, s_2	d		*		
	$s_0,\ s_1,\ s_2$	*	*	*	*
mad d, s_0, s_1, s_2	d		*		
	$s_0,\ s_1,\ s_2$	*	*	*	*
mov d, s	d		*		
	s	*	*	*	*
mul d, s_0, s_1	d		*		
	$s_0,\ s_1$	*	*	*	*
sub d, s_0, s_1	d		*		
	$s_0,\ s_1$	*	*	*	*
tex d	d			*	
texbem d, s	$d,\ s$			*	
texbeml d, s	$d,\ s$			*	
texcoord d	d			*	
texkill s	s			*	
texm3x2pad d, s	$d,\ s$			*	
texm3x2tex d, s	$d,\ s$			*	
texm3x3pad d, s	$d,\ s$			*	
texm3x3spec d, s_0, s_1, s_2	$d,\ s_0,\ s_1$			*	
	s_2	*			
texm3x3tex d, s	$d,\ s$			*	
texm3x3vspec d, s	$d,\ s$			*	
texreg2ar d, s	$d,\ s$			*	
texreg2gb d, s	$d,\ s$			*	

Table 13.4: Pixel shader 1.0 register restrictions.

Pixel Shader 1.1 Register Restrictions

Instruction	Operand	cn	rn	tn	vn
add d, s_0, s_1	d		*	*	
	s_0, s_1	*	*	*	*
cnd d, s_0, s_1, s_2	d		*	*	
	s_0		r0.a		
	s_1, s_2	*	*	*	*
dp3 d, s_0, s_1	d		*	*	
	s_0, s_1	*	*	*	*
lrp d, s_0, s_1, s_2	d		*	*	
	s_0, s_1, s_2	*	*	*	*
mad d, s_0, s_1, s_2	d		*	*	
	s_0, s_1, s_2	*	*	*	*
mov d, s	d		*	*	
	s	*	*	*	*
mul d, s_0, s_1	d		*	*	
	s_0, s_1	*	*	*	*
sub d, s_0, s_1	d		*	*	
	s_0, s_1	*	*	*	*
tex d	d			*	
texbem d, s	d, s			*	
texbeml d, s	d, s			*	
texcoord d	d			*	
texkill s	s			*	
texm3x2pad d, s	d, s			*	
texm3x2tex d, s	d, s			*	
texm3x3pad d, s	d, s			*	
texm3x3spec d, s_0, s_1, s_2	d, s_0, s_1			*	
	s_2	*			
texm3x3tex d, s	d, s			*	
texm3x3vspec d, s	d, s			*	
texreg2ar d, s	d, s			*	
texreg2gb d, s	d, s			*	

Table 13.5: Pixel shader 1.1 register restrictions.

Pixel Shader 1.2 Register Restrictions

Instruction	Operand	cn	rn	tn	vn
add d, s_0, s_1	d		*	*	
	s_0, s_1	*	*	*	*
cmp d, s_0, s_1, s_2	d		*	*	
	s_0, s_1, s_2	*	*	*	*
cnd d, s_0, s_1, s_2	d		*	*	
	s_0		r0.a		
	s_1, s_2	*	*	*	*
dp3 d, s_0, s_1	d		*	*	
	s_0, s_1	*	*	*	*
dp4 d, s_0, s_1	d		*	*	
	s_0, s_1	*	*	*	*
lrp d, s_0, s_1, s_2	d		*	*	
	s_0, s_1, s_2	*	*	*	*
mad d, s_0, s_1, s_2	d		*	*	
	s_0, s_1, s_2	*	*	*	*
mov d, s	d		*	*	
	s	*	*	*	*
mul d, s_0, s_1	d		*	*	
	s_0, s_1	*	*	*	*
sub d, s_0, s_1	d		*	*	
	s_0, s_1	*	*	*	*
tex d	d			*	
texbem d, s	d, s			*	
texbeml d, s	d, s			*	
texcoord d	d			*	
texdp3 d, s	d, s			*	
texdp3tex d, s	d, s			*	
texkill s	s			*	
texm3x2pad d, s	d, s			*	
texm3x2tex d, s	d, s			*	
texm3x3 d, s	d, s			*	
texm3x3pad d, s	d, s			*	
texm3x3spec d, s_0, s_1, s_2	d, s_0, s_1			*	
	s_2	*			
texm3x3tex d, s	d, s			*	
texm3x3vspec d, s	d, s			*	
texreg2ar d, s	d, s			*	
texreg2gb d, s	d, s			*	
texreg2rgb d, s	d, s			*	

Table 13.6: Pixel shader 1.2 register restrictions.

Pixel Shader 1.3 Register Restrictions

Instruction	Operand	cn	rn	tn	vn
add d, s_0, s_1	d		*	*	
	s_0, s_1	*	*	*	*
cmp d, s_0, s_1, s_2	d		*	*	
	s_0, s_1, s_2	*	*	*	*
cnd d, s_0, s_1, s_2	d		*	*	
	s_0		r0.a		
	s_1, s_2	*	*	*	*
dp3 d, s_0, s_1	d		*	*	
	s_0, s_1	*	*	*	*
dp4 d, s_0, s_1	d		*	*	
	s_0, s_1	*	*	*	*
lrp d, s_0, s_1, s_2	d		*	*	
	s_0, s_1, s_2	*	*	*	*
mad d, s_0, s_1, s_2	d		*	*	
	s_0, s_1, s_2	*	*	*	*
mov d, s	d		*	*	
	s	*	*	*	*
mul d, s_0, s_1	d		*	*	
	s_0, s_1	*	*	*	*
sub d, s_0, s_1	d		*	*	
	s_0, s_1	*	*	*	*
tex d	d			*	
texbem d, s	d, s			*	
texbeml d, s	d, s			*	
texcoord d	d			*	
texdp3 d, s	d, s			*	
texdp3tex d, s	d, s			*	
texkill s	s			*	
texm3x2depth d, s	d, s			*	
texm3x2pad d, s	d, s			*	
texm3x2tex d, s	d, s			*	
texm3x3 d, s	d, s			*	
texm3x3pad d, s	d, s			*	
texm3x3spec d, s_0, s_1, s_2	d, s_0, s_1			*	
	s_2	*			
texm3x3tex d, s	d, s			*	
texm3x3vspec d, s	d, s			*	
texreg2ar d, s	d, s			*	
texreg2gb d, s	d, s			*	
texreg2rgb d, s	d, s			*	

Table 13.7: Pixel shader 1.3 register restrictions.

Pixel Shader 1.4 Register Restrictions: Phase 1

Instruction	Operand	cn	rn	tn	vn
add d, s_0, s_1	d		*		
	s_0, s_1	*	*		
bem d.rg, s_0, s_1	d		*		
	s_0	*	*		
	s_1		*		
cmp d, s_0, s_1, s_2	d		*		
	s_0, s_1, s_2	*	*		
cnd d, s_0, s_1, s_2	d		*		
	s_0, s_1, s_2	*	*		
dp3 d, s_0, s_1	d		*		
	s_0, s_1	*	*		
dp4 d, s_0, s_1	d		*		
	s_0, s_1	*	*		
lrp d, s_0, s_1, s_2	d		*		
	s_0, s_1, s_2	*	*		
mad d, s_0, s_1, s_2	d		*		
	s_0, s_1, s_2	*	*		
mov d, s	d		*		
	s	*	*		
mul d, s_0, s_1	d		*		
	s_0, s_1	*	*		
sub d, s_0, s_1	d		*		
	s_0, s_1	*	*		
texcrd d, s	d		*		
	s			*	
texld d, s	d		*		
	s			*	

Table 13.8: Pixel shader 1.4 register restrictions in phase 1.

Pixel Shader 1.4 Register Restrictions: Phase 2

Instruction	**Operand**	cn	rn	tn	vn
`add` d, s_0, s_1	d		$*$		
	$s_0,\ s_1$	$*$	$*$		$*$
`cmp` d, s_0, s_1, s_2	d		$*$		
	$s_0,\ s_1,\ s_2$	$*$	$*$		$*$
`cnd` d, s_0, s_1, s_2	d		$*$		
	$s_0,\ s_1,\ s_2$	$*$	$*$		$*$
`dp3` d, s_0, s_1	d		$*$		
	$s_0,\ s_1$	$*$	$*$		$*$
`dp4` d, s_0, s_1	d		$*$		
	$s_0,\ s_1$	$*$	$*$		$*$
`lrp` d, s_0, s_1, s_2	d		$*$		
	$s_0,\ s_1,\ s_2$	$*$	$*$		$*$
`mad` d, s_0, s_1, s_2	d		$*$		
	$s_0,\ s_1,\ s_2$	$*$	$*$		$*$
`mov` d, s	d		$*$		
	s	$*$	$*$		$*$
`mul` d, s_0, s_1	d		$*$		
	$s_0,\ s_1$	$*$	$*$		$*$
`sub` d, s_0, s_1	d		$*$		
	$s_0,\ s_1$	$*$	$*$		$*$
`texcrd` d, s	d		$*$		
	s			$*$	
`texdepth` d	d		`r5`		
`texkill` s	s		$*$	$*$	
`texld` d, s	d		$*$		
	s		$*$	$*$	

Table 13.9: Pixel shader 1.4 register restrictions in phase 2.

is the `cnd` instruction which can only use `r0.a` as the first source operand. The register restrictions for each PS1.0 instruction are summarized in table 13.4.

A variety of texture addressing instructions (`texXXX`) provide for the same sorts of dependent texture reads supported by the fixed-function pipeline. The generalized arithmetic and texture addressing instructions allow an application to go beyond the fixed-function pipeline scenarios to compute more complicated or customized effects for texture addressing and color computation.

13.3.2 Pixel Shader 1.1

Version 1.1 of the pixel shader architecture extends the 1.0 architecture to allow a larger total number of instructions, as shown in table 13.1. The destination of arithmetic instructions is extended to the texture registers `t`n. The register restrictions for each PS1.0 instruction are summarized in table 13.5.

13.3.3 Pixel Shader 1.2

Version 1.2 of the pixel shader architecture extends version 1.1 with the `cmp`, `dp4`, `texdp3`, `texdp3tex`, `texm3x3`, and `texreg2rgb` instructions. The register restrictions for each PS1.0 instruction are summarized in table 13.6.

13.3.4 Pixel Shader 1.3

Version 1.3 of the pixel shader architecture extends version 1.2 to include the `texm3x2depth` instruction. The register restrictions for each PS1.0 instruction are summarized in table 13.7.

13.3.5 Pixel Shader 1.4

Version 1.4 of the pixel shader architecture generalizes the concept of texture addressing and dependent texture reads through the use of the `phase` instruction. The phase instruction serves as an explicit marker between the computation of an address and the texture to be sampled at that address.

Explain PS 1.4 instruction organization: math, address, phase, math, address.

The allowed register usage in PS1.4 is different depending on the phase in which the instruction is located. The register restrictions for the two phases are given in table 13.8 and table 13.9.

In phase 1, arithmetic instructions can use constant and temporary registers as source operands and temporary registers as destination operands. Texture addressing instructions can use texture registers as source operands and temporary registers as destination operands.

In phase 2, arithmetic instructions can also use the color registers as source operands in addition to temporary and constant registers. The destination

operand for arithmetic instructions must be a temporary register. Texture addressing instructions in phase 2 can use temporary registers as source operands in addition to texture registers.

This organization allows texture coordinates for a dependent read to be computed in phase 1 and stored in a temporary register for use by a texture addressing instruction in phase 2. The phase marker simplifies the computation of a texture address and the sampling of textures so that only a handfull of texture addressing instructions are needed instead of a new texture addressing instruction for each possible dependent read scenario.

13.3.6 Pixel Shader 2.0

13.3.7 Pixel Shader 2.x

13.3.8 Pixel Shader 3.0

13.4 Instruction Set

Arithmetic instructions. Texture addressing instructions. Instruction notation.

Model of 1.0, 1.1, 1.2, 1.3 versus 1.4 model.

13.4.1 Declaration Instructions

Every pixel shader must declare the version of the architecture that will be used with the shader with the `ps` instruction. This instruction must be the first instruction in the shader. The *major* and *minor* operands give the major and minor version numbers, respectively, of the architecture used by the shader.

Pixel shader constant registers may also be bound at the time that a pixel shader is bound to the device with `SetPixelShader`. The `def` instruction defines the contents of a constant register with four scalar floating-point values v_0, v_1, v_2, and v_3.

The `def` instruction must appear after the version instruction and before any computation instructions. When D3DX parses a vertex shader definition, the `def` instruction results in a vertex shader declaration fragment which contains the constant definitions. In this sense, the `def` instruction is not a true instruction and merely a convenience for the programmer to define constants and code that uses those constants in the same file. Therefore, the `def` instruction does not count against the instruction count for a vertex shader definition. However, when using D3DX to assemble vertex shaders, the application must merge the resulting declaration fragment with the vertex shader declaration for the vertex data, or load the constant registers from the fragment.

13.4.2 Basic Arithmetic Instructions

`mov` d, s
$d \leftarrow s$
`add` d, s_0, s_1
$d \leftarrow \langle s_{0x} + s_{1x},\ s_{0y} + s_{1y},\ s_{0z} + s_{1z},\ s_{0w} + s_{1w} \rangle$
`sub` d, s_0, s_1
$d \leftarrow \langle s_{0x} - s_{1x},\ s_{0y} - s_{1y},\ s_{0z} - s_{1z},\ s_{0w} - s_{1w} \rangle$

`mul` d, s_0, s_1
$d \leftarrow \langle s_{0x}s_{1x},\ s_{0y}s_{1y},\ s_{0z}s_{1z},\ s_{0w}s_{1w} \rangle$
`mad` d, s_0, s_1, s_2
$d \leftarrow \langle s_{0x}s_{1x} + s_{2x},\ s_{0y}s_{1y} + s_{2y},\ s_{0z}s_{1z} + s_{2z},\ s_{0w}s_{1w} + s_{2w} \rangle$
`lrp` d, s_0, s_1, s_2
$d \leftarrow \langle f(s_{0x}, s_{1x}, s_{2x}),\ f(s_{0y}, s_{1y}, s_{2y}),\ f(s_{0z}, s_{1z}, s_{2z}),\ f(s_{0w}, s_{1w}, s_{2w}) \rangle$,
where $f(a, b, c) = ab + (1 - a)c$

`dp3` d, s_0, s_1
$d \leftarrow \langle f,\ f,\ f,\ f \rangle, \quad f = s_{0x}s_{1x} + s_{0y}s_{1y} + s_{0z}s_{1z}$
`dp4` d, s_0, s_1
$d \leftarrow \langle f,\ f,\ f,\ f \rangle, \quad f = s_{0x}s_{1x} + s_{0y}s_{1y} + s_{0z}s_{1z} + s_{0w}s_{1w}$

13.4.3 Comparison Instructions

`cmp` d, s_0, s_1, s_2
$d \leftarrow \langle f(s_{0x}, s_{1x}, s_{2x}),\ f(s_{0y}, s_{1y}, s_{2y}),\ f(s_{0z}, s_{1z}, s_{2z}),\ f(s_{0w}, s_{1w}, s_{2w}) \rangle$,
where $f(a, b, c) = \begin{cases} b, & a \geq 0 \\ c, & a < 0 \end{cases}$
`cnd` d, s_0, s_1, s_2
$d \leftarrow \begin{cases} s_1, & s_{0a} > 0.5 \\ s_2, & s_{0a} \leq 0.5 \end{cases}$
`cnd` d, s_0, s_1, s_2
$d \leftarrow \langle f(s_{0x}, s_{1x}, s_{2x}),\ f(s_{0y}, s_{1y}, s_{2y}),\ f(s_{0z}, s_{1z}, s_{2z}),\ f(s_{0w}, s_{1w}, s_{2w}) \rangle$,
where $f(a, b, c) = \begin{cases} b, & a > 0.5 \\ c, & a \leq 0.5 \end{cases}$

13.4.4 Basic Texture Addressing

`tex` d
$d \leftarrow T_d(u_d, v_d, w_d, q_d)$

`texcoord` d
$d \leftarrow \langle f(u_d),\ f(v_d),\ f(w_d),\ 1\rangle,$
where $f(x) = \begin{cases} 0, & x < 0 \\ x, & 0 \le x \le 1 \\ 1, & x > 1 \end{cases}$

`texcrd` d, s
$d \leftarrow \langle u_s,\ v_s,\ w_s,\ *\rangle$

`texld` d, s
$d \leftarrow T_d(u_s, v_s, w_s)$

`texdp3` d, s
$d \leftarrow \langle f,\ f,\ f,\ f\rangle,$
$f = \langle u_d,\ v_d,\ w_d\rangle \cdot \langle s_r,\ s_g,\ s_b\rangle$

`texdp3tex` d, s
$u' = \langle u_d,\ v_d,\ w_d\rangle \cdot \langle s_r,\ s_g,\ s_b\rangle$
$d \leftarrow T_d(u', 0, 0)$

`texkill` s
abort if $u_s < 0$ or $v_s < 0$ or $w_s < 0$

`texm3x2pad` d, s
$d \leftarrow \langle u_d,\ v_d,\ w_d\rangle \cdot \langle s_r,\ s_g,\ s_b\rangle$

`texm3x2tex` d, s
$u' = \langle u_{d-1},\ v_{d-1},\ w_{d-1}\rangle \cdot \langle s_r,\ s_g,\ s_b\rangle$
$v' = \langle u_d,\ v_d,\ w_d\rangle \cdot \langle s_r,\ s_g,\ s_b\rangle$
$d \leftarrow T_d(u', v')$

`texm3x2depth` d, s
$z = \langle u_{d-1},\ v_{d-1},\ w_{d-1}\rangle \cdot \langle s_r,\ s_g,\ s_b\rangle$
$w = \langle u_d,\ v_d,\ w_d\rangle \cdot \langle s_r,\ s_g,\ s_b\rangle$
$f = \begin{cases} 1, & w = 0 \\ \frac{z}{w}, & w \neq 0 \end{cases}$
$d \leftarrow \langle f,\ f,\ f,\ f\rangle$

`texm3x3pad` d, s
$d \leftarrow \langle u_d,\, v_d,\, w_d \rangle \cdot \langle s_r,\, s_g,\, s_b \rangle$

`texm3x3` d, s
$u' = \langle u_{d-2},\, v_{d-2},\, w_{d-2} \rangle \cdot \langle s_r,\, s_g,\, s_b \rangle$
$v' = \langle u_{d-1},\, v_{d-1},\, w_{d-1} \rangle \cdot \langle s_r,\, s_g,\, s_b \rangle$
$w' = \langle u_d,\, v_d,\, w_d \rangle \cdot \langle s_r,\, s_g,\, s_b \rangle$
$d \leftarrow \langle u',\, v',\, w',\, 1 \rangle$

`texm3x3tex` d, s
$u' = \langle u_{d-2},\, v_{d-2},\, w_{d-2} \rangle \cdot \langle s_r,\, s_g,\, s_b \rangle$
$v' = \langle u_{d-1},\, v_{d-1},\, w_{d-1} \rangle \cdot \langle s_r,\, s_g,\, s_b \rangle$
$w' = \langle u_d,\, v_d,\, w_d \rangle \cdot \langle s_r,\, s_g,\, s_b \rangle$
$d \leftarrow T_d(u', v', w')$

`texm3x3spec` d, s_0, s_1
$u' = \langle u_{d-2},\, v_{d-2},\, w_{d-2} \rangle \cdot \langle s_{0r},\, s_{0g},\, s_{0b} \rangle$
$v' = \langle u_{d-1},\, v_{d-1},\, w_{d-1} \rangle \cdot \langle s_{0r},\, s_{0g},\, s_{0b} \rangle$
$w' = \langle u_d,\, v_d,\, w_d \rangle \cdot \langle s_{0r},\, s_{0g},\, s_{0b} \rangle$
$\vec{n} = \langle u',\, v',\, w' \rangle$
$\langle u'',\, v'',\, w'' \rangle = 2 \frac{\vec{n} \cdot \vec{s_1}}{\vec{n} \cdot \vec{n}} \vec{n} - \vec{s_1}$
$d \leftarrow T_d(u'', v'', w'')$

`texm3x3vspec` d, s_0, s_1
$u' = \langle u_{d-2},\, v_{d-2},\, w_{d-2} \rangle \cdot \langle s_{0r},\, s_{0g},\, s_{0b} \rangle$
$v' = \langle u_{d-1},\, v_{d-1},\, w_{d-1} \rangle \cdot \langle s_{0r},\, s_{0g},\, s_{0b} \rangle$
$w' = \langle u_d,\, v_d,\, w_d \rangle \cdot \langle s_{0r},\, s_{0g},\, s_{0b} \rangle$
$\vec{n} = \langle u',\, v',\, w' \rangle$
$\vec{e} = \langle q_{d-2},\, q_{d-1},\, q_d \rangle$
$\langle u'',\, v'',\, w'' \rangle = 2 \frac{\vec{n} \cdot \vec{e}}{\vec{n} \cdot \vec{n}} \vec{n} - \vec{e}$
$d \leftarrow T_d(u'', v'', w'')$

`texreg2ar` d, s
$d \leftarrow T_d(s_a, s_r)$

`texreg2gb` d, s
$d \leftarrow T_d(s_g, s_b)$

`texreg2rgb` d, s
$d \leftarrow T_d(s_r, s_g, s_b)$

`texdepth` `s`

$$z \leftarrow \begin{cases} \frac{s_r}{s_g}, & s_g \neq 0 \\ 1, & s_g = 0 \end{cases}$$

Table 13.10: Pixel shader 2.0 instructions in assembly syntax.

13.4.5 Bump Mapping Instructions

$$
\begin{aligned}
\texttt{bem}\ & d,\ s_0,\ s_1 \\
& d.r \leftarrow s_{0r} + b_{00d}s_{1r} + b_{10d}s_{1g} \\
& d.g \leftarrow s_{0g} + b_{10d}s_{1r} + b_{11d}s_{1g} \\
\texttt{texbem}\ & d,\ s \\
& u' = u_d + b_{00d}s_r + b_{01d}s_g \\
& v' = v_d + b_{10d}s_r + b_{11d}s_g \\
& d \leftarrow T_d(u', v') \\
\texttt{texbeml}\ & d,\ s \\
& u' = u_d + b_{00d}s_r + b_{01d}s_g \\
& v' = v_d + b_{10d}s_r + b_{11d}s_g \\
& d \leftarrow T_d(u', v')(s_b l_d + o_d)
\end{aligned}
$$

13.5 PS 2.0 Instructions

$$
\begin{aligned}
\texttt{abs}\ & d,\ s \\
& d \leftarrow < |s_r|, |s_g|, |s_b|, |s_a| > \\
\texttt{crs}\ & d,\ s_0,\ s_1 \\
\texttt{dcl}\ & d \\
\texttt{dcl\_textureType}\ & d \\
\texttt{dp2add}\ & d,\ s_0,\ s_1,\ s_2 \\
\texttt{exp}\ & d,\ s_c \\
\texttt{frc}\ & d,\ s \\
\texttt{log}\ & d,\ s_c \\
\texttt{m3x2}\ & d,\ s_0,\ s_1 \\
\texttt{m3x3}\ & d,\ s_0,\ s_1 \\
\texttt{m3x4}\ & d,\ s_0,\ s_1 \\
\texttt{m4x3}\ & d,\ s_0,\ s_1 \\
\texttt{m4x4}\ & d,\ s_0,\ s_1 \\
\texttt{max}\ & d,\ s_0,\ s_1 \\
\texttt{min}\ & d,\ s_0,\ s_1 \\
\texttt{nrm}\ & d,\ s \\
\texttt{pow}\ & d,\ s_{0c},\ s_{1c} \\
\texttt{rcp}\ & d,\ s_c \\
\texttt{rsq}\ & d,\ s_c \\
\texttt{sincos}\ & d,\ s_{0c},\ s_1,\ s_2 \\
\texttt{texld}\ & d,\ s_0,\ s_1 \\
\texttt{texldb}\ & d,\ s_0,\ s_1 \\
\texttt{texldp}\ & d,\ s_0,\ s_1
\end{aligned}
$$

Table 13.11: Pixel shader 2.x instructions in assembly syntax.

Table 13.12: Pixel shader 3.0 instructions in assembly syntax.

13.6 PS 2.x Instructions

```
defb
defi

label
call
callnz
callnz_pred
ret

dsx
dsy

if
else
endif
if_comp
if_pred

rep
endrep
break
break_comp
break_pred

setp
texldd
```

13.7 PS 3.0 Instructions

```
dcl_usage

loop
endloop

texldl
```

13.8 Manipulating Pixel Shaders

The pixel shader property on the device is manipulated with the `SetPixelShader` and `GetPixelShader` methods. The shader property is a `DWORD` handle that is associated with an array of `DWORD` tokens, one token for each instruction in the shader.

```
HRESULT GetPixelShader(DWORD *value);
HRESULT SetPixelShader(DWORD value);
```

The association between the tokens and the handle is made with the `CreatePixelShader` function. To destroy the association of a pixel shader handle with a token array – and free the associated token memory in the device – release all references to the pixel shader object. You can obtain the token array associated with a pixel shader handle with the `GetPixelShaderFunction` method.

```
HRESULT CreatePixelShader(const DWORD *function,
            DWORD *result);
HRESULT GetPixelShaderFunction(DWORD handle,
            void *value,
            DWORD *size);
```

When obtaining the tokens for a pixel shader, first call the method with the `value` parameter set to NULL to obtain the size of the token array, then call the method again with an appropriately sized array as shown in this example.

```
DWORD size = 0;
device->GetPixelShaderFunction(handle, NULL, &size);
std::vector<DWORD> tokens(size);
device->GetPixelShaderFunction(handle, &tokens[0], &size);
```

The pixel shader constant property of the device allows the application to manipulate the constant register file available to pixel shaders. The `GetPixelShaderConstantF` and `SetPixelShaderConstantF` methods manipulate this device property.

```
HRESULT GetPixelShaderConstant(DWORD register,
            void *value,
            DWORD number);
HRESULT SetPixelShaderConstant(DWORD register,
            const void *value,
            DWORD number);
```

Each constant is a four dimensional IEEE floating-point value, stored in RGBA order. The `number` argument gives the number of constants to retrieve or store. For example, the following code snippet sets the constant register `c5` to the value $\langle 1, 1, 1, 1 \rangle$.

```
const float opaque_white[4] = { 1, 1, 1, 1 };
device->SetPixelShaderConstant(5, &opaque_white[0], 1);
```

SS Address U	SS Max Anisotropy
SS Address V	SS Max Mip Level
SS Address W	SS Min Filter
SS Border Color	SS Mag Filter
TSS Bump Env L Scale	SS Mip Filter
TSS Bump Env L Offset	SS Mip Map LOD Bias
TSS Bump Env Mat 00	TSS Tex Coord Index
TSS Bump Env Mat 01	TSS Texture Transform Flags
TSS Bump Env Mat 10	
TSS Bump Env Mat 11	

Table 13.13: Texture stage states honored by pixel shaders.

13.9 Texture Stage States

Pixel shaders replace the functionality provided by the fixed-function pipeline through TSS Color Op and TSS Alpha Op, but do not replace the mechanism of texture addressing and filtering. The texture stage states relating to texture sampling and addressing are still honored by pixel shaders and are listed in table 13.13.

13.10 Examples

Fixed-function pipeline pixel shader.

13.11 Debugging a Pixel Shader

How to debug a pixel shader.