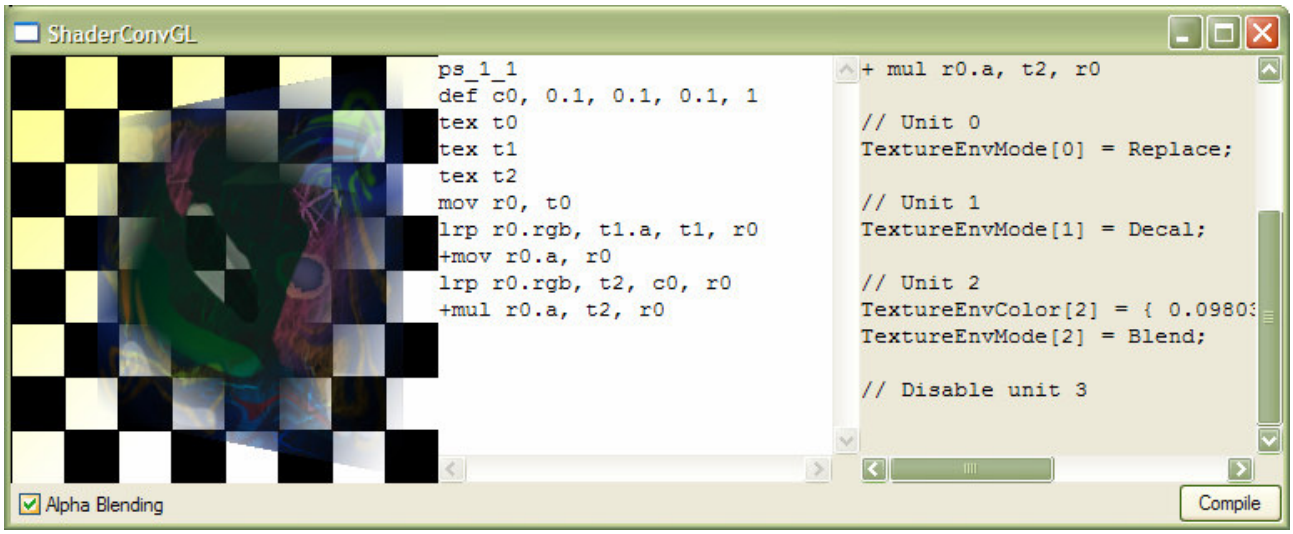
ShaderConv

Example pixel shader to TSS/TexEnv conversion.

Contact: chris@dragan.name

1 Abstract

ShaderConv is a simple utility allowing you to type a pixel shader version 1.0 through 1.3 (Direct3D 9 supports at least version 1.1) and convert it into either Direct3D Texture Stage State pipeline instructions or OpenGL Texture Environment setup.



2 Direct3D TSS implementation

On PC, pixel shaders are available since the DX8-capable hardware. Older hardware that supports multitexturing can be handled by Direct3D through Texture Stage State pipeline. The TSS pipeline is a set of configurable modules, much like texture units in OpenGL. Each module can execute an assigned function. A function is assigned to a stage separately on color and alpha pipelines. Functions take two or three arguments and return result for subsequent stages.
ShaderConvD3D tries to convert ps.1.0-1.3 binary code to TSS pipeline setup. However, the TSS functionality is very limited and the support for certain combinations differs between graphics adapters. Thus not all pixel shaders can be converted.
The following registers are available to the TSS implementation:

Register	**D3DTA**	**Description**
Valid source registers		
r0	CURRENT	The main register which is passed between texture stages.
r1	TEMP	Temporary register. Not supported on some older adapters.
v0	DIFFUSE	Diffuse color.
v1	SPECULAR	Specular color. Not supported on some older adapters.
c0	TFACTOR	A constant register set either inside or outside of the shader. Some older adapters do not handle it properly.
t0-t3	TEXTURE	Texture registers. A texture register t*n* can be used only on the *n*-th stage (i.e. in the *n+1*-th arithmetic instruction pair).
Valid destination registers		
r0	CURRENT	*Note:* Both pipes (color and alpha) must write to the same register.
r1	TEMP	

There are maximum 8 textures available (4 on Direct3D 8), due to nature of Direct3D. The exact number of stages available depends on the graphics adapter and is equivalent to the number of instruction pairs. If an instruction has no write mask, it is executed in exactly the same way on both color and alpha pipes.

The following table lists instructions and argument combinations that can possibly evaluate to a valid TSS pipeline setup. Some instructions support free swapping of the *first* and *second* source arguments (indicated in *Swap* column).

Instruction	D3DTOP	Swap	Remarks
`def c0, R, G, B, A`			
`tex tn`			
`texbem tn, tn-1`	BUMPENVMAP		There must be no arithmetic instruction on stage *n-1*.
`texbeml tn, tn-1`	BUMPENVMAPLUMINANCE		There must be no arithmetic instruction on stage *n-1*.
`mov rn, arg1`	SELECTARG1 or SELECTARG2		D3DTOP instruction is chosen based on argument.
`add rn, arg1, arg2`	ADD	YES	
`add rn, arg1, arg2_bias`	ADDSIGNED	YES	Exactly one argument must have the *_bias* modifier.
`add_x2 rn, arg1, arg2_bias`	ADDSIGNED2X	YES	Exactly one argument must have the *_bias* modifier.
`add rn, arg1, -arg2`	SUBTRACT	YES	Exactly one argument must have the negation modifier.
`sub rn, arg1, arg2`	SUBTRACT		Swapping arguments negates result.
`mul rn, arg1, arg2`	MODULATE	YES	
`mul_x2 rn, arg1, arg2`	MODULATE2X	YES	
`mul_x4 rn, arg1, arg2`	MODULATE4X	YES	
`dp3 rn, arg1_bx2, arg2_bx2`	DOTPRODUCT3	YES	Write mask not supported. Both *arg*s must have *_bx2*.
`mad rn, 1-arg1, arg2, arg1`	ADDSMOOTH	YES	Exactly one instance of *arg1* must be inverted.
`mad rn, 1-tn.a, arg2, arg1`	BLENDTEXTUREALPHAPM	YES	First source argument must be *1-tn.a* (*n* for current stage).
`mad rn.rgb, arg1.a, arg2, arg1`	MODULATEALPHA_ADDCOLOR	YES	Write mask must be *.rgb* to evaluate to this instruction.
`mad rn.rgb, arg1, arg2, arg1.a`	MODULATECOLOR_ADDALPHA	YES	Write mask must be *.rgb* to evaluate to this instruction.
`mad rn.rgb, 1-arg1.a, arg2, arg1`	MODULATEINVALPHA_ADDCOLOR	YES	Write mask must be *.rgb* to evaluate to this instruction.
`mad rn.rgb, 1-arg1, arg2, arg1.a`	MODULATEINVCOLOR_ADDALPHA	YES	Write mask must be *.rgb* to evaluate to this instruction.
`mad rn, arg1, arg2, arg0`	MULTIPLYADD	YES	This is a generic case used if none of the above matches.
`lrp rn, v0.a, arg1, arg2`	BLENDDIFFUSEALPHA		First source argument must be diffuse color alpha.
`lrp rn, r0.a, arg1, arg2`	BLENDCURRENTALPHA		First source argument must be *r0* alpha.
`lrp rn, tn.a, arg1, arg2`	BLENDTEXTUREALPHA		First source argument must be texture alpha.
`lrp rn, c0.a, arg1, arg2`	BLENDFACTORALPHA		First source argument must be constant alpha.
`lrp rn, arg0, arg1, arg2`	LERP		This is a generic case used if none of the above matches.
	PREMODULATE		Unsupported.

The following rules apply to the TSS implementation:

- The *def* instruction cannot be intermixed with other instructions. The *tex** instructions also cannot be intermixed with other instructions. These are actually rules of the pixel shader assembly language.
- The texture sampling must be performed with the *tex* instructions before the *tn* registers are used. The *tex* instructions must declare register sequentially in ascending order. Further, arithmetic instruction forming the shader are explicitly assigned to TSS texture stages in the order they were declared. Instructions can be paired (color vs. alpha pipe).
- No modifiers or masks are supported for the *def*, *tex*, *texbem* and *texbeml* instructions.
- Instruction modifiers (such as result scale) are not supported, except special cases which include *add_x2*, *mul_x2* and *mul_x4*.
- In instructions executing on color or both pipelines, the alpha replicate swizzle (*.a*) is fully supported.
- In most instructions, the *invert* modifier (1-x) is supported. The exception are registers with the *_bias* or *_bx2* modifier, where *negate* (-x) can be used instead. (This is in fact a rule of the shading assembly language.)
- The *negate* (-x) modifier can be used on exactly one argument of the *add* instruction if no result multiplier or *_bias* modifier is used. In this case the *add* instruction evaluates to *sub*.
- In a few instruction, the *_bias* modifier is used to produce special result. It can be used in exactly one operand of *add* or *add_x2* instruction to produce signed result. The *_bx2* modifier must be used on both source arguments of *dp3* instruction.
- Two first source arguments of some instructions (indicated in the table above) can be swapped producing exactly the same result. For example the two following instructions produce the same TSS setup:

```
mad    r0, 1-t1, r0, t1
mad    r0, r0, 1-t1, t1
```

- Be careful when using the *invert* modifier in the *mad* instruction that is supposed to produce *D3DTOP_ADDSMOOTH* instruction. Exactly one instance of the doubled argument must be inverted. One of the following two instructions inverts texture color.

```
mad    r0, 1-t1, r0, t1      ; Normal color
mad    r0, t1, r0, 1-t1      ; Inverted color
```

- The support for *dp3* instruction is very limited. No write mask can be used and the result always goes to all four components (color as well as alpha). Both source arguments must have the *_bx2* modifier. *Negate* (instead of *invert*) and alpha replication modifiers can be used. The performed calculation is as follows:

$$dest.r = dest.g = dest.b = dest.a = 4 * (src1.r - 0.5) * (src2.r - 0.5) + 4 * (src1.g - 0.5) * (src2.g - 0.5) + 4 * (src1.b - 0.5) * (src2.b - 0.5)$$

- There are several special cases for *mad* and *lrp* instructions, indicated in the table above. They are used if the appropriate TSS instructions are supported by hardware and if the combination of arguments is suitable. Otherwise the actual *D3DTOP_MULTIPLYADD* and *D3DTOP_LERP* instructions are used.
- The following sample demonstrates the use of a *texbem* instruction (or alternatively *texbeml*). The first stage is a regular modulation stage (texture * diffuse). In the second stage bump map is sampled, so no arithmetic instruction can be executed. In the third stage, the environment map is applied (added). The environment map texture coordinates are perturbed by the values read from the bump map.

```
tex    t0               ; Surface texture
tex    t1               ; Bump map
texbem t2, t1           ; Environment map

; Stage 0 - Modulate texture with diffuse
mul    r0.rgb, t0, v0
+ mov  r0.a, t0

; Stage 1 - no-op (bump map is sampled)

; Stage 2 - Apply (add) bump-mapped environment map
add    r0.rgb, r0, t2
```

- Remember, that some graphics adapters accept texture only as *arg1* (in the table).
- The *r1* (*D3DTA_TEMP*) register is rare on older hardware.

- The *c0* (*D3DTA_TFACTOR*) does not always behave the way it is meant to behave. Some graphics adapters (or drivers) handle it improperly. Unfortunately there is not direct means to tell whether the *c0* will work properly or not, until you see the results on the screen.
- Although the *_sat* modifier is not required, all arithmetic instructions behave as this modifier was applied. This is because the all the registers can only hold values in range [0,1]. If you want your code to work exactly the same on real pixel shaders and on TSS implementation, use the *_sat* modifier. The modifier does not have to be used on arithmetic instructions *mov*, *mul* (without scale) and *lrp*, because they always provide result in range [0,1], provided that their operands are also in that range (as they always are in TSS implementation).
- ShaderConvD3D tries to convert a pixel shader to TSS shader, given the current adapter capabilities

3 OpenGL TexEnv implementation

OpenGL is the most popular 3D API. It's easy to use and portable. Multitexturing in OpenGL is provided through extensions, although some extensions are included as native part of newer OpenGL versions.
In OpenGL multitexturing is provided through several cascaded configurable texture units. A setup of a texture unit is known as texture environment. Originally, each unit supported only a few basic hard-coded functions, namely *GL_REPLACE*, *GL_MODULATE*, *GL_DECAL* and *GL_BLEND*. This functionality has been enhanced through the definition of *GL_ADD* and a color-alpha decoupled *GL_COMBINE*, that introduces several useful functions.
ShaderConvGL tries to convert ps.1.0-1.3 binary code to TexEnv pipeline setup. It also takes advantage of a few vendor-specific extensions, namely *GL_ATI_texture_env_combine3*, *GL_ATI_envmap_bumpmap* and *GL_NV_texture_env_combine4*.
The functionality is still very limited compared to the actual pixel shader assembly language, thus not all pixel shaders can be successfully converted.
The following registers are available to the TexEnv implementation:

Register	**Source**	**Description**
Valid source registers		
r0	GL_PREVIOUS	The main register that is passed between texture units.
v0	GL_PRIMARY_COLOR	Diffuse color.
c0, c1	GL_CONSTANT	A constant register set either inside or outside of the shader.
t0-t3	GL_TEXTURE	Texture registers. A texture register t*n* can be used only on the *n*-th unit (i.e. in the *n+1*-th instruction pair).
Valid destination registers		
r0	CURRENT	

The number of texture units is usually very limited, but due to the definition of pixel shader assembly language, only maximum of four can be used. There can be defined two constant colors, but only one constant register can be used in a single instruction pair.

The following table lists instructions and argument combinations that can possibly evaluate to a valid TexEnv pipeline setup. Some instructions support free swapping of the *first* and *second* source arguments (indicated in *Swap* column).

Instruction	GL constant	Swap	Remarks
`def c0, R, G, B, A`			
`tex tn`			
`texbem tn, tn-1`	BUMP_ENVMAP_ATI		There must be no arithmetic instruction on stage *n-1*.
`mov r0, tn`	REPLACE (*TEXTURE_ENV_MODE*)		
`mul r0, tn, r0`	MODULATE (*TEXTURE_ENV_MODE*)		*r0* must be replaced with *v0* if this is the first instruction.
`lrp r0.rgb, tn.a, tn, r0` `+mov r0.a, r0`	DECAL (*TEXTURE_ENV_MODE*)		*r0* must be replaced with *v0* if this is the first instruction.
`lrp r0.rgb, tn, c0, r0` `+mul r0.a, tn, r0`	BLEND (*TEXTURE_ENV_MODE*)		*r0* must be replaced with *v0* if this is the first instruction.
`add r0.rgb, tn, r0` `+mul r0.a, tn, r0`	ADD (*TEXTURE_ENV_MODE*)		*r0* must be replaced with *v0* if this is the first instruction.
`mov r0, arg0`	REPLACE		
`add r0, arg0, arg1`	ADD	YES	
`add r0, arg0, arg1_bias`	ADD_SIGNED	YES	Exactly one argument must have the bias modifier.
`add r0, arg0, -arg1`	SUBTRACT	YES	Exactly one argument must have the negation modifier.
`sub r0, arg0, arg1`	SUBTRACT		Swapping arguments negates result.
`mul r0, arg0, arg1`	MODULATE	YES	
`lrp r0, arg2, arg0, arg1`	INTERPOLATE		
`dp3 r0.rgb, arg0_bx2, arg1_bx2`	DOT3_RGB	YES	Both *args* must have *_bx2*.
`dp3 r0, arg0_bx2, arg1_bx2`	DOT3_RGBA	YES	Write mask not supported. Both *arg*s must have *_bx2*.
`mad r0, arg0, arg1, arg2`	ADD (*COMBINE4_NV*)	YES	
`mad r0, arg0, arg1, arg2_bias`	ADD_SIGNED (*COMBINE4_NV*)	YES	The last argument must have *_bias*.
`mad r0, arg0, arg2, arg1`	MODULATE_ADD_ATI	YES	
`mad r0, arg0, arg2, arg1_bias`	MODULATE_SIGNED_ADD_ATI	YES	The last argument must have *_bias*.
`mad r0, arg0, arg2, -arg1`	MODULATE_SUBTRACT_ATI	YES	The last argument must be negated.

The following rules apply to the TexEnv implementation:

- The *def* instruction cannot be intermixed with other instructions. The *tex** instructions also cannot be intermixed with other instructions. These are actually rules of the pixel shader assembly language.
- The texture sampling must be performed with the *tex* instructions before the *tn* registers are used. The *tex* instructions must declare register sequentially in ascending order. Further, arithmetic instruction forming the shader are explicitly assigned to TexEnv texture units in the order they were declared. Instructions can be paired (color vs. alpha pipe).
- No modifiers or masks are supported for the *def*, *tex* and *texbem* instructions.
- Supported instruction modifiers include *_x2*, *_x4* and optional *_sat*. The *_sat* modifier is in fact ignored, and all instructions execute as it was used. It's so because the actual color storage in between texture units is clamped to [0,1].
- In instructions executing on color or both pipelines, the alpha replicate swizzle (*.a*) is fully supported.
- In most instructions, the *invert* modifier (1-x) is supported. The exception are registers with the *_bias* or *_bx2* modifier, where *negate* (-x) can be used instead. (This is in fact a rule of the shading assembly language.)
- The *negate* (-x) modifier can be used only in special cases, indicated in the table above.
- In a few instruction, the *_bias* modifier is used to produce special result. It can be used in exactly one operand of *add* or *mad* instruction to produce signed result. The *_bx2* modifier must be used on both source arguments of *dp3* instruction.
- Two first source arguments of some instructions (indicated in the table above) can be swapped producing exactly the same result. For example the two following instructions produce the same TexEnv setup:

```
mad    r0, 1-t1, r0, t1
mad    r0, r0, 1-t1, t1
```

- The support for *dp3* instruction is very limited. The *.a* write mask cannot be used. Both source arguments must have the *_bx2* modifier. *Negate* (instead of *invert*) and alpha replication modifiers can be used. The performed calculation is as follows:

dest.r = dest.g = dest.b [= dest.a] = *4 * (src1.r – 0.5) * (src2.r – 0.5)*
*+ 4 * (src1.g – 0.5) * (src2.g – 0.5)*
*+ 4 * (src1.b – 0.5) * (src2.b – 0.5)*

- There are five special cases – the setup of the original OpenGL TexEnv pipeline. These are set as *GL_TEXTURE_ENV_MODE*. Otherwise the instructions are set in *GL_COMBINE* mode, or in *GL_COMBINE4_NV* mode in case of the NVidia extension for the *mad* instruction.
- The following sample demonstrates the use of a *texbem* instruction. The first unit performs a regular modulation (texture * diffuse). The second unit samples a bump map, so no arithmetic instruction can be executed. The third unit applies (adds) environment map. The environment map texture coordinates are perturbed by the values read from the bump map.

```
tex    t0              ; Surface texture
tex    t1              ; Bump map
texbem t2, t1          ; Environment map

; Stage 0 - GL_MODULATE
mul    r0, t0, v0

; Stage 1 - no-op (bump map is sampled)

; Stage 2 - GL_COMBINE_RGB( GL_ADD )
add    r0.rgb, r0, t2
```

- ShaderConvGL tries to convert a pixel shader to TexEnv shader, given the current adapter capabilities