NVIDIA™

Texture Compositing With Register Combiners

John Spitzer

NVIDIA Corporation

Contents

Motivation for Using Register Combiners

Combine textures and colors like never before

- **totally customizable per-pixel engines**
- **calculate dot products between any RGB inputs**
- **calculate scalar products or sums between any inputs**
- **RGB and Alpha computations are separate**

Enable the coolest effects

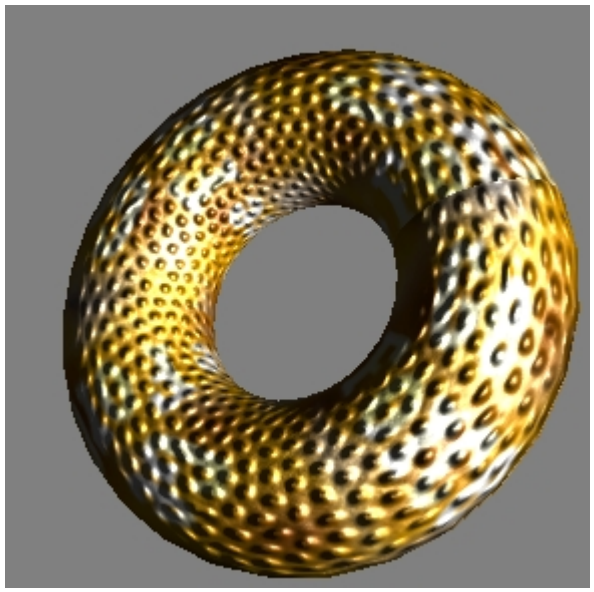
- **per-pixel ambient, diffuse, specular lighting**
- **true bump mapping**
- **modulate specular color with texture map**
- **color space conversions**

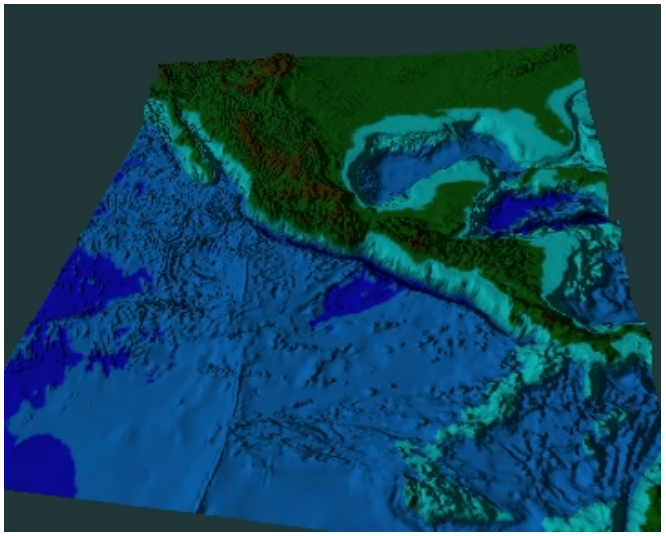
With register combiners, you're programming "to the metal" – what you've been asking for

The payoff is well worth the effort

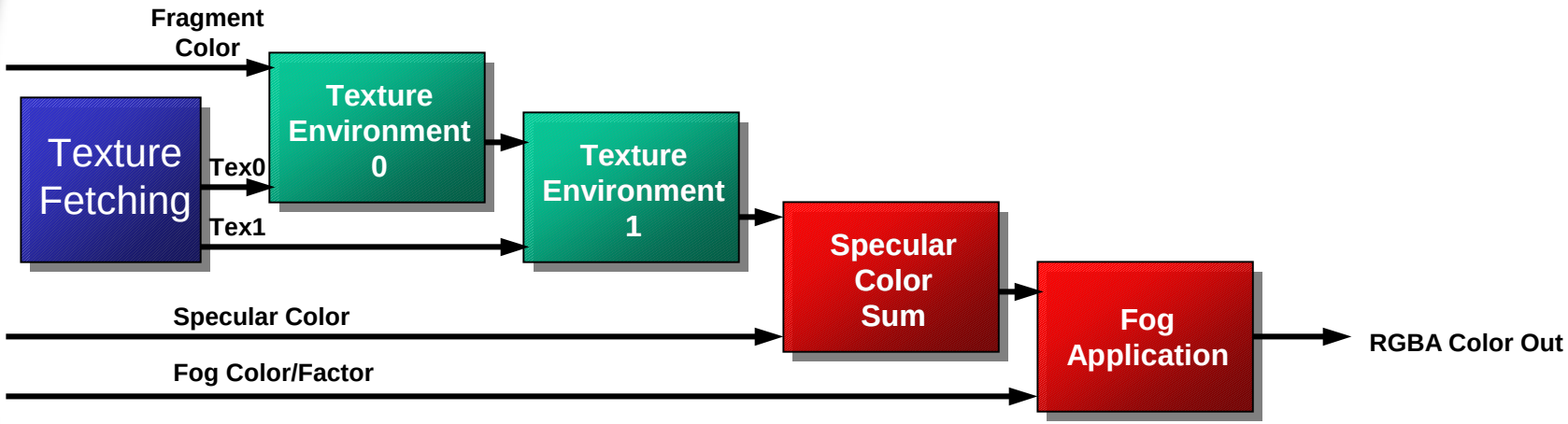
NVIDIA

Examples of Texture Compositing With Register Combiners





OpenGL and DirectX 7 Texture Compositing (2 Textures)



NVIDIA

OpenGL and DirectX 7 Texture Compositing

One texture environment unit per enabled texture (2 textures = 2 TexEnvs)

Linear data flow

- fragment color only available to first unit
- results from one unit passed on to next unit

Unsigned math with [0, 1] range

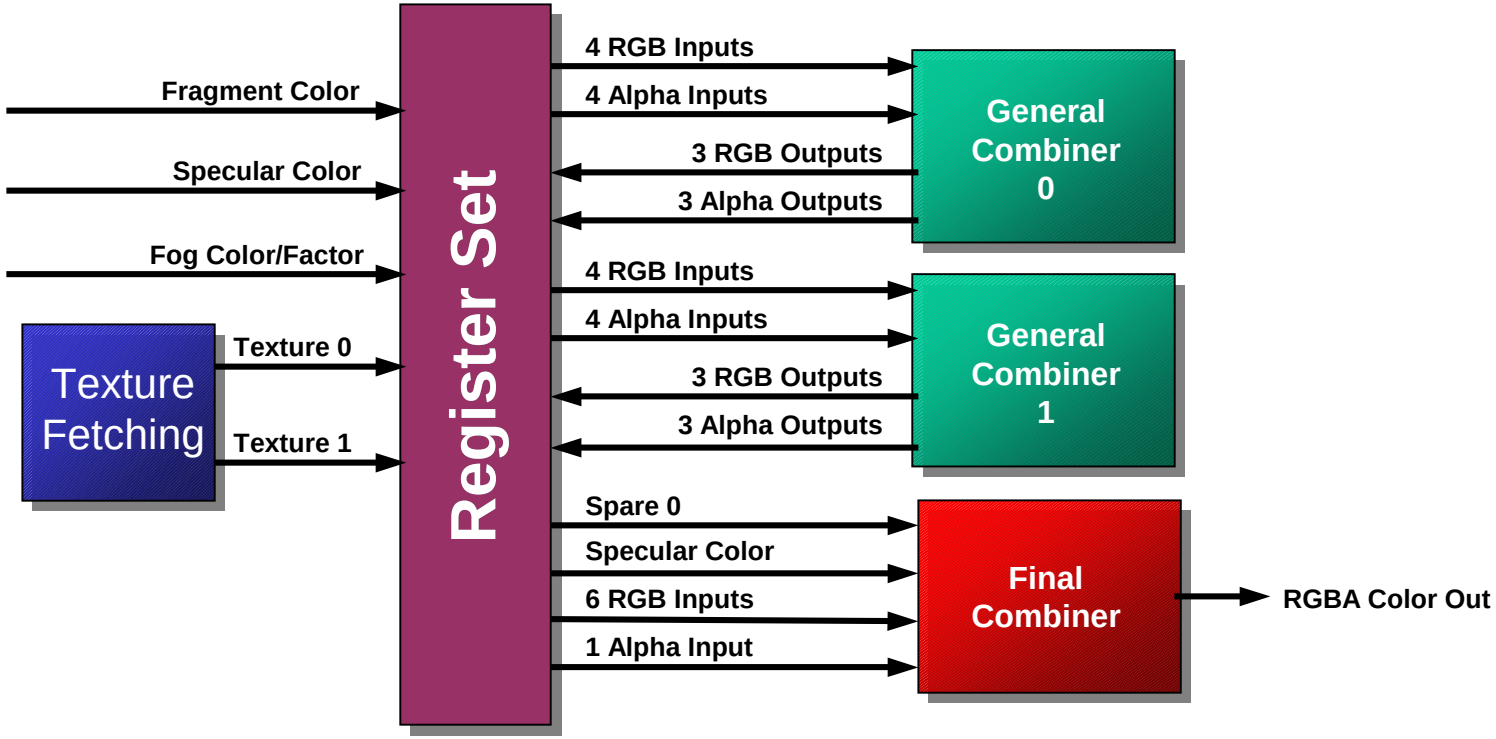
Choice of 5 set functions for RGB and Alpha:

Function	RGB	Alpha
Replace	Ct	At
Modulate	Cf Ct	Af At
Decal	Cf (1 – At) + Ct At	Af
Blend	Cf (1 – Ct) + Cc Ct	Af At
Add	Cf + Ct	Af At

Post-environment specular color addition and fog application

NVIDIA

Texture Compositing With OpenGL Register Combiners Extension



NVIDIA

Texture Compositing With OpenGL Register Combiners Extension

Supersedes texture environment, color sum and fog in OpenGL

One or two combiners can be enabled

Non-linear data flow

- superior to linear chain in current APIs
- results from all textures and colors available in each and every combiner

Signed math with [-1, 1] range

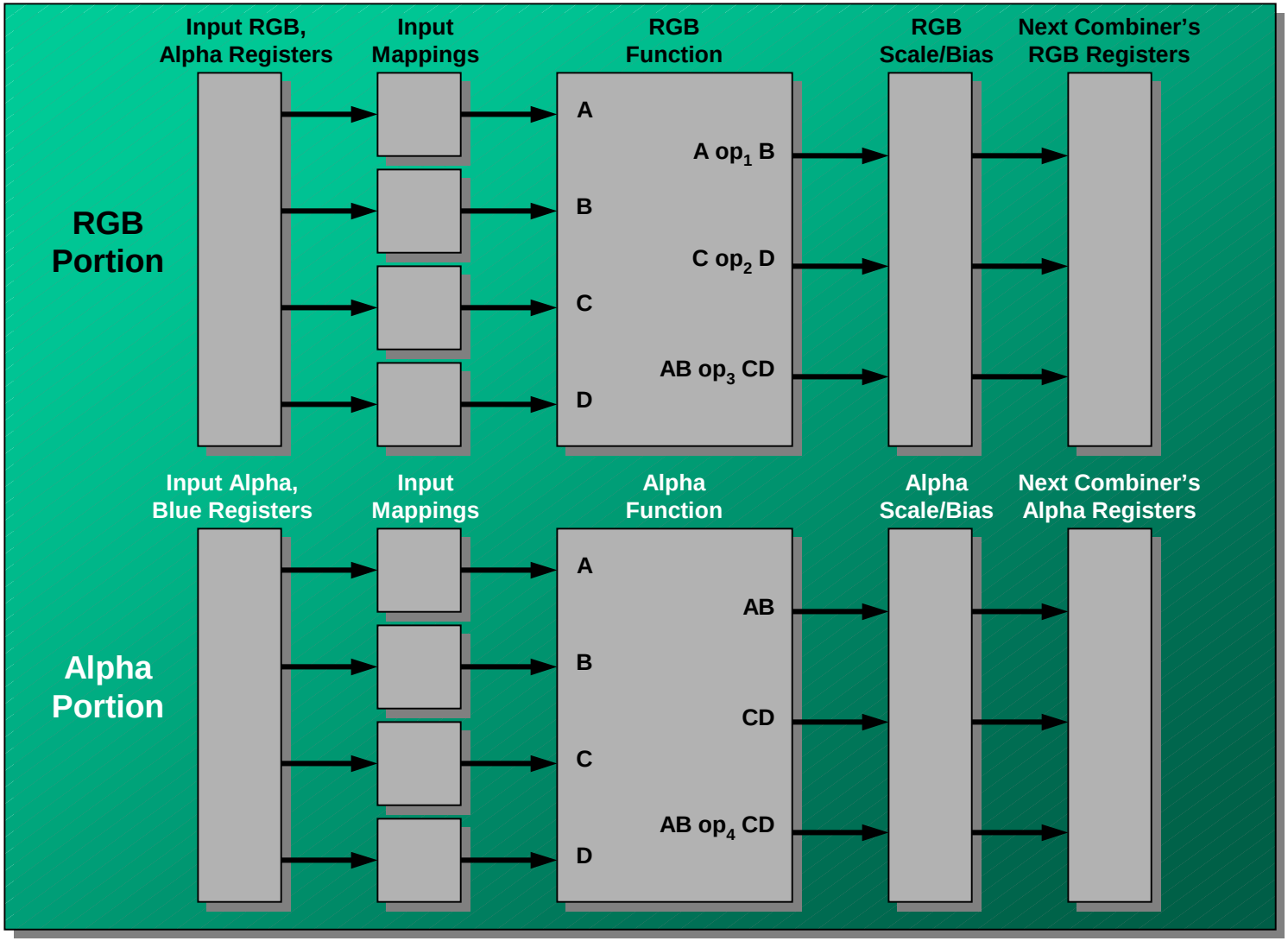
Dot product calculations for lighting and image processing applications

RGB and Alpha calculations separated for added flexibility

Additional "Final Combiner" can be used for specular color sum and fog application, or for custom calculations

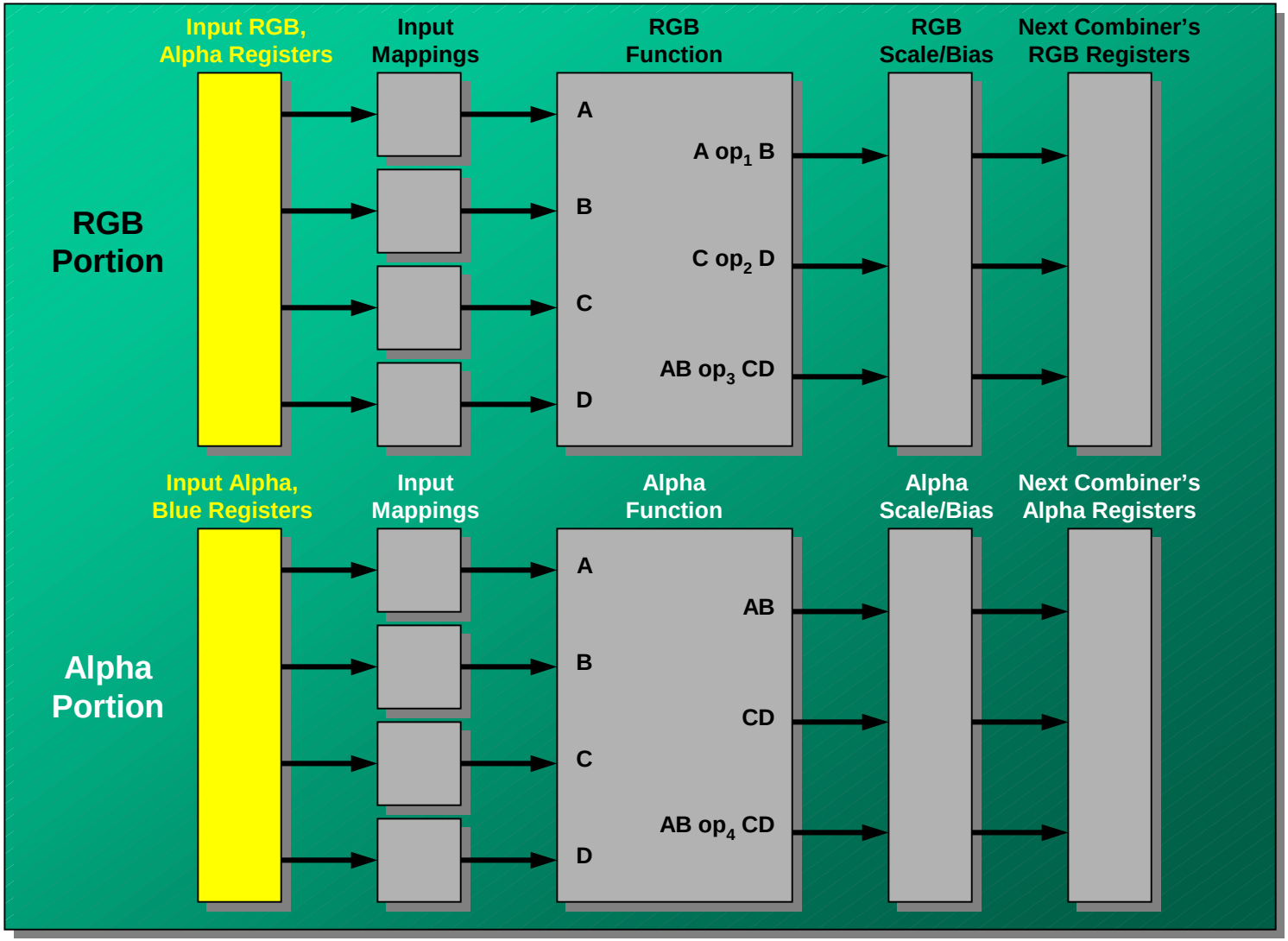
NVIDIA

Diagram of a General Combiner



NVIDIA

General Combiner Input Registers



NVIDIA

The Register Set

Primary (diffuse) color
- initialized to RGBA of fragment's primary color

Secondary (specular) color
- initialized to RGB of fragment's secondary/specular color
- alpha not initialized

Texture 0 and Texture 1 colors
- initialized to fragment's filtered RGBA texel from numbered texture unit
- not initialized if numbered texture unit is disabled or non-existent

Spare 0 and Spare 1
- Alpha of Spare 0 is initialized to alpha of Texture 0 color (if enabled)
- RGB of Spare 0 and all of Spare 1 is not initialized

Fog
- RGB is current fog color
- alpha is fragment's fog factor (only available in final combiner)
- read-only

Constant color 0 and Constant color 1
- initialized to user-defined RGBA value
- read-only

Zero
- constant, read-only value of zero

General Combiner Input Registers

RGB Portion

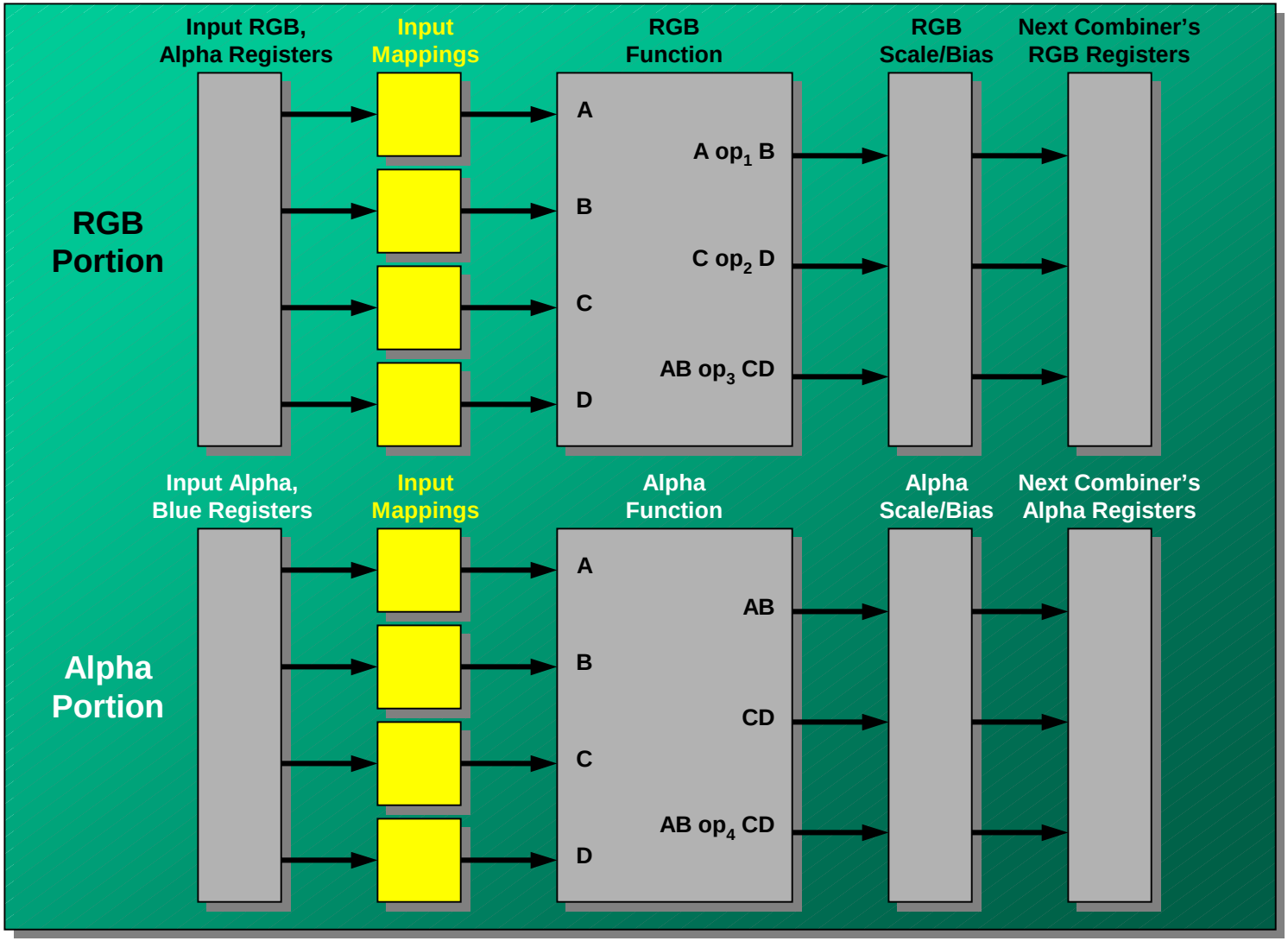
	R	G	B	A
primary color (diffuse color)				
secondary color (specular color)				
texture 0 color				
texture 1 color				
spare 0				
spare 1				
fog color				X
constant color 0				
constant color 1				
zero				

Alpha Portion

	R	G	B	A
primary color (diffuse color)	X	X		
secondary color (specular color)	X	X		
texture 0 color	X	X		
texture 1 color	X	X		
spare 0	X	X		
spare 1	X	X		
fog color	X	X		X
constant color 0	X	X		
constant color 1	X	X		
zero	X	X		

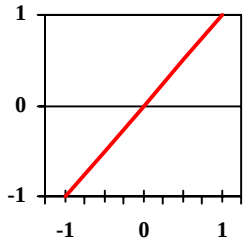
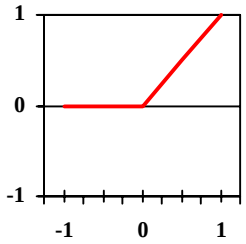
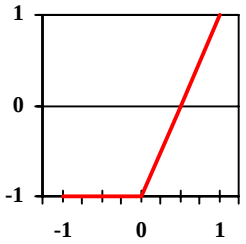
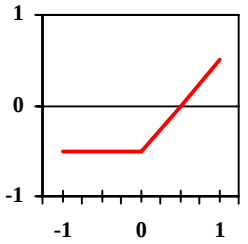
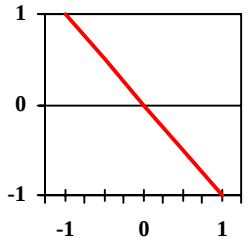
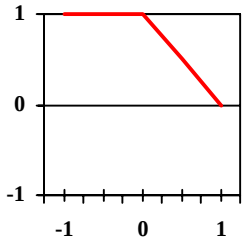
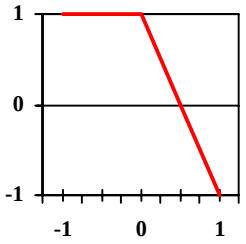
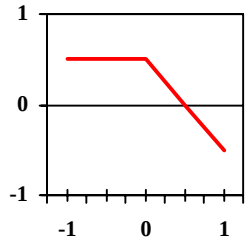
NVIDIA

General Combiner Input Mappings



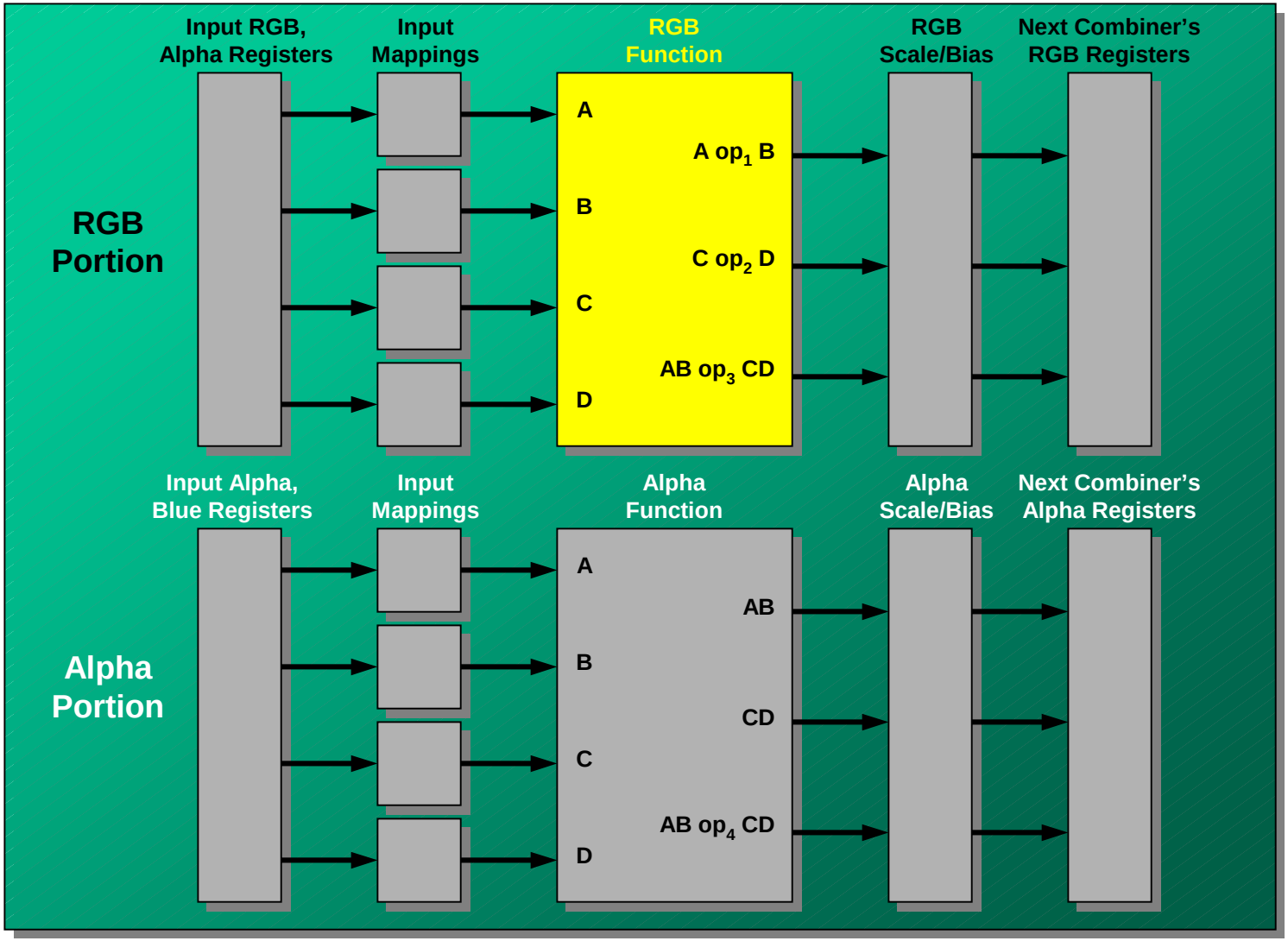
NVIDIA

General Combiner Input Mappings

Signed Identity	Unsigned Identity	Expand Normal	Half Bias Normal
f(x) = x	f(x) = max(0, x)	f(x) = 2 * max(0, x) - 1	f(x) = max(0, x) – ½
[-1, 1] ⟶ [-1, 1]	[0, 1] ⟶ [0, 1]	[0, 1] ⟶ [-1, 1]	[0, 1] ⟶ [-½, ½]
			
Signed Negate	**Unsigned Invert**	**Expand Negate**	**Half Bias Negate**
f(x) = -x	f(x) = 1-min(max(0,x),1)	f(x) = -2 * max(0, x) + 1	f(x) = -max(0, x) + ½
[-1, 1] ⟶ [1, -1]	[0, 1] ⟶ [1, 0]	[0, 1] ⟶ [1, -1]	[0, 1] ⟶ [½, -½]
			

NVIDIA

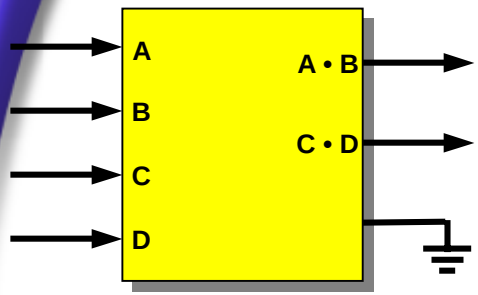
General Combiner RGB Function



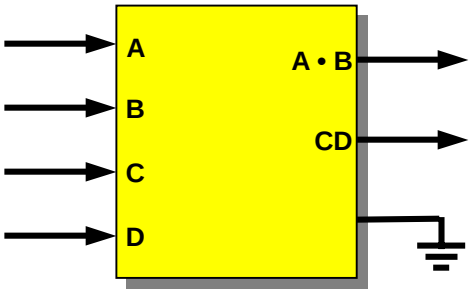
NVIDIA

General Combiner RGB Functions

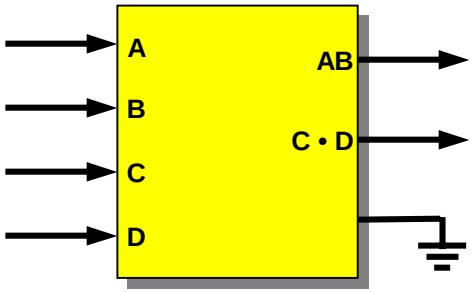
Dot / Dot / Discard



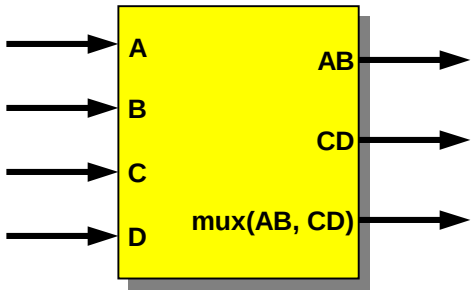
Dot / Mult / Discard



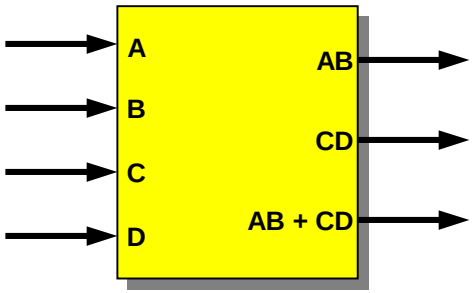
Mult / Dot / Discard



Mult / Mult / Mux



Mult / Mult / Sum



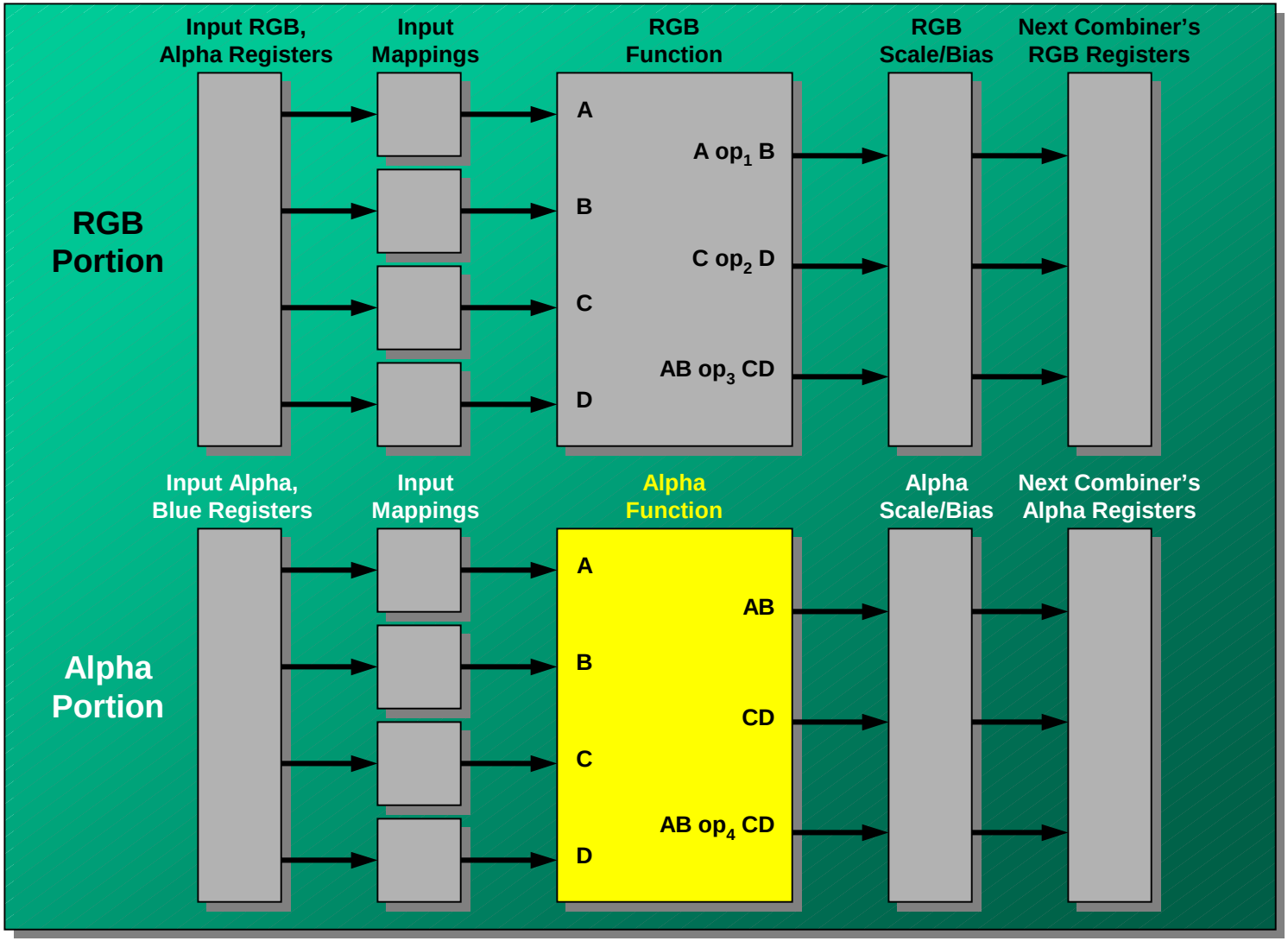
mux(AB, CD) = (Spare0[Alpha] ≥ ½) ? AB : CD

Dot products on RGB registers: A • B = (A[red] * B[red] + A[green] * B[green] + A[blue] * B[blue], A[red] * B[red] + A[green] * B[green] + A[blue] * B[blue], A[red] * B[red] + A[green] * B[green] + A[blue] * B[blue])

Multiplication on RGB registers: AB = (A[red] * B[red], A[green] * B[green], A[blue] * B[blue])

NVIDIA

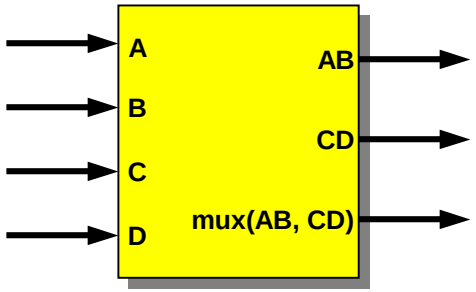
General Combiner Alpha Function



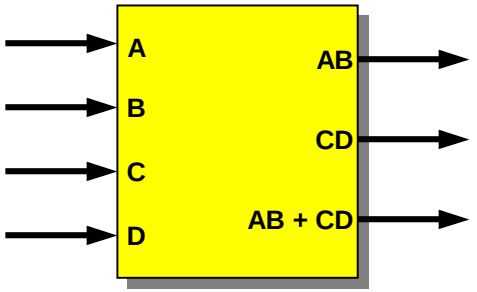
NVIDIA

General Combiner Alpha Functions

Mult / Mult / Mux



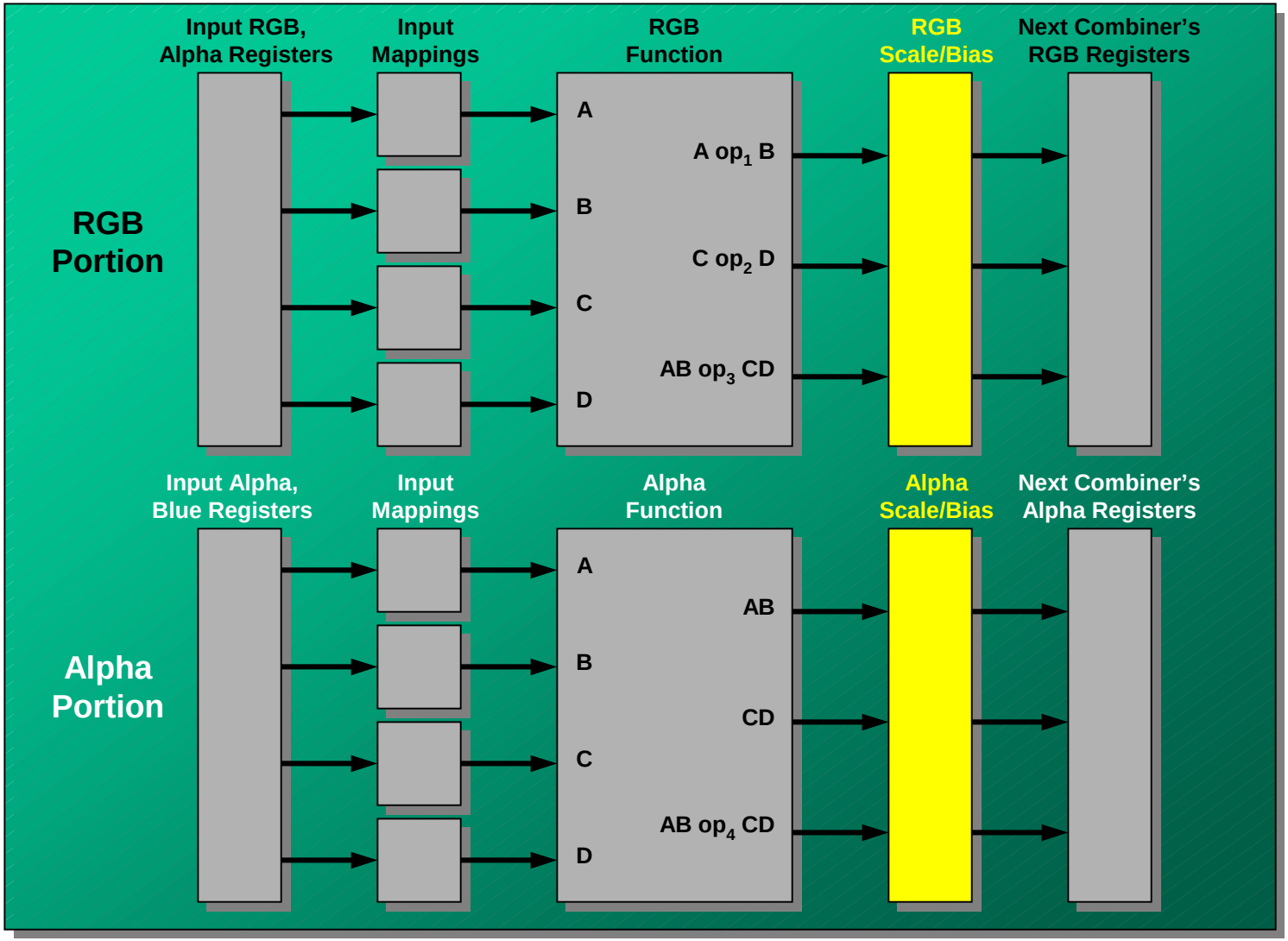
Mult / Mult / Sum



mux(AB, CD) = (Spare0[alpha] ≥ ½) ? AB : CD

NVIDIA

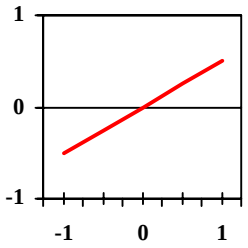
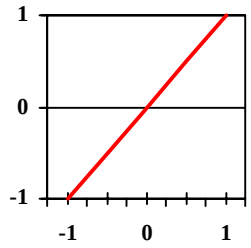
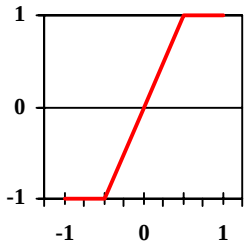
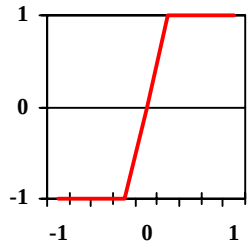
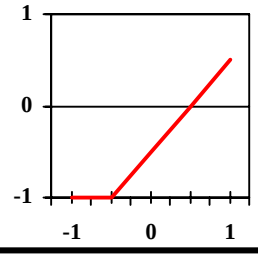
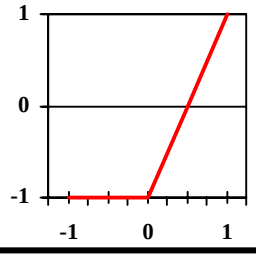
General Combiner Scale and Bias



NVIDIA

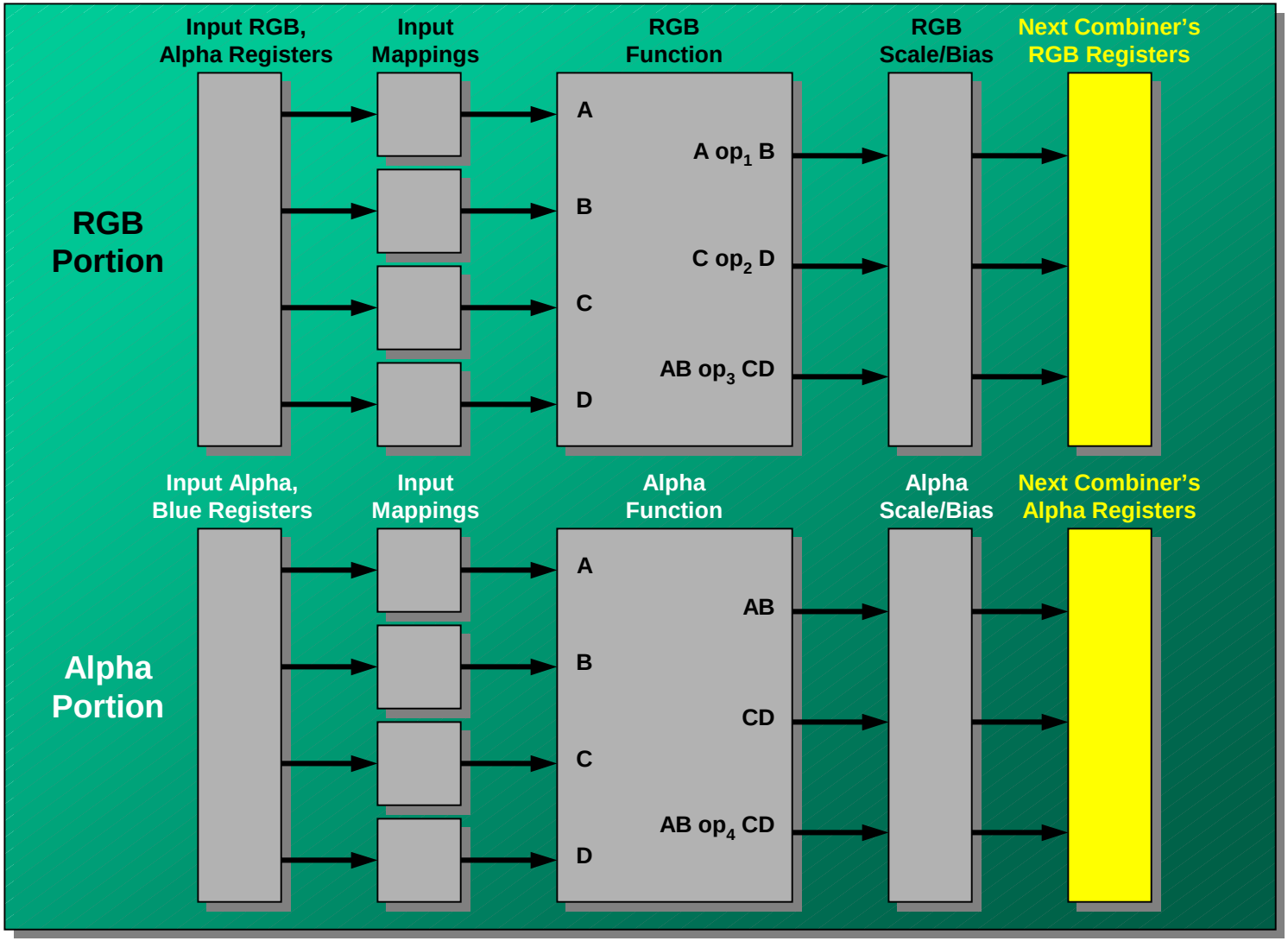
General Combiner Scale and Bias Options

Scale and bias operation is defined as:
ClampNegativeOneToOne(Scale * (x + Bias))
OR max(min(Scale * (x + Bias), 1), -1)

	Scale by ½	No scale	Scale by 2	Scale by 4
No bias	f(x) = x/2 	f(x) = x 	f(x) = 2x 	f(x) = 4x 
Bias by –½	Not supported	f(x) = x – ½ 	f(x) = 2(x – ½) 	Not supported

NVIDIA

General Combiner Output Registers



NVIDIA

General Combiner Output Registers

Up to six outputs can be specified per general combiner:

- three RGB outputs (A op_1 B, C op_2 D, AB op_3 CD) written to RGB portion of writable registers
- three Alpha outputs (AB, CD, AB op_4 CD) written to Alpha portion of writable registers

RGB outputs must be written to distinct registers (that is, two outputs cannot be written to one register)

Alpha outputs must be written to distinct registers

Any output can be discarded

Those RGB functions performing dot products must discard the third result (Dot/Dot/Discard, Dot/Mult/Discard, Mult/Dot/Discard)

General Combiner Output Registers

RGB Portion

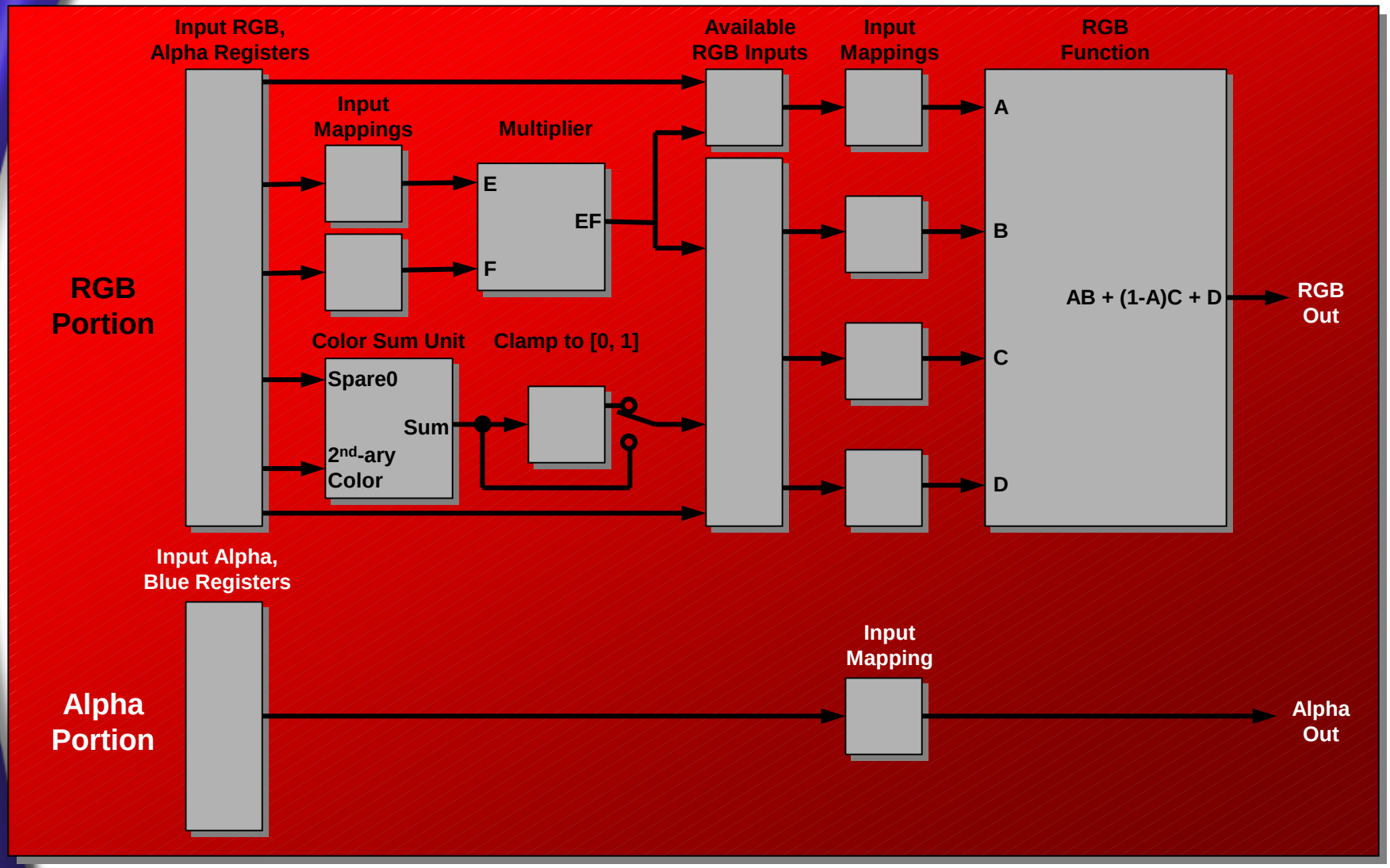
	R	G	B	A
primary color (diffuse color)				X
secondary color (specular color)				X
texture 0 color				X
texture 1 color				X
spare 0				X
spare 1				X
fog color		X		X
constant color 0		X		X
constant color 1		X		X
zero		X		X

Alpha Portion

	R	G	B	A
primary color (diffuse color)		X		
secondary color (specular color)		X		
texture 0 color		X		
texture 1 color		X		
spare 0		X		
spare 1		X		
fog color		X		X
constant color 0		X		X
constant color 1		X		X
zero		X		X

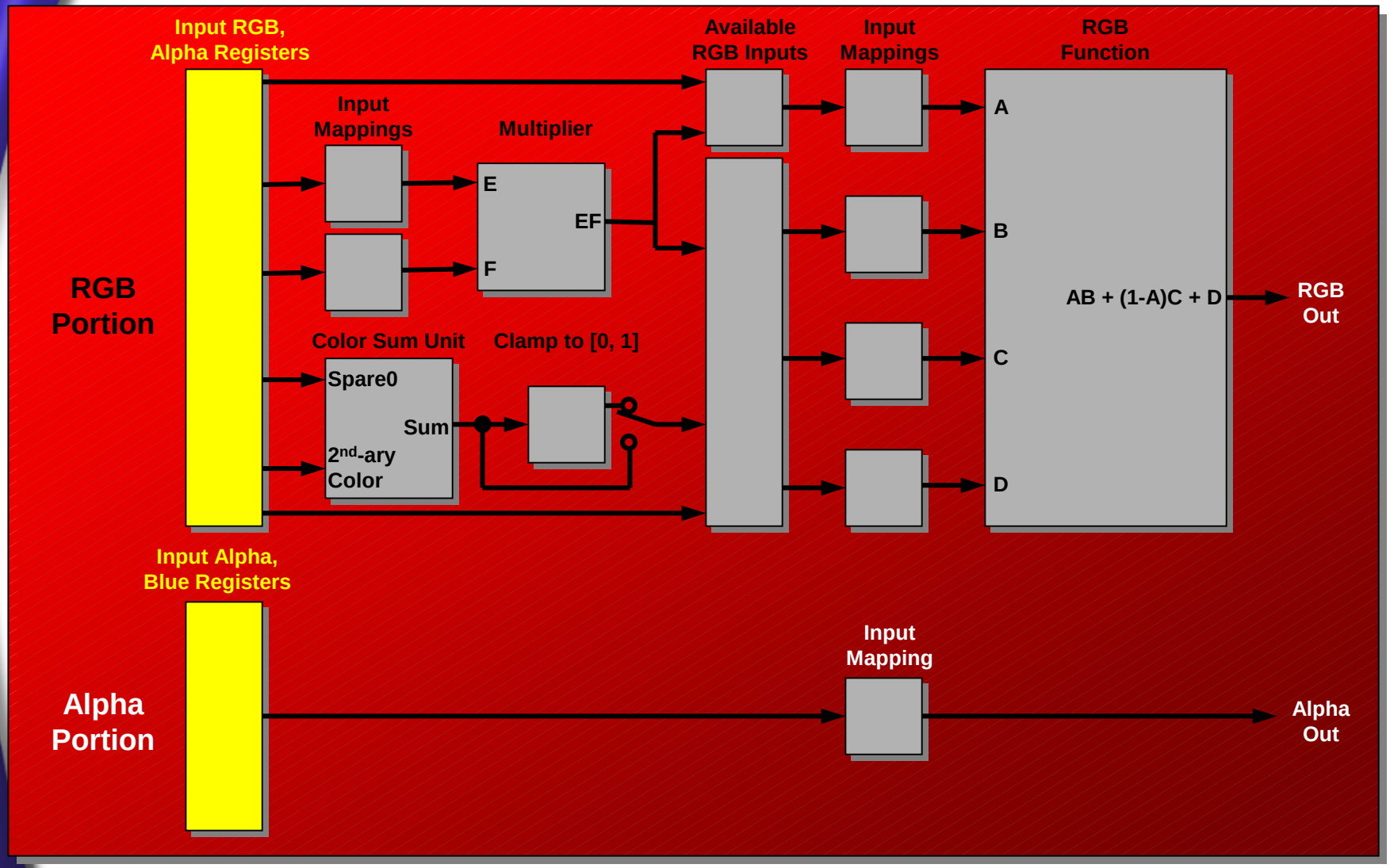
NVIDIA

Diagram of the Final Combiner (OpenGL only)



NVIDIA

Final Combiner Input Registers



NVIDIA

Final Combiner Input Registers

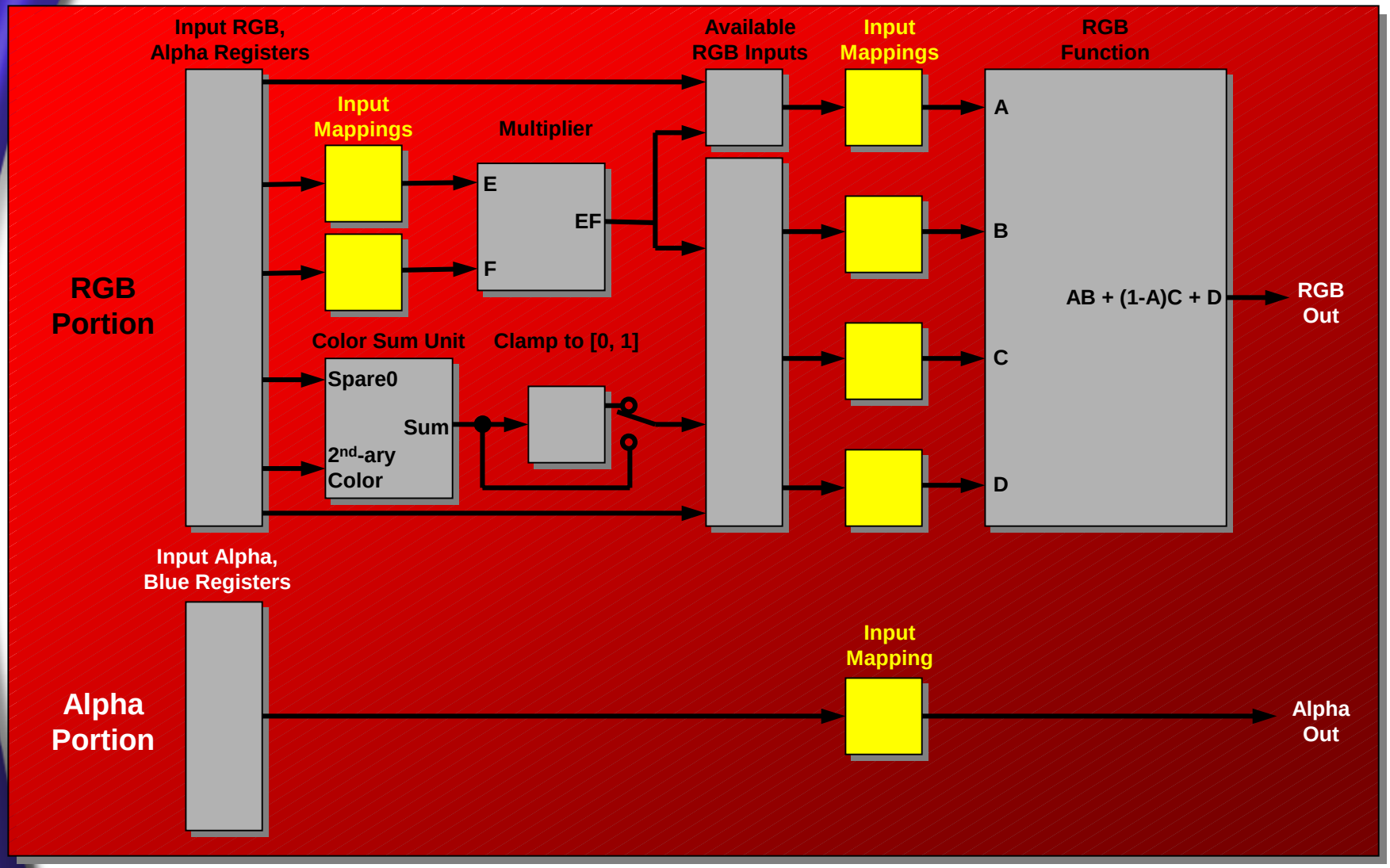
RGB Portion

	R	G	B	A
primary color (diffuse color)				
secondary color (specular color)				
texture 0 color				
texture 1 color				
spare 0				
spare 1				
fog color				
constant color 0				
constant color 1				
zero				

Alpha Portion

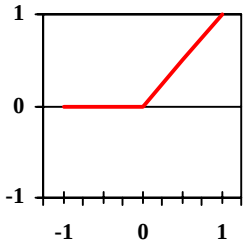
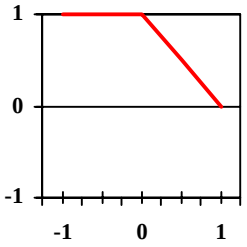
	R	G	B	A
primary color (diffuse color)	X	X		
secondary color (specular color)	X	X		
texture 0 color	X	X		
texture 1 color	X	X		
spare 0	X	X		
spare 1	X	X		
fog color	X	X		
constant color 0	X	X		
constant color 1	X	X		
zero	X	X		

Final Combiner Input Mappings



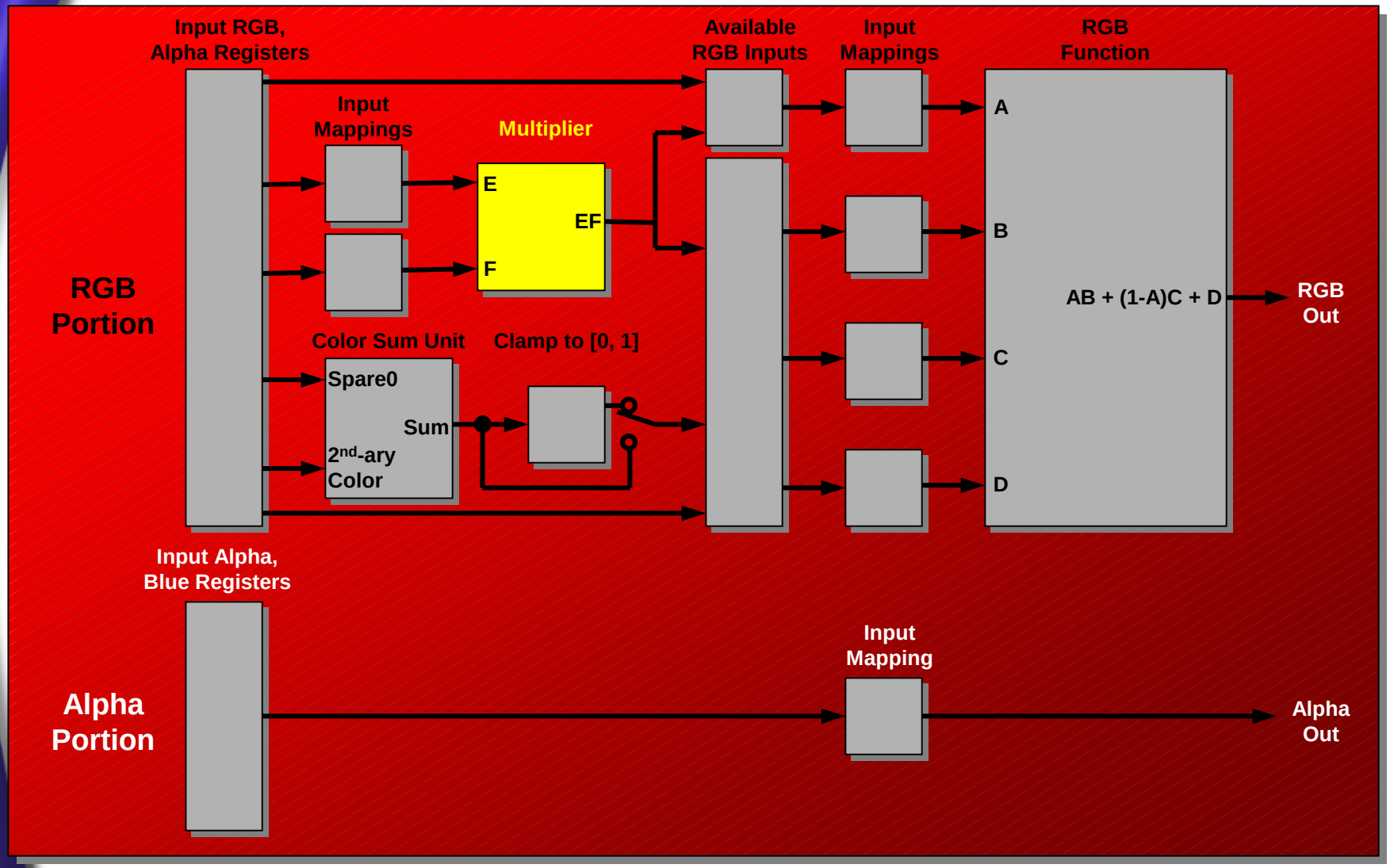
NVIDIA

Final Combiner Input Mappings

Unsigned Identity	Unsigned Invert
f(x) = max(0, x)	f(x) = 1-min(max(0,x),1)
[0, 1] ⟶ [0, 1]	[0, 1] ⟶ [1, 0]
	

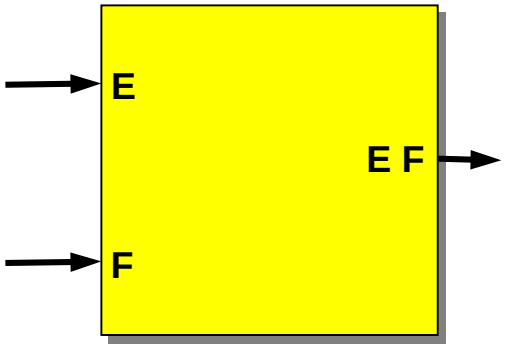
NVIDIA

Final Combiner EF Multiplier



NVIDIA

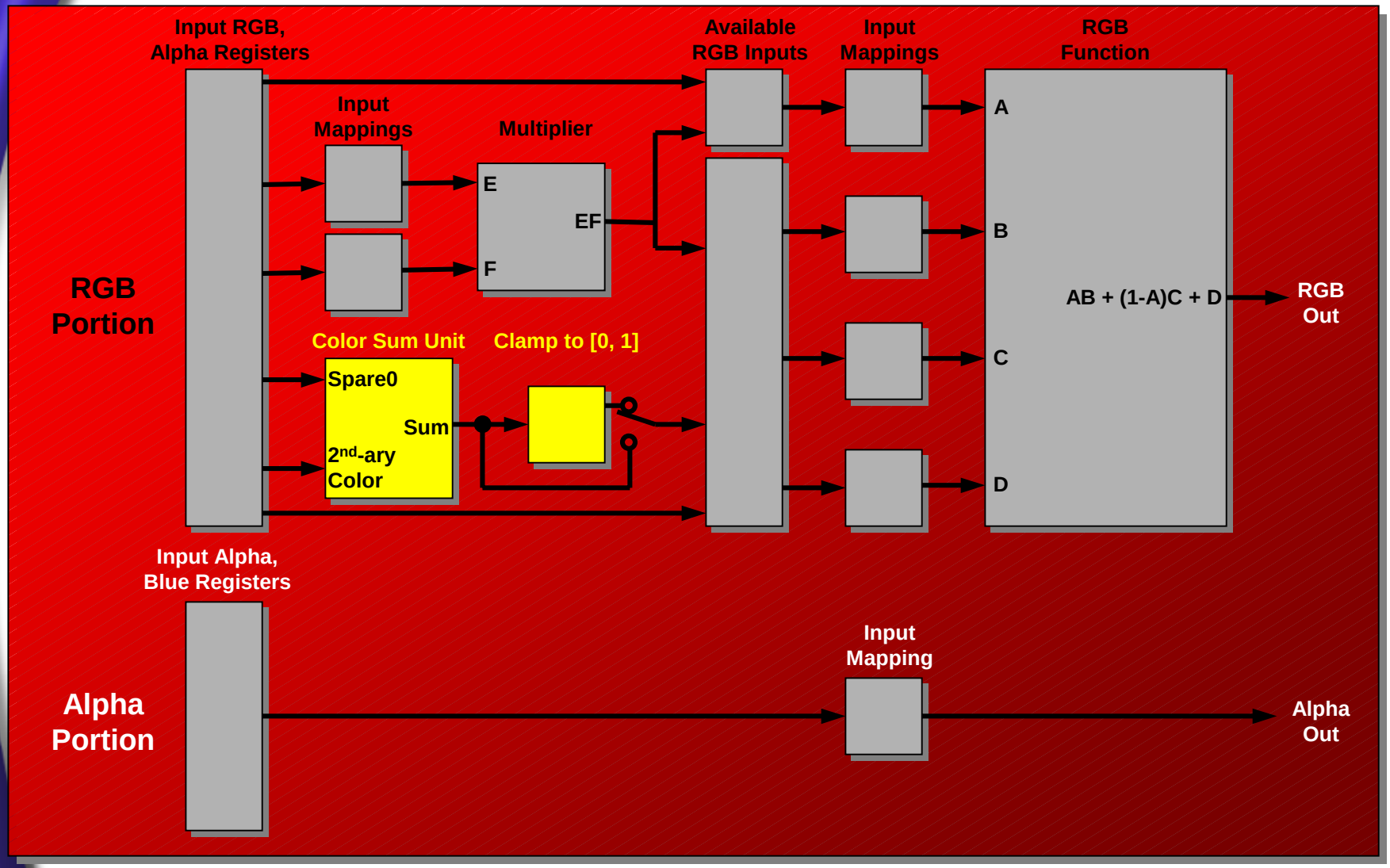
Final Combiner EF Multiplier



Multiplication on RGB registers: **E F = (E[red] * F[red], E[green] * F[green], E[blue] * F[blue])**

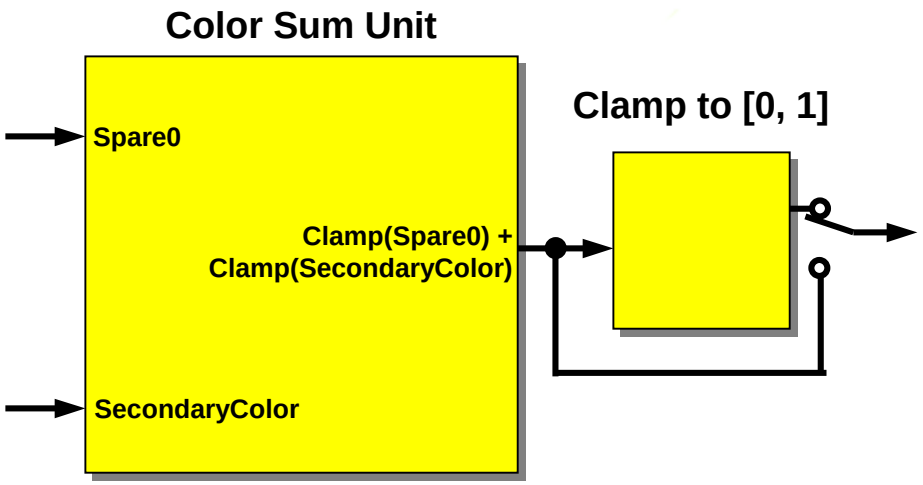
NVIDIA

Final Combiner Color Sum and Optional Clamp



NVIDIA

Final Combiner Color Sum and Optional Clamp



Inputs to this unit are hardwired to Spare0 and SecondaryColor registers

Each input to the color sum unit undergoes an unsigned identity mapping before addition

Unsigned identity = max(0, x), clamping negative input values to 0

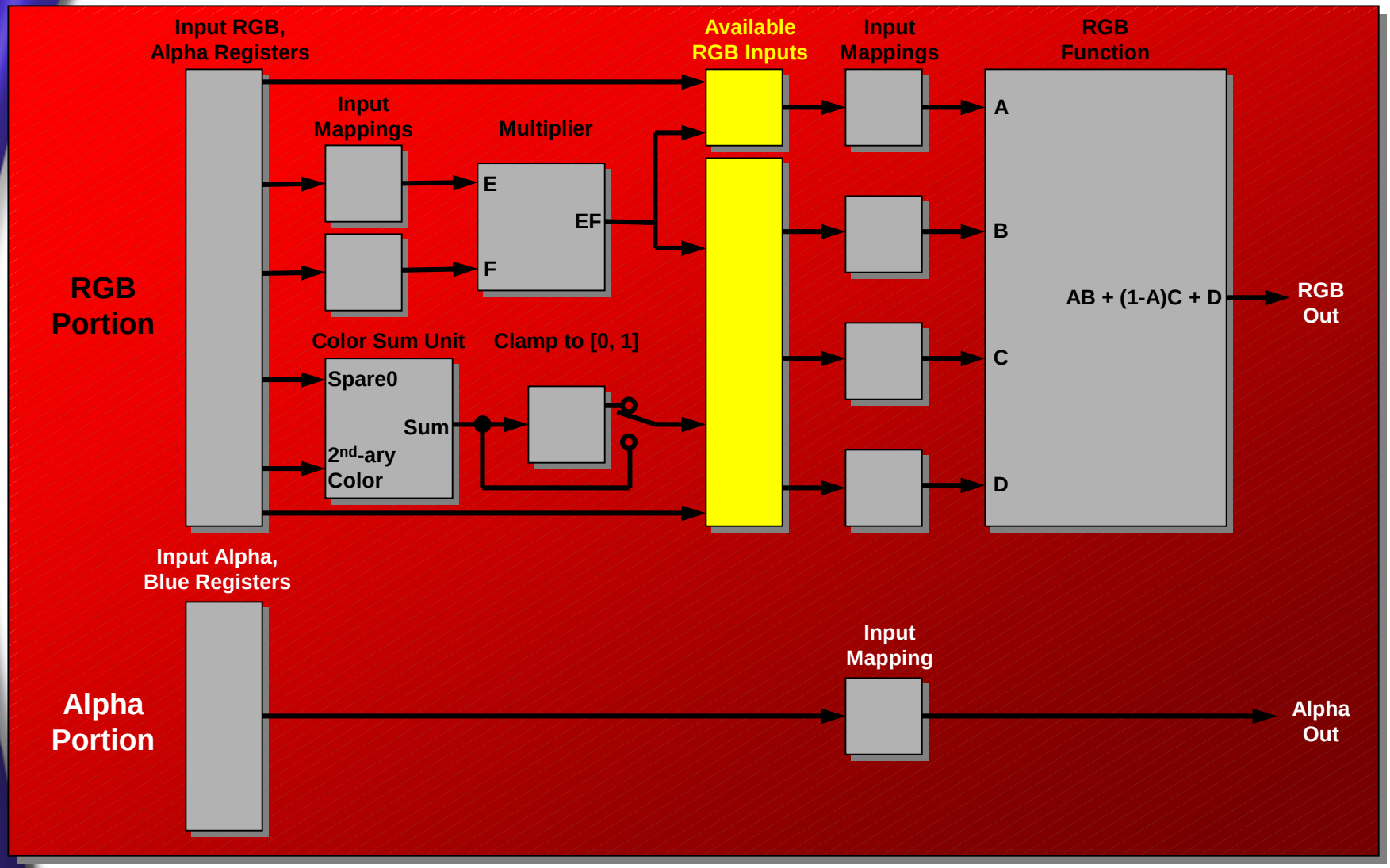
Addition on RGB registers: Spare0 + SecondaryColor =
(Spare0[red] + SecondaryColor[red],
Spare0[green] + SecondaryColor[green],
Spare0[blue] + SecondaryColor[blue])

Output range for sum is [0, 2]

Optional clamping unit clamps the sum to [0, 1]

NVIDIA

Final Combiner RGB Input Set



NVIDIA

Final Combiner RGB Input Set

All registers available to all RGB function inputs (A, B, C and D)

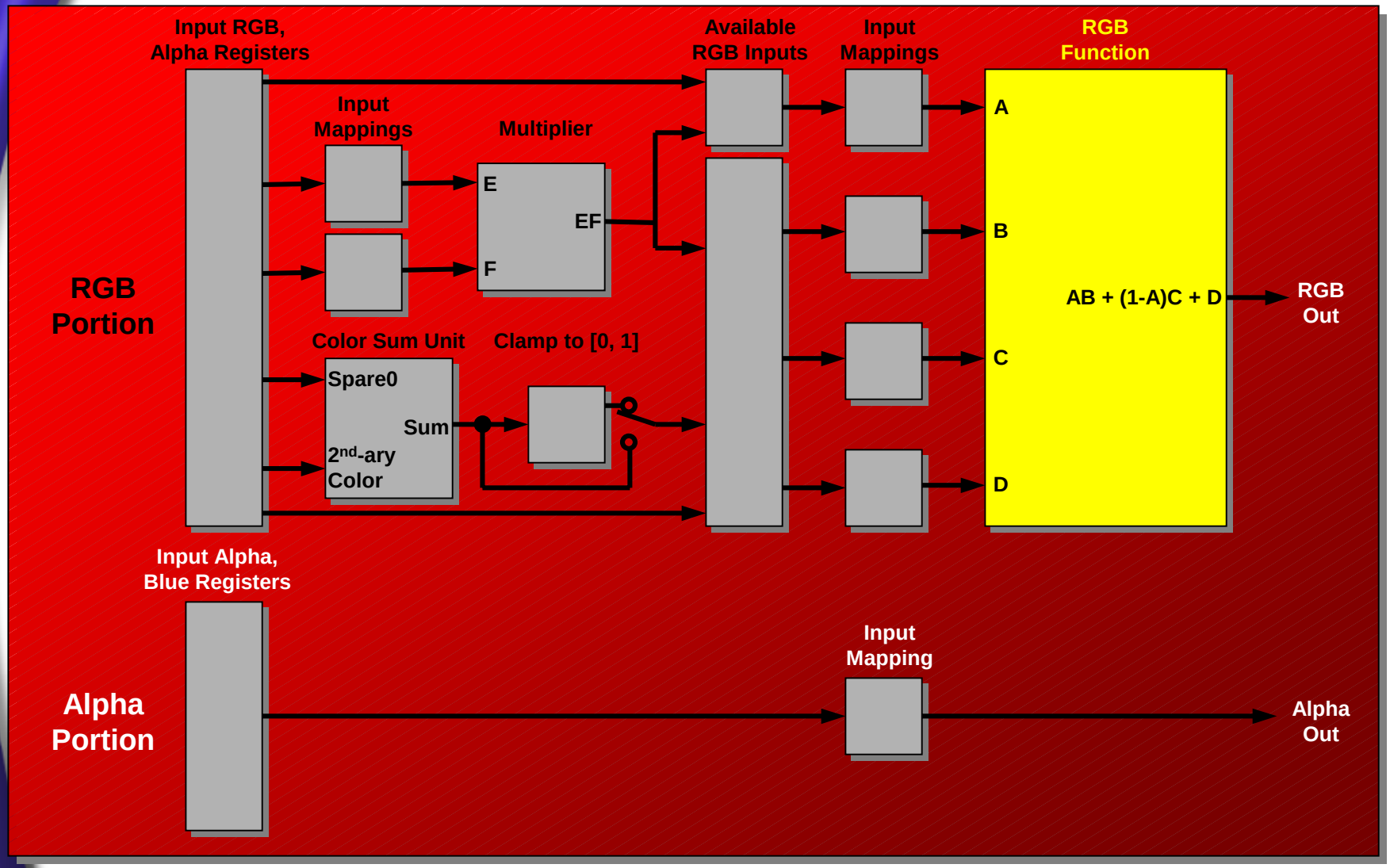
Result of EF multiplier also available to all RGB function inputs

Result of (possibly clamped) color sum available to B, C and D function inputs (but not A)

Neither the result of the EF multiplier nor the color sum is available to the Alpha portion

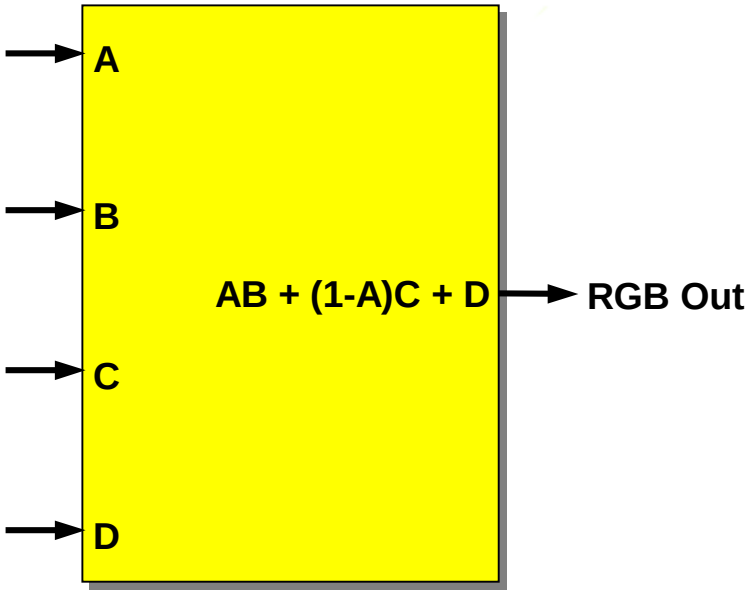
NVIDIA

Final Combiner RGB Function



NVIDIA

Final Combiner RGB Function



Unlike the general combiner, only one function is supported in the final combiner

Addition/multiplication on RGB registers as defined previously

Output range for RGB function is [0, 2]

RGB output combined with Alpha output to form a single RGBA fragment

RGBA fragment continues with subsequent processing (Z-buffering, blending, ...)

NVIDIA

Register Combiner Programming Considerations

When using register combiners, you must do the post-texture specular add and fog application manually, using the final combiner

Though up to 2 general combiners are available for use, enabling only one may yield higher performance

Some OpenGL multitexture TexEnv modes only use 1 general combiner:

- **No Alpha component in Texture1 AND**
- **TexEnv for second texture unit (Texture1) is GL_MODULATE OR**
- **TexEnv for Texture1 is GL_ADD and color sum not used**

NVIDIA

Register Combiner Programming Examples

Light mapped multitexture with fog

Simple and fast (or quick and dirty) bump mapping

Robust bump mapping

Light Mapped Multitexture with Fog

Coded using glTexEnv with GL_MODULATE and GL_MODULATE

Diffuse color C_D provided per vertex

Texture0 is diffuse map T_D, Texture1 is light map T_L

C_F is fog color, A_F is fog factor

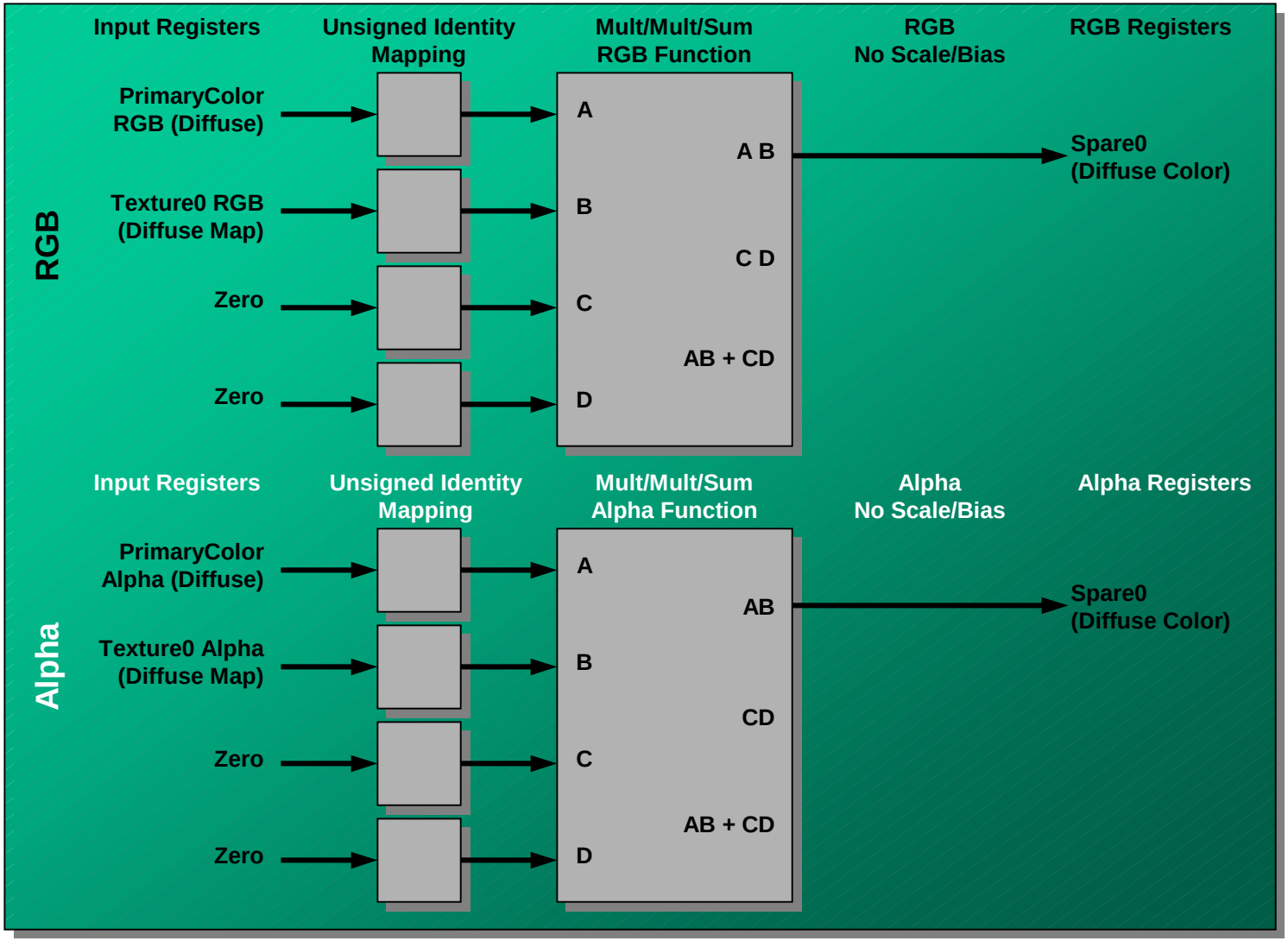
Computes $A_F * C_D * T_D * T_L + (1 - A_F) * C_F$

Uses two textures, but only one general combiner

Advantages: performance (single pass, no blending)

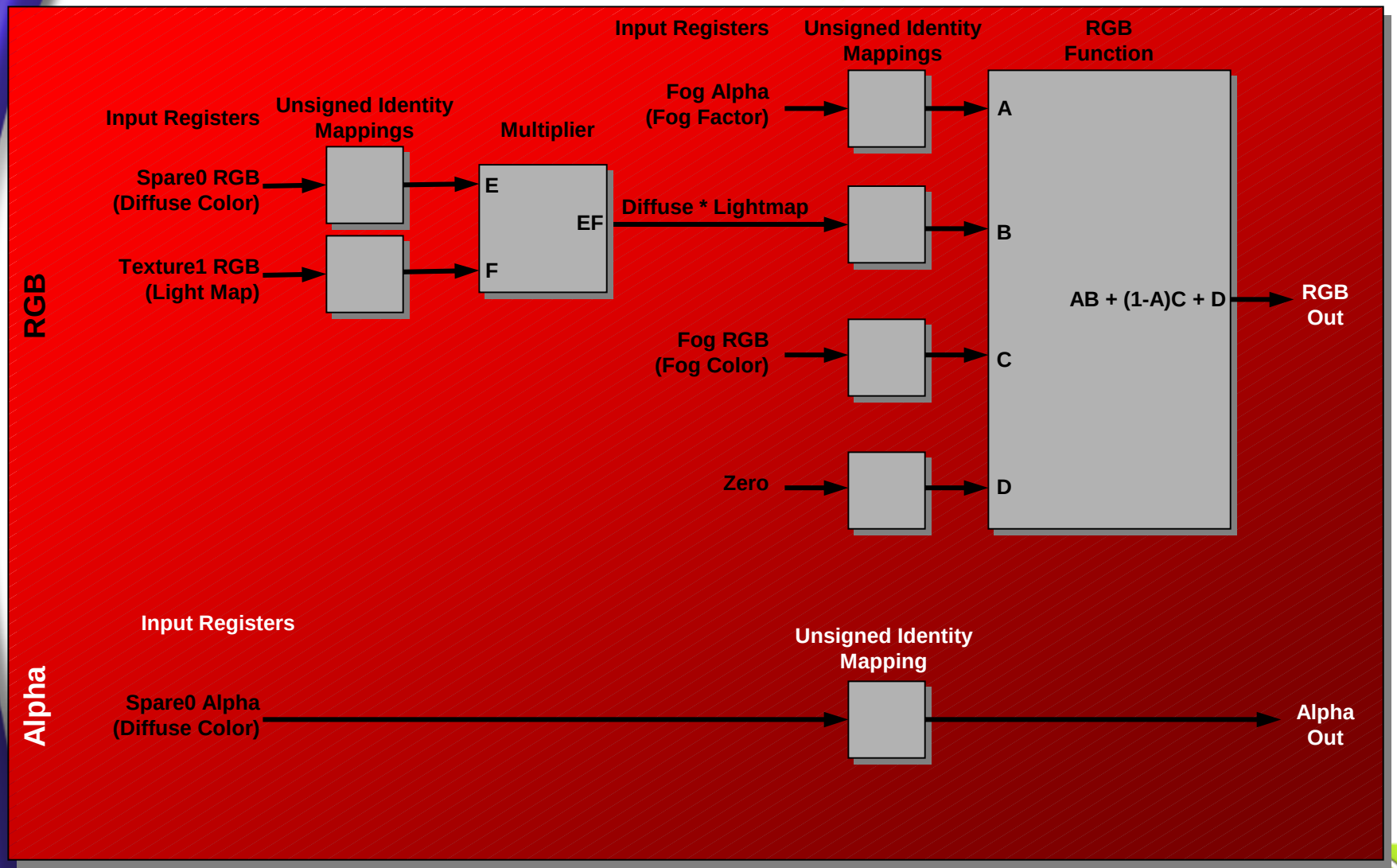
Disadvantages: none

Light Mapped Multitexture with Fog
General Combiner 0 Setup



NVIDIA

Light Mapped Texture with Fog
Final Combiner Setup



NVIDIA

Single Combiner Bump Mapping

Per-pixel diffuse and specular bump mapping using one pass

Texture0 is the normal map (N'), Texture1 is the diffuse texture map C_D

ConstantColor0 is set to specular material color C_S

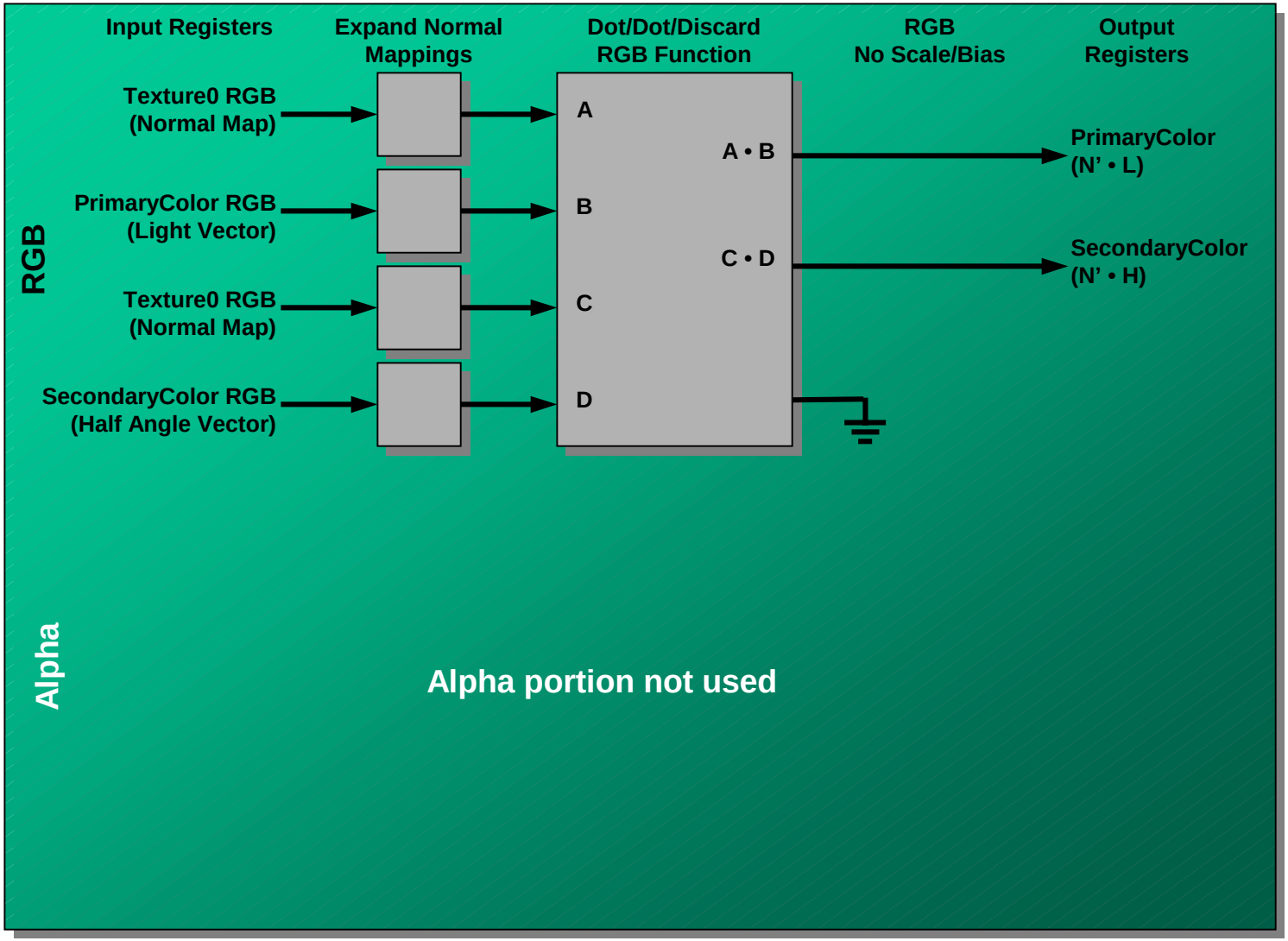
Computes $C_D * (N' \bullet L) + C_S * (N' \bullet H)$

Uses two textures, but only one general combiner

Advantages: performance (single pass, no blending)

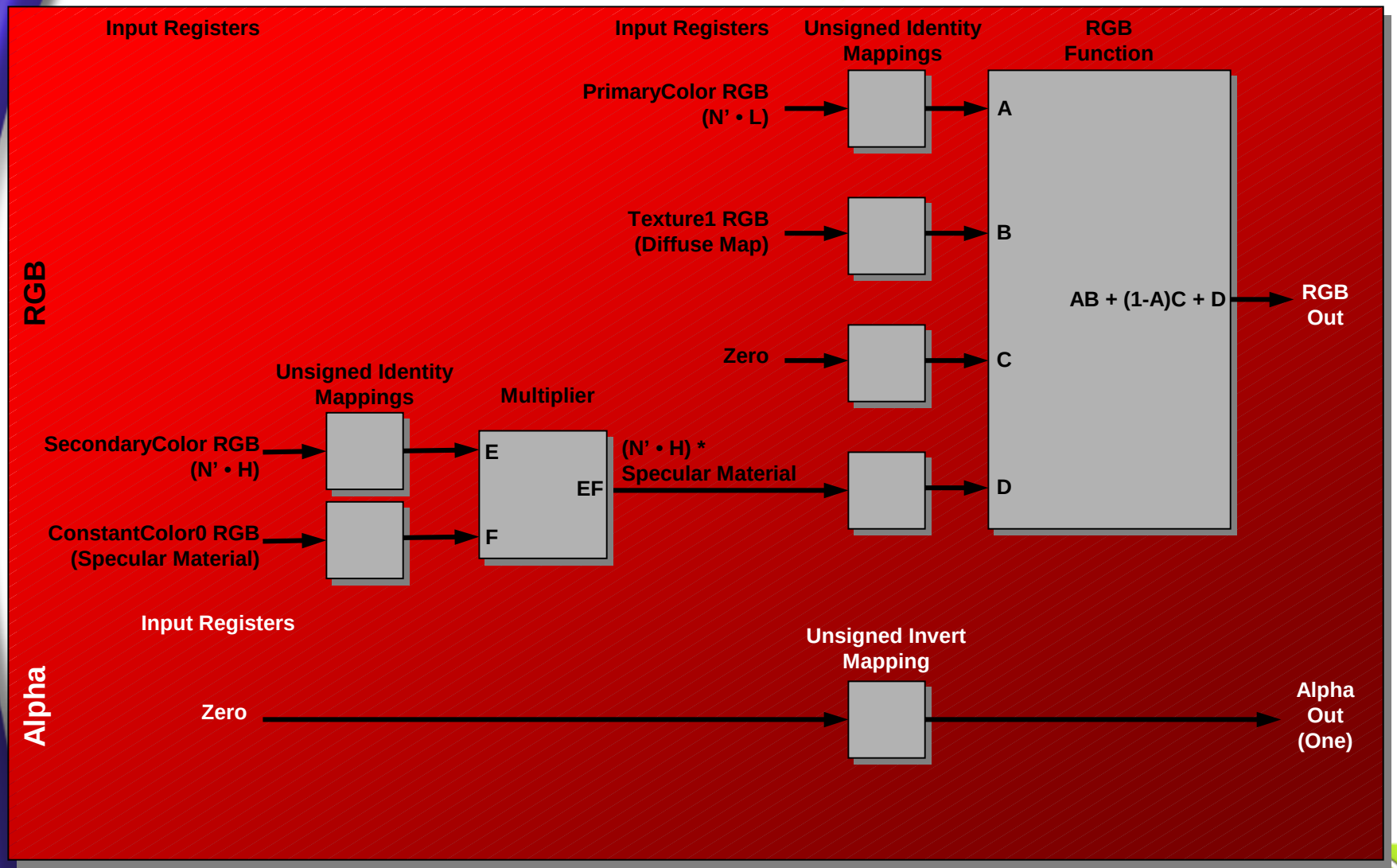
Disadvantages: low image quality (no normalization, no specular exponent, no fog)

Single Combiner Bump Mapping
General Combiner 0 Setup



NVIDIA

Single Combiner Bump Mapping
Final Combiner Setup



NVIDIA

More Robust Bump Mapping

Per-pixel diffuse/specular bump mapping with specular intensity map

Texture0 is the normal map (N'), Texture1 holds diffuse texture map C_D and specular intensity map A_S

Primary (diffuse) color is the unnormalized light vector (L)

Secondary (specular) color is the unnormalized half-angle vector (H)

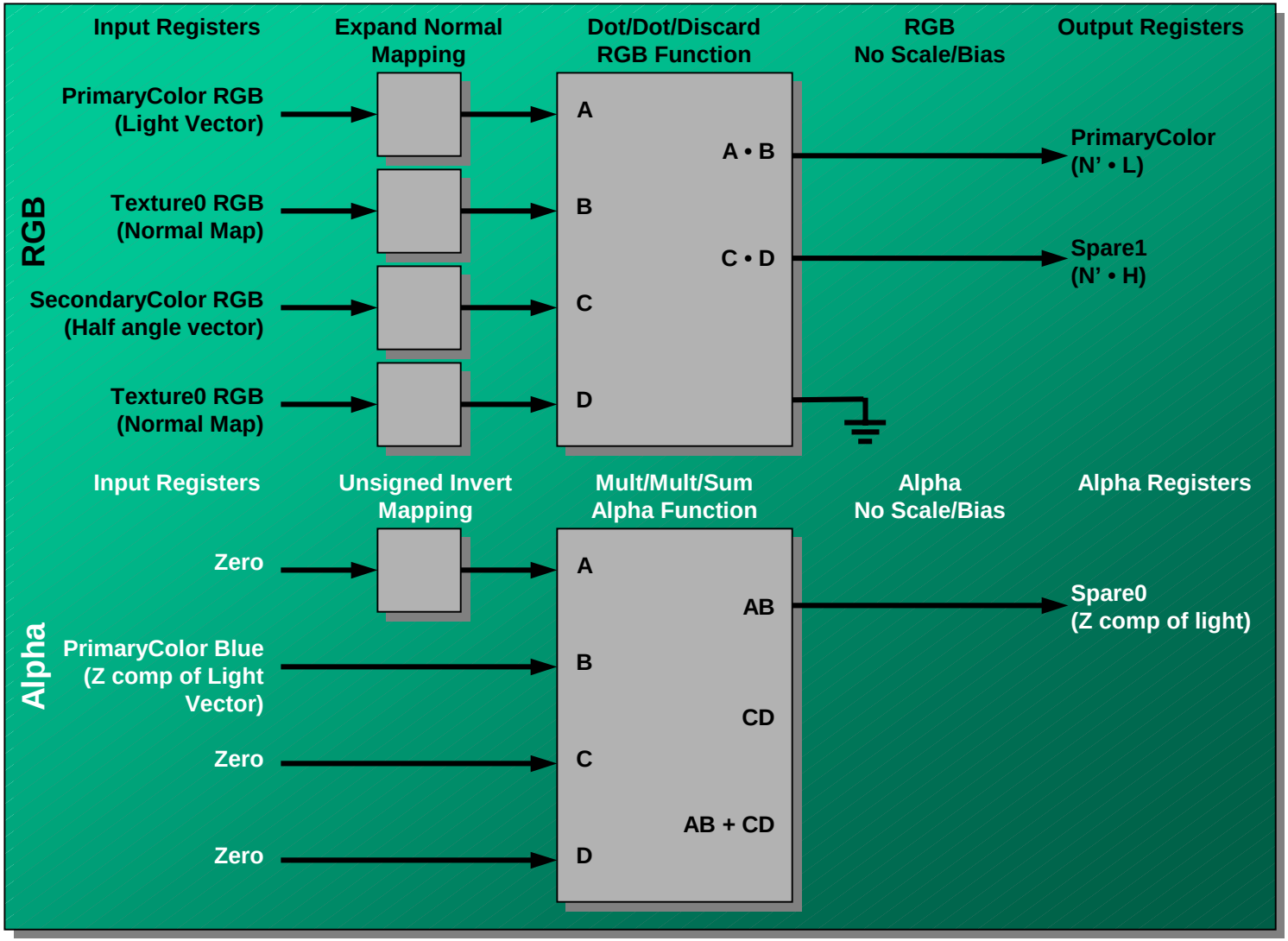
Computes $C_D * (N' \bullet L) + A_S * (N' \bullet H)^5$

Uses one pass, two textures, two general combiner

Advantages: better image quality, self-shadowing, very fast

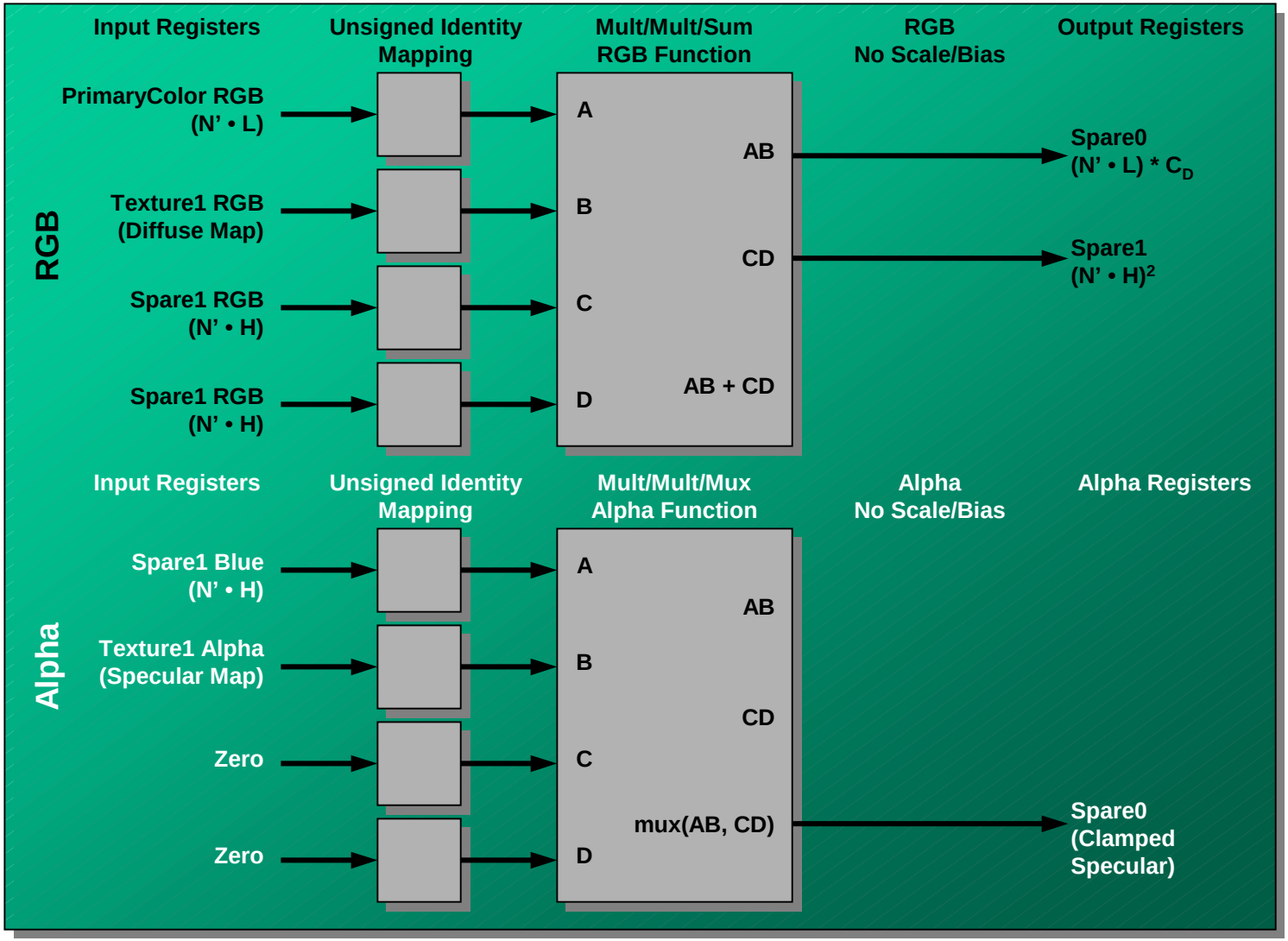
Disadvantages: no normalization of light/half-angle vectors can cause rendering artifacts, particularly with coarse surface tessellation

More Robust Bump Mapping
General Combiner 0 Setup



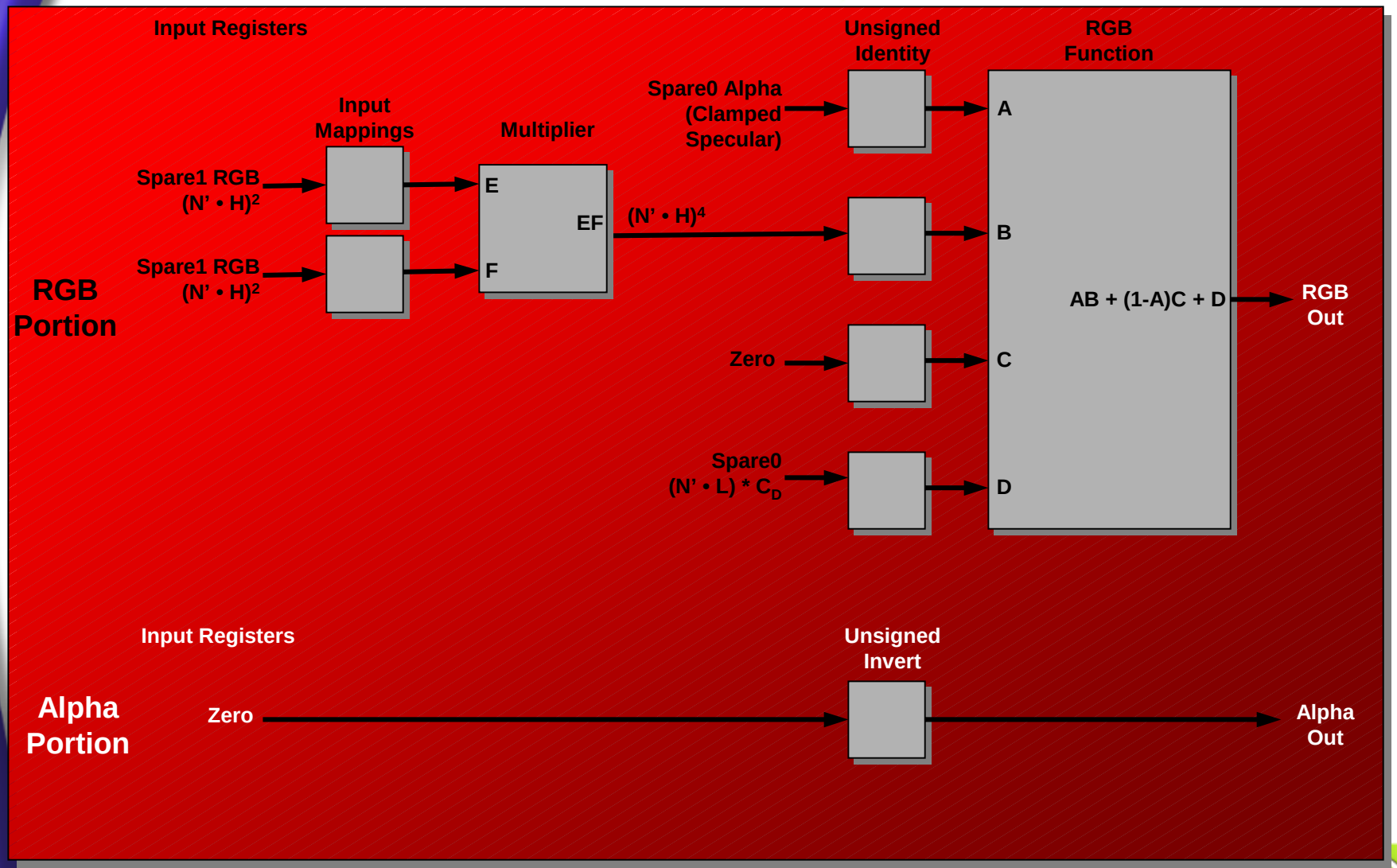
NVIDIA

More Robust Bump Mapping
General Combiner 1 Setup



NVIDIA

More Robust Bump Mapping
Final Combiner Setup



NVIDIA

NV_register_combiners
OpenGL API
Register Combiners Enable/Disable

Enable register combiners

```
glEnable(GL_REGISTER_COMBINERS_NV);
```

Disable register combiners

```
glDisable(GL_REGISTER_COMBINERS_NV);
```

NV_register_combiners
OpenGL API
General Combiner Input/Output Control

glCombinerInputNV(**GLenum stage,**
GLenum portion,
GLenum variable,
GLenum input,
GLenum mapping,
GLenum componentUsage);

glCombinerOutputNV(**GLenum stage,**
GLenum portion,
GLenum abOutput,
GLenum cdOutput,
GLenum sumOutput,
GLenum scale,
GLenum bias,
GLboolean abDotProduct,
GLboolean cdDotProduct,
GLboolean muxSum);

Red indicates specifiers
Blue indicates state values

NVIDIA

NV_register_combiners
OpenGL API
General Combiner Input/Output Control Tokens

stage =	{ GL_COMBINER0_NV, GL_COMBINER1_NV }
portion =	{ GL_RGB, GL_ALPHA }
variable =	{ GL_VARIABLE_A_NV, GL_VARIABLE_B_NV, GL_VARIABLE_C_NV, GL_VARIABLE_D_NV }
input =	{ GL_ZERO, GL_PRIMARY_COLOR_NV, GL_SECONDARY_COLOR_NV, GL_CONSTANT_COLOR0_NV, GL_CONSTANT_COLOR1_NV, GL_TEXTURE0_ARB, GL_TEXTURE1_ARB, GL_FOG, GL_SPARE0_NV, GL_SPARE1_NV }
mapping =	{ GL_UNSIGNED_IDENTITY_NV, GL_UNSIGNED_INVERT_NV, GL_EXPAND_NORMAL_NV, GL_EXPAND_NEGATE_NV, GL_HALF_BIAS_NORMAL_NV, GL_HALF_BIAS_NEGATE_NV, GL_SIGNED_IDENTITY_NV, GL_SIGNED_NEGATE_NV }
componentUsage =	{ GL_RGB, GL_BLUE, GL_ALPHA }
abOutput, cdOutput, sumOutput =	{ GL_DISCARD_NV, GL_PRIMARY_COLOR_NV, GL_SECONDARY_COLOR_NV, GL_TEXTURE0_ARB, GL_TEXTURE1_ARB, GL_SPARE0_NV, GL_SPARE1_NV }
scale =	{ GL_NONE, GL_SCALE_BY_TWO_NV, GL_SCALE_BY_FOUR_NV, GL_SCALE_BY_ONE_HALF_NV }
bias =	{ GL_NONE, GL_BIAS_BY_NEGATIVE_ONE_HALF_NV }

NVIDIA

NV_register_combiners
OpenGL API
Final Combiner Input Control

glFinalCombinerInputNV(**GLenum variable,**
GLenum input,
GLenum mapping,
GLenum componentUsage);

Red indicates specifiers
Blue indicates state values

NVIDIA

NV_register_combiners
OpenGL API
Final Combiner Input Control Tokens

```
variable =          { GL_VARIABLE_A_NV, GL_VARIABLE_B_NV,
                      GL_VARIABLE_C_NV, GL_VARIABLE_D_NV,
                      GL_VARIABLE_E_NV, GL_VARIABLE_F_NV,
                      GL_VARIABLE_G_NV  }
input =             { GL_ZERO, GL_PRIMARY_COLOR_NV, GL_SECONDARY_COLOR_NV,
                      GL_CONSTANT_COLOR0_NV, GL_CONSTANT_COLOR1_NV,
                      GL_TEXTURE0_ARB, GL_TEXTURE1_ARB, GL_FOG,
                      GL_SPARE0_NV, GL_SPARE1_NV,
                      GL_E_TIMES_F_NV, GL_SPARE0_PLUS_SECONDARY_COLOR_NV  }
mapping =           { GL_UNSIGNED_IDENTITY_NV, GL_UNSIGNED_INVERT_NV }
componentUsage =    { GL_RGB, GL_BLUE, GL_ALPHA }
```

NV_register_combiners
OpenGL API
Combiner Parameter Control

```
glCombinerParameterfvNV(GLenum pnamev,
                        const GLfloat *params);

glCombinerParameterivNV(GLenum pname,
                        const GLint *params);

glCombinerParameterfNV(GLenum pname,
                       GLfloat param);

glCombinerParameteriNV(GLenum pnamev,
                       GLint param);
```

NV_register_combiners
OpenGL API
Combiner Parameter Control Tokens

pname = { GL_NUM_COMBINERS_NV, GL_COLOR_SUM_CLAMP_NV }

pnamev = { GL_NUM_COMBINERS_NV, GL_COLOR_SUM_CLAMP_NV, GL_CONSTANT_COLOR0_NV, GL_CONSTANT_COLOR1_NV }

NV_register_combiners OpenGL API Combiner State Queries

```
glGetCombinerInputParameterfvNV(GLenum stage,
                                GLenum portion,
                                GLenum variable,
                                GLenum pname_i,
                                GLfloat *params);

glGetCombinerInputParameterivNV(GLenum stage,
                                GLenum portion,
                                GLenum variable,
                                GLenum pname_i,
                                GLint *params);

glGetCombinerOutputParameterfvNV(GLenum stage,
                                 GLenum portion,
                                 GLenum pname_o,
                                 GLfloat *params);

glGetCombinerOutputParameterivNV(GLenum stage,
                                 GLenum portion,
                                 GLenum pname_o,
                                 GLint *params);

glGetFinalCombinerInputParameterfvNV(GLenum fvariable,
                                     GLenum pname_i,
                                     GLfloat *params);

glGetFinalCombinerInputParameterivNV(GLenum fvariable,
                                     GLenum pname_i,
                                     GLfloat *params);
```

Red indicates specifiers

NVIDIA

NV_register_combiners
OpenGL API
Combiner State Queries Tokens

stage =	{ GL_COMBINER0_NV, GL_COMBINER1_NV }
portion =	{ GL_RGB, GL_ALPHA }
variable =	{ GL_VARIABLE_A_NV, GL_VARIABLE_B_NV, GL_VARIABLE_C_NV, GL_VARIABLE_D_NV }
fvariable =	{ GL_VARIABLE_A_NV, GL_VARIABLE_B_NV, GL_VARIABLE_C_NV, GL_VARIABLE_D_NV, GL_VARIABLE_E_NV, GL_VARIABLE_F_NV, GL_VARIABLE_G_NV }
pname_i =	{ GL_COMBINER_INPUT_NV, GL_COMBINER_MAPPING_NV, GL_COMPONENT_USAGE_NV }
pname_o =	{ GL_COMBINER_AB_DOT_PRODUCT_NV, GL_COMBINER_CD_DOT_PRODUCT_NV, GL_COMBINER_MUX_SUM_NV, GL_COMBINER_SCALE_NV, GL_COMBINER_BIAS_NV, GL_COMBINER_AB_OUTPUT_NV, GL_COMBINER_CD_OUTPUT_NV, GL_COMBINER_SUM_OUTPUT_NV }