Texture Shaders

Sébastien Dominé and John Spitzer
NVIDIA Corporation

Session Agenda

Overview of the texture subsystem

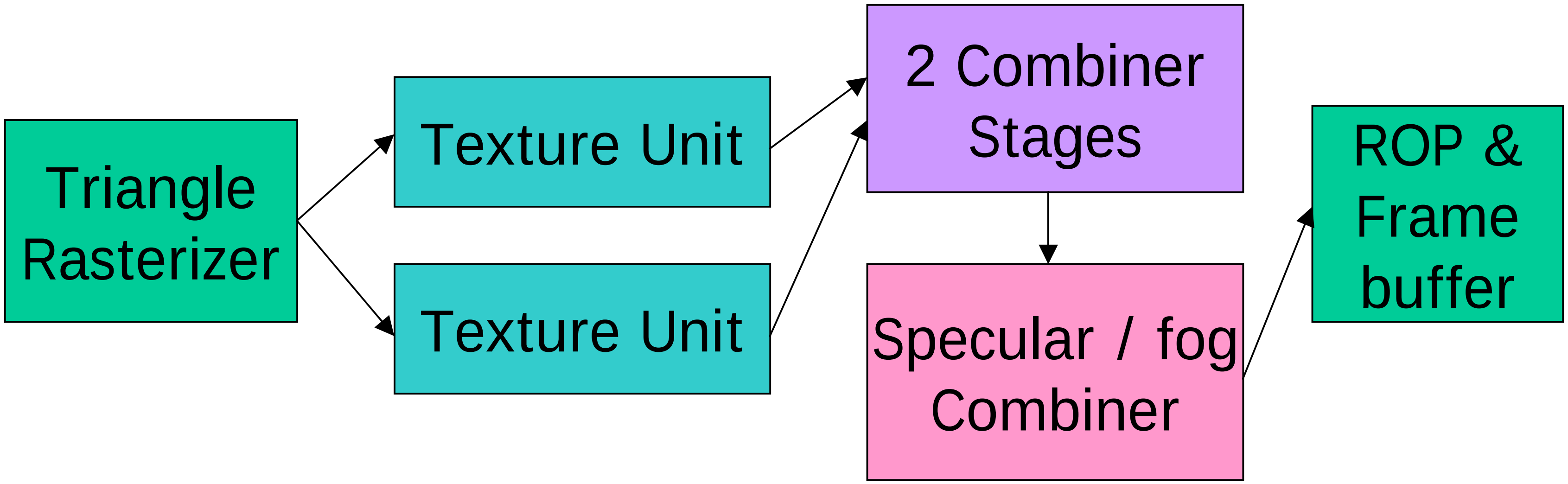
What are texture shader operations?

Texture shader operations

- Conventional textures
- Special modes
- Simple dependent textures
- Dot product dependent textures
- Depth replace

GeForce2 Texture Shading Pipeline

OpenGL + ARB_multitexture
+ ARB_texture_cube_map + NV_register_combiners



GeForce2 Details

Texture Unit

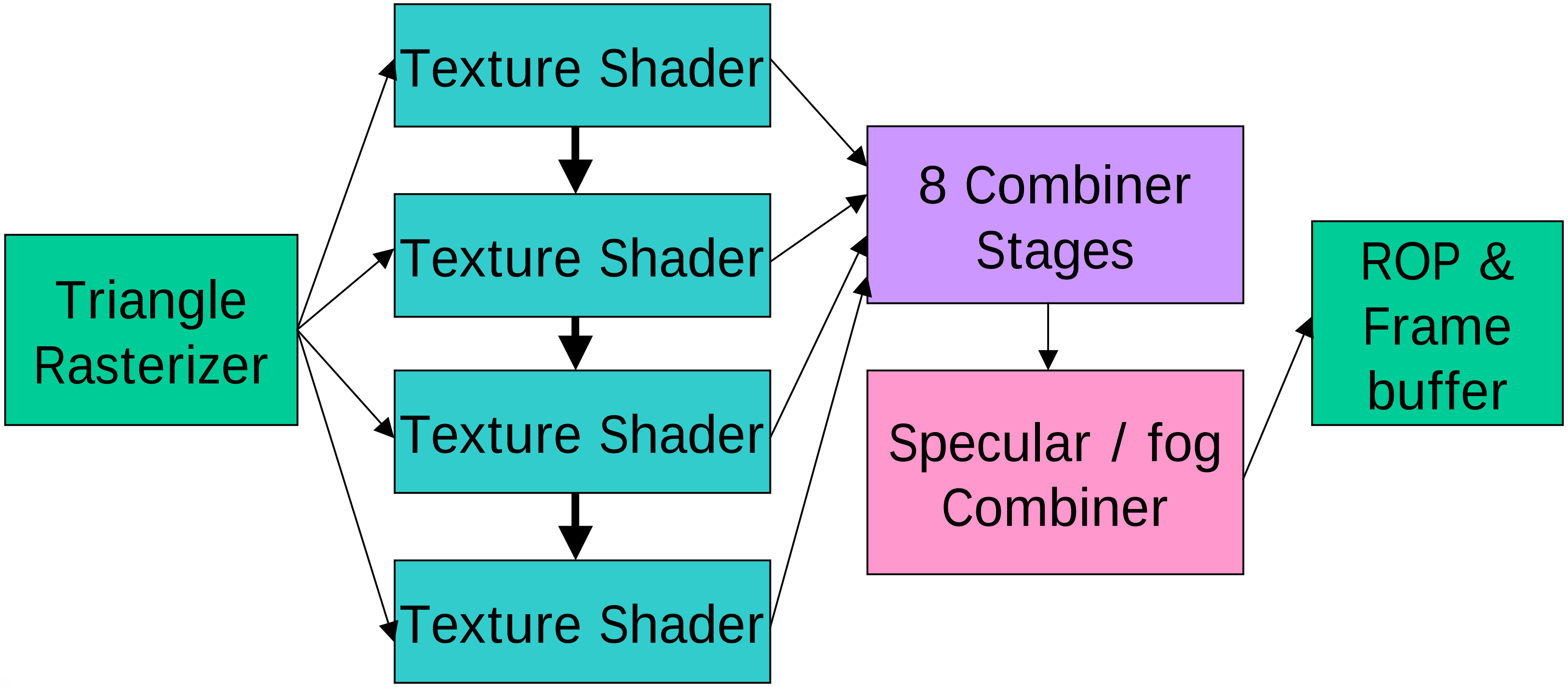
- 2 projective 2D textures
- Performs texture fetch and filtering
- No dependent texture operations
- Cube maps

Register Combiners

- 2 texture blending units
- 1 final combiner (specular/fog)
- Signed math
- Dot products (for bump mapping)

GeForce3 Fragment Pipeline

OpenGL with NV_texture_shader (also includes 4 texture units, 3D textures, and 8 combiners)



GeForce3 Details

Texture shader

- 4 texture units
- 23 different texture shader operations
  - Conventional (1D, 2D, 3D, texture rectangle, cube map)
  - Special case (none, pass through, cull fragment)
  - Dependent texture fetches (result of one texture lookup affects texture coords for subsequent unit)
  - Dependent textures fetches with dot product (and optional reflection) calculations

Register combiners

- 8 stages (general combiners) on GeForce3
- Per-stage constants

Texture Shader “Bridge”

(s_0,t_0,r_0,q_0)

(s_1,t_1,r_1,q_1)

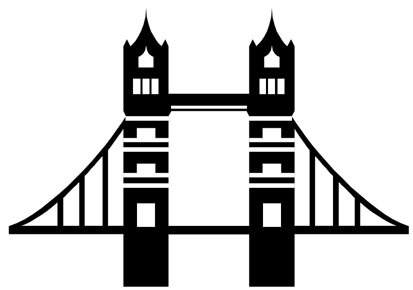
(s_2,t_2,r_2,q_2)

(s_3,t_3,r_3,q_3)

Interpolated texture coordinate sets

32-bit IEEE floating-point

Per-component



(R_0,G_0,B_0,A_0)

(R_1,G_1,B_1,A_1)

(R_2,G_2,B_2,A_2)

(R_3,G_3,B_3,A_3)

RGBA colors

8-bit [0,1] or [-1,1) fixed-point

Per-component

Texture shader and Texture fetch units State

Texture Shaders

Provides a superset of conventional OpenGL texture addressing

Five main categories of shader operations

- Conventional textures
  - 1D, 2D, texture rectangle, cube map
- Special modes
  - none, pass through, cull fragment
- Direct dependent textures
  - dependent AR, dependent GB, offset, offset scaled
- Dot product (DP) dependent textures
  - DP 2D, DP texture rectangle, DP cube map, DP reflect cube map, DP diffuse cube map
- Depth replace operations

Texture Shader Considerations

When texture shaders are enabled, they are *ALL* enabled ("big switch" model)

Select a shader operation of GL_NONE for stages you are not using

Several texture shader operations return texture values of (0,0,0,0) – if not using register combiners, ensure TEX_ENV_MODE is GL_NONE

Shader operations determine which texture is accessed (if any) as opposed to un-extended OpenGL, where enabled texture targets have pre-set precedence

Enabling Texture Shader Mode

- New enable

  glEnable(GL_TEXTURE_SHADER_NV);

  glDisable(GL_TEXTURE_SHADER_NV);

- Existing texture enables are ignored when texture shader mode is enabled,
  i.e. glEnable(GL_TEXTURE_2D), etc.
- Setting texture shader operations:

```
glTexEnvi(GL_TEXTURE_SHADER_NV,
                    GL_SHADER_OPERATION_NV, GL_TEXTURE_2D);
```

- One texture shader operation per texture unit.

Conventional Texture Shaders

Conventional textures

- Texture 1D
- Texture 2D
- Texture 3D
- Texture rectangle
- Texture cube map

Note about GeForce3 3D Texture Support

Conventional textures

- Texture 1D
- Texture 2D
- Texture 3D
- Texture rectangle
- Texture cube map

Production GeForce3 and Quadro DCC boards *do* fully support 3D textures including mipmapping (see the NV_texture_shader2 extension)

Texture 1D

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
i	(S_i,T_i,R_i,Q_i)	Texture 1D	$\left(\frac{S_i}{Q_i}\right)$	1D Any Format	(R,G,B,A)

```
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_TEXTURE_1D);

nvparse( "!!TS1.0
          texture_1d();" );
```

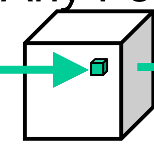
Texture 2D

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
i	(S_i,T_i,R_i,Q_i)	Texture 2D	$(\frac{S_i}{Q_i},\frac{T_i}{Q_i})$	2D Any Format	(R,G,B,A)

```
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_TEXTURE_2D);

nvparse( "!!TS1.0
          texture_2d();" );
```

Texture 3D

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
i	(S_i,T_i,R_i,Q_i)	Texture 3D	$\left(\frac{S_i}{Q_i}, \frac{T_i}{Q_i}, \frac{R_i}{Q_i}\right)$	3D Any Format 	(R,G,B,A)

```
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
            GL_TEXTURE_3D);

nvparse( "!!TS1.0
            texture_3d();" );
```

Texture Rectangle

New texture target defined via new extension – NV_texture_rectangle

Allows for non-power-of-2 width and height (e.g. 640x480)

S and T texture coords address [0,width] and [0,height] respectively, instead of [0,1] as in Texture 2D

No mipmaps

Clamp modes supported:

- GL_CLAMP
- GL_CLAMP_TO_EDGE
- GL_CLAMP_TO_BORDER_ARB

Can be used independently from texture shader (supported by GeForce 1&2)

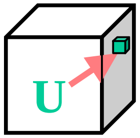
Texture Rectangle

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
i	(S_i,T_i,R_i,Q_i)	Texture Rectangle	$(\frac{S_i}{Q_i},\frac{T_i}{Q_i})$	Texture Rectangle Any Format	(R,G,B,A)

```
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_TEXTURE_RECTANGLE_NV);
```

```
nvparse( "!!TS1.0
          texture_rectangle();" );
```

Texture Cube Map

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
i	(S_i,T_i,R_i)	Texture Cube Map	$U=(S_i, T_i, R_i)$	Cube Map Any Format 	(R,G,B,A)

```
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
        GL_TEXTURE_CUBE_MAP_ARB);

nvparse( "!!TS1.0
            texture_cube_map();" );
```

Special Mode Texture Shaders

Special Modes

- None
- Pass through
- Cull fragment

None

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
i	Ignored	R = 0 G = 0 B = 0 A = 0	None	None →	(R,G,B,A)

```
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV, GL_NONE);
```

```
nvparse( "!!TS1.0
          nop();" );
```

Pass Through

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
i	(S_i,T_i,R_i,Q_i) →	$R = \text{Clamp0to1}(S_i)$ $G = \text{Clamp0to1}(T_i)$ $B = \text{Clamp0to1}(R_i)$ $A = \text{Clamp0to1}(Q_i)$	None	None →	(R,G,B,A)

Pass Through

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
i	(S_i,T_i,R_i,Q_i)	R = Clamp0to1(S_i) G = Clamp0to1(T_i) B = Clamp0to1(R_i) A = Clamp0to1(Q_i)	None	None →	(R,G,B,A)

Pass Through

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
i	(S_i,T_i,R_i,Q_i)	R = Clamp0to1(S_i) G = Clamp0to1(T_i) B = Clamp0to1(R_i) A = Clamp0to1(Q_i)	None	None	(R,G,B,A)

```
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
        GL_PASS_THROUGH_NV);
```

```
nvparse( "!!TS1.0
            pass_through();" );
```

Cull Fragment

Cull the fragment based upon sign of texture coords

- each tex coord (STRQ) has its own settable condition
- each of the 4 conditions is set to one of the following:
  - GL_GEQUAL (tex coord ≥ 0) – pass iff positive or zero
  - GL_LESS (tex coord < 0) – pass iff negative
- all four tex coords are tested
- if any of the four fail, the fragment is rejected

No texture accesses

Texture output for passing fragments is (0,0,0,0)

Very useful for implementing per-pixel user-defined clip planes – up to 4 per texture unit (16 total!)

Cull Fragment

Use:

```
glActiveTextureARB( GL_TEXTURE0_ARB );

glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_CULL_FRAGMENT_NV);

GLint cullmode[4] = { GL_LESS, GL_GEQUAL, GL_LESS, GL_EQUAL };

glTexEnviv(GL_TEXTURE_SHADER_NV, GL_CULL_MODES_NV, cullmode);
```

Cull Fragment

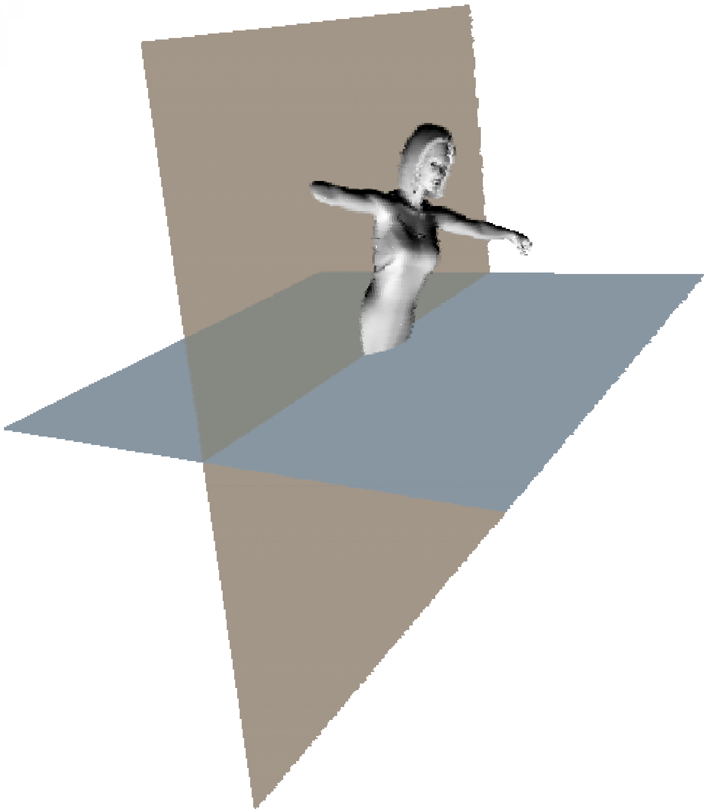
Using nvparse:

```
nvparse( “!!TS1.0
        cull_fragment( LESS_THAN_ZERO,
                       GEQUAL_TO_ZERO,
                       LESS_THAN_ZERO,
                       GEQUAL_TO_ZERO );” );
```

Cull Fragment Applications

- Per-fragment clip planes
  - Up to 4 clip planes per texture unit
  - 16 clip planes maximum
  - Easy to use in conjunction with GL_EYE_LINEAR texgen mode
- Non-planar clipping approaches also possible
  - Vertex programs can compute a distance to a point or line and use that interpolated distance for clipping

Cull Fragment Examples



Clipping a model to two texture shader clip planes

Clipping a 3D grid of cubes based on distance from a point

Simple Dependent Texture Shaders

- Take results of one texture, use them for addressing subsequent texture
- Single stage, not including source texture
- Simple dependent textures (single stage)
  - Dependent alpha-red
  - Dependent green blue
  - Offset texture 2D
  - Offset texture 2D scaled
- All diagrams from here on out start at texture unit 0 and use a contiguous series of texture units
  - This is an artificial restriction to ease in explaining the concepts of these texture shaders

Mipmapping Dependent Texture Accesses

- GeForce3 performs mipmap filtering on dependent texture accesses
- While on the subject . . .
  - GeForce 1/2/3 all properly mipmap cube maps
  - GeForce3 properly mipmaps 3D textures including support for GL_LINEAR_MIPMAP_LINEAR filtering

Dependent Alpha-Red Texturing

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific →	Texture specific	Texture specific	Any type Unsigned RGBA →	$R_0G_0B_0A_0$
1	Ignored	None	(A_0,R_0)	2D RGBA →	$R_1G_1B_1A_1$

Dependent Alpha-Red Texturing

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Unsigned RGBA	$R_0G_0B_0A_0$
1	Ignored	None	(A_0,R_0)	2D RGBA	$R_1G_1B_1A_1$

Dependent Alpha-Red Texturing

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Unsigned RGB[A]	$R_0G_0B_0A_0$
1	Ignored	None	(A_0,R_0)	2D RGBA	$R_1G_1B_1A_1$

```
glActiveTextureARB( GL_TEXTURE0_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_TEXTURE_2D);

glActiveTextureARB( GL_TEXTURE1_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_DEPENDENT_AR_TEXTURE_2D_NV );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
          GL_TEXTURE0_ARB);
```

Dependent Alpha-Red Texturing

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Unsigned RGB[A]	$R_0G_0B_0A_0$
1	Ignored	None	(A_0,R_0)	2D RGBA	$R_1G_1B_1A_1$

```
nvparse( "!!TS1.0
        texture_2d();
        dependent_ar( tex0 );" );
```

Dependent Green-Blue Texturing

Tex #	Texture Coords (S,T,R,Q)		Shader Operations	Texture Fetch	Bound Texture Target/Format		Output Color
0	App specific	→	Texture specific	Texture specific	Any type Unsigned RGB[A]	→	$R_0G_0B_0A_0$
1	Ignored		None	(G_0,B_0)	2D RGBA	→	$R_1G_1B_1A_1$

Dependent Green-Blue Texturing

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Unsigned RGB[A]	$R_0G_0B_0A_0$
1	Ignored	None	(G_0,B_0)	2D RGBA	$R_1G_1B_1A_1$

Dependent Green-Blue Texturing

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Unsigned RGB[A]	$R_0G_0B_0A_0$
1	Ignored	None	(G_0,B_0)	2D RGBA	$R_1G_1B_1A_1$

```
glActiveTextureARB( GL_TEXTURE0_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_TEXTURE_2D);

glActiveTextureARB( GL_TEXTURE1_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_DEPENDENT_GB_TEXTURE_2D_NV );

glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
          GL_TEXTURE0_ARB);
```

Dependent Green-Blue Texturing

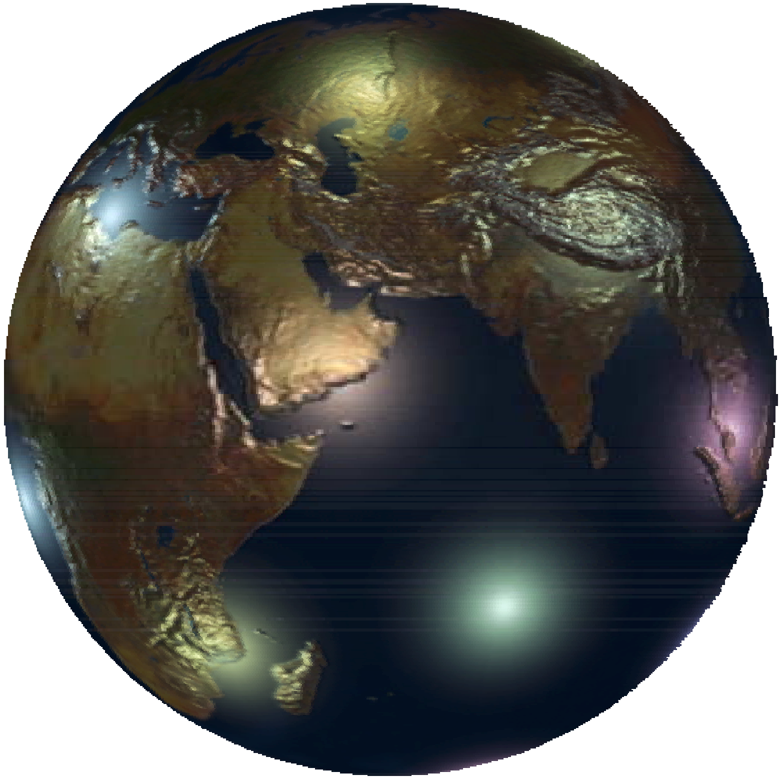
Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Unsigned RGB[A]	$R_0G_0B_0A_0$
1	Ignored	None	(G_0,B_0)	2D RGBA	$R_1G_1B_1A_1$

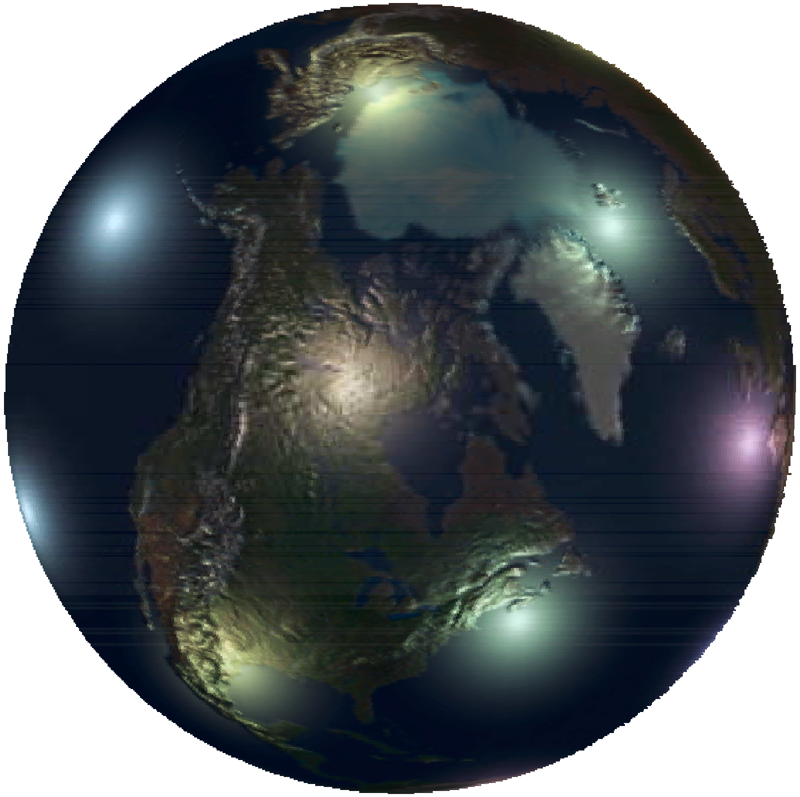
```
nvparse( "!!TS1.0
        texture_2d();
        dependent_gb( tex0 );" );
```

Offset Texture 2D

- Use previous lookup (a signed 2D offset) to perturb the texture coordinates of a subsequent (non-projective) 2D texture lookup
- Signed 2D offset is transformed by user-defined 2x2 matrix (shown in the following diagrams as constants k_0-k_3)
- This 2x2 constant matrix allows for arbitrary rotation/scaling of offset vector
- This shader operation can be used for what is called Environment-Mapped Bump Mapping (EMBM) in DirectX 6 lingo (though it's really a misnomer)
- Offset defined in DS/DT texture

Offset Texture 2D Example



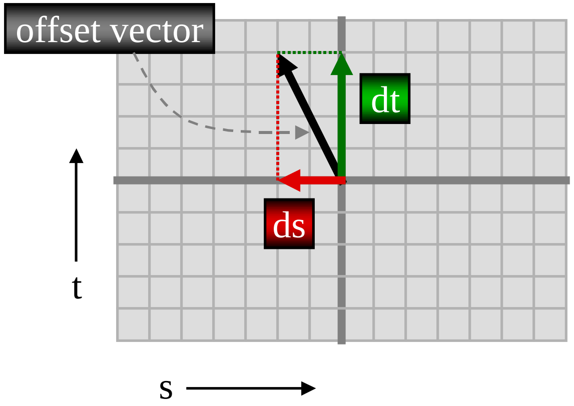


Pseudo bump mapping of a disco earth

Wait... What is a DS/DT Texture?

This format encodes a 2D offset vector in texture space

- ds **and** dt **are mapped to the range [-1,1]**



Magnitude (MAG) and MAG/Intensity flavors use the third and fourth component to optionally include scaling and intensity

OpenGL Formats for DS/DT Textures

New internal texture formats:

- GL_DSDT_NV
- GL_DSDT_MAG_NV
- GL_DSDT_MAG_INTENSITY_NV

New external texture formats:

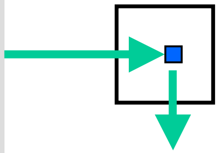
- GL_DSDT_NV
- GL_DSDT_MAG_NV
- GL_DSDT_MAG_VIB_NV

Offset Texture 2D

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	(S_0,T_0,R_0,Q_0) →	Texture 2D	$(\frac{S_0}{Q_0}, \frac{T_0}{Q_0})$	2D DSDT →	(0,0,0,0)
1	(S_1,T_1) →	$S_1' = S_1 + k_0 * ds + k_2 * dt$ $T_1' = T_1 + k_1 * ds + k_3 * dt$	(S_1', T_1')	2D Any Format →	$R_1G_1B_1A_1$

k_0, k_1, k_2 and k_3 define a constant 2x2 floating-point matrix set by glTexEnv

Offset Texture 2D

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	(S_0,T_0,R_0,Q_0) →	Texture 2D	$\left(\frac{S_0}{Q_0}, \frac{T_0}{Q_0}\right)$	2D DSDT  (ds,dt) →	(0,0,0,0)
1	(S_1,T_1) →	$S_1' = S_1 + k_0{*}ds + k_2{*}dt$ $T_1' = T_1 + k_1{*}ds + k_3{*}dt$	(S_1', T_1')	2D Any Format →	$R_1G_1B_1A_1$

k_0, k_1, k_2 and k_3 define a constant 2x2 floating-point matrix set by glTexEnv

Offset Texture 2D

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	(S_0,T_0,R_0,Q_0)	Texture 2D	$\left(\frac{S_0}{Q_0}, \frac{T_0}{Q_0}\right)$	2D DSDT (ds, dt)	(0,0,0,0)
1	(S_1,T_1)	$S_1' = S_1 + k_0 * ds + k_2 * dt$ $T_1' = T_1 + k_1 * ds + k_3 * dt$	(S_1', T_1')	2D Any Format	$R_1G_1B_1A_1$

k_0, k_1, k_2 and k_3 define a constant 2x2 floating-point matrix set by glTexEnv

Offset Texture 2D

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	(S_0,T_0,R_0,Q_0)	Texture 2D	$(\frac{S_0}{Q_0}, \frac{T_0}{Q_0})$	2D DSDT (ds,dt)	(0,0,0,0)
1	(S_1,T_1)	$S_1' = S_1 + k_0*ds + k_2*dt$ $T_1' = T_1 + k_1*ds + k_3*dt$	(S_1', T_1')	2D Any Format	$R_1G_1B_1A_1$

k_0, k_1, k_2 and k_3 define a constant 2x2 floating-point matrix set by glTexEnv

Offset Texture 2D GL_OFFSET_TEXTURE_2D_NV

Previous texture input internal texture format must be one of:

- **GL_DSDT_NV**
- **GL_DSDT_MAG_NV**
- **GL_DSDT_MAG_INTENSITY_NV**

```
glActiveTextureARB( GL_TEXTURE0_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_TEXTURE_2D);

glActiveTextureARB( GL_TEXTURE1_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
           GL_OFFSET_TEXTURE_2D_NV );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
           GL_TEXTURE0_ARB);
glTexEnvfv(GL_TEXTURE_SHADER_NV,GL_OFFSET_TEXTURE_MATRIX_NV, mat2d);
```

Offset Texture 2D
GL_OFFSET_TEXTURE_2D_NV

Using nvparse:

```
nvparse( "!!TS1.0
          texture_2d();
          offset_2d( tex0, .5, 0, 0, .5 );" );
```

Offset Texture 2D Scale

- Same as Offset Texture 2D, except that subsequent (non-projective) 2D texture RGB output is scaled
- Scaling factor is the MAG component (from previous texture) scaled/biased by user-defined constants (k_{scale} and k_{bias} in the following diagrams)
- Alpha component is *NOT* scaled
- Unless GL_DSDT_MAG_INTENSITY_NV format is used, the previous texture output is (0,0,0,0)
- For GL_DSDT_MAG_INTENSITY_NV, the previous texture output is the intensity component

Offset Texture 2D Scale

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	(S_0,T_0,R_0,Q_0) →	Texture 2D	$(\frac{S_0}{Q_0}, \frac{T_0}{Q_0})$	2D DSDT_Mag (ds,dt,mag) →	(0,0,0,0)
1	(S_1,T_1) →	$S_1' = S_1 + k_0 * ds + k_2 * dt$ $T_1' = T_1 + k_1 * ds + k_3 * dt$ $M = k_{scale} * mag + k_{bias}$	(S_1', T_1')	2D RGBA →	$(R * M,$ $G * M,$ $B * M, A)$

k_0, k_1, k_2 and k_3 define a constant floating-point 2x2 matrix set by glTexEnv
k_{scale} and k_{bias} define constant floating-point scale/bias set by glTexEnv

Offset Texture 2D Scale

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	(S_0,T_0,R_0,Q_0)	Texture 2D	$(\frac{S_0}{Q_0}, \frac{T_0}{Q_0})$	2D DSDT_Mag (ds, dt, mag)	(0,0,0,0)
1	(S_1,T_1)	$S_1' = S_1 + k_0 * ds + k_2 * dt$ $T_1' = T_1 + k_1 * ds + k_3 * dt$ $M = k_{scale} * mag + k_{bias}$	(S_1', T_1')	2D RGBA	$(R*M, G*M, B*M, A)$

k_0, k_1, k_2 and k_3 define a constant 2x2 floating-point matrix set by glTexEnv

k_{scale} and k_{bias} define constant floating-point scale/bias set by glTexEnv

Offset Texture 2D Scale

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	(S_0,T_0,R_0,Q_0)	Texture 2D	$(\frac{S_0}{Q_0}, \frac{T_0}{Q_0})$	2D DSDT_Mag (ds,dt,mag)	(0,0,0,0)
1	(S_1,T_1)	$S_1' = S_1 + k_0*ds + k_2*dt$ $T_1' = T_1 + k_1*ds + k_3*dt$ $M = k_{scale} * mag + k_{bias}$	(S_1', T_1')	2D RGBA (R,G,B,A)	$(R*M, G*M, B*M, A)$

k_0, k_1, k_2 and k_3 define a constant 2x2 floating-point matrix set by glTexEnv

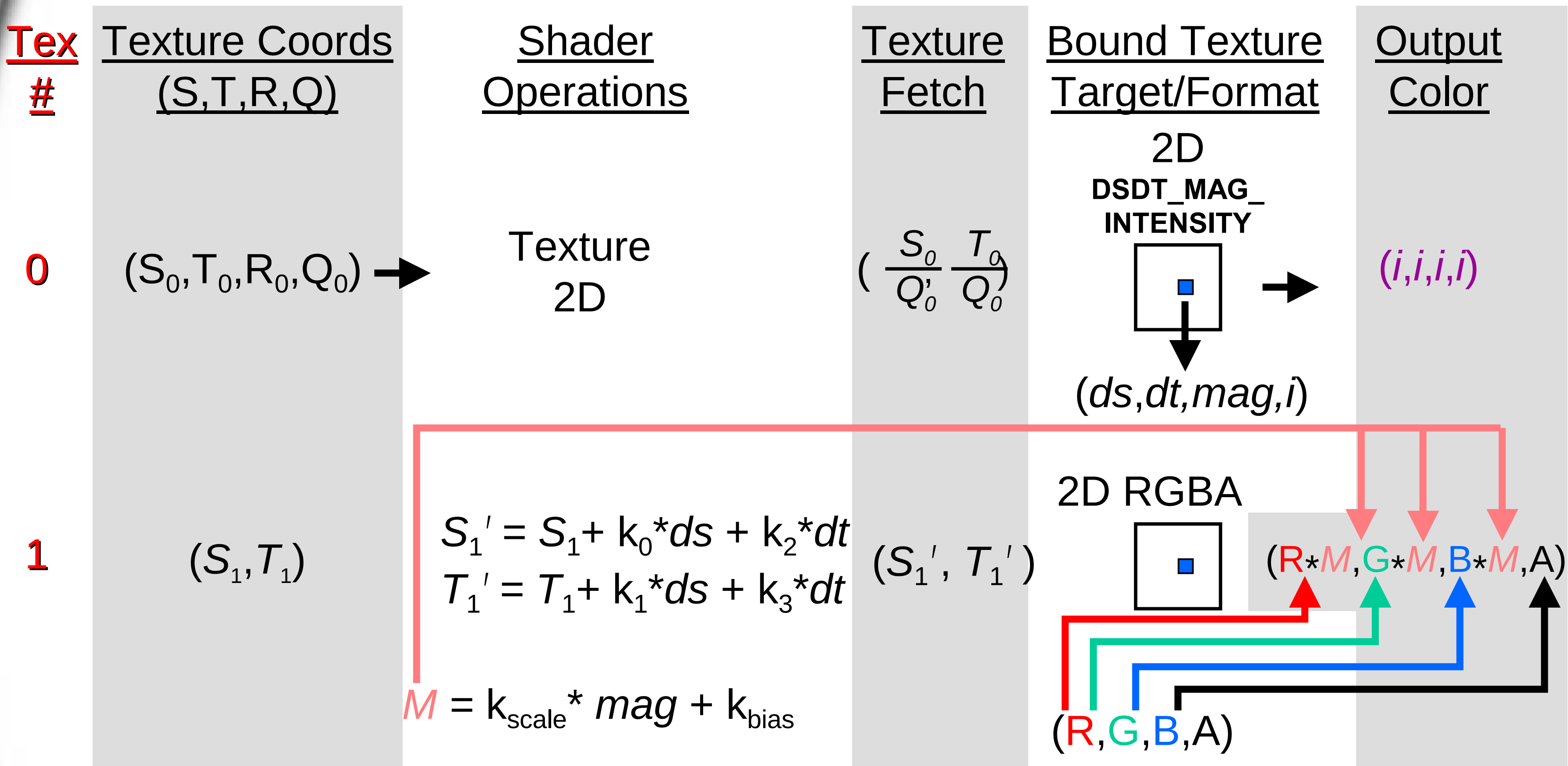
k_{scale} and k_{bias} define constant scale/bias set by glTexEnv

Offset Texture 2D Scale

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	(S_0,T_0,R_0,Q_0)	Texture 2D	$(\frac{S_0}{Q_0}, \frac{T_0}{Q_0})$	2D DSDT_Mag (ds, dt, mag)	(0,0,0,0)
1	(S_1,T_1)	$S_1' = S_1 + k_0*ds + k_2*dt$ $T_1' = T_1 + k_1*ds + k_3*dt$ $M = k_{scale} * mag + k_{bias}$	(S_1', T_1')	2D RGBA (R,G,B,A)	(R*M,G*M,B*M,A)

As with all output colors, each scaled RGB component is clamped to [0,1]

Offset Texture 2D Scale using DSDT_MAG_INTENSITY



DSDT_MAG_INTENSITY format outputs intensity instead of 0s in tex unit 0

Offset Texture 2D and Scale
GL_OFFSET_TEXTURE_2D_SCALE_NV

Previous texture input base internal texture format must be either GL_DSDT_MAG_NV or GL_DSDT_MAG_INTENSITY_NV

```
glActiveTextureARB( GL_TEXTURE0_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV, GL_TEXTURE_2D);

glActiveTextureARB( GL_TEXTURE1_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_OFFSET_TEXTURE_2D_NV );

glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
          GL_TEXTURE0_ARB);

glTexEnvf(GL_TEXTURE_SHADER_NV,GL_OFFSET_TEXTURE_2D_BIAS_NV,0.5);

glTexEnvf(GL_TEXTURE_SHADER_NV,GL_OFFSET_TEXTURE_2D_SCALE_NV,2.0);
```

Offset Texture 2D and Scale
GL_OFFSET_TEXTURE_2D_SCALE_NV

Using nvparse:

```
nvparse( “!!TS1.0
        texture_2d();                          /* must be of DSDT_Mag format */
        offset_2d_scale( tex0, .5, 0, 0, .5, .5, 2.0 );” );
```

Offset Texture Issues

Limited precision in DSDT formats (max 8-bits per component)

- Don’t scale DS/DT values by more than 8 so as to preserve sub-texel precision
- Limits texture coord perturbation to [-8,8] (or so)
- Applications needing to perturb texture coords by more than this should use Dot Product Texture 2D (explained in next section) with HILO textures

Offset texturing also available for texture rectangles, in addition to 2D textures

- GL_OFFSET_TEXTURE_RECTANGLE_NV
- GL_OFFSET_TEXTURE_RECTANGLE_SCALE_NV

New Signed NV_texture_shader Texture Formats

- Signed [-1,1] Range Textures
  - DSDT formats are inherently signed for DS & DT
  - HILO (to be discussed) either signed or unsigned
    - GL_SIGNED_HILO16
  - Color signed internal formats (also un-sized versions)
    - GL_SIGNED_RGBA8_NV
    - GL_SIGNED_RGB8_NV
    - GL_SIGNED_LUMINANCE8_NV
    - GL_SIGNED_LUMINANCE8_ALPHA8_NV
    - GL_SIGNED_ALPHA8_NV
    - GL_SIGNED_INTENSITY8_NV
    - GL_SIGNED_RGB8_UNSIGNED_ALPHA8_NV

Signed Texture Formats Semantics

- DSDT formats useful only for *texture offset* shader operations
- Signed HILO and color formats are useful for dot product shader operations
- Signed color formats are signed for fragment-coloring
  - New signed conventional texture environment behavior
  - Register combiners texture registers initialized with signed color values if using signed color textures
  - 8-bit [-1,1) range; essentially 7 bits magnitude + sign

Dot Product Dependent Texture Shaders

Take results of one texture, perform 2 or 3 dot products with it and incoming texture coordinates, then use results for addressing subsequent texture(s)

Multiple contiguous stages, not including source texture

Dot product dependent textures

- Dot product texture 2D
- Dot product texture rectangle
- Dot product texture cube map
- Dot product constant eye reflect cube map
- Dot product reflect cube map
- Dot product diffuse cube map
- Dot product depth replace

Dot Product

- Simply calculates a high-precision dot product
- All dot product operations can be considered to perform this operation, the others just do something with the resulting scalars
- Source (previous) texture can have one of the following internal formats:
  - Signed RGBA (used in all the diagrams)
  - Unsigned RGBA (expandable to [-1,1])
  - Signed HILO
  - Unsigned HILO

RGBA texture formats

Very useful for arbitrary vector encoding

Signed RGB[A]

- New formats (GL_SIGNED_RGB_NV, etc.)
- Three (or four) 8-bit signed components
- All components are [-1,1]

Unsigned RGB[A]

- Three (or four) 8-bit unsigned components
- All components are [0,1]
- All components can be expanded to [-1,1] range prior to any dot product shader operation

  (2*R – 1, 2*G – 1 , 2*B – 1 , 2*A – 1)

HILO texture formats

Two 16-bit channels (high and low)

Signed HILO (GL_SIGNED_HILO_NV)

- Both components are [-1,1]
- Useful for encoding normals with high precision
- Third channel is hemispherical projection of first 2

$$\left(HI, LO, \sqrt{1 - HI^2 - LO^2}\right)$$

Unsigned HILO (GL_HILO_NV)

- Both components are [0,1]
- Useful for encoding 32-bit values, like depth
- Third channel is set to 1

$$\left(HI, LO, 1\right)$$

HILO Advantages

- **Filtering for each component done in 16-bits**
- **Hemispherical projection performed *after* filtering**
- **Always results in unit length vector**
- **External format relatively unimportant**

Single, bump mapped, quad with different normal map precision

Dot Product
GL_DOT_PRODUCT_NV

Used in intermediate stages only

Does not access any textures

Previous texture input base internal texture format must be either RGBA or HILO

```
glActiveTextureARB( GL_TEXTURE0_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
            GL_TEXTURE_2D);

glActiveTextureARB( GL_TEXTURE1_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
           GL_DOT_PRODUCT_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV,
            GL_PREVIOUS_TEXTURE_INPUT_NV, GL_TEXTURE0_ARB);
```

Dot Product Texture 2D

Previous stage must be Dot Product

Two dot products same as 3x2 matrix/vector mult:

$$\begin{bmatrix} S' \\ T' \end{bmatrix} = \mathbf{M}\vec{n} = \begin{bmatrix} S_0 & T_0 & R_0 \\ S_1 & T_1 & R_1 \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}$$

Matrix can be thought of as the "Texel Matrix", and transforms previous texture result (e.g. a normal) from R^3 to R^2, then uses transformed 2D vector to access a 2D texture

Dot Product Texture 2D

Tex #	Texture Coords (S,T,R,Q)		Shader Operations	Texture Fetch	Bound Texture Target/Format		Output Color
0	App specific	→	Texture specific	Texture specific	Any type Signed RGB[A]	→	$R_0G_0B_0$
1	(S_1, T_1, R_1)	→	$U_x = [S_1, T_1, R_1] \bullet [R_0, G_0, B_0]$	None	None	→	(0,0,0,0)
2	(S_2, T_2, R_2)	→	$U_y = [S_2, T_2, R_2] \bullet [R_0, G_0, B_0]$	(U_x, U_y)	2D RGBA	→	$R_2G_2B_2A_2$

Dot Product Texture 2D

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Signed RGB[A]	$R_0G_0B_0$
1	(S_1, T_1, R_1)	$U_x = [S_1, T_1, R_1] \bullet [R_0, G_0, B_0]$	None	None	(0,0,0,0)
2	(S_2, T_2, R_2)	$U_y = [S_2, T_2, R_2] \bullet [R_0, G_0, B_0]$	(U_x, U_y)	2D RGBA	$R_2G_2B_2A_2$

Dot Product Texture 2D

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Signed RGB[A]	$R_0G_0B_0$
1	(S_1, T_1, R_1)	$U_x = [S_1,T_1,R_1] \bullet [R_0,G_0,B_0]$	None	None	(0,0,0,0)
2	(S_2, T_2, R_2)	$U_y = [S_2, T_2, R_2] \bullet [R_0,G_0,B_0]$	(U_x, U_y)	2D RGBA	$R_2G_2B_2A_2$

Dot Product Texture 2D

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Signed RGB[A]	$R_0G_0B_0$
1	(S_1, T_1, R_1)	$U_x = [S_1, T_1, R_1] \bullet [R_0, G_0, B_0]$	None	None	(0,0,0,0)
2	(S_2, T_2, R_2)	$U_y = [S_2, T_2, R_2] \bullet [R_0, G_0, B_0]$	(U_x, U_y)	2D RGBA	$R_2G_2B_2A_2$

Dot Product Texture 2D
GL_DOT_PRODUCT_TEXTURE_2D_NV

```
glActiveTextureARB( GL_TEXTURE0_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
            GL_TEXTURE_2D);

glActiveTextureARB( GL_TEXTURE1_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
            GL_DOT_PRODUCT_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
            GL_TEXTURE0_ARB);

glActiveTextureARB( GL_TEXTURE2_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
            GL_DOT_PRODUCT_TEXTURE_2D_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
            GL_TEXTURE0_ARB);
```

Dot Product Texture 2D
GL_DOT_PRODUCT_TEXTURE_2D_NV

Using nvparse:

```
nvparse( "!!TS1.0
        texture_2d();
        dot_product_2d_1of2( tex0 );
        dot_product_2d_2of2( tex0 );" );
```

Dot Product Texture 2D Application

- High-quality bump-mapping
  - 2D HILO texture stores normals
    - Per-fragment tangent-space normal, N′
  - Vertex programs supplies tangent-space light (L) and half-angle (H) vectors in (s,t,r) texture coordinates
  - Two dot products compute
    - Diffuse L dot N′
    - Specular H dot N′
  - Illumination stored in 2D texture accessed by L dot N′ and H dot N′
    - Excellent specular appearance

HILO Normal Map Dot Product Texture 2D Bump Mapping

Bump mapping the Holy Grail

Dot Product Texture Rectangle

Previous stage must be Dot Product

Similar to Dot Product Texture 2D, except that subsequent texture target is a texture rectangle, instead of a 2D texture

Dot Product Texture Rectangle

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific →	Texture specific	Texture specific	Any type Signed RGB[A]	→ $R_0G_0B_0$
1	(S_1, T_1, R_1) →	$U_x = [S_1, T_1, R_1] \bullet [R_0, G_0, B_0]$	None	None	→ (0,0,0,0)
2	(S_2, T_2, R_2) →	$U_y = [S_2, T_2, R_2] \bullet [R_0, G_0, B_0]$	(U_x, U_y)	Texture Rectangle RGBA	→ $R_2G_2B_2A_2$

Dot Product Texture Rectangle

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Signed RGB[A]	$R_0G_0B_0$
1	(S_1, T_1, R_1)	$U_x = [S_1, T_1, R_1] \bullet [R_0, G_0, B_0]$	None	None	(0,0,0,0)
2	(S_2, T_2, R_2)	$U_y = [S_2, T_2, R_2] \bullet [R_0, G_0, B_0]$	(U_x, U_y)	Texture Rectangle RGBA	$R_2G_2B_2A_2$

Dot Product Texture Rectangle

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Signed RGB[A]	$R_0G_0B_0$
1	(S_1, T_1, R_1)	$U_x = [S_1,T_1,R_1] \bullet [R_0,G_0,B_0]$	None	None	(0,0,0,0)
2	(S_2, T_2, R_2)	$U_y = [S_2, T_2, R_2] \bullet [R_0,G_0,B_0]$	(U_x, U_y)	Texture Rectangle RGBA	$R_2G_2B_2A_2$

Dot Product Texture Rectangle

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Signed RGB[A]	$R_0G_0B_0$
1	(S_1, T_1, R_1)	$U_x = [S_1, T_1, R_1] \bullet [R_0, G_0, B_0]$	None	None	(0,0,0,0)
2	(S_2, T_2, R_2)	$U_y = [S_2, T_2, R_2] \bullet [R_0, G_0, B_0]$	(U_x, U_y)	Texture Rectangle RGBA	$R_2G_2B_2A_2$

Dot Product Texture Rectangle

GL_DOT_PRODUCT_TEXTURE_RECTANGLE_NV

```
glActiveTextureARB( GL_TEXTURE0_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV, GL_TEXTURE_2D);

glActiveTextureARB( GL_TEXTURE1_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_DOT_PRODUCT_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
          GL_TEXTURE0_ARB);

glActiveTextureARB( GL_TEXTURE2_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_DOT_PRODUCT_TEXTURE_RECTANGLE_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
          GL_TEXTURE0_ARB);
```

Dot Product Texture Rectangle
GL_DOT_PRODUCT_TEXTURE_RECTANGLE_NV

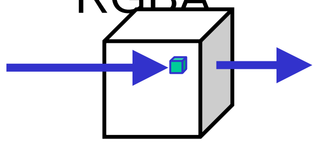
Using nvparse:

```
nvparse( "!!TS1.0
        texture_2d();
        dot_product_rectangle_1of2( tex0 );
        dot_product_rectangle_2of2( tex0 );" );
```

Dot Product Texture 3D

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific →	Texture specific	Texture specific	Any type Signed RGB[A] →	$R_0G_0B_0$
1	(S_1, T_1, R_1) →	$S' = [S_1,T_1,R_1] \bullet [R_0,G_0,B_0]$	None	None →	(0,0,0,0)
2	(S_2, T_2, R_2) →	$T' = [S_2,T_2,R_2] \bullet [R_0,G_0,B_0]$	None	None →	(0,0,0,0)
3	(S_3, T_3, R_3) →	$R' = [S_3,T_3,R_3] \bullet [R_0,G_0,B_0]$	(S',T',R')	3D RGBA →	$R_3G_3B_3A_3$

Dot Product Texture 3D

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Signed RGB[A]	$R_0G_0B_0$
1	(S_1, T_1, R_1)	$S' = [S_1,T_1,R_1] \bullet [R_0,G_0,B_0]$	None	None	(0,0,0,0)
2	(S_2, T_2, R_2)	$T' = [S_2,T_2,R_2] \bullet [R_0,G_0,B_0]$	None	None	(0,0,0,0)
3	(S_3, T_3, R_3)	$R' = [S_3,T_3,R_3] \bullet [R_0,G_0,B_0]$	(S',T',R')	3D RGBA 	$R_3G_3B_3A_3$

Dot Product Texture 3D

```
glActiveTextureARB( GL_TEXTURE0_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV, GL_TEXTURE_2D);

glActiveTextureARB( GL_TEXTURE1_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_DOT_PRODUCT_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
          GL_TEXTURE0_ARB);

glActiveTextureARB( GL_TEXTURE2_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_DOT_PRODUCT_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
          GL_TEXTURE0_ARB);

glActiveTextureARB( GL_TEXTURE3_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_DOT_PRODUCT_TEXTURE_3D_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
          GL_TEXTURE0_ARB);
```

Dot Product Texture 3D

Using nvparse:

```
nvparse( "!!TS1.0
        texture_2d();
        dot_product_3d_1of3( tex0 );
        dot_product_3d_2of3( tex0 );
        dot_product_3d_3of3( tex0 );" );
```

Dot Product Texture Cube Map

Previous two stages must be Dot Product

Three dot products same as 3x3 matrix/vector mult:

$$\vec{n}' = \mathbf{M}\vec{n} = \begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}$$

Matrix can be thought of as the "Texel Matrix", and transforms previous texture result (e.g. a normal) from one space to another, then uses transformed vector to access a cube map

Matrix shown above moves normal map vector from surface-local space to object space

Dot Product Texture Cube Map

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific →	Texture specific	Texture specific	Any type Signed RGB[A] →	$R_0G_0B_0$
1	(T_x, B_x, N_x) →	$U_x = [T_x, B_x, N_x] \bullet [R_0, G_0, B_0]$	None	None →	(0,0,0,0)
2	(T_y, B_y, N_y) →	$U_y = [T_y, B_y, N_y] \bullet [R_0, G_0, B_0]$	None	None →	(0,0,0,0)
3	(T_z, B_z, N_z) →	$U_z = [T_z, B_z, N_z] \bullet [R_0, G_0, B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$		Cube map RGBA →	$R_3G_3B_3A_3$

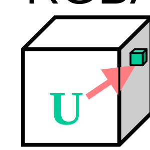
Dot Product Texture Cube Map

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Signed RGB[A]	$R_0G_0B_0$
1	(T_x, B_x, N_x)	$U_x = [T_x, B_x, N_x] \bullet [R_0, G_0, B_0]$	None	None	(0,0,0,0)
2	(T_y, B_y, N_y)	$U_y = [T_y, B_y, N_y] \bullet [R_0, G_0, B_0]$	None	None	(0,0,0,0)
3	(T_z, B_z, N_z)	$U_z = [T_z, B_z, N_z] \bullet [R_0, G_0, B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$		Cube map RGBA	$R_3G_3B_3A_3$

Dot Product Texture Cube Map

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific →	Texture specific	Texture specific	Any type Signed RGB[A] →	$R_0G_0B_0$
1	(T_x, B_x, N_x)	$U_x = [T_x,B_x,N_x] \bullet [R_0,G_0,B_0]$	None	None →	(0,0,0,0)
2	(T_y, B_y, N_y)	$U_y = [T_y,B_y,N_y] \bullet [R_0,G_0,B_0]$	None	None →	(0,0,0,0)
3	(T_z, B_z, N_z)	$U_z = [T_z,B_z,N_z] \bullet [R_0,G_0,B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$		Cube map RGBA →	$R_3G_3B_3A_3$

Dot Product Texture Cube Map

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific →	Texture specific	Texture specific	Any type Signed RGB[A] →	$R_0G_0B_0$
1	(T_x, B_x, N_x)	$U_x = [T_x,B_x,N_x] \bullet [R_0,G_0,B_0]$	None	None →	(0,0,0,0)
2	(T_y, B_y, N_y)	$U_y = [T_y,B_y,N_y] \bullet [R_0,G_0,B_0]$	None	None →	(0,0,0,0)
3	(T_z, B_z, N_z)	$U_z = [T_z,B_z,N_z] \bullet [R_0,G_0,B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ →		Cube map RGBA  →	$R_3G_3B_3A_3$

Dot Product Texture Cube Map
GL_DOT_PRODUCT_TEXTURE_CUBE_MAP_NV

```
glActiveTextureARB( GL_TEXTURE0_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV,GL_SHADER_OPERATION_NV,GL_TEXTURE_2D);

glActiveTextureARB( GL_TEXTURE1_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV,GL_SHADER_OPERATION_NV,
          GL_DOT_PRODUCT_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
          GL_TEXTURE0_ARB);

glActiveTextureARB( GL_TEXTURE2_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV, GL_DOT_PRODUCT_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
          GL_TEXTURE0_ARB);

glActiveTextureARB( GL_TEXTURE3_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_DOT_PRODUCT_TEXTURE_CUBE_MAP_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
          GL_TEXTURE0_ARB);
```

Dot Product Texture Cube Map
GL_DOT_PRODUCT_TEXTURE_CUBE_MAP_NV

Using nvparse:

```
nvparse( "!!TS1.0
        texture_2d();
        dot_product_cube_map_1of3( tex0 );
        dot_product_cube_map_2of3( tex0 );
        dot_product_cube_map_3of3( tex0 );" );
```

Dot Product Constant Eye Reflect Cube Map

Similar to Dot Product Texture Cube Map, except that the vector accessing the cube map (R) is computed as the reflection of the eye vector about the transformed normal

The eye vector is passed in as constants (i.e. an infinite viewer)

$$\vec{n}' = \mathbf{M}\vec{n} = \begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \qquad \mathbf{E} = (E_x, E_y, E_z)$$

$$\mathbf{R} = \frac{2\boldsymbol{n}'\,(\boldsymbol{n}' \bullet \mathbf{E})}{(\boldsymbol{n}' \bullet \boldsymbol{n}')} - \mathbf{E}$$

Dot Product Constant Eye Reflect Cube Map

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific →	Texture specific	Texture specific	Any type Signed RGB[A] →	$R_0G_0B_0$
1	(T_x, B_x, N_x) →	$U_x = [T_x, B_x, N_x] \bullet [R_0, G_0, B_0]$	None	None →	(0,0,0,0)
2	(T_y, B_y, N_y) →	$U_y = [T_y, B_y, N_y] \bullet [R_0, G_0, B_0]$	None	None →	(0,0,0,0)
3	(T_z, B_z, N_z) →	$U_z = [T_z, B_z, N_z] \bullet [R_0, G_0, B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ $\mathbf{E} = (E_x, E_y, E_z)$ $\mathbf{R} = \frac{\mathbf{2U(U \bullet E)}}{\mathbf{(U \bullet U)}} - \mathbf{E}$	$\mathbf{R} = (R_x, R_y, R_z)$	Cube map RGBA →	$R_3G_3B_3A_3$

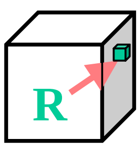
Dot Product Constant Eye Reflect Cube Map

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Signed RGB[A]	$R_0G_0B_0$
1	(T_x, B_x, N_x)	$U_x = [T_x,B_x,N_x] \bullet [R_0,G_0,B_0]$	None	None	(0,0,0,0)
2	(T_y, B_y, N_y)	$U_y = [T_y,B_y,N_y] \bullet [R_0,G_0,B_0]$	None	None	(0,0,0,0)
3	(T_z, B_z, N_z)	$U_z = [T_z,B_z,N_z] \bullet [R_0,G_0,B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ $\mathbf{E} = (E_x, E_y, E_z)$ $\mathbf{R} = \frac{2\mathbf{U}(\mathbf{U}\bullet\mathbf{E})}{(\mathbf{U}\bullet\mathbf{U})} - \mathbf{E}$	$\mathbf{R} = (R_x, R_y, R_z)$	Cube map RGBA	$R_3G_3B_3A_3$

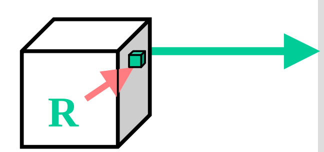
Dot Product Constant Eye Reflect Cube Map

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Signed RGB[A]	$R_0G_0B_0$
1	(T_x, B_x, N_x)	$U_x = [T_x, B_x, N_x] \bullet [R_0, G_0, B_0]$	None	None	(0,0,0,0)
2	(T_y, B_y, N_y)	$U_y = [T_y, B_y, N_y] \bullet [R_0, G_0, B_0]$	None	None	(0,0,0,0)
3	(T_z, B_z, N_z)	$U_z = [T_z, B_z, N_z] \bullet [R_0, G_0, B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ $\mathbf{E} = (E_x, E_y, E_z)$ $\mathbf{R} = \frac{2\mathbf{U}(\mathbf{U}\bullet\mathbf{E})}{(\mathbf{U}\bullet\mathbf{U})} - \mathbf{E}$	$\mathbf{R} = (R_x, R_y, R_z)$	Cube map RGBA	$R_3G_3B_3A_3$

Dot Product Constant Eye Reflect Cube Map

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	→ Texture specific	Texture specific	Any type → Signed RGB[A]	$R_0G_0B_0$
1	(T_x, B_x, N_x)	$U_x = [T_x, B_x, N_x] \bullet [R_0, G_0, B_0]$	None	None →	(0,0,0,0)
2	(T_y, B_y, N_y)	$U_y = [T_y, B_y, N_y] \bullet [R_0, G_0, B_0]$	None	None →	(0,0,0,0)
3	(T_z, B_z, N_z)	$U_z = [T_z, B_z, N_z] \bullet [R_0, G_0, B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ $\mathbf{E} = (E_x, E_y, E_z)$ $\mathbf{R} = \frac{2\mathbf{U}(\mathbf{U}\bullet\mathbf{E})}{(\mathbf{U}\bullet\mathbf{U})} - \mathbf{E}$	$\mathbf{R} = (R_x, R_y, R_z)$	Cube map RGBA  →	$R_3G_3B_3A_3$

Dot Product Constant Eye Reflect Cube Map

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific →	Texture specific	Texture specific	Any type Signed RGB[A] →	$R_0G_0B_0$
1	(T_x, B_x, N_x)	$U_x = [T_x,B_x,N_x] \bullet [R_0,G_0,B_0]$	None	None →	(0,0,0,0)
2	(T_y, B_y, N_y)	$U_y = [T_y,B_y,N_y] \bullet [R_0,G_0,B_0]$	None	None →	(0,0,0,0)
3	(T_z, B_z, N_z)	$U_z = [T_z,B_z,N_z] \bullet [R_0,G_0,B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ $\mathbf{E} = (E_x, E_y, E_z)$ $\mathbf{R} = \frac{2\mathbf{U}(\mathbf{U}\bullet\mathbf{E})}{(\mathbf{U}\bullet\mathbf{U})} - \mathbf{E}$	$\mathbf{R} = (R_x, R_y, R_z)$	Cube map RGBA  	$R_3G_3B_3A_3$

Dot Product Constant Eye Reflect Cube Map

GL_DOT_PRODUCT_CONST_EYE_REFLECT_CUBE_MAP_NV

```
glActiveTextureARB( GL_TEXTURE0_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV,GL_SHADER_OPERATION_NV,GL_TEXTURE_2D);

glActiveTextureARB( GL_TEXTURE1_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV,GL_SHADER_OPERATION_NV,
    GL_DOT_PRODUCT_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV, GL_TEXTURE0_ARB);

glActiveTextureARB( GL_TEXTURE2_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV, GL_DOT_PRODUCT_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV, GL_TEXTURE0_ARB);

glActiveTextureARB( GL_TEXTURE3_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
    GL_DOT_PRODUCT_CONST_EYE_REFLECT_CUBE_MAP_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV, GL_TEXTURE0_ARB);
```

Dot Product Constant Eye Reflect Cube Map

GL_DOT_PRODUCT_CONST_EYE_REFLECT_CUBE_MAP_NV

Using nvparse:

```
nvparse( "!!TS1.0
        texture_2d();
        dot_product_reflect_cube_map_const_eye_1of3( tex0, 0, 0, 1 );
        dot_product_reflect_cube_map_const_eye_2of3( tex0 );
        dot_product_reflect_cube_map_const_eye_3of3( tex0 );" );
```

Dot Product Reflect Cube Map

- Same as Dot Product Constant Eye Reflect Cube Map, except that the eye vector is passed in through the Q coordinate of the three dot product stages
- Eye in this case is "local", resulting in better, more realistic, images as it is interpolated across all polygons

$$\vec{n}' = \mathbf{M}\vec{n} = \begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \qquad \mathbf{E} = (q_0, q_1, q_2)$$

$$\mathbf{R} = \frac{2\boldsymbol{n}(\boldsymbol{n}' \bullet \mathbf{E})}{(\boldsymbol{n}' \bullet \boldsymbol{n}')} - \mathbf{E}$$

Dot Product Reflect Cube Map

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific →	Texture specific	Texture specific	Any type → Signed RGB[A]	$R_0G_0B_0$
1	$([T_x, B_x, N_x], E_x)$ →	$U_x = [T_x, B_x, N_x] \bullet [R_0, G_0, B_0]$	None	None →	(0,0,0,0)
2	$([T_y, B_y, N_y], E_y)$ →	$U_y = [T_y, B_y, N_y] \bullet [R_0, G_0, B_0]$	None	None →	(0,0,0,0)
3	$([T_z, B_z, N_z], E_z)$ →	$U_z = [T_z, B_z, N_z] \bullet [R_0, G_0, B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ $\mathbf{E} = (E_x, E_y, E_z)$ $\mathbf{R} = \dfrac{2\mathbf{U}(\mathbf{U}\bullet\mathbf{E})}{(\mathbf{U}\bullet\mathbf{U})} - \mathbf{E}$	$\mathbf{R} = (R_x, R_y, R_z)$	Cube map RGBA →	$R_3G_3B_3A_3$

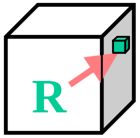
Dot Product Reflect Cube Map

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Signed RGB[A]	$R_0G_0B_0$
1	$([T_x, B_x, N_x], E_x)$	$U_x = [T_x, B_x, N_x] \bullet [R_0, G_0, B_0]$	None	None	(0,0,0,0)
2	$([T_y, B_y, N_y], E_y)$	$U_y = [T_y, B_y, N_y] \bullet [R_0, G_0, B_0]$	None	None	(0,0,0,0)
3	$([T_z, B_z, N_z], E_z)$	$U_z = [T_z, B_z, N_z] \bullet [R_0, G_0, B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ $\mathbf{E} = (E_x, E_y, E_z)$ $\mathbf{R} = \frac{2\mathbf{U}(\mathbf{U}\bullet\mathbf{E})}{(\mathbf{U}\bullet\mathbf{U})} - \mathbf{E}$	$\mathbf{R} = (R_x, R_y, R_z)$	Cube map RGBA	$R_3G_3B_3A_3$

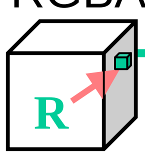
Dot Product Reflect Cube Map

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Signed RGB[A]	$R_0G_0B_0$
1	$([T_x, B_x, N_x], E_x)$	$U_x = [T_x,B_x,N_x] \bullet [R_0,G_0,B_0]$	None	None	(0,0,0,0)
2	$([T_y, B_y, N_y], E_y)$	$U_y = [T_y,B_y,N_y] \bullet [R_0,G_0,B_0]$	None	None	(0,0,0,0)
3	$([T_z, B_z, N_z], E_z)$	$U_z = [T_z,B_z,N_z] \bullet [R_0,G_0,B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ $\mathbf{E} = (E_x, E_y, E_z)$ $\mathbf{R} = \frac{2\mathbf{U}(\mathbf{U}\bullet\mathbf{E})}{(\mathbf{U}\bullet\mathbf{U})} - \mathbf{E}$	$\mathbf{R} = (R_x, R_y, R_z)$	Cube map RGBA	$R_3G_3B_3A_3$

Dot Product Reflect Cube Map

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific →	Texture specific	Texture specific	Any type Signed RGB[A] →	$R_0G_0B_0$
1	$([T_x, B_x, N_x], E_x)$	$U_x = [T_x, B_x, N_x] \bullet [R_0, G_0, B_0]$	None	None →	(0,0,0,0)
2	$([T_y, B_y, N_y], E_y)$	$U_y = [T_y, B_y, N_y] \bullet [R_0, G_0, B_0]$	None	None →	(0,0,0,0)
3	$([T_z, B_z, N_z], E_z)$	$U_z = [T_z, B_z, N_z] \bullet [R_0, G_0, B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ $\mathbf{E} = (E_x, E_y, E_z)$ $\mathbf{R} = \frac{2\mathbf{U}(\mathbf{U}\bullet\mathbf{E})}{(\mathbf{U}\bullet\mathbf{U})} - \mathbf{E}$ →	$\mathbf{R} = (R_x, R_y, R_z)$	Cube map RGBA  →	$R_3G_3B_3A_3$

Dot Product Reflect Cube Map

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Signed RGB[A]	$R_0G_0B_0$
1	$([T_x, B_x, N_x], E_x)$	$U_x = [T_x, B_x, N_x] \bullet [R_0, G_0, B_0]$	None	None	(0,0,0,0)
2	$([T_y, B_y, N_y], E_y)$	$U_y = [T_y, B_y, N_y] \bullet [R_0, G_0, B_0]$	None	None	(0,0,0,0)
3	$([T_z, B_z, N_z], E_z)$	$U_z = [T_z, B_z, N_z] \bullet [R_0, G_0, B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ $\mathbf{E} = (E_x, E_y, E_z)$ $\mathbf{R} = \frac{2\mathbf{U}(\mathbf{U} \bullet \mathbf{E})}{(\mathbf{U} \bullet \mathbf{U})} - \mathbf{E}$	$\mathbf{R} = (R_x, R_y, R_z)$	Cubemap RGBA 	$R_3G_3B_3A_3$

Dot Product Reflect Cube Map

GL_DOT_PRODUCT_REFLECT_CUBE_MAP_NV

```
glActiveTextureARB( GL_TEXTURE0_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV,GL_SHADER_OPERATION_NV,GL_TEXTURE_2D);

glActiveTextureARB( GL_TEXTURE1_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV,GL_SHADER_OPERATION_NV,
          GL_DOT_PRODUCT_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV, GL_TEXTURE0_ARB);

glActiveTextureARB( GL_TEXTURE2_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV, GL_DOT_PRODUCT_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV, GL_TEXTURE0_ARB);

glActiveTextureARB( GL_TEXTURE3_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_DOT_PRODUCT_REFLECT_CUBE_MAP_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV, GL_TEXTURE0_ARB);
```

Dot Product Reflect Cube Map
GL_DOT_PRODUCT_REFLECT_CUBE_MAP_NV

Using nvparse:

```
nvparse( "!!TS1.0
        texture_2d();
        dot_product_reflect_cube_map_eye_from_qs_1of3( tex0 );
        dot_product_reflect_cube_map_eye_from_qs_2of3( tex0 );
        dot_product_reflect_cube_map_eye_from_qs_3of3( tex0 );" );
```

Dot Product Cube Map Reflect Example

Old NVIDIA headquarters lobby with floating bumpy shiny patch

Dot Product Diffuse Cube Map

- Dot product reflect cube map programs transform the normal en route to computing the reflection vector
- This intermediate vector may also be used to access a separate cube map – yielding Dot Product Texture Cube Map and Dot Product Reflect Cube Map operations in a single pass!
- Texture output for second-to-last stage represents the transformed normal lookup (i.e. diffuse)
- Texture output for the last stage represents the reflection vector lookup (i.e. specular)
- If the cube map targets for these stages hold identity (or normalization) cube maps, the vectors can be used for further computation with register combiners

Dot Product Diffuse Cube Map
(with Dot Product Reflect Cube Map)

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific →	Texture specific	Texture specific	Any type Signed RGB[A] →	$R_0G_0B_0$
1	$([T_x, B_x, N_x], E_x)$ →	$U_x = [T_x, B_x, N_x] \bullet [R_0, G_0, B_0]$	None	None →	(0,0,0,0)
2	$([T_y, B_y, N_y], E_y)$ →	$U_y = [T_y, B_y, N_y] \bullet [R_0, G_0, B_0]$	$\mathbf{U} = (U_x, U_y, U_z)$	Cube map RGBA →	$R_2G_2B_2A_2$
3	$([T_z, B_z, N_z], E_z)$ →	$U_z = [T_z, B_z, N_z] \bullet [R_0, G_0, B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ $\mathbf{E} = (E_x, E_y, E_z)$ $\mathbf{R} = \frac{2\mathbf{U}(\mathbf{U} \bullet \mathbf{E})}{(\mathbf{U} \bullet \mathbf{U})} - \mathbf{E}$	$\mathbf{R} = (R_x, R_y, R_z)$	Cube map RGBA →	$R_3G_3B_3A_3$

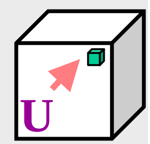
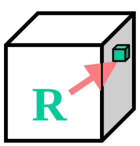
Dot Product Diffuse Cube Map
(with Dot Product Reflect Cube Map)

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Signed RGB[A]	$R_0G_0B_0$
1	$([T_x, B_x, N_x], E_x)$	$U_x = [T_x,B_x,N_x] \bullet [R_0,G_0,B_0]$	None	None	(0,0,0,0)
2	$([T_y, B_y, N_y], E_y)$	$U_y = [T_y,B_y,N_y] \bullet [R_0,G_0,B_0]$	$\mathbf{U} = (U_x, U_y, U_z)$	Cube map RGBA	$R_2G_2B_2A_2$
3	$([T_z, B_z, N_z], E_z)$	$U_z = [T_z,B_z,N_z] \bullet [R_0,G_0,B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ $\mathbf{E} = (E_x, E_y, E_z)$ $\mathbf{R} = \frac{2\mathbf{U}(\mathbf{U}\bullet\mathbf{E})}{(\mathbf{U}\bullet\mathbf{U})} - \mathbf{E}$	$\mathbf{R} = (R_x, R_y, R_z)$	Cube map RGBA	$R_3G_3B_3A_3$

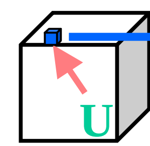
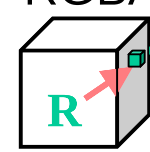
Dot Product Diffuse Cube Map
(with Dot Product Reflect Cube Map)

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	Texture specific	Texture specific	Any type Signed RGB[A]	$R_0G_0B_0$
1	$([T_x, B_x, N_x], E_x)$	$U_x = [T_x, B_x, N_x] \bullet [R_0, G_0, B_0]$	None	None	(0,0,0,0)
2	$([T_y, B_y, N_y], E_y)$	$U_y = [T_y, B_y, N_y] \bullet [R_0, G_0, B_0]$	$\mathbf{U} = (U_x, U_y, U_z)$	Cube map RGBA	$R_2G_2B_2A_2$
3	$([T_z, B_z, N_z], E_z)$	$U_z = [T_z, B_z, N_z] \bullet [R_0, G_0, B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ $\mathbf{E} = (E_x, E_y, E_z)$ $\mathbf{R} = \frac{2\mathbf{U}(\mathbf{U} \bullet \mathbf{E})}{(\mathbf{U} \bullet \mathbf{U})} - \mathbf{E}$	$\mathbf{R} = (R_x, R_y, R_z)$	Cube map RGBA	$R_3G_3B_3A_3$

Dot Product Diffuse Cube Map
(with Dot Product Reflect Cube Map)

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	→ Texture specific	Texture specific	Any type → Signed RGB[A]	$R_0G_0B_0$
1	$([T_x, B_x, N_x], E_x)$	$U_x = [T_x, B_x, N_x] \bullet [R_0, G_0, B_0]$	None	None →	(0,0,0,0)
2	$([T_y, B_y, N_y], E_y)$	$U_y = [T_y, B_y, N_y] \bullet [R_0, G_0, B_0]$	 $\mathbf{U} = (U_x, U_y, U_z)$	Cube map RGBA	→ $R_2G_2B_2A_2$
3	$([T_z, B_z, N_z], E_z)$	$U_z = [T_z, B_z, N_z] \bullet [R_0, G_0, B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ $\mathbf{E} = (E_x, E_y, E_z)$ $\mathbf{R} = \frac{2\mathbf{U}(\mathbf{U} \bullet \mathbf{E})}{(\mathbf{U} \bullet \mathbf{U})} - \mathbf{E}$	$\mathbf{R} = (R_x, R_y, R_z)$	Cube map RGBA 	→ $R_3G_3B_3A_3$

Dot Product Diffuse Cube Map (with Dot Product Reflect Cube Map)

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	App specific	→ Texture specific	Texture specific	Any type Signed RGB[A]	→ $R_0G_0B_0$
1	$([T_x, B_x, N_x], E_x)$	$U_x = [T_x,B_x,N_x] \bullet [R_0,G_0,B_0]$	None	None	→ (0,0,0,0)
2	$([T_y, B_y, N_y], E_y)$	$U_y = [T_y,B_y,N_y] \bullet [R_0,G_0,B_0]$	$\mathbf{U} = (U_x, U_y, U_z)$	Cube map RGBA 	→ $R_2G_2B_2A_2$
3	$([T_z, B_z, N_z], E_z)$	$U_z = [T_z,B_z,N_z] \bullet [R_0,G_0,B_0]$ $\mathbf{U} = (U_x, U_y, U_z)$ $\mathbf{E} = (E_x, E_y, E_z)$ $\mathbf{R} = \frac{2\mathbf{U}(\mathbf{U} \bullet \mathbf{E})}{(\mathbf{U} \bullet \mathbf{U})} - \mathbf{E}$	$\mathbf{R} = (R_x, R_y, R_z)$	Cube map RGBA 	→ $R_3G_3B_3A_3$

Dot Product Diffuse Cube Map

GL_DOT_PRODUCT_DIFFUSE_CUBE_MAP_NV

```
glActiveTextureARB( GL_TEXTURE0_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV,GL_SHADER_OPERATION_NV,GL_TEXTURE_2D);

glActiveTextureARB( GL_TEXTURE1_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV,GL_SHADER_OPERATION_NV,
   GL_DOT_PRODUCT_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
   GL_TEXTURE0_ARB);

glActiveTextureARB( GL_TEXTURE2_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
   GL_DOT_PRODUCT_DIFFUSE_CUBE_MAP_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
   GL_TEXTURE0_ARB);

glActiveTextureARB( GL_TEXTURE3_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
   GL_DOT_PRODUCT_REFLECT_CUBE_MAP_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
   GL_TEXTURE0_ARB);
```

Dot Product Diffuse Cube Map
GL_DOT_PRODUCT_DIFFUSE_CUBE_MAP_NV

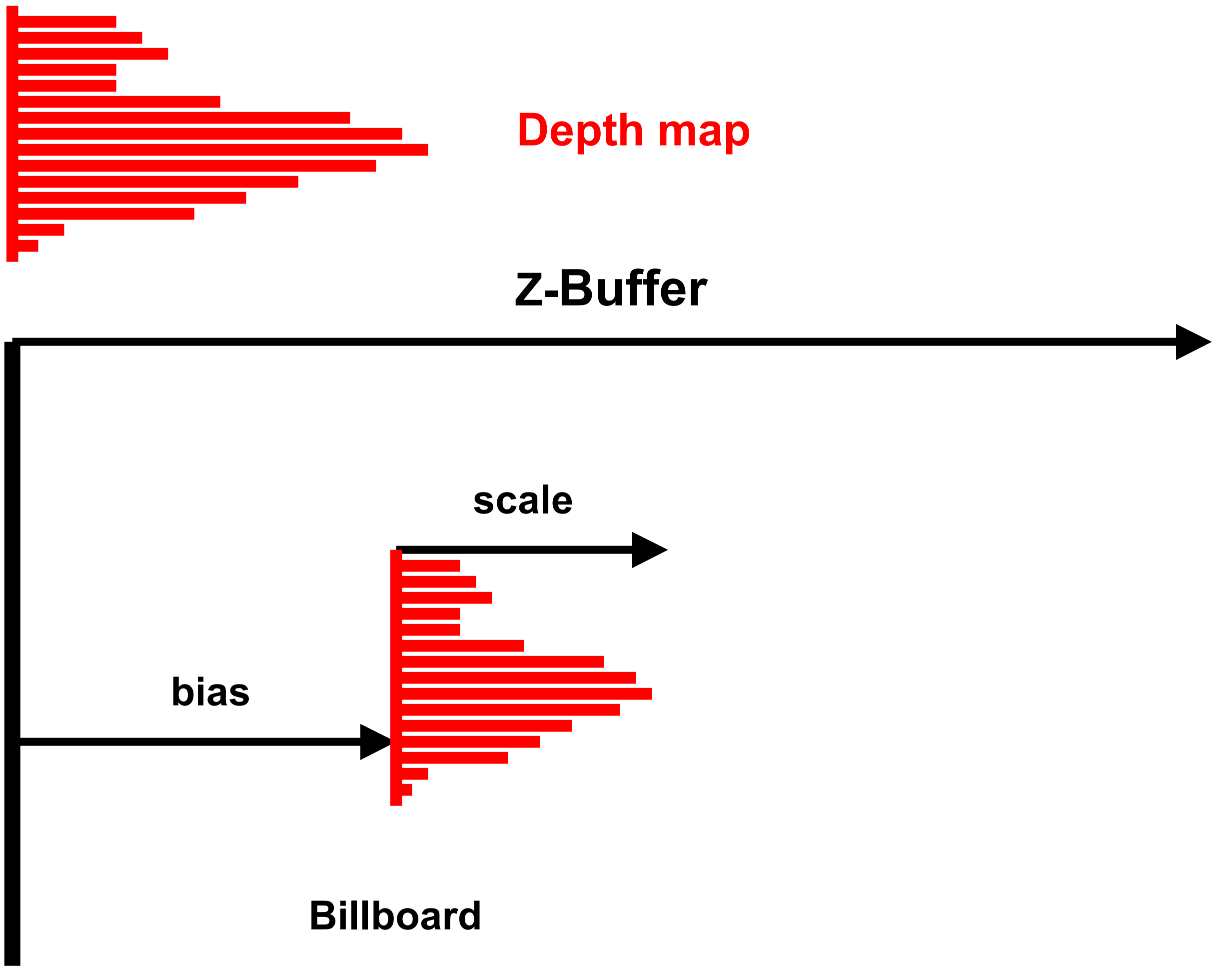
Using nvparse:

```
nvparse( "!!TS1.0
        texture_2d();
        dot_product_cube_map_and_reflect_cube_map_eye_from_qs_1of3( tex0 );
        dot_product_cube_map_and_reflect_cube_map_eye_from_qs_2of3( tex0 );
        dot_product_cube_map_and_reflect_cube_map_eye_from_qs_3of3( tex0 );" );
```

Dot Product Depth Replace

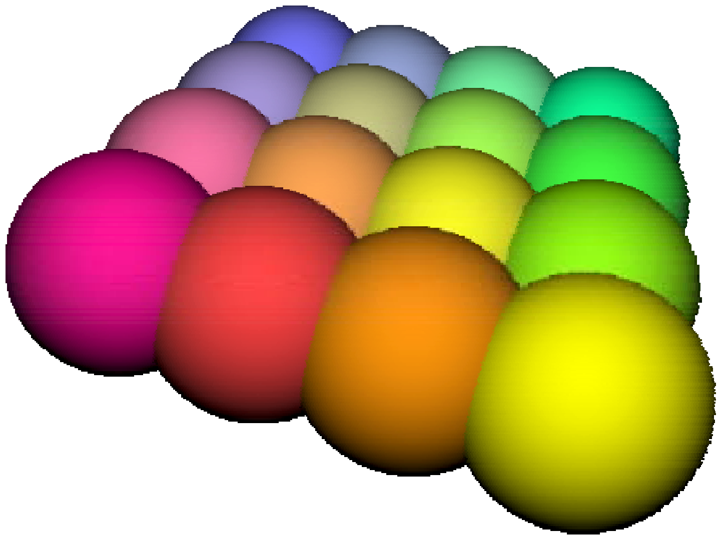
- Used for “depth sprites”, where a screen aligned image can also have correct depth
- Previous stage must be Dot Product program
- Best precision if source texture is unsigned HILO, though other formats may be used
- If the new depth value is outside of the range of the near and far depth range values, the fragment is rejected (that is, it’s clipped to near/far planes)
- Calculates two dot products, and replaces the fragment (window) depth with their quotient
- Output color is always (0,0,0,0)

Dot Product Depth Replace

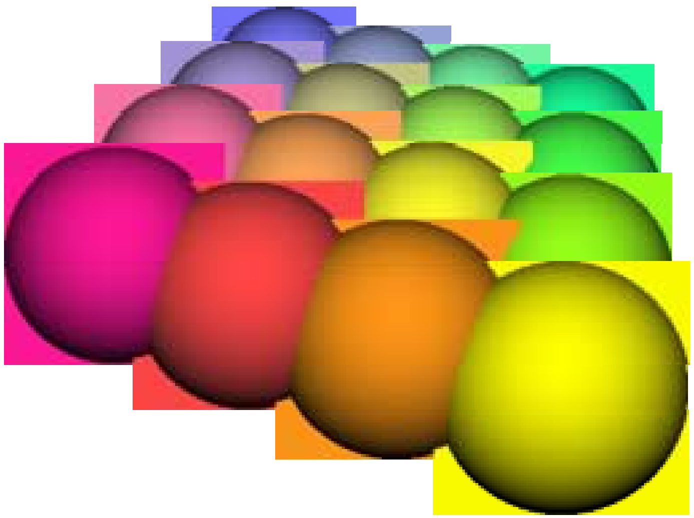


Dot Product Depth Replace Example

Per-pixel diffuse lighting of properly depth occluded spheres?



Normal mode with alpha testing



Without alpha testing to show polygon count

How many polygons required? Just 16.

Dot Product Depth Replace

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	(S_0,T_0,R_0,Q_0)	Texture 2D	$\left(\frac{S_0}{Q_0},\frac{T_0}{Q_0}\right)$	2D Unsigned HILO	(0,0,0,0)
1	$\left(Z_{scale},\frac{Z_{scale}}{2^{16}}, Z_{bias}\right)$	$Z=\left(Z_{scale},\frac{Z_{scale}}{2^{16}}, Z_{bias}\right)\bullet[H,L,1]$	None	None	(0,0,0,0)
2	$\left(W_{scale},\frac{W_{scale}}{2^{16}}, W_{bias}\right)$	$W=\left(W_{scale},\frac{W_{scale}}{2^{16}}, W_{bias}\right)\bullet[H,L,1]$	None	None	(0,0,0,0)

$$Z_{window}=\frac{Z}{W}$$

Replaces current fragment's depth

Dot Product Depth Replace

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
				2D Unsigned HILO	
0	(S_0,T_0,R_0,Q_0)	Texture 2D	$(\frac{S_0}{Q_0},\frac{T_0}{Q_0})$		(0,0,0,0)
1	$(Z_{scale},\frac{Z_{scale}}{2^{16}},Z_{bias})$	$Z=(Z_{scale},\frac{Z_{scale}}{2^{16}},Z_{bias})\bullet[H,L,1]$	None	None	(0,0,0,0)
2	$(W_{scale},\frac{W_{scale}}{2^{16}},W_{bias})$	$W=(W_{scale},\frac{W_{scale}}{2^{16}},W_{bias})\bullet[H,L,1]$	None	None	(0,0,0,0)

$$Z_{window}=\frac{Z}{W}$$

Replaces current fragment's depth

Dot Product Depth Replace

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	(S_0,T_0,R_0,Q_0)	Texture 2D	$\left(\frac{S_0}{Q_0},\frac{T_0}{Q_0}\right)$	2D Unsigned HILO	(0,0,0,0)
1	$(Z_{scale},\frac{Z_{scale}}{2^{16}}, Z_{bias})$	$Z = (Z_{scale},\frac{Z_{scale}}{2^{16}}, Z_{bias}) \bullet [H,L,1]$	None	None	(0,0,0,0)
2	$(W_{scale},\frac{W_{scale}}{2^{16}}, W_{bias})$	$W = (W_{scale},\frac{W_{scale}}{2^{16}}, W_{bias}) \bullet [H,L,1]$	None	None	(0,0,0,0)

$$Z_{window} = \frac{Z}{W}$$

Replaces current fragment's depth

Dot Product Depth Replace

Tex #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	Output Color
0	(S_0,T_0,R_0,Q_0)	Texture 2D	$(\frac{S_0}{Q_0}, \frac{T_0}{Q_0})$	2D Unsigned HILO	(0,0,0,0)
1	$(Z_{scale}, \frac{Z_{scale}}{2^{16}}, Z_{bias})$	$Z = (Z_{scale}, \frac{Z_{scale}}{2^{16}}, Z_{bias}) \bullet [H,L,1]$	None	None	(0,0,0,0)
2	$(W_{scale}, \frac{W_{scale}}{2^{16}}, W_{bias})$	$W = (W_{scale}, \frac{W_{scale}}{2^{16}}, W_{bias}) \bullet [H,L,1]$	None	None	(0,0,0,0)

$$Z_{window} = \frac{Z}{W}$$

Replaces current fragment's depth

Dot Product Depth Replace
GL_DOT_PRODUCT_DEPTH_REPLACE_NV

```
glActiveTextureARB( GL_TEXTURE0_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_TEXTURE_2D);

glActiveTextureARB( GL_TEXTURE1_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_DOT_PRODUCT_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
          GL_TEXTURE0_ARB);

glActiveTextureARB( GL_TEXTURE2_ARB );
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
          GL_DOT_PRODUCT_DEPTH_REPLACE_NV);
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_PREVIOUS_TEXTURE_INPUT_NV,
          GL_TEXTURE0_ARB);
```

Dot Product Depth Replace
GL_DOT_PRODUCT_DEPTH_REPLACE_NV

Using nvparse:

```
nvparse( "!!TS1.0
        texture_2d();                       /* Probably unsigned HILO format */
        dot_product_depth_replace_1of2( tex0 );
        dot_product_depth_replace_2of2( tex0 );" );
```

Texture Shader Fragment Coloring

- Fragment coloring is the post-texture shader process of computing a final fragment color based on texture results and other interpolated values
- Texture shaders can work with either
  - Conventional OpenGL texture environment functionality,
  - Or NVIDIA's register combiners functionality
- Both work though register combiners is more powerful in its ability to use signed texture results

New Conventional Texture Environment Semantics

- GL_NONE texture environment function allows the texture environment to ignore a texture stage not generating a useful color
  - Example usage: to ignore the RGBA result used by a GL_DEPENDENT_AR_TEXTURE_2D fetch
  - Texture shader operations that do not generate a meaningful RGBA color (dot product, cull fragment, etc.) automatically default to GL_NONE
- New signed texture environment behavior
  - GL_ADD clamps to [-1,1] range, etc.
  - EXT_texture_env_combine & related extensions clamp inputs & results to [0,1] always

Register Combiners with Texture Shaders

- Result of a texture shader stage initializes correspondingly numbered register combiner texture register
  - Signed color results show up signed
- Texture shader operations that do not generate meaningful RGBA results initialize their corresponding register combiner texture register to (0,0,0,0)

Texture Shader Precision

- Interpolated texture coordinates are IEEE 32-bit floating-point values
- Texture projections, dot products, texture offset, post-texture offset scaling, reflection vector, and depth replace division computations are performed in IEEE 32-bit floating-point
- HILO texture components are filtered as 16-bit values
- DSDT, MAG, intensity, and color components are filtered as 8-bit values

Texture Shader Consistency

- Texture shader operations sometimes depend on other texture shader operations and texture format and texture mipmap consistency
- Not all texture shader configurations are "consistent"
  - Inconsistent operations operate like GL_NONE
  - Exact consistency rules are spelled out in the NV_texture_shader OpenGL extension specification
- If texture shader programs are not working for you, *check texture shader consistency*

Checking Texture Shader Consistency

- If texture shader functionality is not working the way you think it should, then
  - Check for OpenGL errors with glGetError
    - Always a good idea
  - Also, query texture consistency of all 4 stages
- Query texture shader consistency

```
for (i=0; i<4; i++) {
  GLint isConsistent;

  glTexEnviv(GL_TEXTURE_SHADER_NV,
    GL_SHADER_CONSISTENT_NV, &isConsistent);
  printf("Texture shader stage %d is %s.\n",
    isConsistent ? "consistent" : "NOT consistent");
}
```

Questions?

Sebastien Domine, sdomine@nvidia.com

John Spitzer, jspitzer@nvidia.com

www.nvidia.com/developer

Thanks to Mark Kilgard and Chris Wynn for adding to and improving this presentation.