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

- Triangle Rasterizer
- Texture Unit
- Texture Unit
- 2 Combiner Stages
- Specular / fog Combiner
- ROP & Frame buffer
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)
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”

Interpolated texture coordinate sets

32-bit IEEE floating-point
Per-component

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 preset precedence
Enabling Texture Shader Mode

- **New enable**
  
  ```c
  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:**
  
  ```c
  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**

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S, T, R, Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>(S_i, T_i, R_i, Q_i)</td>
<td></td>
<td>(\frac{S_i}{Q_i})</td>
<td>1D</td>
<td>(R, G, B, A)</td>
</tr>
</tbody>
</table>

```c
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV, GL_TEXTURE_1D);
nvparse( "!!TS1.0
texture_1d();" );
```
## Texture 2D

<table>
<thead>
<tr>
<th>( \text{Tex #} )</th>
<th>( (S_i, T_i, R_i, Q_i) )</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i )</td>
<td>( (S_i, T_i, R_i, Q_i) )</td>
<td>Texture 2D</td>
<td>( \frac{S_i}{Q_i}, \frac{T_i}{Q_i} )</td>
<td>2D Any Format</td>
<td>(R,G,B,A)</td>
</tr>
</tbody>
</table>

\[
glTexEnvi(GL\_TEXTURE\_SHADER\_NV, GL\_SHADER\_OPERATION\_NV, GL\_TEXTURE\_2D);
\]

\[
nvparse( "!TS1.0
texture_2d();" );
\]
Texture 3D

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color (R,G,B,A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>(S_i,T_i,R_i,Q_i)</td>
<td>Texture 3D</td>
<td>(S_i/Q_i, T_i/Q_i, R_i/Q_i)</td>
<td>3D Any Format</td>
<td></td>
</tr>
</tbody>
</table>

```gl
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

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>(S_i, T_i, R_i, Q_i)</td>
<td>Texture Rectangle</td>
<td>(S_i/Q_i, T_i/Q_i)</td>
<td>Texture Rectangle</td>
<td>(R, G, B, A)</td>
</tr>
</tbody>
</table>

```c
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV, GL_TEXTURE_RECTANGLE_NV);
	nvparse( "!!TS1.0
texture_rectangle();" );
```
Texture Cube Map

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>(S_i, T_i, R_i)</td>
<td></td>
<td></td>
<td>Cube Map</td>
<td>(R,G,B,A)</td>
</tr>
</tbody>
</table>

$$\text{glTexImage}(\text{GL\_TEXTURE\_SHADER\_NV, GL\_SHADER\_OPERATION\_NV, GL\_TEXTURE\_CUBE\_MAP\_ARB});$$

$$\text{nvparse( "!!TS1.0}
\quad\text{texture_cube_map();" );}$$
Special Mode Texture Shaders

Special Modes

- None
- Pass through
- Cull fragment
**None**

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Ignored</td>
<td>R = 0</td>
<td>None</td>
<td>None</td>
<td>(R,G,B,A)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>G = 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B = 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A = 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```c
glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV, GL_NONE);
nvparse("!!TS1.0	nop();");
```
## Pass Through

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
</table>
| i     | (S<sub>i</sub>, T<sub>i</sub>, R<sub>i</sub>, Q<sub>i</sub>) | R = Clamp0to1(S<sub>i</sub>)  
G = Clamp0to1(T<sub>i</sub>)  
B = Clamp0to1(R<sub>i</sub>)  
A = Clamp0to1(Q<sub>i</sub>) | None | None | (R,G,B,A) |
Pass Through

\[
\begin{align*}
\text{Tex} & \quad \text{Texture Coords} \quad (S, T, R, Q) \\
\text{#} & \quad (S_i, T_i, R_i, Q_i) \\
\text{Shader} & \quad \text{Operations} \\
\text{R} & = \text{Clamp0to1}(S_i) \\
\text{G} & = \text{Clamp0to1}(T_i) \\
\text{B} & = \text{Clamp0to1}(R_i) \\
\text{A} & = \text{Clamp0to1}(Q_i) \\
\text{Texture} & \quad \text{Fetch} \quad \text{None} \\
\text{Bound Texture} & \quad \text{Target/Format} \quad \text{None} \\
\text{Output} & \quad \text{Color} \quad (R, G, B, A)
\end{align*}
\]
Pass Through

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S, T, R, Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>(S_i, T_i, R_i, Q_i)</td>
<td>R = Clamp0to1(S_i)</td>
<td>None</td>
<td>None</td>
<td>(R, G, B, A)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>G = Clamp0to1(T_i)</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B = Clamp0to1(R_i)</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A = Clamp0to1(Q_i)</td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

`glTexEnvi(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV, GL_PASS_THROUGH_NV);`

`nvparsе( "!!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:

```c
glActiveTextureARB( GL_TEXTURE0_ARB );
glTexEnviv(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV,
            GL_CULL_FRAGMENT_NV);
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

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>(A₀,R₀)</td>
<td>Any type Unsigned RGBA</td>
<td>R₀G₀B₀A₀</td>
</tr>
<tr>
<td>1</td>
<td>Ignored</td>
<td>None</td>
<td>(A₁,R₁)</td>
<td>2D RGBA</td>
<td>R₁G₁B₁A₁</td>
</tr>
</tbody>
</table>
# Dependent Alpha-Red Texturing

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>(A₀,R₀)</td>
<td>2D RGBA</td>
<td>R₀,G₀,B₀,A₀</td>
</tr>
<tr>
<td>1</td>
<td>Ignored</td>
<td>None</td>
<td>(A₀,R₀)</td>
<td>2D RGBA</td>
<td>R₁,G₁,B₁,A₁</td>
</tr>
</tbody>
</table>
## Dependent Alpha-Red Texturing

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>Texture specific</td>
<td>Any type Unsigned RGB[A]</td>
<td>R₀G₀B₀A₀</td>
</tr>
<tr>
<td>1</td>
<td>Ignored</td>
<td>None</td>
<td>(A₀,R₀)</td>
<td>2D RGBA</td>
<td>R₁G₁B₁A₁</td>
</tr>
</tbody>
</table>

```c
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

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>Texture specific</td>
<td>Any type Unsigned RGB[A]</td>
<td>$R_0G_0B_0A_0$</td>
</tr>
<tr>
<td>1</td>
<td>Ignored</td>
<td>None</td>
<td>(A₀,R₀)</td>
<td>2D RGBA</td>
<td>$R_1G_1B_1A_1$</td>
</tr>
</tbody>
</table>

```c
nvparsen("!!TS1.0
texture_2d();
dependent_ar( tex0 );");
```
## Dependent Green-Blue Texturing

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>Texture specific</td>
<td>Any type Unsigned RGB[A]</td>
<td>R₀G₀B₀A₀</td>
</tr>
<tr>
<td>1</td>
<td>Ignored</td>
<td>None</td>
<td>(G₀,B₀)</td>
<td>2D RGBA</td>
<td>R₁G₁B₁A₁</td>
</tr>
</tbody>
</table>
# Dependent Green-Blue Texturing

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>(G₀,B₀)</td>
<td>Any type Unsigned RGBA[A]</td>
<td>R₀,G₀,B₀,A₀</td>
</tr>
<tr>
<td>1</td>
<td>Ignored</td>
<td>None</td>
<td>2D RGBA</td>
<td></td>
<td>R₁,G₁,B₁,A₁</td>
</tr>
</tbody>
</table>
**Dependent Green-Blue Texturing**

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>(G₀,B₀)</td>
<td>Any type Unsigned RGBA[A]</td>
<td>R₀G₀B₀A₀</td>
</tr>
<tr>
<td>1</td>
<td>Ignored</td>
<td>None</td>
<td>(G₀,B₀)</td>
<td>2D RGBA</td>
<td>R₁G₁B₁A₁</td>
</tr>
</tbody>
</table>

```c
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

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>Texture specific</td>
<td>Any type Unsigned RGB[A]</td>
<td>$R_0 G_0 B_0 A_0$</td>
</tr>
<tr>
<td>1</td>
<td>Ignored</td>
<td>None</td>
<td>$(G_0, B_0)$</td>
<td>2D RGBA</td>
<td>$R_1 G_1 B_1 A_1$</td>
</tr>
</tbody>
</table>

```c
nvparsen("!!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

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(S₀, T₀, R₀, Q₀)</td>
<td>Text 2D</td>
<td>(S₀/Q₀, T₀/Q₀)</td>
<td>2D DSDT</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>(S₁, T₁)</td>
<td>S₁' = S₁ + k₀<em>ds + k₂</em>dt</td>
<td></td>
<td>2D Any Format</td>
<td>R₁G₁B₁A₁</td>
</tr>
</tbody>
</table>

S₁’ = S₁ + k₀*ds + k₂*dt
T₁’ = T₁ + k₁*ds + k₃*dt

k₀, k₁, k₂ and k₃ define a constant 2x2 floating-point matrix set by glTexEnv
### Offset Texture 2D

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(S₀, T₀, R₀, Q₀)</td>
<td>→ Texture 2D</td>
<td>(S₀/Q₀, T₀/Q₀)</td>
<td>2D DSDT (ds, dt)</td>
<td>(0,0,0,0)</td>
</tr>
</tbody>
</table>
| 1    | (S₁, T₁)        | → S₁' = S₁ + k₀*ds + k₂*dt  
|     |                |                   |              | 2D Any Format               | R₁,G₁,B₁,A₁  |
|      |                | T₁' = T₁ + k₁*ds + k₃*dt  |

k₀, k₁, k₂ and k₃ define a constant 2x2 floating-point matrix set by glTexEnv
# Offset Texture 2D

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(S₀, T₀, R₀, Q₀)</td>
<td>Texture 2D</td>
<td></td>
<td>2D DSDT</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>(S₁, T₁)</td>
<td>S₁' = S₁ + k₀<em>ds + k₂</em>dt</td>
<td>(S₀/Q₀, T₀/Q₀)</td>
<td>2D Any Format</td>
<td>R₁,G₁,B₁,A₁</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T₁' = T₁ + k₁<em>ds + k₃</em>dt</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

k₀, k₁, k₂ and k₃ define a constant 2x2 floating-point matrix set by `glTexEnv`
## Offset Texture 2D

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords ((S,T,R,Q))</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>((S_0,T_0,R_0,Q_0))</td>
<td>→ Texture 2D</td>
<td></td>
<td>2D DSDT</td>
<td>((0,0,0,0))</td>
</tr>
</tbody>
</table>
| 1     | \((S_1,T_1)\)              | \(S_1' = S_1 + k_0 \cdot ds + k_2 \cdot dt\) \(T_1' = T_1 + k_1 \cdot ds + k_3 \cdot dt\) | \(
begin{align*}
\frac{S_0}{Q_0}, \frac{T_0}{Q_0}
\end{align*}
\) | 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 \texttt{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 ($k_{\text{scale}}$ and $k_{\text{bias}}$ 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

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords ((S_0,T_0,R_0,Q_0))</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( (S_0,T_0,R_0,Q_0) )</td>
<td>( S_1' = S_1 + k_0<em>ds + k_2</em>dt )</td>
<td>( \left( \frac{S_0}{Q_0}, \frac{T_0}{Q_0} \right) )</td>
<td>( DSDT_Mag )</td>
<td>( (0,0,0,0) )</td>
</tr>
<tr>
<td>1</td>
<td>( (S_1,T_1) )</td>
<td>( T_1' = T_1 + k_1<em>ds + k_3</em>dt )</td>
<td>( (S_1',T_1') )</td>
<td>( 2D ) RGBA</td>
<td>( (R<em>M, G</em>M, B*M, A) )</td>
</tr>
</tbody>
</table>

\( k_0, k_1, k_2 \) and \( k_3 \) define a constant floating-point 2x2 matrix set by \( \text{glTexEnv} \)

\( k_{\text{scale}} \) and \( k_{\text{bias}} \) define constant floating-point scale/bias set by \( \text{glTexEnv} \)
Offset Texture 2D Scale

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(S₀, T₀, R₀, Q₀)</td>
<td>Texture 2D</td>
<td>(S₀/Q₀, T₀/Q₀)</td>
<td>2D DSDT_Mag</td>
<td>(0,0,0,0)</td>
</tr>
</tbody>
</table>
| 1     | (S₁, T₁)                 | S₁' = S₁ + k₀*ds + k₂*dt 
T₁' = T₁ + k₁*ds + k₃*dt | (S₁', T₁') | 2D RGBA                    | (R*M, G*M, B*M,A) |

k₀, k₁, k₂ and k₃ 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**

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(S₀, T₀, R₀, Q₀)</td>
<td>Texture 2D</td>
<td>(S₀/Q₀', T₀/Q₀')</td>
<td>DSDT_Mag (ds, dt, mag)</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>(S₁, T₁)</td>
<td></td>
<td>(S₁', T₁')</td>
<td>2D RGBA (R<em>M, G</em>M, B*M, A)</td>
<td></td>
</tr>
</tbody>
</table>

\[
S₁' = S₁ + k₀ * ds + k₂ * dt \\
T₁' = T₁ + k₁ * ds + k₃ * dt \\
M = k_{scale} * mag + k_{bias}
\]

k₀, k₁, k₂ and k₃ 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

Texture Coords: \((S, T, R, Q)\)

Shader Operations:
- Texture 2D

Texture Fetch:
- \((S_0, T_0, R_0, Q_0)\)
- \((S, T)\)
- \((S_1', T_1')\)
- \((R, G, B, A)\)

Bound Texture Target/Format:
- 2D DSDT_Mag
- \((ds, dt, mag)\)

Output Color:
- \((0, 0, 0, 0)\)

\[
S_1' = S_1 + k_0 * ds + k_2 * dt \\
T_1' = T_1 + k_1 * ds + k_3 * dt \\
M = k_{\text{scale}} * mag + k_{\text{bias}}
\]

As with all output colors, each scaled RGB component is clamped to \([0, 1]\)
Offset Texture 2D Scale using DSDT_MAG_INTENSITY

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(S₀,T₀,R₀,Q₀)</td>
<td>Texture 2D</td>
<td>(S₀/Q₀, T₀/Q₀)</td>
<td>2D DSDT_MAG_INTENSITY</td>
<td>(i,i,i,i)</td>
</tr>
<tr>
<td>1</td>
<td>(S₁,T₁)</td>
<td></td>
<td>(S₁’, T₁’)</td>
<td>2D RGBA</td>
<td>(R<em>ₘ,G</em>ₘ,B*ₘ,A)</td>
</tr>
</tbody>
</table>

\[
M = k_{\text{scale}} \times \text{mag} + k_{\text{bias}}
\]

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

```gl
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:

```c
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_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

\[ (HI, LO, 1) \]
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

```c
glActiveTextureARB(GL_TEXTURE0_ARB);
glTexEnviv(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV, GL_TEXTURE_2D);

glActiveTextureARB(GL_TEXTURE1_ARB);
glTexEnviv(GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV, GL_DOT_PRODUCT_NV);
glTexEnviv(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:

\[
\begin{bmatrix}
S' \\
T'
\end{bmatrix} = M \tilde{\mathbf{n}} =
\begin{bmatrix}
S_0 & T_0 & R_0 \\
S_1 & T_1 & R_1
\end{bmatrix}
\begin{bmatrix}
x' \\
y' \\
z'
\end{bmatrix}
\]

Matrix can be thought of as the “Texel Matrix”, and transforms previous texture result (e.g. a normal) from \( \mathbb{R}^3 \) to \( \mathbb{R}^2 \), then uses transformed 2D vector to access a 2D texture.
# Dot Product Texture 2D

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>None</td>
<td>Any type Signed RGBA[A]</td>
<td>R₀G₀B₀</td>
</tr>
<tr>
<td>1</td>
<td>(S₁, T₁, R₁)</td>
<td>( U_x = [S_1, T_1, R_1] \cdot [R_0, G_0, B_0] )</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(S₂, T₂, R₂)</td>
<td>( U_y = [S_2, T_2, R_2] \cdot [R_0, G_0, B_0] )</td>
<td>( (U_x, U_y) )</td>
<td>2D RGBA</td>
<td>R₂G₂B₂A₂</td>
</tr>
</tbody>
</table>
## Dot Product Texture 2D

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch Target/Format</th>
<th>Bound Texture</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>TEXTURE</td>
<td>Any type</td>
<td>R0G0B0</td>
</tr>
<tr>
<td>1</td>
<td>(Sₙ, Tₙ, Rₙ₁)</td>
<td>Uₓ = [S₁, T₁, R₁] • [R₀, G₀, B₀]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(S₂, T₂, R₂)</td>
<td>Uᵧ = [S₂, T₂, R₂] • [R₀, G₀, B₀]</td>
<td>(Uₓ, Uᵧ)</td>
<td>2D RGBA</td>
<td>R₂G₂B₂A₂</td>
</tr>
</tbody>
</table>
# Dot Product Texture 2D

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td></td>
<td>Texture specific</td>
<td>Any type Signed RGB[A]</td>
<td>R₀G₀B₀</td>
</tr>
<tr>
<td>1</td>
<td>(S₁, T₁, R₁)</td>
<td>Uₓ = [S₁, T₁, R₁] · [R₀, G₀, B₀]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(S₂, T₂, R₂)</td>
<td>Uᵧ = [S₂, T₂, R₂] · [R₀, G₀, B₀]</td>
<td>(Uₓ, Uᵧ)</td>
<td>2D RGBA</td>
<td>R₂G₂B₂A₂</td>
</tr>
</tbody>
</table>
## Dot Product Texture 2D

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td></td>
<td></td>
<td></td>
<td>R_0G_0B_0</td>
</tr>
<tr>
<td>1</td>
<td>(S_1, T_1, R_1)</td>
<td>U_x = [S_1, T_1, R_1] • [R_0, G_0, B_0]</td>
<td>None</td>
<td>Any type Signed RGBA</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(S_2, T_2, R_2)</td>
<td>U_y = [S_2, T_2, R_2] • [R_0, G_0, B_0]</td>
<td>(U_x, U_y)</td>
<td>2D RGBA</td>
<td>R_2G_2B_2A_2</td>
</tr>
</tbody>
</table>
Dot Product Texture 2D
GL_DOT_PRODUCT_TEXTURE_2D_NV

```c
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

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>Texture specific</td>
<td>None</td>
<td>Any type Signed RGB[A]</td>
</tr>
<tr>
<td>1</td>
<td>(S₁, T₁, R₁)</td>
<td>Uₓ=[S₁, T₁, R₁] • [R₀, G₀, B₀]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(S₂, T₂, R₂)</td>
<td>Uᵧ=[S₂, T₂, R₂] • [R₀, G₀, B₀]</td>
<td>(Uₓ, Uᵧ)</td>
<td>Texture Rectangle RGBA</td>
<td>R₂G₂B₂A₂</td>
</tr>
</tbody>
</table>
Dot Product Texture Rectangle

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>None</td>
<td>R₀G₀B₀</td>
</tr>
<tr>
<td>1</td>
<td>(S₁, T₁, R₁)</td>
<td>Uₓ = [S₁, T₁, R₁] • [R₀, G₀, B₀]</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(S₂, T₂, R₂)</td>
<td>Uᵧ = [S₂, T₂, R₂] • [R₀, G₀, B₀]</td>
<td>(Uₓ, Uᵧ)</td>
<td>R₂G₂B₂A₂</td>
</tr>
</tbody>
</table>
# Dot Product Texture Rectangle

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td></td>
<td>None</td>
<td>R₀G₀B₀</td>
</tr>
<tr>
<td>1</td>
<td>(S₁, T₁, R₁)</td>
<td>$U_x = [S_1, T_1, R_1] \cdot [R_0, G_0, B_0]$</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(S₂, T₂, R₂)</td>
<td>$U_y = [S_2, T_2, R_2] \cdot [R_0, G_0, B_0]$ ($U_x, U_y$)</td>
<td>RGBA</td>
<td>R₂G₂B₂A₂</td>
</tr>
</tbody>
</table>
# Dot Product Texture Rectangle

<table>
<thead>
<tr>
<th>Text</th>
<th>Texture Coords</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>Texture specific</td>
<td>Any type Signed RGB[A]</td>
<td>R0G0B0</td>
</tr>
<tr>
<td>1</td>
<td>(S1, T1, R1)</td>
<td>Ux = [S1, T1, R1] • [R0, G0, B0]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(S2, T2, R2)</td>
<td>Uy = [S2, T2, R2] • [R0, G0, B0]</td>
<td>(Ux, Uy)</td>
<td>Texture Rectangle RGBA</td>
<td>R2G2B2A2</td>
</tr>
</tbody>
</table>
Dot Product Texture Rectangle
GL_DOT_PRODUCT_TEXTURE_RECTANGLE_NV

```c
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

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td></td>
<td></td>
<td>Any type Signed RGB[A]</td>
<td>R₀G₀B₀</td>
</tr>
<tr>
<td>1</td>
<td>(S₁, T₁, R₁)</td>
<td>S' = [S₁, T₁, R₁] • [R₀, G₀, B₀]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(S₂, T₂, R₂)</td>
<td>T' = [S₂, T₂, R₂] • [R₀, G₀, B₀]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>3</td>
<td>(S₃, T₃, R₃)</td>
<td>R' = [S₃, T₃, R₃] • [R₀, G₀, B₀]</td>
<td>(S', T', R')</td>
<td>3D RGBA</td>
<td>R₃G₃B₃A₃</td>
</tr>
</tbody>
</table>
# Dot Product Texture 3D

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>Texture specific</td>
<td>Any type Signed RGB[A]</td>
<td>R₀G₀B₀</td>
</tr>
<tr>
<td>1</td>
<td>(S₁, T₁, R₁)</td>
<td>S’ = [S₁, T₁, R₁] • [R₀, G₀, B₀]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(S₂, T₂, R₂)</td>
<td>T’ = [S₂, T₂, R₂] • [R₀, G₀, B₀]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
</tbody>
</table>
Dot Product Texture 3D

```c
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:

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

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

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td></td>
<td></td>
<td>Any type Signed RGB[A]</td>
<td>R₀G₀B₀</td>
</tr>
<tr>
<td>1</td>
<td>(Tₓ, Bₓ, Nₓ)</td>
<td>$Uₓ = [Tₓ, Bₓ, Nₓ] \cdot [R₀, G₀, B₀]$</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(Tᵧ, Bᵧ, Nᵧ)</td>
<td>$Uᵧ = [Tᵧ, Bᵧ, Nᵧ] \cdot [R₀, G₀, B₀]$</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>3</td>
<td>(Tᶻ, Bᶻ, Nᶻ)</td>
<td>$Uᶻ = [Tᶻ, Bᶻ, Nᶻ] \cdot [R₀, G₀, B₀]$</td>
<td>None</td>
<td>Cube map RGBA</td>
<td>R₃G₃B₃A₃</td>
</tr>
</tbody>
</table>
## Dot Product Texture Cube Map

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Text specific</td>
<td>Text specific</td>
<td>Any type Signed RGB[A]</td>
<td>R0G0B0</td>
</tr>
<tr>
<td>1</td>
<td>(Tx, Bx, Nx)</td>
<td>Ux = [Tx, Bx, Nx] · [R0, G0, B0]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(Ty, By, Ny)</td>
<td>Uy = [Ty, By, Ny] · [R0, G0, B0]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>3</td>
<td>(Tz, Bz, Nz)</td>
<td>Uz = [Tz, Bz, Nz] · [R0, G0, B0]</td>
<td>None</td>
<td>Cube map RGBA</td>
<td>R3G3B3A3</td>
</tr>
</tbody>
</table>

\[
U = (U_x, U_y, U_z)
\]
# Dot Product Texture Cube Map

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td></td>
<td></td>
<td></td>
<td>R0G0B0</td>
</tr>
<tr>
<td>1</td>
<td>(Tx, Bx, Nx)</td>
<td>(U_x = [T_x, B_x, N_x] \cdot [R_0, G_0, B_0])</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(Ty, By, Ny)</td>
<td>(U_y = [T_y, B_y, N_y] \cdot [R_0, G_0, B_0])</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>3</td>
<td>(Tz, Bz, Nz)</td>
<td>(U_z = [T_z, B_z, N_z] \cdot [R_0, G_0, B_0])</td>
<td>None</td>
<td>Cube map RGBA</td>
<td>R3G3B3A3</td>
</tr>
</tbody>
</table>

\(U = (U_x, U_y, U_z)\)
## Dot Product Texture Cube Map

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>Any type</td>
<td>None</td>
<td>R₀G₀B₀</td>
</tr>
<tr>
<td>1</td>
<td>( (T_x, B_x, N_x) )</td>
<td>( U_x = [T_x, B_x, N_x] \cdot [R_0, G_0, B_0] )</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>( (T_y, B_y, N_y) )</td>
<td>( U_y = [T_y, B_y, N_y] \cdot [R_0, G_0, B_0] )</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>3</td>
<td>( (T_z, B_z, N_z) )</td>
<td>( U_z = [T_z, B_z, N_z] \cdot [R_0, G_0, B_0] )</td>
<td>None</td>
<td>Cube map RGBA</td>
<td>R₃G₃B₃A₃</td>
</tr>
</tbody>
</table>

\[ U = (U_x, U_y, U_z) \]
Dot Product Texture Cube Map
GL_DOT_PRODUCT_TEXTURE_CUBE_MAP_NV

```c
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)

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

\[ E = (E_x, E_y, E_z) \]

\[ R = \frac{2n'(n' \cdot E)}{(n' \cdot n')} - E \]
# Dot Product Constant Eye Reflect Cube Map

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td></td>
<td>Texte specific</td>
<td></td>
<td>R0G0B0</td>
</tr>
<tr>
<td>1</td>
<td>(Tx, Bx, Nx)</td>
<td>Ux = [Tx, Bx, Nx] · [R0, G0, B0]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(Ty, By, Ny)</td>
<td>Uy = [Ty, By, Ny] · [R0, G0, B0]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>3</td>
<td>(Tz, Bz, Nz)</td>
<td>Uz = [Tz, Bz, Nz] · [R0, G0, B0] U = (Ux, Uy, Uz) E = (Ex, Ey, Ez) R = (Rx, Ry, Rz) R = \frac{2U(U \cdot E)}{(U \cdot U)} - E</td>
<td>None</td>
<td>Cube map RGBA</td>
<td>R3G3B3A3</td>
</tr>
</tbody>
</table>

U = (Ux, Uy, Uz)
E = (Ex, Ey, Ez)
R = (Rx, Ry, Rz)

R = \frac{2U(U \cdot E)}{(U \cdot U)} - E

Texture Coords: (S, T, R, Q)

Output Color: R0G0B0

Bound Texture Target/Format: Any type Signed RGB[A]
# Dot Product Constant Eye Reflect Cube Map

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch Target/Format</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td></td>
<td>Text specific</td>
<td></td>
<td>R₀G₀B₀</td>
</tr>
<tr>
<td>1</td>
<td>(Tx, Bx, Nx)</td>
<td>Uₓ = [Tx, Bx, Nx] • [R₀, G₀, B₀]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(Ty, By, Ny)</td>
<td>Uᵧ = [Ty, By, Ny] • [R₀, G₀, B₀]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>3</td>
<td>(Tz, Bz, Nz)</td>
<td>Uₚ = [Tz, Bz, Nz] • [R₀, G₀, B₀]</td>
<td>Cube map RGBA</td>
<td>R = (Rₓ, Rᵧ, Rₚ)</td>
<td>RₚGₚBₚAₚ</td>
</tr>
</tbody>
</table>

\[
U = (Uₓ, Uᵧ, Uₚ)
\]
\[
E = (Eₓ, Eᵧ, Eₚ)
\]
\[
R = \frac{2U(U • E)}{(U • U)} - E
\]
## Dot Product Constant Eye Reflect Cube Map

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td></td>
<td></td>
<td>Any type</td>
<td>R₀G₀B₀</td>
</tr>
<tr>
<td>1</td>
<td>(Tₓ, Bₓ, Nx)</td>
<td></td>
<td></td>
<td>Signed RGB[A]</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(Tᵧ, Bᵧ, Ny)</td>
<td></td>
<td></td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>3</td>
<td>(Tz, Bz, Nz)</td>
<td></td>
<td></td>
<td>Cube map RGBA</td>
<td>R₃G₃B₃A₃</td>
</tr>
</tbody>
</table>

\[ U_x = [T_x, B_x, N_x] \cdot [R_0, G_0, B_0] \]
\[ U_y = [T_y, B_y, N_y] \cdot [R_0, G_0, B_0] \]
\[ U_z = [T_z, B_z, N_z] \cdot [R_0, G_0, B_0] \]
\[ U = (U_x, U_y, U_z) \]
\[ E = (E_x, E_y, E_z) \]
\[ R = \frac{2U(U \cdot E)}{(U \cdot U)} - E \]
# Dot Product Constant Eye Reflect Cube Map

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>Texture specific</td>
<td>Any type → R₀G₀B₀</td>
<td>R₀G₀B₀</td>
</tr>
<tr>
<td>1</td>
<td>(Tx, Bx, Nx)</td>
<td>Uₓ = [Tx, Bx, Nx] • [R₀, G₀, B₀]</td>
<td>None</td>
<td>None → (0,0,0,0)</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(Ty, By, Ny)</td>
<td>Uᵧ = [Ty, By, Ny] • [R₀, G₀, B₀]</td>
<td>None</td>
<td>None → (0,0,0,0)</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>3</td>
<td>(Tz, Bz, Nz)</td>
<td>Uz = [Tz, Bz, Nz] • [R₀, G₀, B₀]</td>
<td>R = 2U(U•E)</td>
<td>Cube map RGBA → R₃G₃B₃A₃</td>
<td>R₃G₃B₃A₃</td>
</tr>
</tbody>
</table>

\[
R = \frac{(U \cdot E)}{(U \cdot U)} - E
\]
# Dot Product Constant Eye Reflect Cube Map

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords ((S,T,R,Q))</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>Texture specific</td>
<td>Any type Signed RGB[A]</td>
<td>(R_0G_0B_0)</td>
</tr>
<tr>
<td>1</td>
<td>((T_x, B_x, N_x))</td>
<td>(U_x = [T_x, B_x, N_x] \cdot [R_0, G_0, B_0])</td>
<td>None</td>
<td>None</td>
<td>((0,0,0,0))</td>
</tr>
<tr>
<td>2</td>
<td>((T_y, B_y, N_y))</td>
<td>(U_y = [T_y, B_y, N_y] \cdot [R_0, G_0, B_0])</td>
<td>None</td>
<td>None</td>
<td>((0,0,0,0))</td>
</tr>
<tr>
<td>3</td>
<td>((T_z, B_z, N_z))</td>
<td>(U_z = [T_z, B_z, N_z] \cdot [R_0, G_0, B_0])</td>
<td>None</td>
<td>Cube map RGBA</td>
<td>(R_3G_3B_3A_3)</td>
</tr>
</tbody>
</table>
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.

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

\[
E = (q_0, q_1, q_2)
\]

\[
R = \frac{2n(n' \cdot E)}{(n' \cdot n')} - E
\]
### Dot Product Reflect Cube Map

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Texture specific</td>
<td>Any type RGB[A]</td>
<td>R₀G₀B₀</td>
</tr>
<tr>
<td>1</td>
<td>([Tₓ, Bₓ, Nₓ], Eₓ)</td>
<td>(Uₓ = [Tₓ, Bₓ, Nₓ] \cdot [R₀, G₀, B₀])</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>([Tᵧ, Bᵧ, Nᵧ], Eᵧ)</td>
<td>(Uᵧ = [Tᵧ, Bᵧ, Nᵧ] \cdot [R₀, G₀, B₀])</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>3</td>
<td>([Tᶻ, Bᶻ, Nᶻ], Eᶻ)</td>
<td>(Uᶻ = [Tᶻ, Bᶻ, Nᶻ] \cdot [R₀, G₀, B₀])</td>
<td>Cube map RGBA</td>
<td>R₃G₃B₃A₃</td>
</tr>
</tbody>
</table>

\[
\text{R} = \left( Rₓ, Rᵧ, Rᶻ \right) \quad \text{E} = \left( Eₓ, Eᵧ, Eᶻ \right) \\
\text{R} = \frac{2\text{U}(\text{U} \cdot \text{E})}{(\text{U} \cdot \text{U})} - \text{E}
\]
Dot Product Reflect Cube Map

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td></td>
<td></td>
<td>Any type Signed RGB[A]</td>
<td>$R_0 G_0 B_0$</td>
</tr>
<tr>
<td>1</td>
<td>$([T_x, B_x, N_x], E_x)$</td>
<td>$U_x = [T_x, B_x, N_x] \cdot [R_0, G_0, B_0]$</td>
<td>None</td>
<td>None</td>
<td>$(0,0,0,0)$</td>
</tr>
<tr>
<td>2</td>
<td>$([T_y, B_y, N_y], E_y)$</td>
<td>$U_y = [T_y, B_y, N_y] \cdot [R_0, G_0, B_0]$</td>
<td>None</td>
<td>None</td>
<td>$(0,0,0,0)$</td>
</tr>
<tr>
<td>3</td>
<td>$([T_z, B_z, N_z], E_z)$</td>
<td>$U_z = [T_z, B_z, N_z] \cdot [R_0, G_0, B_0]$</td>
<td></td>
<td>Cube map RGBA</td>
<td>$R_3 G_3 B_3 A_3$</td>
</tr>
</tbody>
</table>

$\mathbf{R} = \frac{2\mathbf{U}(\mathbf{U} \cdot \mathbf{E})}{(\mathbf{U} \cdot \mathbf{U})} - \mathbf{E}$
## Dot Product Reflect Cube Map

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>![Equation](U = (U_x, U_y, U_z))</td>
<td>Texture specific</td>
<td>Any type Signed RGB[A]</td>
<td>R &lt;sub&gt;0&lt;/sub&gt; G &lt;sub&gt;0&lt;/sub&gt; B &lt;sub&gt;0&lt;/sub&gt;</td>
</tr>
<tr>
<td>1</td>
<td>([&lt;i&gt;T_x&lt;/i&gt;, &lt;i&gt;B_x&lt;/i&gt;, &lt;i&gt;N_x&lt;/i&gt;], &lt;i&gt;E_x&lt;/i&gt;)</td>
<td>![Equation](U_x = [T_x, B_x, N_x] \cdot [R_0, G_0, B_0])</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>([&lt;i&gt;T_y&lt;/i&gt;, &lt;i&gt;B_y&lt;/i&gt;, &lt;i&gt;N_y&lt;/i&gt;], &lt;i&gt;E_y&lt;/i&gt;)</td>
<td>![Equation](U_y = [T_y, B_y, N_y] \cdot [R_0, G_0, B_0])</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>3</td>
<td>([&lt;i&gt;T_z&lt;/i&gt;, &lt;i&gt;B_z&lt;/i&gt;, &lt;i&gt;N_z&lt;/i&gt;], &lt;i&gt;E_z&lt;/i&gt;)</td>
<td>![Equation](U_z = [T_z, B_z, N_z] \cdot [R_0, G_0, B_0])</td>
<td></td>
<td>Cube map RGBA</td>
<td>R &lt;sub&gt;3&lt;/sub&gt; G &lt;sub&gt;3&lt;/sub&gt; B &lt;sub&gt;3&lt;/sub&gt; A &lt;sub&gt;3&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

Where:
- **<i>U</i>** = (U_x, U_y, U_z)
- **<i>E</i>** = (E_x, E_y, E_z)
- **<i>R</i>** = \(\frac{2U(U \cdot E)}{(U \cdot U)} - E\)
# Dot Product Reflect Cube Map

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td></td>
<td>Texture specific</td>
<td>Any type Signed RGBA[A]</td>
<td>R0G0B0</td>
</tr>
<tr>
<td>1</td>
<td>([Tx, Bx, Nx], Ex)</td>
<td>Ux = [Tx, Bx, Nx] ∙ [R0, G0, B0]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>([Ty, By, Ny], Ey)</td>
<td>Uy = [Ty, By, Ny] ∙ [R0, G0, B0]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>3</td>
<td>([Tz, Bz, Nz], Ez)</td>
<td>Uz = [Tz, Bz, Nz] ∙ [R0, G0, B0]</td>
<td>None</td>
<td>Cube map RGBA</td>
<td>R3G3B3A3</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
R &= \left( \frac{2U(U \cdot E)}{(U \cdot U)} - E \right)
\end{align*}
\]
# Dot Product Reflect Cube Map

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>Text specific</td>
<td>Text specific</td>
<td>Any type</td>
<td>R_0G_0B_0</td>
</tr>
<tr>
<td>1</td>
<td>([T_x, B_x, N_x], E_x)</td>
<td>U_x = [T_x, B_x, N_x] \cdot [R_0, G_0, B_0]</td>
<td>None</td>
<td>Cubemap RGBA</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>([T_y, B_y, N_y], E_y)</td>
<td>U_y = [T_y, B_y, N_y] \cdot [R_0, G_0, B_0]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>3</td>
<td>([T_z, B_z, N_z], E_z)</td>
<td>U_z = [T_z, B_z, N_z] \cdot [R_0, G_0, B_0]</td>
<td>None</td>
<td>R = (R_x, R_y, R_z)</td>
<td>R_3G_3B_3A_3</td>
</tr>
</tbody>
</table>
Dot Product Reflect Cube Map
GL_DOT_PRODUCT_REFLECT_CUBE_MAP_NV

```c
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)

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td></td>
<td>Any type Signed RGBA</td>
<td>R0G0B0</td>
</tr>
<tr>
<td>1</td>
<td>([Tₓ, Bₓ, Nₓ], Eₓ)</td>
<td>Uₓ = [Tₓ, Bₓ, Nₓ] • [R₀, G₀, B₀]</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>([Tᵧ, Bᵧ, Nᵧ], Eᵧ)</td>
<td>Uᵧ = [Tᵧ, Bᵧ, Nᵧ] • [R₀, G₀, B₀]</td>
<td>Cube map RGBA</td>
<td>R₂G₂B₂A₂</td>
</tr>
<tr>
<td>3</td>
<td>([T₂, B₂, N₂], E₂)</td>
<td>Uz = [T₂, B₂, N₂] • [R₀, G₀, B₀]</td>
<td>Cube map RGBA</td>
<td>R₃G₃B₃A₃</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
U &= (Uₓ, Uᵧ, Uz) \\
E &= (Eₓ, Eᵧ, Ez) \\
R &= \frac{2U(U\cdot E)}{(U\cdot U)} - E
\end{align*}
\]
# Dot Product Diffuse Cube Map
(with Dot Product Reflect Cube Map)

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch Target/Format</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td></td>
<td>Any type Signed RGB[A]</td>
<td>None</td>
<td>$R_0G_0B_0$</td>
</tr>
<tr>
<td>1</td>
<td>$([T_x, B_x, N_x], E_x)$</td>
<td>$U_x = [T_x, B_x, N_x] \cdot [R_0, G_0, B_0]$</td>
<td>Texture specific</td>
<td>None</td>
<td>$(0,0,0,0)$</td>
</tr>
<tr>
<td>2</td>
<td>$([T_y, B_y, N_y], E_y)$</td>
<td>$U_y = [T_y, B_y, N_y] \cdot [R_0, G_0, B_0]$</td>
<td>Texture specific</td>
<td>Cube map RGBA</td>
<td>$R_2G_2B_2A_2$</td>
</tr>
<tr>
<td>3</td>
<td>$([T_z, B_z, N_z], E_z)$</td>
<td>$U_z = [T_z, B_z, N_z] \cdot [R_0, G_0, B_0]$</td>
<td>$U = (U_x, U_y, U_z)$</td>
<td>Cube map RGBA</td>
<td>$R_3G_3B_3A_3$</td>
</tr>
</tbody>
</table>

$$R = \frac{2U(U \cdot E)}{(U \cdot U)} - E$$
Dot Product Diffuse Cube Map (with Dot Product Reflect Cube Map)

<table>
<thead>
<tr>
<th>Tex</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td></td>
<td>None (0,0,0,0)</td>
<td>R₀G₀B₀</td>
</tr>
<tr>
<td>1</td>
<td>([Tₓ, Bₓ, Nₓ], Eₓ)</td>
<td>Uₓ = [Tₓ,Bₓ,Nₓ] ⊖ [R₀,G₀,B₀]</td>
<td>None (0,0,0,0)</td>
<td>R₂G₂B₂A₂</td>
</tr>
<tr>
<td>2</td>
<td>([Tᵧ, Bᵧ, Nᵧ], Eᵧ)</td>
<td>Uᵧ = [Tᵧ,Bᵧ,Nᵧ] ⊖ [R₀,G₀,B₀]</td>
<td>Any type Signed RGB[A]</td>
<td>R₂G₂B₂A₂</td>
</tr>
<tr>
<td>3</td>
<td>([Tᶻ, Bᶻ, Nᶻ], Eᶻ)</td>
<td>Uᵢ = [Tₓ,Bₓ,Nₓ] ⊖ [R₀,G₀,B₀]</td>
<td>Cube map RGBA</td>
<td>R₃G₃B₃A₃</td>
</tr>
</tbody>
</table>

Uᵢ = (Uₓ, Uᵧ, Uᵢ)

R = \( \frac{2U(U \cdot E)}{(U \cdot U)} - E \)
# Dot Product Diffuse Cube Map
(with Dot Product Reflect Cube Map)

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch Target/Format</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td></td>
<td>Text specific</td>
<td>Any type Signed RGB[A]</td>
<td>R₀G₀B₀</td>
</tr>
<tr>
<td>1</td>
<td>([Tx, Bx, Nx], Ex)</td>
<td>Uₓ = [Tx, Bx, Nx] • [R₀, G₀, B₀]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>([Ty, By, Ny], Ey)</td>
<td>Uᵧ = [Ty, By, Ny] • [R₀, G₀, B₀]</td>
<td>Cube map RGBA</td>
<td>Cube map RGBA</td>
<td>R₂G₂B₂A₂</td>
</tr>
<tr>
<td>3</td>
<td>([Tz, Bz, Nz], Ez)</td>
<td>Uₙ = [Tz, Bz, Nz] • [R₀, G₀, B₀]</td>
<td>Cube map RGBA</td>
<td>Cube map RGBA</td>
<td>R₃G₃B₃A₃</td>
</tr>
</tbody>
</table>

\[ \mathbf{U} = (Uₓ, Uᵧ, Uₙ) \]

\[ \mathbf{R} = \frac{2 \mathbf{U} (\mathbf{U} \cdot \mathbf{E})}{(\mathbf{U} \cdot \mathbf{U})} - \mathbf{E} \]
## Dot Product Diffuse Cube Map (with Dot Product Reflect Cube Map)

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>App specific</td>
<td>→</td>
<td>None</td>
<td>Cube map RGBA</td>
<td>R₀G₀B₀</td>
</tr>
<tr>
<td>1</td>
<td>([T_x, B_x, N_x, E_x])</td>
<td>(U_x = [T_x, B_x, N_x] \cdot [R_0, G_0, B_0])</td>
<td>Texture specific</td>
<td>Any type Signed RGBA[A]</td>
<td>((0,0,0,0))</td>
</tr>
<tr>
<td>2</td>
<td>([T_y, B_y, N_y, E_y])</td>
<td>(U_y = [T_y, B_y, N_y] \cdot [R_0, G_0, B_0])</td>
<td>None</td>
<td>Cube map RGBA</td>
<td>R₂G₂B₂A₂</td>
</tr>
<tr>
<td>3</td>
<td>([T_z, B_z, N_z, E_z])</td>
<td>(U_z = [T_z, B_z, N_z] \cdot [R_0, G_0, B_0])</td>
<td>None</td>
<td>Cube map RGBA</td>
<td>R₃G₃B₃A₃</td>
</tr>
</tbody>
</table>

\[U = (U_x, U_y, U_z)\]
\[R = \frac{2U(U \cdot E)}{(U \cdot U)} - E\]
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

Depth map

Z-Buffer

scale

bias

Billboard
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

### Shader Operations

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords</th>
<th>Shader Operations</th>
<th>Texture Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>((S_0, T_0, R_0, Q_0))</td>
<td>Texture 2D</td>
<td>(\frac{S_0}{Q_0}, \frac{T_0}{Q_0})</td>
<td>(0, 0, 0, 0)</td>
</tr>
<tr>
<td>1</td>
<td>(\left(\frac{Z_{\text{scale}}}{2^{16}}, Z_{\text{bias}}\right))</td>
<td>(Z = \left(\frac{Z_{\text{scale}}}{2^{16}}, Z_{\text{bias}}\right) \cdot [H, L, 1])</td>
<td>None</td>
<td>(0, 0, 0, 0)</td>
</tr>
<tr>
<td>2</td>
<td>(\left(\frac{W_{\text{scale}}}{2^{16}}, W_{\text{bias}}\right))</td>
<td>(W = \left(\frac{W_{\text{scale}}}{2^{16}}, W_{\text{bias}}\right) \cdot [H, L, 1])</td>
<td>None</td>
<td>(0, 0, 0, 0)</td>
</tr>
</tbody>
</table>

\[Z_{\text{window}} = \frac{Z}{W}\]

Replaces current fragment’s depth
## Dot Product Depth Replace

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(S₀,T₀,R₀,Q₀)</td>
<td>Texture 2D</td>
<td>(S₀/Q₀, T₀/Q₀)</td>
<td>2D Unsigned HILO</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>(Z_{scale}, Z_{scale}/2^{16}, Z_{bias})</td>
<td>(Z = (Z_{scale}, Z_{scale}/2^{16}, Z_{bias}) \cdot [H, L, 1])</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(W_{scale}, W_{scale}/2^{16}, W_{bias})</td>
<td>(W = (W_{scale}, W_{scale}/2^{16}, W_{bias}) \cdot [H, L, 1])</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
</tbody>
</table>

\[Z_{window} = \frac{Z}{W}\]

Replaces current fragment’s depth
# Dot Product Depth Replace

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$(S_0, T_0, R_0, Q_0)$</td>
<td>Texture 2D</td>
<td>$(S_0/Q_0, T_0/Q_0)$</td>
<td>None</td>
<td>$(0,0,0,0)$</td>
</tr>
<tr>
<td>1</td>
<td>$(Z_{scale}, Z_{scale}/2^{16}, Z_{bias})$</td>
<td>$Z = (Z_{scale}, Z_{scale}/2^{16}, Z_{bias}) \cdot [H,L,1]$</td>
<td>None</td>
<td>None</td>
<td>$(0,0,0,0)$</td>
</tr>
<tr>
<td>2</td>
<td>$(W_{scale}, W_{scale}/2^{16}, W_{bias})$</td>
<td>$W = (W_{scale}, W_{scale}/2^{16}, W_{bias}) \cdot [H,L,1]$</td>
<td>None</td>
<td>None</td>
<td>$(0,0,0,0)$</td>
</tr>
</tbody>
</table>

\[ Z_{window} = \frac{Z}{W} \]

Replaces current fragment’s depth
## Dot Product Depth Replace

<table>
<thead>
<tr>
<th>Tex #</th>
<th>Texture Coords (S,T,R,Q)</th>
<th>Shader Operations</th>
<th>Texture Fetch</th>
<th>Bound Texture Target/Format</th>
<th>Output Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(S₀, T₀, R₀, Q₀)</td>
<td>Texture 2D</td>
<td>(S₀, T₀, Q₀)</td>
<td>2D Unsigned HILO</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>(Z_{scale}, Z_{bias} \cdot 2^{16})</td>
<td>Z = (Z_{scale}, Z_{bias} \cdot 2^{16}) \cdot [H, L, 1]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>2</td>
<td>(W_{scale}, W_{bias} \cdot 2^{16})</td>
<td>W = (W_{scale}, W_{bias} \cdot 2^{16}) \cdot [H, L, 1]</td>
<td>None</td>
<td>None</td>
<td>(0,0,0,0)</td>
</tr>
</tbody>
</table>

Replaces current fragment’s depth
Dot Product Depth Replace
GL_DOT_PRODUCT DEPTH_REPLACE_NV

```c
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 HIL0 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 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

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

    glGetIntiv(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.