Reflective Bump Mapping

Cass Everitt

NVIDIA Corporation

cass@nvidia.com

Overview

- Review of per-vertex reflection mapping
  - Bump mapping and reflection mapping
- Reflective Bump Mapping
  - Pseudo-reflective bump mapping
    - Offset bump mapping, or EMBM (misnomer)
    - Tangent-space support?
    - Common usage
  - True Reflective bump mapping
    - Simple object-space implementation
    - Supports tangent-space, and more normal control

Per-Vertex Reflection Mapping

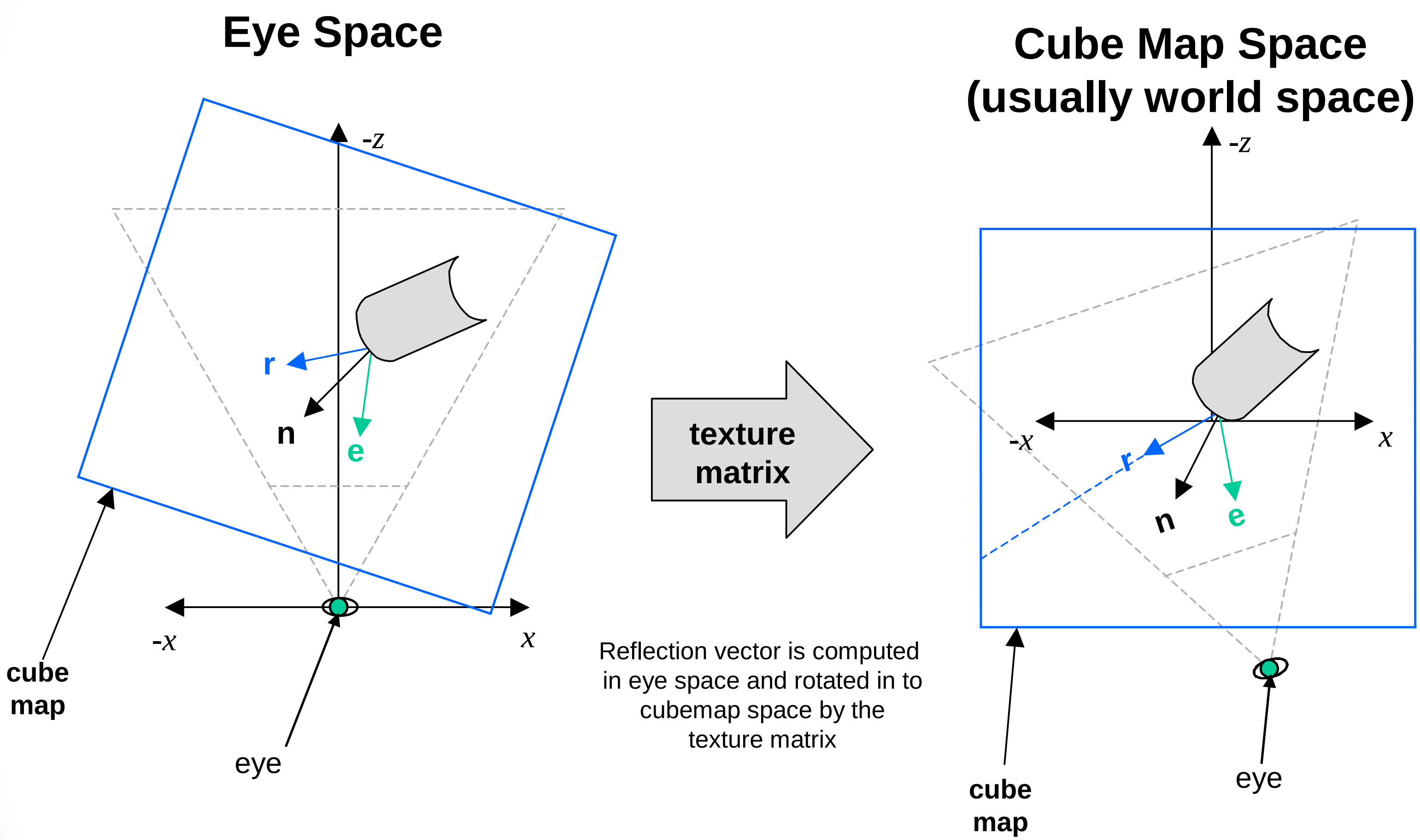
- **Normals are transformed into eye-space**
- **"u" vector is the normalized eye-space vertex position**
- **Reflection vector is calculated in eye-space as**

$$\mathbf{r} = \mathbf{u} - 2\mathbf{n}(\mathbf{n} \bullet \mathbf{u})$$

Note that this equation depends on **n** being unit length

- **Reflection vector is transformed into cubemap-space with the texture matrix**
  - **Since the cubemap represents the environment, cubemap-space is typically the same as world-space**
  - **OpenGL does not have an explicit world-space, but the application usually does**

Per-Vertex Reflection Mapping Diagram



Reflection vector is computed in eye space and rotated in to cubemap space by the texture matrix

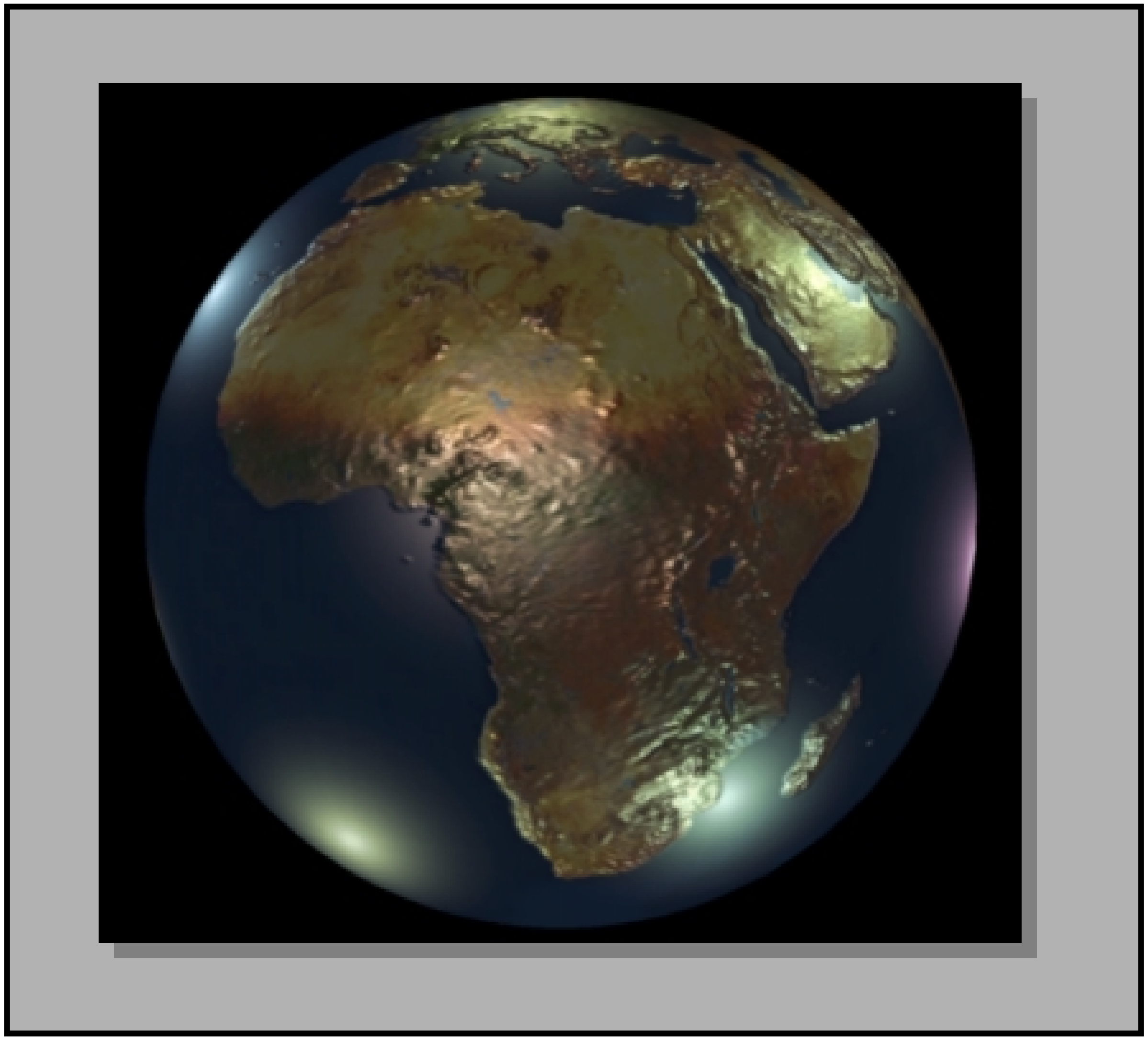
Bump Mapping and Reflection Mapping

- **Bump mapping and (per-vertex) reflection mapping don't look right together**
  - **Reflection Mapping is a form of specular lighting**
  - **Would be like combining per-vertex specular with per-pixel diffuse**
  - **Looks like a bumpy surface with a smooth enamel gloss coat**
- **Really need per-fragment reflection mapping**
  - **Doing it right requires a <u>lot</u> of high-precision per-fragment math!**

Pseudo-Reflective (offset) Bump Mapping

- **Correct per-fragment reflection mapping requires**
  - **Dependent texturing support**
  - **Complex (expensive) per-fragment math**
- **Solution: approximate the math**
  - **This approach is (unfortunately) called "Environment Map Bump Mapping" or EMBM**
  - **The environment map is a 2D texture, and the "bump map" texture supplies a per-fragment perturbation to the environment map**
  - **Offset texturing is implemented using the OFFSET_TEXTURE_2D texture shader operation on GeForce3**

Example



What are Offset Texture Shaders? (1)

TEX #	Texture Coords (S,T,R,Q)	Shader Operations	Texture Fetch	Bound Texture Target/Format	
0	$(S_0,T_0,R_0,Q_0) \rightarrow$	Texture 2D	$\left(\frac{S_0}{Q_0}, \frac{T_0}{Q_0}\right)$	2D DSDT; (ds, dt)	$\rightarrow (0,0,0,0)$
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	$\rightarrow R_1G_1B_1A_1$

k_0, k_1, k_2 and k_3 define a ***constant*** 2x2 "offset matrix" set by glTexEnv

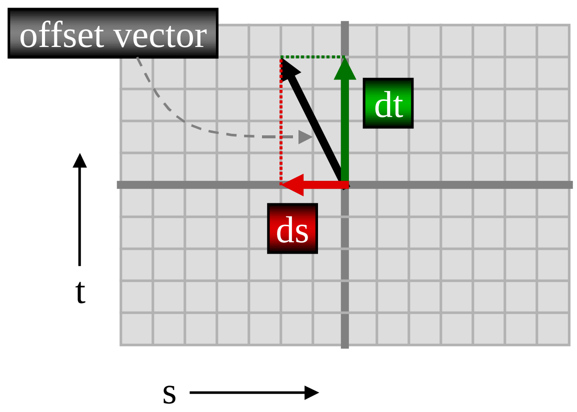
What are Offset Texture Shaders? (2)

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 “offset matrix” set by glTexEnv

DSDT Texture Format

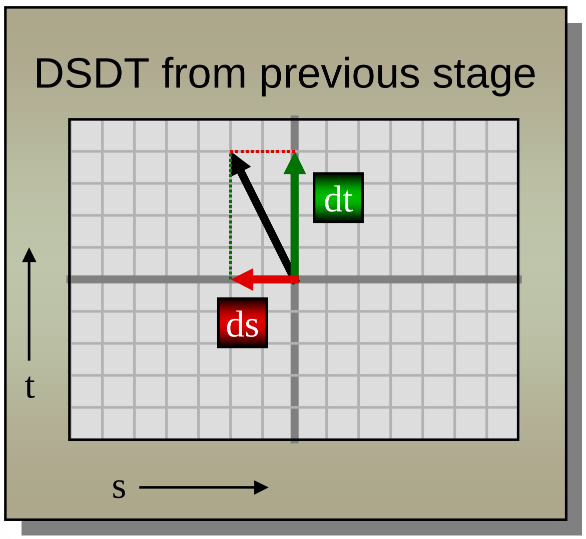
- **GL_DSDT_NV**
  - **This format encodes an offset vector in texture space**
  - ds **and** dt **are mapped to the range [-1,1]**



The constant 2x2 texel transform matrix

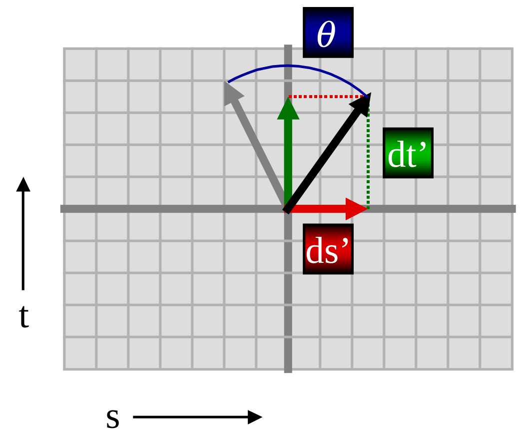
- **Per-stage 2x2 matrix (O) transforms the [ds,dt]^t before biasing the incoming (s,t)**

$$\begin{bmatrix} ds' \\ dt' \end{bmatrix} = \mathbf{O}\begin{bmatrix} ds \\ dt \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} ds \\ dt \end{bmatrix}$$



$$\mathbf{O} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

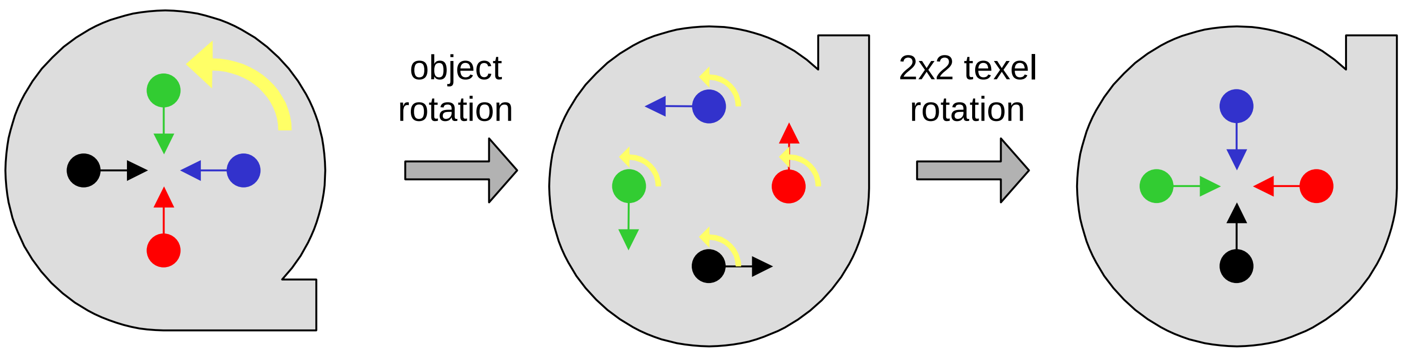
Offset texture 2D matrix



- **Offset texture 2D matrix should also include scaling since ds and dt are low precision**

Why the 2x2 matrix?

- When texels have <u>spatial meaning</u> the orientation of the surface matters!



dimple effect

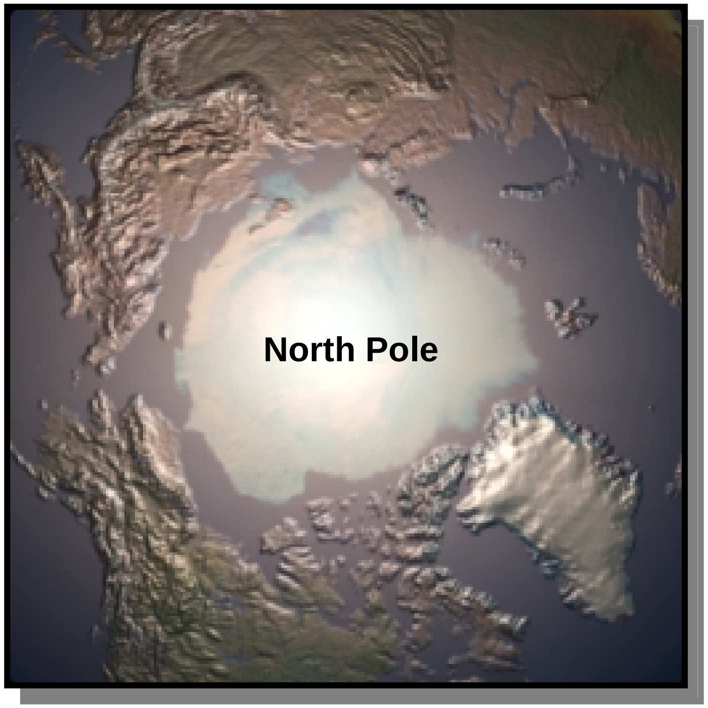
object rotation changes dimple to vortical distortion!

object rotation with dimple effect requires 2x2 transform per-texel

Limitations of the constant 2x2 matrix

- **The constant 2x2 matrix limits the usefulness of this technique to flat objects.**

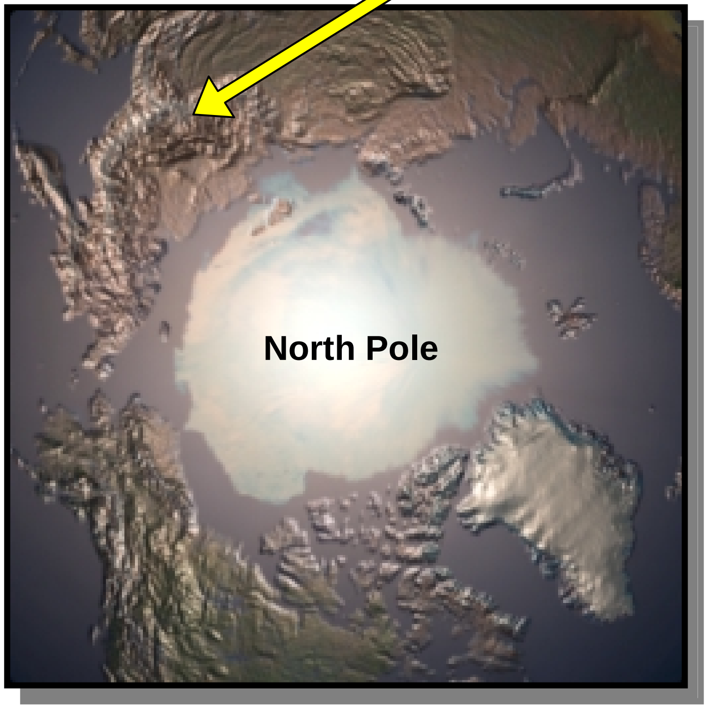
Correct



Per-Vertex 2x2 Texel Matrix
(using DOT_PRODUCT_TEXTURE_2D)

Incorrect

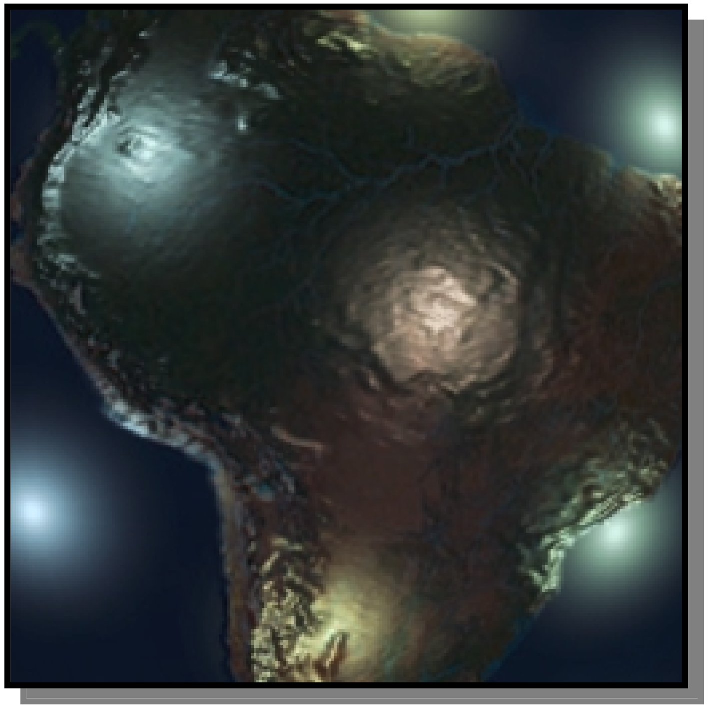
Mountains poke in!



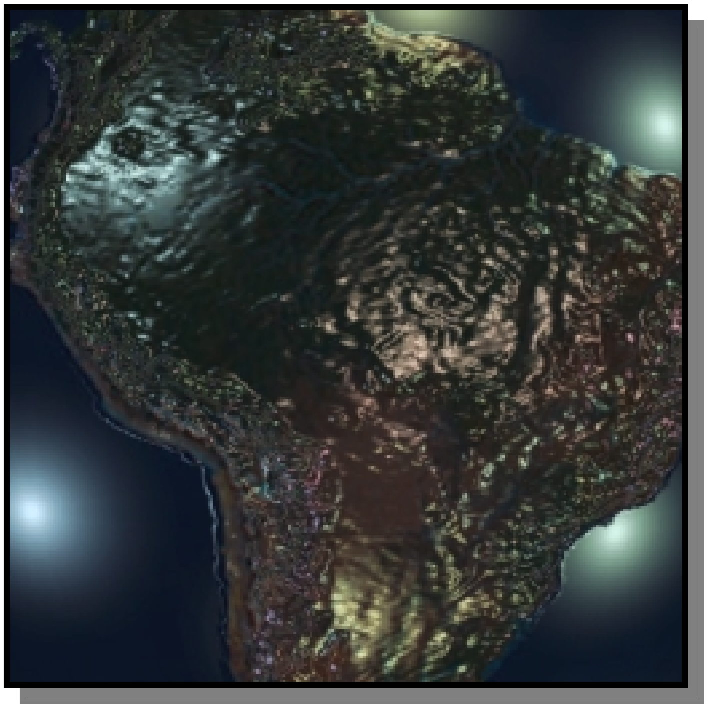
Constant 2x2 Texel Matrix
(using OFFSET_TEXTURE_2D)

Other Limitations of Pseudo-Reflective Bump Mapping

- It simply applies a per-fragment perturbation to a 2D reflection map lookup
- If perturbation is too great, weird results…



normal bump scale



large bump scale

True Reflective Bump Mapping

- **True reflective bump mapping solves the shortcomings of offset bump mapping by evaluating the reflection equation per-fragment**
  - **More complicated than you might think…**
    - **Must transform normals into cubemap space per-fragment (3x3 texel matrix)**
    - **Must interpolate cubemap space eye vector**
  - **Per-fragment reflection vector looked up into cube map**

Example

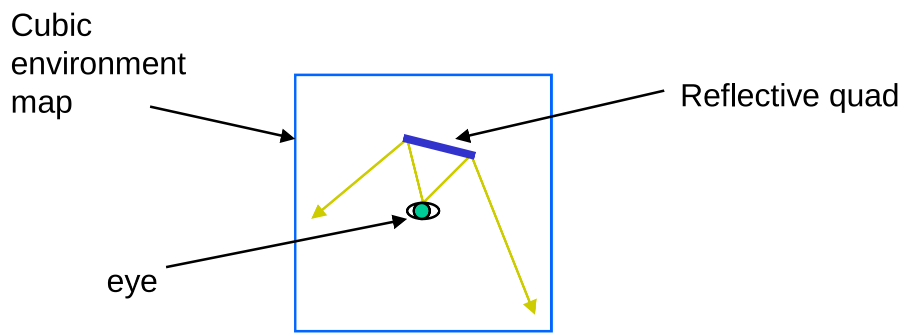
Basic shader configuration

- This is the standard configuration for reflective bump mapping with NV_texture_shader
- The normal map can be HILO or RGB

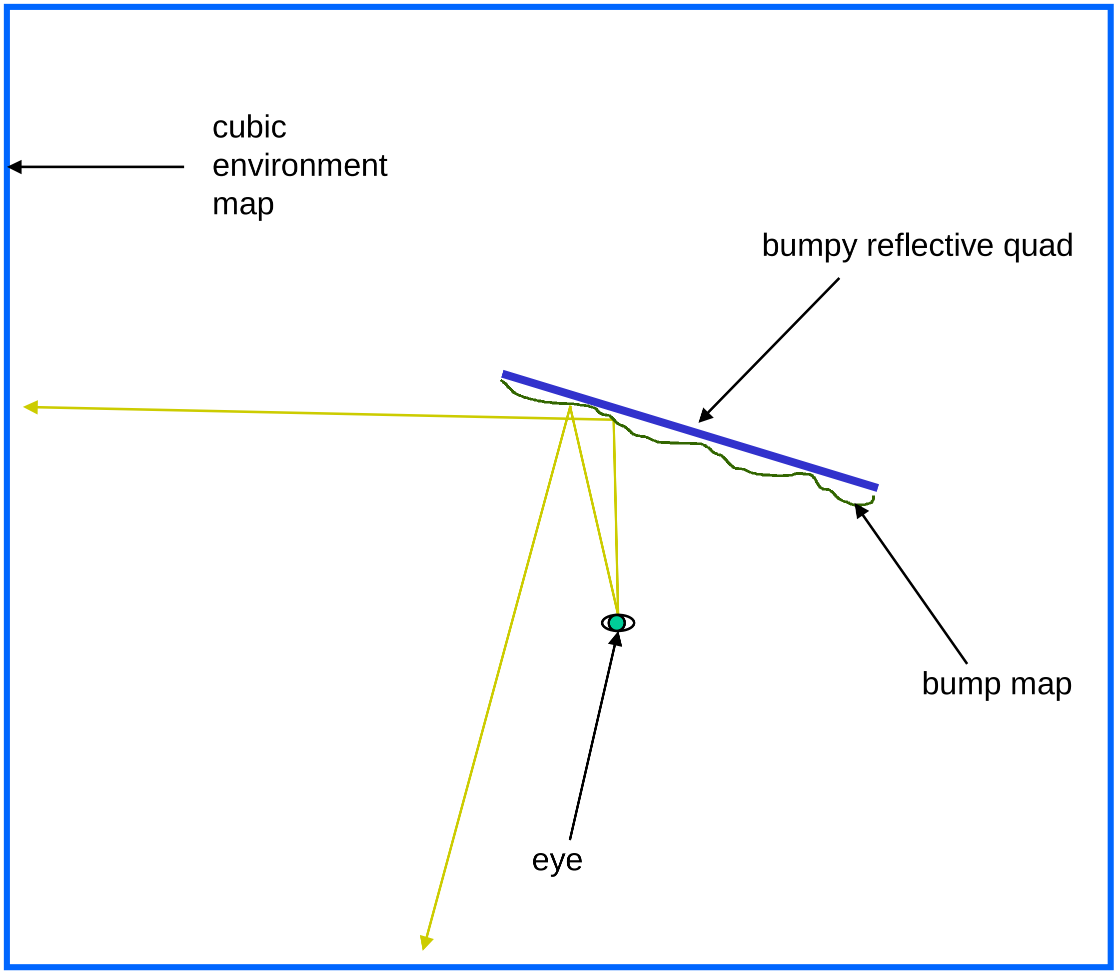
  - -stage0: TEXTURE_2D
    - texture image is normal map
  - -stage1: DOT_PRODUCT
    - no texture image
  - -stage2: DOT_PRODUCT
    - no texture image
  - -stage3: DOT_PRODUCT_REFLECT_CUBE_MAP
    - texture image is cubic environment map

Object-Space Reflective Bump Mapping

- **The `dot_product_reflect` demo renders a single bumpy, reflective quad**
  - **Normal map defined in *object-space***
  - **Cubic environment map space is same as eye-space in this example**
  - **Reflection vector is calculated per-pixel**



Reflective Bump Mapping



Rendering

- The normal vector and eye vector must be transformed into cubemap-space (which is the same as eye space in this example)
  - Normal vector is multiplied by the upper 3x3 of the inverse transpose of the MODELVIEW matrix, the same as object-space per-vertex normals are treated for per-vertex lighting in OpenGL
  - The eye vector is calculated per-vertex, and because the eye is defined to be at (0,0,0) in eye-space, it is simply the negative of the eye-space vertex position

Rendering (2)

- Given the normal vector (**n'**) and the eye vector (**e**) both defined in cubemap-space, the reflection vector (**r**) is calculated as

$$\mathbf{r} = \frac{2\mathbf{n}'(\mathbf{n}' \bullet \mathbf{e})}{(\mathbf{n}' \bullet \mathbf{n}')} - \mathbf{e}$$

- The reflection vector is used to look into a cubic environment map
- This is the same as per-vertex cubic environment mapping except that the reflection calculation ***must*** happen in cubemap-space

Details (for `dot_product_reflect`)

- **The per-vertex data is passed in as the texture coordinates of texture shader stages 1, 2, and 3**
  - **The upper-left 3x3 of the inverse transpose of the modelview matrix (M^{-T}) is passed in the s, t, and r coordinates**
    - **note: $M^{-T} \equiv M$ for rotation-only matrices**
  - **The (unnormalized) eye vector (e_x, e_y, e_z) is specified per-vertex in the q coordinates**

$$(s_1, t_1, r_1, q_1) = (M^{-T}_{00}, M^{-T}_{01}, M^{-T}_{02}, e_x)$$
$$(s_2, t_2, r_2, q_2) = (M^{-T}_{10}, M^{-T}_{11}, M^{-T}_{12}, e_y)$$
$$(s_3, t_3, r_3, q_3) = (M^{-T}_{20}, M^{-T}_{21}, M^{-T}_{22}, e_z)$$

“True Reflective Bump Mapping”?

- **Unlike the “EMBM” technique, this method performs <u>real</u> 3D vector calculations <u>per-pixel</u>!**
  - **Calculations:**
    - **Transform of the normal map normal ($\mathbf{n}$) by the texel matrix ($\mathbf{T}$) to yield ($\mathbf{n}$’)**

$$\mathbf{n}' = \mathbf{T}\mathbf{n}$$

    - **Evaluation of the reflection equation using $\mathbf{n}$’ and $\mathbf{e}$**

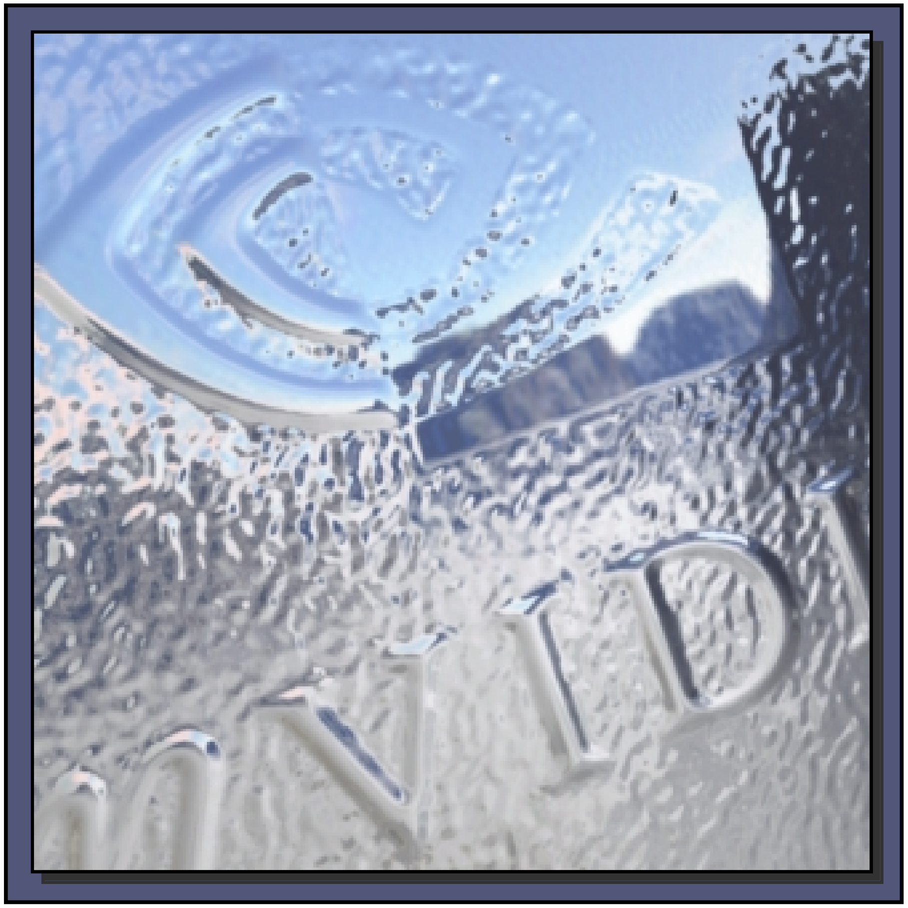
$$\mathbf{r} = \frac{2\mathbf{n}'(\mathbf{n}' \bullet \mathbf{e})}{(\mathbf{n}' \bullet \mathbf{n}')} - \mathbf{e}$$

Note that this equation does not require $\mathbf{n}$’ to be normalized

- **The resulting 3D reflection vector is looked up into a cubic environment map**
- **This <u>IS</u> true reflective bump mapping**

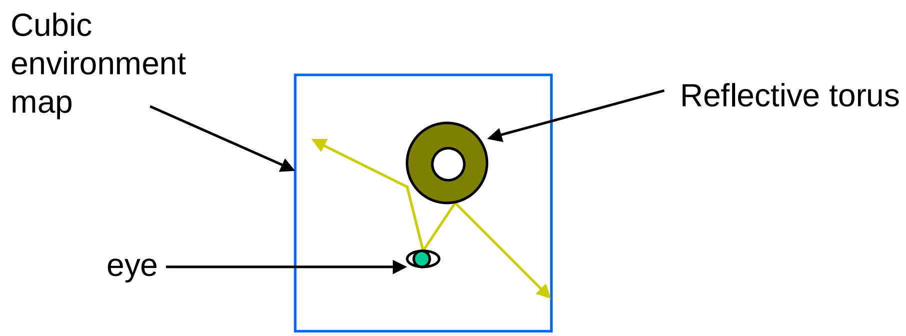
`dot_product_reflect` Results

- **A screen shot from the running demo**

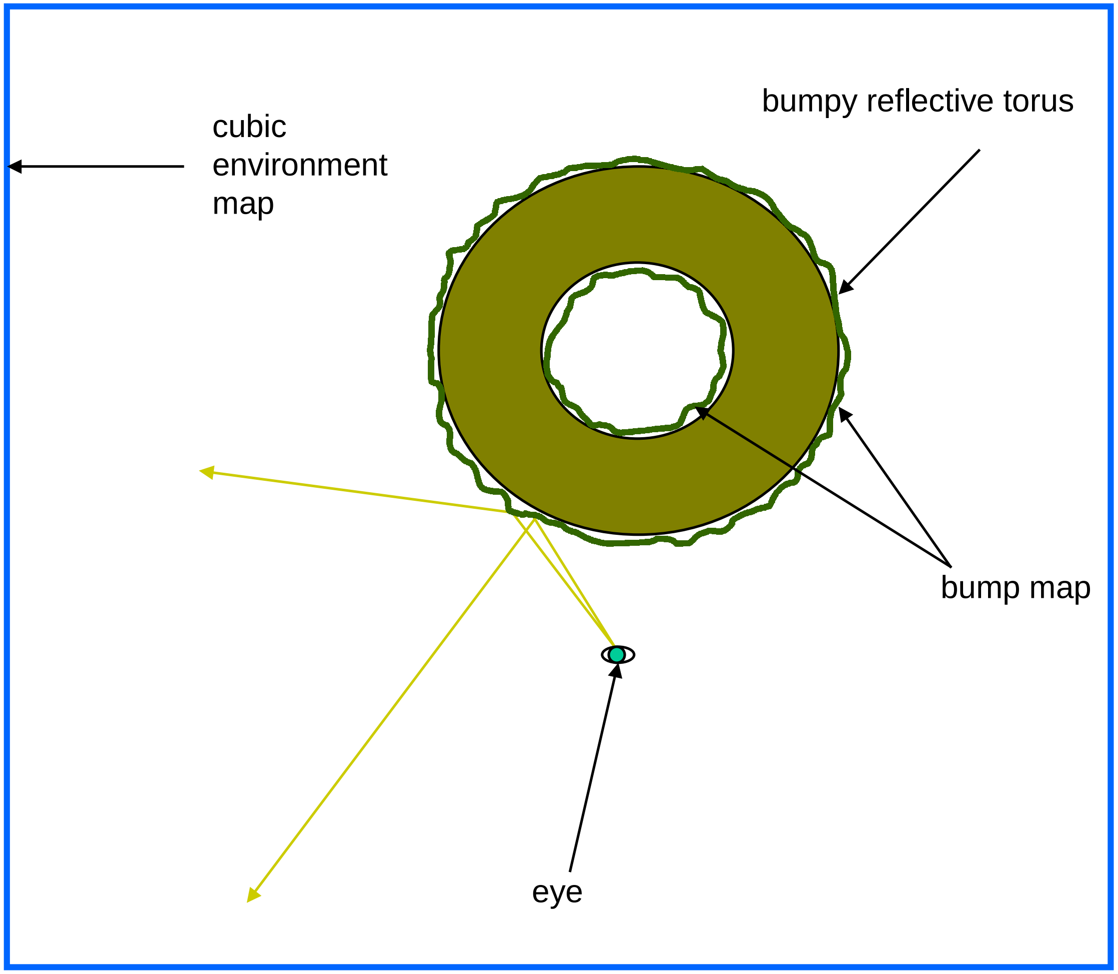


Tangent-Space Reflective Bump Mapping

- **The `dot_product_reflect_torus` demo renders a bumpy, reflective torus**
  - **Normal map defined in *tangent-space***
  - **Cubemap-space is same as eye-space**
  - **Reflection vector is calculated per-pixel**



Reflective Bump Mapping
(in `dot_product_reflect_torus`)



Rendering

- The texture coordinates are the same in this example as in `dot_product_reflect`, with the notable exception that the surface-local transform ($\mathbf{S}$) must also be applied to the normals in the normal map
  - Normal vector is multiplied by the product of the upper-left 3x3 of the inverse transpose of the MODELVIEW matrix ($\mathbf{M}^{-T}$) and the matrix ($\mathbf{S}$) whose columns are the tangent, binormal, and normal surface-local basis vectors

Rendering (2)

- **The texel matrix ($\mathbf{T}$) is defined as the product of the upper-left 3x3 of the inverse transpose of the modelview matrix ($\mathbf{M}^{-\mathrm{T}}$) and the surface-local-space to object-space matrix ($\mathbf{S}$)**

$$\mathbf{T} = \mathbf{M}^{-\mathrm{T}}\mathbf{S} = \begin{bmatrix} M_{00}^{-\mathrm{T}} & M_{01}^{-\mathrm{T}} & M_{02}^{-\mathrm{T}} \\ M_{10}^{-\mathrm{T}} & M_{11}^{-\mathrm{T}} & M_{12}^{-\mathrm{T}} \\ M_{20}^{-\mathrm{T}} & M_{21}^{-\mathrm{T}} & M_{22}^{-\mathrm{T}} \end{bmatrix} \begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix}$$

`dot_product_reflect_torus` Details

- **The texel matrix ($\mathbf{T}$) and eye vector ($\mathbf{e}$) are specified in the texture coordinates of stages 1, 2, and 3**

$$(s_1, t_1, r_1, q_1) = (T_{00}, T_{01}, T_{02}, e_x)$$
$$(s_2, t_2, r_2, q_2) = (T_{10}, T_{11}, T_{12}, e_y)$$
$$(s_3, t_3, r_3, q_3) = (T_{20}, T_{21}, T_{22}, e_z)$$

`dot_product_reflect_torus` Results

- **A screen shot from the running demo**

Related Information

- See the `bumpy_shiny_patch` presentation and demo for
  - Using NV_evaluators
  - Tangent-space reflective bump mapping
  - NV_vertex_program for performing setup

Questions, comments, feedback

- Cass Everitt, cass@nvidia.com
- www.nvidia.com/developer