NVIDIA®

Vertex Program 1.1
Texture Shader 3

Hansong Zhang

Vertex Program 1.1 New Instructions

- SUB and ABS
  - Use to be accomplished as ADD and MAX
  - No new functionality – just usability enhancement
- DPH: Homogeneous Dot Product
  - DPH vout, v0, v1
  - Four component dot product
  - v0's w component is assumed to be 1.0
  - Result register contains replicated scalar result

Vertex Program 1.1 New Instructions

- RCC: Reciprocal Clamped
  - Same as RCP, but clamped
  - Infinity clamped to a really large number
  - -Infinity clamped to a really small number

Vertex Program 1.1 Options

- Vertex Program 1.1 Declaration
  - !!VP1.1 <options> <instructions> END
- Options
  - New for vertex program 1.1
  - Currently only one: NV_position_invariant

Position Invariance

- Why is it important?
  - In multi-pass rendering, different passes may use either the fixed function pipeline or vertex programs
  - To avoid Z fighting, positions output by the vertex program and the fixed function pipe should be identical
- By default, vertex programs are *not* guaranteed to be position-invariant
  - A vertex program can compute its homogeneous position (HPOS) in whatever way it wants
  - Can be different from how the OpenGL fixed function pipeline does it

Position-Invariant Vertex Programs

- By using NV_position_invariant, homogeneous positions are guaranteed to be identical
  - However, the vertex program no long outputs HPOS
    - HPOS is not an valid result register when the option's specified
  - It will be computed by the OpenGL implementation
  - Limitations: there can be no vertex blending, where HPOS must be computed in a custom way

Texture Shader 3 Overview

- GL_NV_texture_shader3
- New texture shader operations
  - 14 of them
- New HILO texture formats
  - HILO8_NV
  - SIGNED_HILO8_NV
- Unsigned RGBA dot product mapping mode
- Blue-to-one

New Texture Shader Operations

- New dependent texture operations (3)
  - HILO, 3D, and cube map dependent operations
- New offset texture operations (8)
  - Projective, HILO variants
- New dot product texture operations (2)
  - Dot product texture 1D
  - Dot product texture pass through
- New depth replace (1)
  - Affine (not projective)

Dependent Textures

- A previous stage looks up a texture
- Current stage interpret previous texture lookup result as texture coordinates
- Current stage use such coordinates to access its texture

New Dependent Texture Operations

- Texture shader 2 dependent texture functionality
  - Use 8-bit RGBA as dependent textures
    - Either AR or GB as texture coordinates
    - Inadequate precision for addressing large textures
  - Current stage can only access 2D textures
    - No 3D textures or cube maps
- Thus, new functionality:
  - Support HILO format as texture coordinate sources, 16-bit precision
  - Support for dependent 3D and cube map textures

HILO as Source of Texture Coordinates

- HILO textures as texture coordinate source
  - Previous stage looks up a HILO texture
  - Current stage set SHADER_OPERATION to GL_DEPENDENT_HILO_TEXTURE_2D_NV

Dependent 3-D and Cube-Map Textures

- Dependent 3-D Texturing
  - Previous stage provides RGB
  - Current stage takes RGB as 3-D texture coordinates and index into a 3-D texture
    - GL_DEPENDENT_RGB_TEXTURE_3D_NV
- Dependent cube-map texturing
  - Previous stage provides RGB
  - Current stage interpret RGB as cube-map texture coordinates and index into a cube-map
    - NV_DEPENDENT_RGB_TEXTURE_CUBE_MAP_NV

Offset Textures

- Previous stage looks up a texture
- Current stage treats the look-up result, optionally transformed by a matrix, as texture coordinate offset
- Current stage offsets input texture coordinates accordingly, and access its texture

New Offset Texture Operations

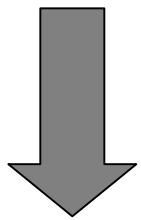
- Texture shader 2 offset texture operations
  - Offset texture 2D:
    - s' = s + a1 * ds + a3 * dt,

      t' = t + a2 * ds + a4 * dt
  - Offset texture 2D with scale & bias
    - First do offset texture 2D, then scale & bias
  - Previous Limitations
    - Non-projective
    - ds, dt have only 8-bit signed precision
- New functionality
  - Projection before offset
  - HILO 16-bit as offsets

Projection before Offset

- Divide by q before applying the offset

$$s' = s + a1 * \underline{ds} + a3 * \underline{dt},\ t' = t + a2 * \underline{ds} + a4 * \underline{dt}$$



$$s' = s/q + a1 * \underline{ds} + a3 * \underline{dt},\ t' = t/q + a2 * \underline{ds} + a4 * \underline{dt}$$

- Add "PROJECTIVE" in the token
  - All offset texture modes have a projective version, e.g. GL_OFFSET_PROJECTIVE_TEXTURE_2D_NV

HILO Offset Texture

- Goal: higher precision
  - Use 16-bit HILO or the new HILO8 texture format
- GL_OFFSET_HILO_TEXTURE_2D_NV
- ds = *hi*, dt = *lo*, therefore

  s’ = s + a1 * *hi* + a3 * *lo*, t’ = t + a2 * *hi* + a4 * *lo*

New HILO8 Texture Formats

- New texture format: HILO8
  - Two 8-bit channels (hi and lo) with 16-bit filtering
- GL_HILO8_NV (signed)
  - Both hi and lo are [-1,1]
  - Useful for encoding normals with high precision
  - The full normal is $\left(HI, LO, \sqrt{\max\left(0, 1 - HI^2 - LO^2\right)}\right)$
- GL_UNSIGNED_HILO8_NV
  - Both components are [0,1]
  - Useful for encoding 32-bit values, like depth
  - Third channel is set to 1

HILO8 Advantages

- Storage is 8-bit but filtering for each component done in 16-bits
  - Often HILO8 is almost indistinguishable from HILO16
- Saves texture memory and (more importantly) reduces texture cache footprint

New Dot Product Operations

- Current limitations
  - No support for 1-D textures using dot product result
  - Dot product result is not output as a stage's RGBA
- New operation
  - GL_DOT_PRODUCT_TEXTURE_1D_NV
  - GL_DOT_PRODUCT_PASS_THROUGH_NV

Dot Product Texture 1D

- 1D table lookup based on a single dot product

  s = (s, t, r) * N
- Example use: access a 1-D texture map encoding illumination information, with L • N or H • N
  - L and H are passed in per-vertex and interpolated
  - N is usually from a normal map
  - The result: custom per-pixel diffuse and specular controlled by two 1-D textures

Dot Product Pass-Through

- GL_DOT_PRODUCT_PASS_THROUGH_NV
- Smears dot-product to RGBA
- Requires no texture access
- Later stages can use the RGBA anyway they want

New Depth Replace Operation

- Texture shader 2 depth replace
  - Depth replace is always projective making it more expensive than necessary when projection is not required
  - Uses two texture shader stages for two dot products
- New operation
  - Dot product affine depth replace
    - Removes Z/W projection
    - Uses one stage

Affine Depth Replace

- In projective depth replace, depth = dotP / dotC
- In affine depth replace, depth = dotC
  - GL_DOT_PRODUCT_AFFINE_DEPTH_REPLACE_NV
  - Uses just one stage (still references a previous stage)
  - Set texture coordinates to

    (Zscale, Zscale/2^16, Zbias)
  - dotC = (Zscale, Zscale/2^16, Zbias) • (hi, lo, 1), where (hi, lo) is from the previous stage texture lookup

New Dot Product Mapping Mode

- Defines how RGB is interpreted in dot products
- Previous modes
  - GL_UNSIGNED_IDENTITY_NV
    - [s, t, r] • [R, G, B]
  - GL_EXPAND_NORMAL_NV
    - [s, t, r] • [2×R-1, 2×G-1, 2×B-1]
- New mode: GL_FORCE_BLUE_TO_ONE_NV
  - [s, t, r] • [R, G, 1]
  - Blue and alpha still available in combiners without influencing the dot product

Questions?

- **Hzhang@nvidia.com**