



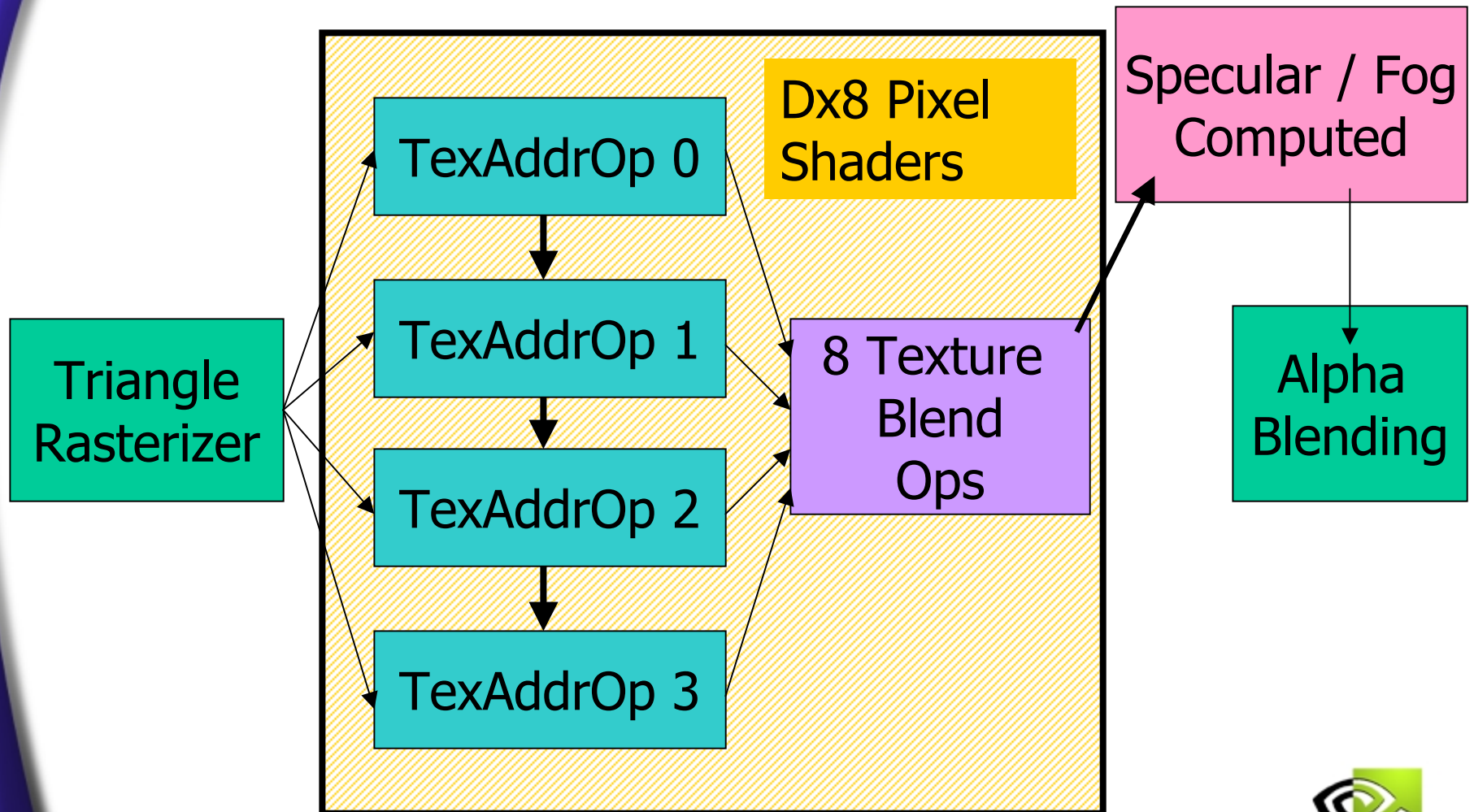
*n*VIDIA™

DX8 Pixel Shader Detail

Sim Dietrich

sim.dietrich@nvidia.com

DX8 Pixel Shading Pipeline



nVIDIA™

Dx8 Pixel Shaders

- A Pixel Shader is a byte stream of instructions compiled from a text file
- You can compile the pixel shader at runtime during development, and then change to pre-compiled byte-streams for release mode
- Assemble a Pixel shader like so :
- `D3DXAssembleShader(strParsedProgram.c_str(), // source
strParsedProgram.size() - 1, //size
0, // no flags
&pConstants, // constant floats
pCode, // where to put code
&pCompileErrors); // get errors`



NVIDIA

Dx8 Pixel Shaders

- Now ask D3D for a handle to the compiled pixel shader :

```
pD3DDev->CreatePixelShader( pCode→GetBufferPointer(),  
                           &m_dwMyHandle );
```

- Now select this pixel shader program like so:
- ```
pD3DDev->SetPixelShader(m_dwMyHandle);
```



NVIDIA™

# DX8 Pixel Shaders

---

- **Three types of Instructions**
  - **Constant Definitions**
    - **Similar to Setting the TFACTOR**
  - **Texture Address Ops**
    - **Fetching texels**
  - **Texture Blending Ops**
    - **Combining texels, constant colors and iterated colors to produce SrcColor and SrcAlpha**



NVIDIA™

# Setting Constants

---

**def c#, x, y, z, w**

**Sets the Constant, from 0 to 7 with the appropriate floating point value , which will be clamped to be between 0.0f and 1.0f :**

**def c0, 1.0f, 4.0f, -10.0f, 1.0f,**

**Note that not all Constants are visible to all instructions**

**Instructions 0-1 reference Constants 0-1**

**Instructions 2-3 reference Constants 2-3, etc.**



**nVIDIA**

# Texture Address Ops

---

- Each Texture Address Op represents the use of a particular set of texture coordinates
- Either :
  - Look up a filtered texel color
  - Use as a vector
  - Use as the part of a matrix



nVIDIA™

# Texture Address Ops

---

- **tex t0**
  - Just fetch a filtered texel color
- **texbem tDest, tSrc0**
  - Bump Environment Map
    - $U += 2 \times 2 \text{ matrix}(dU)$
    - $V += 2 \times 2 \text{ matrix}(dV)$
    - Then Sample at ( U, V )
- **Texbem1 tDest, tSrc0**
  - Bump Environment Map w/ Luminance
    - $U += 2 \times 2 \text{ matrix}(dU)$
    - $V += 2 \times 2 \text{ matrix}(dV)$
    - Then Sample at ( U, V ) & Apply Luminance



NVIDIA



# Texture Address Ops

---

- **texcoord tDest**
  - Just turn the texture coordinate into a color
- **texkill tDest**
  - Kill any texels where at least one of s,t,r,q is  $< 0$
- **Texm3x2pad t1, t0**
  - “padding” instruction as part of the texm3x2tex instruction – performs a dot product of t0’s color with these texture coordinates
- **Texm3x2tex t2, t0**
  - Take previous dot product from “pad” instruction as the S coordinate
  - Perform dot product of t0’s color with this texture coordinate and use as T
  - Sample from a 2D texture using ( S, T )



NVIDIA

# Texture Address Ops

---

- **texreg2ar tDest, tSrc**
  - Sample from ( tSrc.A, tSrc.R )
- **texreg2gb tDest, tSrc**
  - Sample from ( tSrc.G, tSrc.B )
- These are the general dependent texture read operations
- They take part of a color from the tSrc texture to use as S,T coordinates of the tDest texture fetch



NVIDIA

## 3x3 Texture Address Ops

---

- **Texm3x3pad**
  - Padding for 3x3 matrix operation
  - Uses the 3D texture coordinate as a row of the matrix
- **Texm3x3spec**
  - Compute Non-Local Viewer Specular reflection about Normal from Normal Map
    - `tex t0` ; Normal Map
    - `texm3x3pad t1, t0` ; 1<sup>st</sup> row of matrix
    - `texm3x3pad t2, t0` ; 2<sup>nd</sup> row of matrix
    - `texm3x3spec t3, t0, c0` ; 3<sup>rd</sup> row, reflect & sample
    - `mov r0, t3`



NVIDIA

# Local Viewer Reflection

---

- **Texm3x3vspec**
  - **Compute Local Viewer Specular reflection about Normal from Normal Map**
  - **Eye vector comes from q coordinates of the 3 sets of 4D textures**
    - **tex t0 ; Normal Map**
    - **texm3x3pad t1, t0 ; 1<sup>st</sup> matrix row, x of eyevector**
    - **texm3x3pad t2, t0 ; 2<sup>nd</sup> matrix row, y of eyevector**
    - **texm3x3spec t3, t0, c0 ; 3<sup>rd</sup> row & eye z, reflect & sample**
    - **mov r0, t3**



NVIDIA<sup>™</sup>

# 3x3 Per-Pixel Vector Rotation

---

- **texm3x3mat**
  - Rotate vector through 3x3 matrix, then sample a CubeMap or 3D texture
    - `tex t0` ; Normal Map
    - `texm3x3pad t1, t0` ; 1<sup>st</sup> matrix row
    - `texm3x3pad t2, t0` ; 2<sup>nd</sup> matrix row
    - `texm3x3mat t3, t0, c0` ; 3<sup>rd</sup> matrix row & sample
    - `mov r0, t3`



NVIDIA™

# Texture Blending Ops

---

- After all Texture Address Ops, you can have up to 8 texture blending instruction slots
- Each slot can hold a color and an alpha operation to be executed simultaneously
- These are analogous to the old TextureStageState COLOROP and ALPHAOPs
- You must add your own specular if using the Pixel Shader pipeline



NVIDIA<sup>™</sup>

## Texture Blending Ops

---

**add    dest, src1, src2**  
**dest = src1 + src2**

**sub dest, src1, src2**  
**dest = src1 – src2**

**dp3    dest, src1, src2**  
**dest = ( src1.x \* src2.x + src1.y \* src2.y ...)**

**lrp    dest, factor, src1, src2**

**dest = (factor)src1 + ( 1-factor)src2**



*n*VIDIA™

# Texture Blending Ops

---

**mul dest, src0, src1**  
**dest = src0 \* src1**

**mad dest, src0, src1, src2**  
**dest = ( src0 + src1 \* src2 )**

**mov dest, src**  
**dest = src**

**cnd dest, r0.a, src1, src2**  
**if ( r0.a > 0.5 ) { dest = src1; }**  
**else { dest = src2; }**



*NVIDIA*™



# Argument Modifiers

---

- **Alpha Replicate**
  - $r0.a$
- **Invert**
  - $1 - r0$
- **Negate**
  - $-r0$
- **Bias – subtract 0.5**
  - $r0\_bias$  ;
- **Signed Scale –  $2 * (x - 0.5f)$** 
  - $r1+bx2$



NVIDIA™

# Instruction Modifiers

---

**\_x2    // double result**

**\_x4    // quadruple result**

**\_d2    // halve result**

**\_sat   // clamp  $< 0$  to 0 and  $< 1$  to 1**

**You can use \_sat together with scaling :**

**For instance :**

**add\_bx2\_sat r0, r1, t2**

**Common Example :**

**dp3\_sat r1, r0\_bx2, t0\_bx2**



**nVIDIA**

## Example Pixel Shader

---

- **ps.1.0** ; **DirectX8 Version**
- **tex t0** ; **sample normal map**
- **texm3x2pad t1, t0\_bx2** ; **N dot L**
- **texm3x2tex t2, t0\_bx2** ; **N dot H and sample**
- **add r0, t2, c0** ; **add in ambient**
- **mov r0.a, t0.a** ; **normal map alpha into r0**



**nVIDIA**™