ARB Fragment Program

Fragment level programmability in OpenGL

Evan Hart
ehart@ati.com

Plan of Action

- What is it
- How does it work
- What is it good for

Introduction to ARB FP

- New standardized programming model
- What it replaces
- What it does not replace

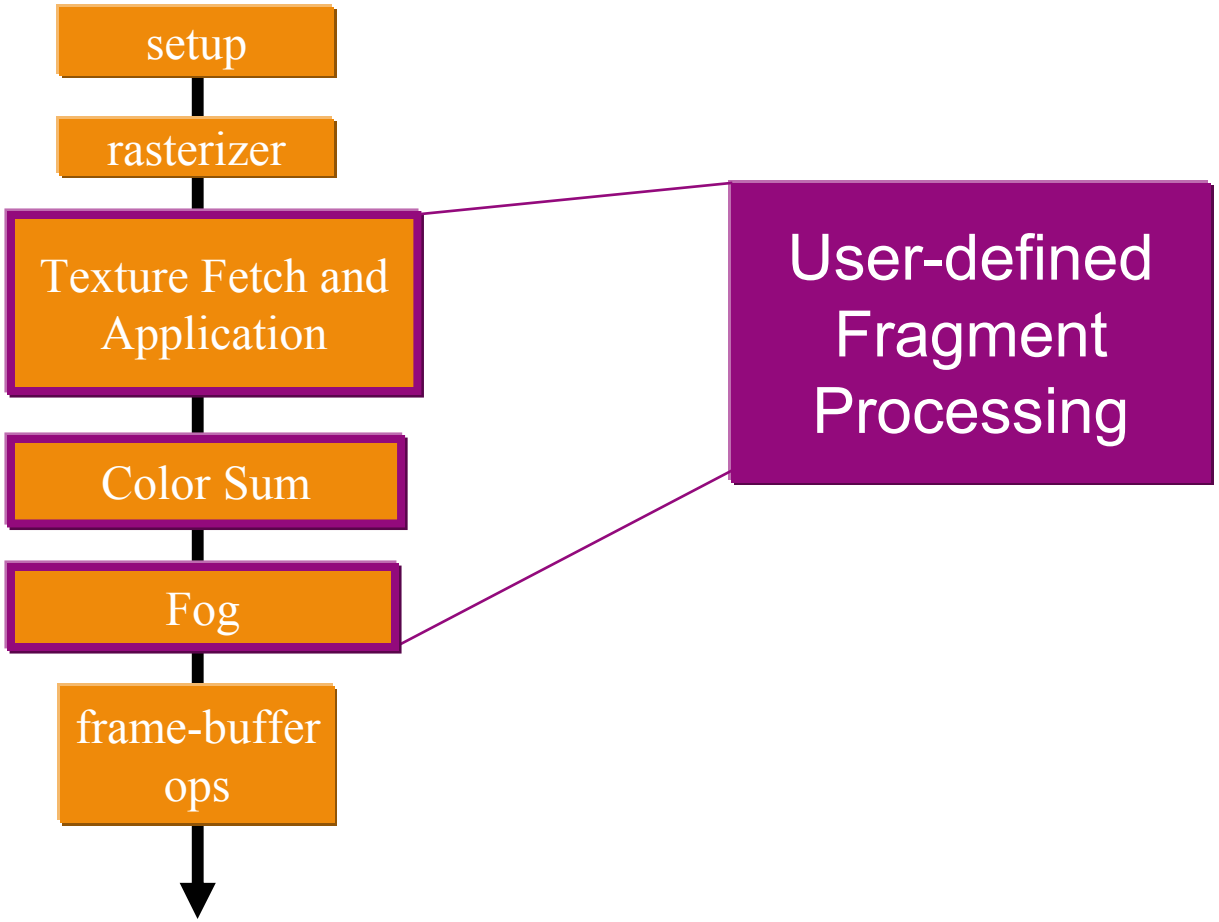
Standardized Fragment Programmability

- ARB standard for fragment-level programmability
- Derived from ARB_vertex_program

Replaces Multitexture Pipe

- **Replaces**
  - **Texture blending**
  - **Color Sum**
  - **Fog**
- **Subsumes**
  - **texture enables**
  - **texture target priorities**

Fragment Processing Pipe



Preserves Backend Operations

- Coverage application
- Alpha Test
- Stencil and depth test
- Blending

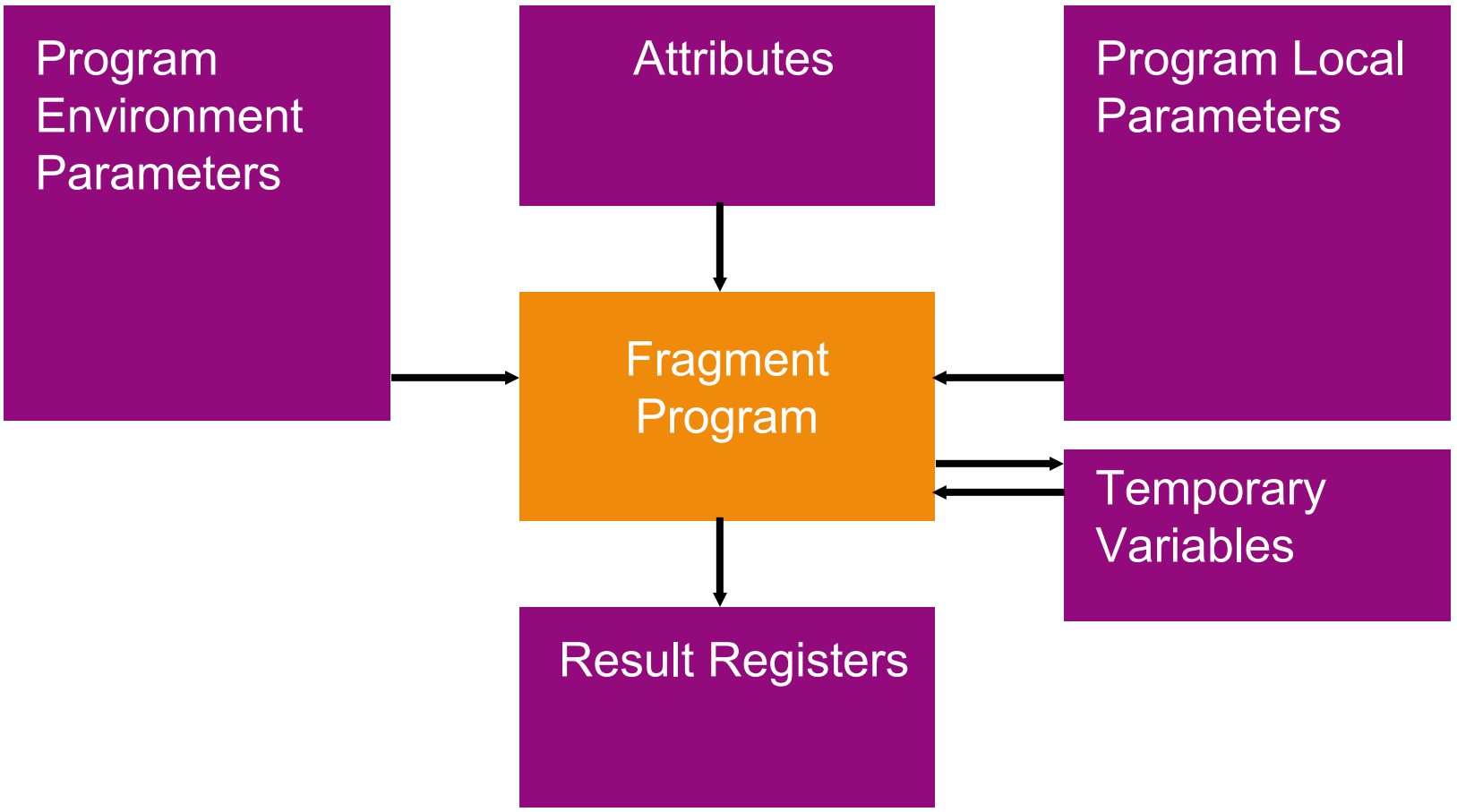
Programming Model

- ASM based similar to ARB_vertex_program
- Rich SIMD instruction set
- Requires resource management
  - Texture accesses
  - Interpolators
  - Temporaries
  - Constants

Similar to ARB_vertex_program

- Assembly syntax
- Same basic grammar rules
  - Ops are upper case
  - Statements terminated by semicolons
  - Comments begin with #

Fragment Program Machine



Large Instruction Set

- Three basic categories
  - Scalar (single value) operations
  - Vector operations
  - Texture operations
- Component swizzling
- Component masking

Scalar Instructions

- COS – cosine
- EX2 – base 2 exponential
- LG2 – base 2 logarithm
- RCP – reciprocal
- RSQ – reciprocal square root
- SIN – sine
- SCS – sine and cosine
- POW – power

New Scalar Instructions

- COS – cosine
- SIN – sine
- SCS – sine and cosine

Removed Scalar Instructions

- **EXP – Partial Precision EX2**
- **LOG – Partial Precision LG2**

Vector Instructions

- Standard Arithmetic
- Special purpose vector

Standard Arithmetic Ops

- ABS – absolute value
- FLR – floor
- FRC – fraction component
- SUB – subtract
- XPD – cross product
- CMP – compare
- LRP – linearly interpolate
- MAD – multiply accumulate
- MOV – move
- ADD – add
- DP3 – three component dot product
- DP4 – four component dot product

Special Vector Ops

- LIT – compute lighting
- DPH – homogeneous dot product
- DST – compute distance vector

New Instructions

- LRP – Linearly Interpolate
- CMP - Compare

Texture Instructions

- TEX
- TXP
- TXB
- KIL

Standard Texture Fetch

```
TEX <dest>, <src>, texture[n], <type>;
```

- **Does not divide by q**

Extended Texture Fetches

- TXP
  - Same syntax as TEX
  - Divides first three components by the fourth
- TXB
  - Adds the fourth component to the computed LOD

Killing pixels

- **KIL instruction**
  - **Terminates shader program if any component is less than 0**

Resource Categories in ARB FP

- Temporaries
- Textures
- Attributes
- Parameters
- Instructions
  - Texture Instructions
  - Arithmetic Instructions

Identifying Limits

- **Standard Resource limits**
  - **Number of temps etc**
- **Texture Indirections**

Using Fragment Program

- API
- Simple shaders
- Complex shaders

Simple API

- Loading programs
- Setting Parameters
- Making active

Shared API

```
glGenProgramsARB( num, id );

glBindProgramARB( GL_FRAGMENT_PROGRAM_ARB,
  id );

glProgramStringARB(
  GL_FRAGMENT_PROGRAM_ARB,
  GL_PROGRAM_FORMAT_ASCII_ARB, length,
  string );
```

Simple shaders

- **Pre-programmable functionality**
  - **Texture * color**

Simple Shader Example

```
!!ARBfp1.0

TEMP temp;    #temporary
ATTRIB tex0 = fragment.texcoord[0];
ATTRIB col0 = fragment.color;

PARAM pink = { 1.0, 0.4, 0.4, 1.0};

OUTPUT out = result.color;

TEX temp, tex0, texture[0], 2D;  #Fetch texture

MOV out, temp; #replace

#MUL out, col0, temp; #modulate
#MUL out, temp, pink; #modulate with constant color
```

Complex Shaders

- **Lighting**
- **Procedural**

Phong Lighting

```
#compute half angle vector
ADD spec.rgb, view, lVec;
DP3 spec.a, spec, spec;
RSQ spec.a, spec.a;
MUL spec.rgb, spec, spec.a;

#compute specular intensisty
DP3_SAT spec.a, spec, tmp;
LG2 spec.a, spec.a;
MUL spec.a, spec.a, const.w;
EX2 spec.a, spec.a;

#compute diffuse illum
DP3_SAT dif, tmp, lVec;
ADD_SAT dif.rgb, dif, const;
```

Procedural Bricks

```
#Apply the stagger
MUL r2.w, r0.y, half;
FRC r2.w, r2.w;
SGE r2.w, half, r2.w;
MAD r0.x, r2.w, half, r0.x;

#determine whether it is brick or mortar
FRC r0.xy, r0;
SGE r2.xy, freq, r0;
SUB r3.xy, 1.0, freq;
SGE r0.xy, r3, r0;
SUB r0.xy, r2, r0;
MUL r0.w, r0.x, r0.y;
```

Caveats

- Remember the difference between
  - {3.0}
  - 3.0
- Ensure Continuity at Boundaries
  - Needed to compute LOD
- Programs under limits may not load

Wrap up

- **Fragment Program is the Standard for Fragment Programmability**
- **Assembly Language Shaders**
- **Enhances Pixel Level Effects**

Questions?

- **ehart@ati.com**