AMD HLSL

AMD HLSL (25 min)

HLSL (High Level Shading Language) - shading language developed by Microsoft and part of DirectX

AMD HLSL is a derivative of HLSL (versions 3.0 and 4.0) with extensions for ATI hardware

- scatter
- scratch buffers
- double precision

Compiler compiles AMD HLSL to AMD IL

Compiler works in both Windows and Linux

What is HLSL?

```
Texture2D tex0;
Texture2D tex1;

uniform float2 offset;

float4 main(float2 tcoord : TEXCOORD0) : COLOR0
{
    float4 a = tex2D(tex0, tcoord + offset);
    float4 b = tex2D(tex1, tcoord);

    return floor(a+b);
}
```

High Level Shading Language

Loose C++ like syntax

Pre-defined graphics centric functions (e.g. texture fetches)

Data Types

bool

float

uint

int

double

struct

Data types, continued…

Vector types
- <type>[1, 2, 3, 4]
- e.g. "float3 var0;" declares var as a vector of length 3

Uniform modifier
- uniform <type> <var>
- variables are visible outside of the shader (i.e. constants)
- e.g. "uniform float4 var1;"


```
Texture2D tex0;
Texture2D tex1;

uniform float2 offset;

float4 main(float2 tcoord : TEXCOORD0) : COLOR0
{
    float4 a = tex2D(tex0, tcoord + offset);
    float4 b = tex2D(tex1, tcoord);

    return floor(a+b);
}
```

C++ Supported Operators

. member access (structs) / swizzle operator
++ pre/post increment
-- pre/post decrement
- unary minus
+ unary plus
Casts
* multiply
+ addition
- subtraction
?: ternary operator
= assignment operator
+= increment and assign
-= decrement and assign
*= multiply and assign

More Supported Operators

/ divide

% modulus

/= divide and assign

%= mod and assign

< compare

<= compare

> compare

>= compare

== compare

!= compare

Operators supported for Array types:

[] index operator

Operators supported for Boolean, uint and integer types:

! logical negate

~ logical compliment

Operators supported for uint and integer types:

<< left shift

>> right shift

&= bitwise and and assign

^= bitwise exclusive or and assign

|= bitwise or and assign

<<= bitwise shift and assign

>>= bitwise shift and assign

What's Not Supported?

AMD Smarter Choice

:: scope

-> pointer access

* dereference

& address of

Control Flow Operations

AMD Smarter Choice

break – exit the surrounding loop (do, while, or for) or switch

continue – go to the next iteration of the surrounding loop (do, while, or for) or switch

do – repeats a block of code, do { block } while (condtion)

for – as in C for(init; condition; increment){ block}

if – as in C if(condition) { block }

switch – as in C

while – while(cond) { block)

Built In Functions

...

dot(x,y) returns the dot product of x and y. (x,y must be vectors)

exp(x) returns e^x (float only) and approximate

floor(x) returns the largest integer <= x

fmod(x,y) re turns the float reminder of x/y such that

frac(x) returns the fractional part of x (works for floats and doubles)

...

(See MS HLSL sec for more info.)

Memory/Texture Data Arrays

Full Gamut of Data Types

UINT, INT, FLOAT

Vector Types

1, 2, 3

Examples:

R32G32B32A32_FLOAT (4 32 bit float values)

R32G32B32A32_INT (4 32 bit integer values)

R32G32B32A32_TYPELESS for mixed data

Memory/Texture Fetches

AMD Smarter Choice

DX9 Style

Declaration:

Texture<1,2,3>D <name>;

Call:

tex2D(<name>, <address>);

DX10 Style

<name>.Load(<address>);

AMD Smarter Choice

```
Texture2D tex0;
Texture2D tex1;

uniform float2 offset;

float4 main(float2 tcoord : TEXCOORD0) : COLOR0
{
    float4 a = tex2D(tex0, tcoord + offset);
    float4 b = tex2D(tex1, tcoord);

    return floor(a+b);
}
```

Constant Buffers

Instead of single constants can use constant arrays

```
cbuffer name
{
  list of variable declarations;
//
<type> temp[<size>];
};
```


AMD HLSL

AMD's derivative of HLSL

Why?
- Expose ATI hardware features
- Compiler can run in both Windows and Linux

Includes everything previously discussed plus...

Double Precision

ATI Radeon HD 3000+ series exposes first double precision GPU units

What's available in AMD HLSL?

- Raw converts for doubles: convert uint2 to double

Scatter

Write data to an arbitrary address

1. Declare a global buffer

global float4 <name>[]

2. Write as array

global float4 result[];

...

result[<addr>] = <val>;

...

Bottom Line…

Programming in shader assembly isn't fun

We're used to a high level language

AMD HLSL provides Linux and Windows compatibility

For more info see the documentation