DirectX 11
Compute Shader

Chas. Boyd
Architect
Windows Desktop and Graphics Technology
Microsoft
Outline

• Compute Shader Objectives
  – DirectX
  – Data-Parallel Processing
  – Target Applications
  – Design

• Compute Shader Details
  – Syntax and Features
DirectX

- DirectX API set shipping since 1995
- Direct3D is popular graphics API for PCs
- Historically for games, but broadening in scope
  - Windows OS components, media apps, etc.
- GPU performance has grown at faster rate than CPU graphic performance has
- New customers want that performance
Data Parallel Processing

• A programming model and hardware architecture
• Assign processor resources on a per-data-element basis
• Scales very well with core-count growth
  – Applications written in DirectX3 for 1 ALU still run on 800 core processors
Introducing: the Compute Shader

- A new processing model for GPUs
  - Data-parallel programming for mass market client apps
- Integrated with Direct3D
  - For efficient inter-op with graphics in client scenarios
- Supports more general constructs than before
  - Cross thread data sharing
  - Un-ordered access I/O operations
- Enables more general data structures
  - Irregular arrays, trees, etc.
- Enables more general algorithms
  - Far beyond shading
Target: Interactive Graphics/Games

• Image/Post processing:
  – Image Reduction, Histogram, Convolution, FFT

• Effect physics
  – Particles, smoke, water, cloth, etc.

• Advanced renderers:
  – A-Buffer/OIT, Reyes, Ray-tracing, radiosity, etc.

• Gameplay physics, AI, etc.

• Production pipelines
Target: Media Processing

• Video:
  – Transcode, superResolution, etc.

• Photo/imaging:
  – Consumer applications

• Non-client scenarios:
  – HPC, server workloads, etc.
Optimized for Client Scenarios

• Simpler setup syntax
  – Balance between power and complexity

• Real-time rendering of results
  – Working to reduce cost of transition from compute mode to graphics mode

• Better integration with media data types:
  – Pixels, samples, text, vs only floats

• Need consistency between implementations
  – Both across vendors and over time/generations
Component Relationships

Applications

Media playback or processing, media UI, recognition, etc.

Domain Libraries

Accelerator, Brook+, Rapidmind, Ct MKL, ACML, cuFFT, D3DX, etc.

Domain Languages

DirectX11 Compute, CUDA, CAL, OpenCL, LRB Native, etc.

Compute Languages

Processors

CPU, GPU, Larrabee nVidia, Intel, AMD, S3, etc.
Compute Shader Features

• Predictable Thread Invocation
  – Regular arrays of threads: 1-D, 2-D, 3-D
  – Don’t have to ‘draw a quad’ anymore

• Shared registers between threads
  – Reduces register pressure
  – Can eliminate redundant compute and i/o

• Scattered Writes
  – Can read/write arbitrary data structures
  – Enables new classes of algorithms
  – Integrates with Direct3D resources
Integrated with Direct3D

- Fully supports all Direct3D resources
- Targets graphics/media data types
- Evolution of DirectX HLSL
- Graphics pipeline updated to emit general data structures via addressable writes
- Which can then be manipulated by compute shader
- And then rendered by Direct3D again
Integration with Graphics Pipeline

- Render scene
- Write out scene image
- Use Compute for image post-processing
- Output final image

Input Assembler
Vertex Shader
Tessellation
Geometry Shader
Rasterizer
Pixel Shader
Output Merger

Scene Image
Data Structure
Compute Shader
Final Image
Pixel Shader Programming Model

- For imaging or GPGPU
- Millions of threads
- Each can only write to its own destination
  - No write contention
- No inter-thread communication
- Pure data-parallel model
Compute Shader Programming

- 1000s of thread groups
- Registers shareable within each group
- Arbitrary access writes to video memory
Memory Objects

• DXGI Resources
  – Used for textures, images, vertices, hulls, etc.
  – Enables out-of-bounds memory checking
    • Returns 0 on reads
    • Writes are No-Ops
  – Improves security, reliability of shipped code

• Exposed as HLSL ‘Resource Variables’
  – Declared in the language as data objects
Optimized I/O Intrinsics

• **Textures & Buffers**
  – RWTexture2D, RWBuffer
  – Act just like existing types

• **Structured I/O**
  – RWStructuredBuffer
  – StructuredBuffer (read-only)
  – Template type can be any struct definition

• **Fast Structured I/O**
  – AppendStructuredBuffer, ConsumeStructuredBuffer
  – Work like streams
  – Do not preserve ordering
Atomic Operator Intrinsics

Enable basic operations w/o lock/contention:

InterlockedAdd( rVar, val );
InterlockedMin( rVar, val );
InterlockedMax( rVar, val );
InterlockedOr( rVar, val );
InterlockedXOr( rVar, val );
InterlockedCompareWrite( rVar, val );
InterlockedCompareExchange( rVar, val );
Texture Sampling

- All 1-D, 2-D, 3-D and cube map resource topologies
Texture Sampling Operations

• All DirectX11 texture formats
  – Including new compressed HDR format
  – Sizes extended to 2GB, 16k x 16k,

• Standard HLSL sampling intrinsics
  – Sample()
  – Load()
  – Gather()
More DirectX11 Language Features

• SIMD-optimized method support
  – Facilitates SIMD version of OOP
  – Minimizes register utilization of method instances
  – Enables combinatoric shaders to be specialized

• Arbitrarily addressable writes in Pixel Shader

• Optional double precision
  – New `double` and `long` types
DirectX 11 Foundation

• Support for runtime compilation
  – Very nice during prototyping and development

• Support for runtime data binding
  – Consequence of above

• Compiler provided for off-line use as well
Reduction Compute Code

Buffer<

OutputBuffer<

ImageAverage()
{

groupshared uint Total;  // Total so far

groupshared uint Count;  // Count added

float3 vPixel = load( sampler, sv_ThreadID );
float fLuminance = dot( vPixel, LUM_VECTOR );
uint value = fLuminance*65536;

InterlockedAdd( Count, 1 );
InterlockedAdd( Total, value );

GroupMemoryBarrier();  // Let all threads in group complete
FFT Performance Evolution

CPU

Direct3D9

DirectX11 CS

New Hardware

New Algorithm

SIGGRAPH 2008
Additional Algorithms

• New rendering methods
  – Ray-tracing, collision detection, etc.
  – Rendering elements at different resolutions

• Non-rendering algorithms
  – IK, Physics, AI, simulation, fluid simulation, radiosity

• More general data structures
  – Quad/octrees, irregular arrays, sparse arrays

• Linear Algebra
Summary

- DirectX 11 Compute Shader delivers the performance of 3-D games to new applications
- Demonstrates tight integration between computation and rendering
- Supported by all processor vendors
- Scalable parallel processing model
  - Code should scale for several generations