GPU Architecture Overview

John Owens
UC Davis
The Right-Hand Turn

[H&P Figure 1.1]
Why? [Architecture Reasons]

• ILP increasingly difficult to extract from instruction stream

• Control hardware dominates µprocessors
  - Complex, difficult to build and verify
  - Takes substantial fraction of die
  - Scales poorly
    • Pay for max throughput, sustain average throughput
    • Quadratic dependency checking
  - Control hardware doesn’t do any math!
    • Intel Core Duo: 48 GFLOPS, ~10 GB/s
    • NVIDIA G80: 330 GFLOPS, 80+ GB/s
AMD “Deerhound” (K8L)
Why? [Technology Reasons]

- Industry moving from “instructions per second” to “instructions per watt”
  - “Power wall” now all-important
  - Traditional µproc techniques are not power-efficient

- We can continue to put more transistors on a chip …
  - … but we can’t scale their voltage like we used to …
  - … and we can’t clock them as fast …
Go Parallel

• Time of architectural innovation
  - GPUs let us explore using hundreds of processors now, not 10 years from now

• Major CPU vendors supporting multicore

• Interest in general-purpose programmability on GPUs

• Universities must teach thinking in parallel
What’s Different about the GPU?

- The future of the desktop is parallel
  - We just don’t know what kind of parallel

- GPUs and multicore are different
  - Multicore: Coarse, heavyweight threads, better performance per thread
  - GPUs: Fine, lightweight threads, single-thread performance is poor

- A case for the GPU
  - Interaction with the world is visual
  - GPUs have a well-established programming model
  - Market for GPUs is 500M+ total/year
The Rendering Pipeline

- Application
  - Compute 3D geometry
    - Make calls to graphics API
  - Geometry
    - Transform geometry from 3D to 2D \((in\ parallel)\)
  - Rasterization
    - Generate fragments from 2D geometry \((in\ parallel)\)
  - Composite
    - Combine fragments into image
The **Programmable Pipeline**

- **Application**
  - **Geometry**
    - **Rasterization**
      - **Composite**

**GPU**

- **Compute 3D geometry**
  - Make calls to graphics API

- **Transform geometry from 3D to 2D** [vertex programs]

- **Generate fragments from 2D geometry** [fragment programs]

- **Combine fragments into image**
DirectX 10 Pipeline

Courtesy David Blythe, Microsoft
Characteristics of Graphics

- Large computational requirements
- Massive parallelism
  - Graphics pipeline designed for independent operations
- Long latencies tolerable
- Deep, feed-forward pipelines
- Hacks are OK—can tolerate lack of accuracy
- GPUs are good at parallel, arithmetically intense, streaming-memory problems
Graphics Hardware—Task Parallel

Application
Command
Vertex
Geometry
Rasterization
Fragment
Display

Application/Command (CPU)
Command
Display
Fragment
Rasterization
GPU

Mem

GPGPU
Rage 128
NVIDIA GeForce 6800 3D Pipeline

- Vertex
- Triangle Setup
- Z-Cull
- Shader Instruction Dispatch
- Fragment Crossbar
- Fragment
- Composite
- Memory Partition
- Memory Partition
- Memory Partition
- Memory Partition

GPGPU

Courtesy Nick Triantos, NVIDIA
Programmable Pipeline

[From Akeley and Hanrahan, Real-Time Graphics Architectures]
Generalizing the Pipeline

- **Transform A to B**
  - Ex: Rasterization (triangles to fragments)
  - Historically fixed function

- **Process A to A**
  - Ex: Fragment program
  - Recently programmable, and becoming more so
GeForce 8800 GPU

- Built around programmable units
- Unified shader

[courtesy of Ian Buck, NVIDIA]
Unified Shaders

Application
Command
Vertex
Geometry
Rasterization
Fragment
Display

Application/Command (CPU)
Command
Display
Programmable
Rasterization
GPU

Mem
Mem
Towards Programmable Graphics

• Fixed function
  - Configurable, but not programmable

• Programmable shading
  - Shader-centric
  - Programmable shaders, but fixed pipeline

• Programmable graphics
  - Customize the pipeline
  - Neoptica asserts the major obstacle is programming models and tools

http://www.neoptica.com/NeopticaWhitepaper.pdf
http://www.graphicshardware.org/previous/www_2006/presentations/pharr-keynote-gh06.pdf
Yesterday’s Vendor Support

High-Level Graphics Language

OpenGL δ  D3D δ

Low-Level Device Driver

GPGPU
Today’s New Vendor Support

High-Level Graphics Language

OpenGL \& D3D \& Compute \&

Low-Level Device Drivers

High-Level Compute Lang.

CUDA

CTM CAL

CTM HAL

GPGPU
Architecture Summary

• GPU is a massively parallel architecture
  - Many problems map well to GPU-style computing
  - GPUs have large amount of arithmetic capability
  - Increasing amount of programmability in the pipeline

• New features map well to GPGPU
  - Unified shaders
  - Direct access to compute units in new APIs

• Challenge:
  - How do we make the best use of GPU hardware?
    - Techniques, programming models, languages, evaluation tools ...