Current Trends in Computer Graphics Hardware

Dirk Reiners
University of Louisiana
Lafayette, LA

Quick Introduction

• Assistant Professor in Computer Science at University of Louisiana, Lafayette (since 2006)
  – Iowa State University/VRAC (2003-2006)
  – Fraunhofer IGD (Germany) (1994-2003)
• Area: Interactive 3D Graphics, Software Systems for Virtual Reality
  – Project lead for OpenSG Open Source scenegraph (www.opensg.org)
LITE

- Louisiana Immersive Technologies Enterprise
  - State-initiated, independent R&D lab
  - Good relations to University & Industry
- Provide Visualization, Supercomputing and Networking Resources
  - For University researchers
  - For smaller companies

LITE Venues

- Flex
- Executive Room
- Theatre
- TIS
LITE Resources

• Compute
  – 352 processor (32*11) SGI Altix 350 cluster
  – 160 core, 4 TB main memory single-system-image Altix 4700

• Network
  – 20 Gb Connection to LONI
    • Bridges into National Lambda Rail and out
  – 20 Gb Connection for commercial projects

COMPUTER GRAPHICS HARDWARE
Motivation

• Why bother with special hardware?
• Graphics is very computation-intensive
• Graphics work is different from CPU work
  – Special hardware can be much more efficient
• Interactive (3D) Graphics can never be fast enough
  – People are willing to pay good money for it

First Graphics Hardware

• CRT Displays
  – Need refresh per frame
• Line Drawing Processor
  – Simple Commands
    • Move, Draw
  – Limited amount of memory
• Evolved into simple programs
  – Loops, Subroutines
2D Processors

- Fairly soon after raster-based displays
  - Raster operations need a lot of memory bandwidth
- Simple colored pixel fill
  - Points, Lines, Parallelograms, Triangles
- The world is not flat, though…
- 3D is a lot more challenging
  - 3D models consist of thousands/millions of vertices and triangles

3D Graphics Pipeline

- End: Integer data (pixel coords, colors)
  - Easy and efficient to do in hardware
- Beginning: Floating-point data
  - Fundamental operation: 4D dot product
The Geometry Engine

- Vector-Vector Multiply/Add Unit
  - 5 MFlops
- Front-end to 2D drawing hardware
  - Lines
  - Z-sorted polygons
- SGI Iris Workstation
  - First general-purpose graphics workstation

J. Clark "The Geometry Engine", ACM SIGGRAPH 82

Making it faster

- Pipeline really is a pipeline
- Pipelining allows the use of multiple chips together
  - Don’t have to be different
  - Different Microcode or configuration
  - Iris: 12 (see above)
- A balanced pipeline scales performance linearly
  - Does not reduce latency, though
Parallelism

- Graphics: very parallel
  - Thousands of vertices, millions of pixels
  - Very similar/the same operations on each
- Parallelism improves performance and latency

Pushing the Envelope

- RE: 12 Geometry Engines
  - Intel i860XP
- IR: 4 Geometry Engines
  - Custom ASICs
- 64K Vertex Geo/Raster Buffer
- 4 Raster Memory Boards
  - 80 Image Engines each
  - 1 MB Framebuffer/IE
  - 64 MB Texmem (IR)

K. Akeley “Reality Engine Graphics” SIGGRAPH 93
J. Montrym “Infinite Reality – A Real-Time Graphics System”, SIGGRAPH 97
Making it cheap

- All thanks to John Carmack
- Invented First Person Shooter
  - Doom, 1993
  - Fully software-rendered
- Showed 3D graphics for games
- Jump-started the PC 3D chip industry

PCs Coming Up

- Followed the same evolution as the workstations
  - 2D Pixel Filling (Image Operations, Textures, Depth Buffer)
  - 2D Triangle Drawing
  - 3D Transformations (nVidia GeForce 256, 1999)
- And surpass the workstations…
Workstations vs. PCs

Reaching Limits

• By 2000 the full pipeline was in hardware
• What to do next?
  – Point features: Vertex Blending, Sprites, …
  – Just minor advances
• Problem: Differentiating Features
  – Games need to look better than competition
• Give flexibility to developers
  – Fixed operations, configure connections
Programming the GPU

• First approach: low-level, assembly languages
  – Standardized between manufacturers
  – Not directly mapped to hardware

• Better: High-level, C/C++-style languages
  – Microsoft Direct3D HLSL
  – OpenGL GLSL
  – nVidia CG

How to program?

• Need to keep graphics hardware advantages
  – Pipelining & Parallelism
  – Simplicity: want a lot of simple processors

• Need to keep ordering
  – Final image depends on order

• Solution: Stream Processing
  – Programs (kernels) work on individual units
    (vertices and pixels)
  – No communication between them
What to program?

- 2 Big Parts: Vertex & Pixel Handling
- Separate cooperating programs

The Quest for More Power

- Hunger for performance insatiable
  - More realism, more complexity
  - Need more memory bandwidth, more compute
- GPUs develop much faster than CPUs
  - CPU: doubles every ~18 months
  - GPU: quadruples every ~18 months
CPUs vs. GPUs: Bandwidth

From http://gpu4vision.icg.tugraz.at

CPUs vs. GPUs: FLOPS

From http://gpu4vision.icg.tugraz.at
Processing vs. Cache

- Intel Tukwila
  - 2 B Transistors
  - 30 MB Cache
  - 4 Cores / 16 ALUs
- nVidia GT200
  - 1.4 B Transistors
  - ~2.5 MB Cache
  - > 500 SIMD ALUs
  - 933 GFlops peak

No Cache?

- Caches main tool against memory latency
  - Memory Cycle ~ 200 Compute Cycles
  - Same for graphics cards
- Alternative: Multithreading
  - While waiting, compute on other thread
    - Intel’s HyperThreading
    - GT200: 10240 threads
      - 10 groups with 1024 each
Massive Multithreading

- Allows Very High Performance
  - For the right problems
  - Lots of compute, not a lot of memory access
    - Write worse than read
- Imposes new limitations
  - Register pressure: need to keep state for all threads
  - Max performance for < 10 variables used
  - Ok for graphics: calculate lighting for pixels

GT200 Architecture

From www.beyond3d.com
GPGPU

- General Programming on GPUs
- Use GPU for non-graphics usage
- First: using graphics APIs
  - Render result, read back as image
- Hard to learn, painful to use
  - Still inherently a graphics program
  - Output to pixels only, no scattered write
  - No communication between threads at all
- New API: CUDA / OpenCL

Compute Unified Device Architecture

- Variant of C/C++
  - Vector types
  - Special global shared memory
  - No recursion
  - CPU/GPU specific functions and variables
- Explicit thread blocking
- Limited thread communication
  - None between blocks, some within block
nVidia Tesla

- Graphics card without monitor outputs
- Purely as a compute device
  - Also rack-mounted
- Possible, but not necessary
  - CUDA works on most current nVidia graphics cards

OpenCL

- CUDA is proprietary, only on nVidia systems
  - AMD’s variant is CTM
- OpenCL is an open standard for GPGPU
  - Initiated June 08 by Khronos
- No results yet, but an important step
Example Applications And Speedups

• Some extreme examples:
  – K-Nearest Neighbor Search: 470x
  – Sum Product Solving: 270x
• Image Processing:
  – Sliding-Windows for Rapid Object Class Localization: 30x
  – Optical Flow using CUDA and OpenCV: 90x
• More examples at http://www.nvidia.com/object/cuda_home.html
• See also http://gpu4vision.icg.tugraz.at

Summary

• Graphics operations can be accelerated massively by graphics hardware
• GPUs today are more powerful than CPUs
  – For specific tasks: different programming model
  – Focused on stream processing operations
• Thanks to CUDA/OpenCL can be used nearly as easily as CPUs