Introduction to and History of GPU Algorithms

Jeff M. Phillips

November 9, 2011
Early Computer Graphics

Draw each pixel on screen.
For each pixel $p$:
- Determine if pixel could “see” triangle
- Determine which object “in front”
- If we can “see through” object, what is behind?
- Does light reach that object?
Draw each pixel on screen.
For each pixel $p$:

- Determine if pixel could “see” triangle
- Determine which object “in front”
- If we can “see through” object, what is behind?
- Does light reach that object?

All done on CPU
Early Computer Graphics

Draw each pixel on screen.
For each pixel $p$:

- Determine if pixel could “see” triangle
- Determine which object “in front”
- If we can “see through” object, what is behind?
- Does light reach that object?

All done on CPU
Early Computer Graphics

Draw each pixel on screen.
For each pixel $p$:

- Determine if pixel could “see” triangle
- Determine which object “in front”
- If we can “see through” object, what is behind?
- Does light reach that object?

All done on CPU
Blitters in Hardware

1980s.

- Commodore Amiga, IBM
- Block copying of memory; in parallel on CPU
- Copied image bitmaps quickly (for moving GUIs)
1990s.

3D Gaming!

- OpenGL and DirectX APIs
- GPU directly implemented these APIs
  fixed functional pipeline
- nVidia vs. ATI vs. 3dfx
Early GPUs

“Fixed Functional Pipeline”

- All games / 3D Graphics looked all about the same
- Triangle Rasterization = very efficient
- RayTracing looked, better, but too slow, took much memory!
OpenGL

Was OpenGL the first GPU language?
Was OpenGL the first GPU language? No.

- Just a specification!
- Hardware vendor implemented specification (sometimes slight variation).
- before 2.0, entirely fixed-function
- after 2.0, some different effects added
OpenGL

Was OpenGL the first GPU language?
No.

- Just a specification!
- Hardware vendor implemented specification (sometimes slight variation).
- before 2.0, entirely fixed-function
- after 2.0, some different effects added

DirectX: a Windows library.
- Direct3D is the graphics component
Early GPU programming

Direct3D 8.0 (2000) and OpenGL 2.0 (2004) added support for assembly language programming for shaders.

- nVidia GeForce 3
- ATI Radeon 8000

More minor increments...
Early GPU programming

Direct3D 8.0 (2000) and OpenGL 2.0 (2004) added support for assembly language programming for shaders.

- nVidia GeForce 3
- ATI Radeon 8000

Direct3D 9.0 added *High Level Shader Language* (HLSL)

- nVidia GeForce FX 5000
- ATI Radeon 9000
Early GPU programming

Direct3D 8.0 (2000) and OpenGL 2.0 (2004) added support for assembly language programming for shaders.

- nVidia GeForce 3
- ATI Radeon 8000

Direct3D 9.0 added *High Level Shader Language* (HLSL)

- nVidia GeForce FX 5000
- ATI Radeon 9000

More minor increments...
Early GPU Pipeline

- Vertex data sent via graphics API (e.g. OpenGL, DirectX)
- vertex data processed by **vertex shader**
- vertex shader outputs pixels
- **fragment shader** processes pixels
Early GPU Pipeline

Early-on (Direct3D 10, GeForce 8000, Radeon 2000): vertex / fragment shaders had different hardware.

- slightly different rules
- Direct3D 10 (Windows Vista) added geometry shader, unified hardware

GPUs now use same core to run all shaders
Early GPU Pipeline

Early-on (Direct3D 10, GeForce 8000, Radeon 2000): vertex / fragment shaders had different hardware.

- slightly different rules
- Direct3D 10 (Windows Vista) added geometry shader, unified hardware

GPUs now use same core to run all shaders
Shader Languages

No longer write in assembly!

- GLSL, HLSL, cG, offer C-style shader programming
- write two main() functions which are run on each vertex/pixel
- Auxiliary functions and local variables
- output by setting position and color (write to special variables)
CUDA

Compute Unified Device Architecture

- created by nVidia
- came with GeForce 8000 line
- runs general C code (not restricted graphics APIs)
- Linear Memory Access (no buffer objects)
- runs *thousands* of separate scalar cores
Other GPU patterns

ATI Stream SDK
- closer to assembly
Other GPU patterns

ATI Stream SDK
  ▶ closer to assembly

Apple / Kronos Group (OpenGL) started OpenCL initiative (2008)
  ▶ released 2009
  ▶ supported by nVidia and ATI
  ▶ not specific to GPU (support for CPU SSE)
Other GPU patterns

ATI Stream SDK
  ▶ closer to assembly
Apple / Kronos Group (OpenGL) started OpenCL initiative (2008)
  ▶ released 2009
  ▶ supported by nVidia and ATI
  ▶ not specific to GPU (support for CPU SSE)
DirectX 11 added *DirectCompute Shaders*
  ▶ similar to OpenCL
  ▶ tied with Direct3D
  ▶ added *hull* and *domain* shaders to pipeline
  ▶ allows high-detail geometry created on GPU, not PCI-E bus
Other GPU patterns

ATI Stream SDK
  ▶ closer to assembly

Apple / Kronos Group (OpenGL) started OpenCL initiative (2008)
  ▶ released 2009
  ▶ supported by nVidia and ATI
  ▶ not specific to GPU (support for CPU SSE)

DirectX 11 added *DirectCompute Shaders*
  ▶ similar to OpenCL
  ▶ tied with Direct3D
  ▶ added *hull* and *domain* shaders to pipeline
  ▶ allows high-detail geometry created on GPU, not PCI-E bus

OpenGL 4 similar to Direct 11
  ▶ also added two stages to pipeline
GPU Programming

Top of line:
- 3 Teraflops
- 100+ GB/s memory access bandwidth
- high-speed atomic operations

Now easier to program:
- nVidia’s Fermi architecture supports C++
- MATLAB integration

Many applications:
- Folding@Home
- Photoshop
- Mathematica 8
- large scale data mining
- physics fluid simulation
- computational ecology
GPU Program Model

We will focus on computational properties and data analysis (not graphics)

- Suited for **highly** parallel, fine-grain parallel programs
- Suited for **regular** number-crunching
GPU Program Model

We will focus on computational properties and data analysis (not graphics)

- Suited for **highly** parallel, fine-grain parallel programs
- Suited for **regular** number-crunching
- Need to model hierarchy of processors and memory
GPU Hierarchy

Each processor (SM) has private L1 Cache
- 16-48 kB (small)
- not coherent (CRCW causes problems)
- (256-512 kB on CPU)
GPU Hierarchy

Each processor (SM) has private L1 Cache
- 16-48 kB (small)
- not coherent (CRCW causes problems)
- (256-512 kB on CPU)

Modern systems, L2 Cache
- 512 - 768 kB
- (8-15 MB on CPU)
Each processor (SM) has private L1 Cache
- 16-48 kB (small)
- not coherent (CRCW causes problems)
- (256-512 kB on CPU)

Modern systems, L2 Cache
- 512 - 768 kB
- (8-15 MB on CPU)

Memory bandwidth is fast!
- 100 - 200 GB/s
- but ... separate from CPU
- (24-32 GB/s on CPU)
GPU Hierarchy

Each processor (SM) has private L1 Cache
- 16-48 kB (small)
- not coherent (CRCW causes problems)
- (256-512 kB on CPU)

Modern systems, L2 Cache
- 512 - 768 kB
- (8-15 MB on CPU)

Memory bandwidth is fast!
- 100 - 200 GB/s
- but ... separate from CPU
- (24-32 GB/s on CPU)

Memory size is small!
- 768MB - 6GB
- and ... separate from CPU
- (6 - 64 GB on CPU)
NVidia GeForce 8800 GTX

G80 series

- 128 stream processors:
- 16 multiprocessors
- a multiprocessor has 8 processor units

Higher in hierarchy, more shared memory
Lower in hierarchy, less shared/private memory
GPU Hype

Much hype of 100-200x speed-up on GPU!
- not always fair comparison: 128 GPU cores vs 1 CPU core
- optimized GPU code vs. un-optimized CPU code
- work in single precision (double precision slow on GPU)
- not counting memory transfer time

- As CUDA functionality increased, so did its overhead!

But sometimes GPU is very useful.

Cheap, highly parallel computer!
GPU in Matlab

pMatlab: Parallel Matlab Toolbox v2.0.1

cpu_x = rand(1,100000000)*10*pi;
gpu_x = gpuArray(cpu_x);
gpu_y = sin(gpu_x);
cpu_y = gather(gpu_y);

![Graph showing Elementwise sine on large arrays]
GPU in Matlab

pMatlab: Parallel Matlab Toolbox v2.0.1

cpu_x = rand(1,100000000)*10*pi;
gpu_x = gpuArray(cpu_x);
gpu_y = big-trig-function(gpu_x);
cpu_y = gather(gpu_y);
Attribution

These slides borrow from material by

- Mathieu Desbrun