CIS 565: Study Guide
Spring 2011

Studying Tips

- Go through the course slides and use this to guide to know where to focus
  - If we didn’t cover a topic in class, it will not be on the final even if it is on the slides
  - Material in student presentations and guest lectures will not be covered
- Think about what we emphasised in class - what questions am I likely to ask? If you were teaching this course, what would you ask?
- Go through the written questions in the homeworks
- Study alone, then review in a group
- Study hard early, then ramp down as exam day approaches

Graphics

- Graphics Pipeline
  - Programmable vertex and fragment shaders
  - Fixed function stages
  - Types of parallelism: data-parallel, pipeline parallel, CPU/GPU working in parallel
  - Evolution
    - Software rendering
    - Fixed function hardware
    - Programmable stages with different shader cores, e.g., GeForce 6800 (NV40)
    - Unified shader architecture, e.g., GeForce 8800 (G80)
- GLSL
  - Fixed function vs. programmable pipeline from the developer’s perspective
  - Shader execution model: SIMT
  - Shaders
    - Inputs: vertex attribute, uniforms, and textures
      - Vertex vs. attribute vs. component
    - Communication between vertex and fragment shaders
    - Fragment shader output
  - Given some verbose GLSL code and a list of GLSL functions, make the code concise.
- OpenGL
  - What is it? Why does it exist?
General understanding of the OpenGL API. You will not be asked specific questions that require you to remember the 5th argument to `glShaderSource` (tip: it only has four arguments).
- Shader objects vs. shader programs
- Drawing: immediate mode vs. display lists vs. client-side vertex arrays vs. vertex buffer objects (VBO)
- Multi-threaded OpenGL drivers

**GLSL Applications**
- Per-vertex vs. per-fragment lighting
- Using a fragment shader for image processing
- Multitexturing, globe and terrain examples
- Framebuffer Object (FBO), deferred shading example
- Ambient Occlusion concepts

**GPGPU**
- Hijacking fragment shaders for general-purpose computations
- Parallel reduction
- Limitations

**Scan**
- Sequential Scan
- Naive Parallel Scan
- Use cases
  - Stream Compaction
  - Summed Area Tables (SAT)

**GPU Architecture**
- CPU vs. GPU
- Throughput vs. latency
- What are transistors spent on?
- How are branches handled?
- How is memory access handled?
- Context size vs. latency hiding ability

**CUDA**
- Define: Host, Device, Kernel, Grid, Block, Warp
- Thread Hierarchies
- Memory Model
- Synchronization
  - Within blocks via `__syncthreads`
  - Across blocks via atomics
- Scheduling threads via warps
- SM limits (in general, not exact numbers) that create a performance cliff:
  - Number of registers
  - Number of threads/warps(blocks
  - Amount of shared memory
- Performance
  - Warp partitioning
- Memory coalescing
- Data prefetching
- Loop unrolling
- Thread granularity
- Bank conflicts
- Page-locked host memory
- Graphics interoperability
  - Parallel reduction in CUDA
  - Matrix multiple example and optimizations
- OpenCL
  - General idea compared to CUDA and OpenGL
- Fermi
  - Caches
  - Dual-warp scheduling
  - Parallel kernel execution