GPUs
Why GPUs?

In order to render a scene, we must determine the color assigned to each pixel (usually based on light transport)
Work Per Fragment

Fixed work per fragment
Ideally process several hundred thousands of these at 60Hz
Working on the CPU

CPU is big and complex but fast on a single thread
...But even a really fast thread isn’t sufficient for shader execution...
Graphics Processing Unit

Built for rendering pipeline
  • Process large number of vertices
  • Assume similar, relatively simple, operations

What sort of architecture facilitates this?
Throughput Architecture

Simpler cores but lots of them in parallel!
Remember the Rendering Pipeline?
Modern GPU Characteristics

- Homogeneous programmable cores for all programmable stages
- Relatively few special purpose texture units
- Even fewer fixed function units
- Task parallel at pipeline level
SIMD

• Single instruction, multiple data
• Large vectors of data that have the same operation applied to individual elements in parallel
• Based on old super computing techniques but has regained popularity in modern architectures (both CPU and GPU)
• Same thing is done in parallel for all fragments/verts/etc
• SIMD amortizes instruction handling over multiple ALUs
Multiple Types of Processing

GPUs do more than shading
  • Allow execution of more than one program

Replicate SIMD processors for different SIMD computations in parallel

8 programs running in parallel, 128 threads in parallel
Problems?

What situations does this throughput style of architecture not handle well?
Branching and Stalling

- Threads stall when next instruction depends on previous instruction’s result
- Pipeline dependencies
- Memory latency
- Complex CPU hardware not available on GPU
- How to handle these?
Multithreading

• We can assume there are more threads (scheduled computations) than processors
• Threads with similar code executed in “warps” to maintain minimal divergence
• Interleaving warp execution keeps hardware busy when an individual warp stalls
Working with Latency

• Latency hiding
  • Executing many warps can minimize latency (delay in processing)
• More context switching requires more storage (values in registers etc)
GPU Memory and Architecture

Designed for throughput, so bandwidth is critical

- Wide bus (150 GB/s+)
- High bandwidth DRAM organization
- Warp scheduling for latency hiding
- Small execution contexts and local memory
- Limited cache hierarchy
Example: Pascal Architecture
Programming on the GPU

Idea: Programmable shader pipeline is highly specific to rendering. Create a language that can harness GPU throughput with more accessible programming paradigms.
GPGPUs

• Solve non-graphics problems on GPUs
  • Textures act as memory
  • Compute shaders allow for small, highly parallel executions
  • Methods like map, reduce, scatter, gather, etc provided for convenience
• Languages like CUDA and OpenCL facilitate development
GPGPU Challenges

- Parallelization algorithms
- Memory for throughput architecture
- Work scheduling on throughput architecture
- Hiding latency
Toward Heterogeneous Architecture

Idea: CPUs are good at some things and GPUs are good at others. Why not have them closer together to get the best of both worlds?

• Already commonly used in embedded devices (e.g. system on a chip)
• Has attractive properties for general computing as well
• Also presents numerous software and hardware challenges at all levels of programming!