OpenCL for realtime graphics
Matthäus G. Chajdas
Why OpenCL?

• Because OpenCL does it all
  – **Single source** for all graphics APIs (D3D9, D3D10, D3D11, OpenGL 2.x – 4.x)
  – Mobile support
  – Web support
• Clean interop API
• Easy debugging (printf on CPU!)
Interop API

• Acquire resource, map/unmap
• Automatic synchronization
• Minimal code required
  – Full production implementation: 900 LoC
  – Includes error checking & comments
  – Supports OpenCL 1.1 & 1.2, OpenGL, Direct3D11 KHR/NV
Tile based deferred shading

1. Divide screen into tiles and determine which lights affects which tiles
2. Only apply the visible light sources on pixels
   ✓ Custom shader with multiple lights
   ✓ Reduced bandwidth & setup cost

How can we do this best in DX11?
Tile-based Deferred Shading

1. Divide screen into tiles and determine which lights affects which tiles

2. Only apply the visible light sources on pixels
   - Custom shader with multiple lights
   - Reduced bandwidth & setup cost

How can we do this best in OpenCL?
Tile based deferred shading
Tile based deferred shading

- Works!
- Performance roughly the same as DirectCompute
- Some limitations apply though ...
Tile based deferred shading

- No image read/write
  - Just map image twice & get the desired undefined behavior :)  
  - Fixed in OpenCL 2.0 

- No depth images 
  - Fixed by extension (for OpenGL) 

- No MSAA images 
  - Fixed by extension (for OpenGL)
Real compute

- Voxel raytracing
  - 600 LoC of kernel code
  - Tree traversal
  - Per-thread stacks
- Run out of GPU memory?
  - Current GPUs max out at 16 GiB (AMD) or 12 GiB (NVIDIA)
  - CPUs max out at ... well ...
Voxel raytracing

- Same code
- Really!
- Portable performance is not a myth!
Voxel raytracing

- GPU optimizations benefit CPU and vice versa
- No special casing
  - Avoid some “obviously” slow paths
  - AMD & NVIDIA* get comparable performance/efficiency
  - CPU performance ~ 30% slower than extremely optimized ISPC code
Please select the part of the volume you want to work on.
• Voxel editor
• All editing is done in OpenCL only
• Graphics interop running all the time
  – OpenCL prepares data for visualization
  – Minimal CPU intervention (to minimize buffer sizes)
VOTA, lessons learned

- All users have OpenCL installed
- AMD doesn't have a kernel cache (just do it on your own)
- Compiler quality is “good enough”
  - Still, we eagerly wait for SPIR
  - Initial startup takes quite some time due to compiling
- Query everything which is implementation specific
OpenCL/Graphics interop state

- **AMD**
  - Direct3D: Works
  - OpenGL: Works mostly* (Linux & Windows)

- **NVIDIA**
  - Direct3D: Through _NV extension, _KHR not supported.
  - OpenGL: Works (Linux & Windows)
OpenCL/Graphics interop state

- AMD's implementation is much stricter (similar to OpenGL)
- Both have occasionally issues with memory management
  - Need extension to force GPU memory defragment & query where buffer ended up
- For debugging/profiling, use CodeXL/KernelAnalyzer
  - NVIDIA stopped supporting OpenCL profiling some time ago
Reporting bugs

- AMD: Just report through normal channels. They got back to me quickly & and you get status updates (cool)
- NVIDIA, best report through CUDA bug report form. They don't say much about it, but they still fix OpenCL bugs.
- Intel: The forums seem to work fine.
OpenCL for graphics, the road ahead

• With OpenCL 2.0, OpenCL will far surpass DirectCompute/OpenGL compute shaders
  - GPU pipelining
  - Shared CPU/GPU memory data structures
  - AMD is aggressively working on it, first OpenCL 2.0 SDK should come this June
  - Intel most likely working on it as well
Conclusion

- You can ship an application built on OpenCL **today**
- For games, you'll likely want the MSAA/depth extensions
  - Vendors should do it instantly **once requested**
  - Can start writing the game today without problems
- In future, hard to avoid
  - OpenCL 2.0 allows GPU to traverse scene graph, prepare draw commands, then issue draw calls from CPU, with zero copy overhead
Thanks to ...

- The whole Volumerics team
  - @janjorrit, @mreitinger
- @repi
- AMD
  - @grahamsellers, @JCBaratault
- Codeplay & Khronos
  - @codeandrew, @neilt3d