Understanding Shaders and WebGL

Chris Dalton & Olli Etuaho
Agenda

- Introduction: Accessible shader development with WebGL
- Understanding WebGL shader execution: from JS to GPU
- Common shader bugs
Accessible Shader Development with WebGL
Accessible Shader Development with WebGL

WebGL Shaders

- WebGL shading language is a restricted variant of The OpenGL ES Shading Language 1.00
- Enables flexible shading and some general purpose usage of GPUs on the web
Accessible Shader Development with WebGL

Web-based tools

- Start experimenting immediately
- ShaderToy http://shadertoy.com/
- Shdr http://shdr.bkcore.com/
- http://glslsandbox.com/
Accessible Shader Development with WebGL

Browser tools

- Shader Editor in Firefox
- For existing projects
- Highlighting errors
- Highlighting objects based on shader
Power of WebGL Shaders

Demos

► http://labs.gooengine.com/examples/painter
► http://david.li/flow
► Will only get better with WebGL 2.0, with a more flexible version of GLSL!
Understanding WebGL Shader Execution
Understanding WebGL Shader Execution

Path from JS to GPU

- Browser validates shader source in compliance with the GLSL ES spec
- Browser translates the shader code for the platform drivers, exact process depends on the browser
- Platform driver translates the shader program to GPU bytecode
- GPU bytecode runs
Understanding WebGL Shader Execution

Shader Translation

- ANGLE is used for shader validation and translation in Chrome and Firefox
- Translates WebGL shaders to HLSL or to GLSL that's appropriate to the platform
- Workarounds and emulation are used where needed

https://www.khronos.org/registry/webgl/sdk/tests/extra/webgl-translate-shader.html
Understanding WebGL Shader Execution

Quirks of Shader Translation

- Dynamic indexing is clamped - some additional overhead
- Some built-ins are emulated on HLSL, can have slight overhead:
  mod, faceForward, atan(y, x)
- Short-circuiting || and && operators sometimes need to be converted to if statements on HLSL
Understanding WebGL Shader Execution

Shader Translation on WebGL 2

- WebGL 2 on HLSL will add emulated built-ins:
  - hyperbolics, roundEven (!), pack/unpack, matrix inverse
- switch statements with fall-through need complex workarounds on HLSL
Understanding WebGL Shader Execution

Modern GPU pipeline (simplified)

- Shader cores execute compiled GLSL
  - “Unified” means vertex/fragment shaders run on same unit
- Rasterizer converts triangles to pixels
- ROP (Render/Raster Output Unit) does per-pixel operations
  - Depth/stencil test
  - Blending
- Texture units do the math for sampling
  - Derivative/LOD computation
  - Clamping
  - Filtering

Framebuffer (main memory)
Understanding WebGL Shader Execution

GPU multiprocessing model

- **Basis:** SIMD (Single Instruction Multiple Data)
  - The same set of instructions executes for every pixel/vertex
- **Conceptually:** Each pixel has its own dedicated processor, with a centralized controller
  - As controller decodes instructions, processors execute in sync
  - Results of one pixel/vertex shader can't affect other shaders
  - Can run simultaneously
Understanding WebGL Shader Execution

The Maxwell SMM

- Tegra X1 has 2 SMMs, GTX 980 has 16 SMMs
- An SMM == Streaming Multiprocessor
- An SMM has 4 partitioned blocks
- A partitioned block executes warps
- A warp is a group of 32 threads
  - Can't swap threads in or out
  - Can't “context-switch” to a different block
- A thread is a single execution of your vertex/fragment shader for a specific vertex/pixel
Understanding WebGL Shader Execution

Partitioned Blocks

- **16,834 registers (32-bit)**
  
  Every warp has its own registers (zero context switch cost)
  
  Partition switches warps on a per-instruction basis
  
  Amount of warps limited by register requirement

- **32 basic math units (aka cuda cores)**
  
  mad, mul, add, mov, min, max, abs, sub, neg, slt, sge, etc.
  
  Cost: 1 cycle per instruction

- **8 special function units**
  
  sin, cos, 1/x, sqrt, pow, log, exp, etc.
  
  Cost: 4 cycles per instruction

- **8 load/store units**
Understanding WebGL Shader Execution

Demo

http://swankybird.com/play/

Water, waves, pipes all ray traced with special functions (sin, cos, 1/x, sqrt, pow, log)

Special functions are cheap!
Understanding WebGL Shader Execution

Optimization tips

- Avoid multiple or dependent memory accesses (300-600 cycles!)
  A few can be hidden by running other warps
  Flops/fillrate is around 32 on modern GPUs
  Memory bandwidth is shared with CPU on mobile

- Maximize the number of warps that can be run on a partition
  - Less registers => more threads

- Avoid divergent branching
  - If one thread takes a branch, they all wait
Understanding WebGL Shader Execution

Divergent branching example

```c
attribute float a;

void main() {
    float v = 0.0;
    if (a > 0.0) v = 1.0; // relatively lightweight
    doSomething(v);
}
```
attribute float a;

void main() {
    // this can get expensive!
    if (a > 0.0) doSomethingComplex();
    else doSomethingElse();
}
Common shader bugs
Common Shader Bugs

- Most frequent issue in the wild: incorrect use of shader precision
- Two variations
  - #1: use of mediump or lowp when the shader requires 32-bit precision for correct results
  - #2: use of highp when 16-bit precision would be enough
Common Shader Bugs

Precision bug example

Flawed code: http://glslsandbox.com/e#20238.2
Common Shader Bugs

Solving precision issues

- Easy way out: use highp everywhere
- Shaders will act the same on mobile and desktop
- Cost: loss of performance on some mobile devices, only compatible with 85% of WebGL-capable mobile clients
  (data: webglstats.com late 2014)
Common Shader Bugs

Solving precision issues

▷ Alternative solution: test with precision emulation and fix
▷ Emulate mobile-like shader hardware on desktop
▷ `chrome --emulate-shader-precision`
  Add `--use-gl=desktop` on Windows
  (requires Chrome version 41)
▷ Note the large performance cost of emulation
Common Shader Bugs

Precision bug example

- Precision emulation: http://glslsandbox.com/e#20238.6
- Fixed code: http://glslsandbox.com/e#20238.4