1 Abstract

WebGL is a new Khronos Group standard for GPU-accelerated rendering by in-browser JavaScript applications. WebGL introduces a new language, WebGLSL, to the WWW ecosystem. WebGL implementations consume WebGLSL source to produce GPU instructions for graphics rendering. WebGLSL lacks many high-level language features such as user-defined namespaces and modules. The language also includes several powerful but poorly defined features such as a lexical preprocessor and non-orthogonal overloading.

In the interest of engineering large-scale 3D rendering and numerical geometry applications, Ashima Arts has developed a tool, gloc, to manipulate and analyze WebGLSL. gloc 1.0 includes a first-class partially-evaluating lexical preprocessor, free variable namespace analysis, high-fidelity lexical information, comment-tagged tokens, a linked data JSON shader source/analysis format (glol), js_of_ocaml cross-compilation, and a JavaScript translation of glol, the glo linker. gloc is BSD licensed. A JSON-based S-expression DSL for GLSL module-level functions, a constraint type system to handle ad hoc polymorphism, and a Chrome extension to scrape shader source are presently being developed and progress on these subsystems will be presented.
2 Use

Shader and numerical kernel module developers may use either the gloc online development environment (glocode [http://ashimaarts.com/gloc/glocode/]), a node.js package of the js_of_ocaml compile of gloc, or a native OCaml version of the compiler to produce JSON or XML glo files. These glo files can then be linked together into complete shader programs with either an OCaml implementation of the linking algorithm, glol, or a JavaScript translation of the same. The JavaScript translation is $\sim300$ source lines and $\sim3$kb compressed. gloc 1.0 leverages ulex, menhir, lwt, js_of_ocaml, atdgen, cohttp, ocaml-re, ocaml-uri, and OCaml 3.12.

To the author’s knowledge, the gloc toolchain is the only existing solution for modular WebGL shaders that does not introduce additional annotations or semantics to the source language. gloc is primarily implemented in OCaml, giving the tool access to a powerful ecosystem of specification and transformation tools to aid metaprogramming and platform-independence.

3 Demonstration

My talk will feature demonstrations of an in-browser compiler interface, live shader module editing, dynamic shader effects, and WebGL API proxying to overload dynamic shader modules in third-party applications.
4 Experience

I am interested in learning more about best practices for developing program transformations, syntax tree representations, type checking algorithms, and semantic web applications in OCaml. I will invite the audience to join me in researching and developing the libraries and tools that will form the basis for a common, universally targetable hyperlinked numerical kernel library. WebGLSL syntax extensions, algebraic optimizations, alternative targets (JavaScript, OCaml, LLVM), and extension-based decomposition of the standard will be discussed. Development experience with lwt, js_of_ocaml, and atdgen will be detailed. My experience with the language standard and the standards body will also be presented.

5 References

https://github.com/ashima/gloc
https://github.com/ashima/webgl-noise
http://ashimaarts.com/gloc/
http://ashimaarts.com/gloc/glocode/
http://ashimagroup.net/demo/game/ooman/tutorial/
http://ashimagroup.net/demo/recon/corner/
http://ashimagroup.net/demo/pano/mars/
Figure 2: Screenshot of Ashima Arts’ geometric puzzle platformer, *Ooman*, with Ashima’s simplex noise shader on the blue platform