Cross-Compiled 3D Web Applications – Problems and Solutions

Christoph Müller, Fabian Gärtner
Furtwangen University, Germany*

1 Introduction

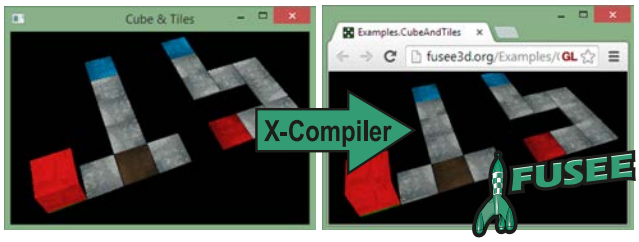

Figure 1: *A cross-compiled .NET game running inside Chrome*

A number of projects, including our FUSEE engine, provide the means to cross-compile real-time 3D applications from more traditional programming languages such as C, C#, or Java into JavaScript/HTML5 in order to run those applications natively in web browsers. The current state of the FUSEE project shows that the use of a cross-compiler and an appropriate software architecture enables real-time 3D applications written in C# and on top of FUSEE to be automatically ported to plugin-free browser applications (see Figure 1). In some cases, however, the cross-compiled result suffers from limitations of the cross compiler, underlying libraries, or the JavaScript/HTML5 platform [Müller et al. in press].

In this study we identify the three most significant obstacles to cross-compiling arbitrary real-time 3D applications into equivalent counterparts of their native versions. Furthermore, we present our work in progress on approaches for addressing those obstacles.

2 Solution Approaches

2.1 Performance

The runtime behavior of JavaScript applications is generally weak compared to C# code. This is not necessarily obvious as both languages yield just-in-time compiled highly optimized machine code, at least with contemporary JavaScript execution engines. The reason for the lack of performance is the weakly typed object model behind JavaScript. Here, the content of an object member results in a number of recursive hash table lookups to memory locations scattered all over the physical address space. This consumes much more processor cycles than adding an offset to a memory address in contiguously allocated data chunks. Our approach tries to leverage TypedArrays [The Khronos Group et al. 2013], a rather new addition to JavaScript. FUSEE's cross-compiled approach offers the chance to modify the current cross compiler to translate well-performing and, at the same time, object-oriented and easily readable C# code into JavaScript using TypedArrays.

*e-mail: {mch, gaertner}@hs-furtwangen.de

SIGGRAPH 2014, August 10 – 14, 2014, Vancouver, British Columbia, Canada.

ACM 978-1-4503-2958-3/14/08

2.2 File and Asset Serialization

Asset data should be present in a fast-to-read binary format. The serialization code for reading and writing such data from and to memory (i.e., programming language constructs such as structs, classes, and arrays) should be easy to create and maintain. A number of serialization techniques exist which require minimal coding to create serialization code from existing object structures to and from data streams. The "Protocol Buffers" (protobuf) project [Google Inc. 2001–2014] is a serialization library already available for multiple platforms, such as JavaScript and C#. Unfortunately, the structures of structs and classes are different after cross-compiling, so an automatic type cast on the deserialized data would not work. Therefore, using protobuf in a cross-compiled scenario would require manual re-coding of functionality for both platforms. We found a way to automatically transform the protobuf serialization code from C# into JavaScript. This allows for the writing and maintenance of serialization code with a single-source approach for all platforms.

2.3 GPU Programming

The growing number of platforms supported by FUSEE require a platform abstraction for the CPU-programming part of a FUSEE application as well as an abstraction of the underlying GPU-programming interface. Currently, FUSEE applications need to supply different shader code for the different GPU platforms. To enable developers to handle GPU programming with the same platform-independent single-source approach as CPU programming, we follow an approach in which a defined subset of C# will be used as a common shader language. In the same way the JavaScript cross compiler creates JavaScript code from compiled C# code, another cross compiler will translate compiled C# code (IL) into the respective GPU platform language, such as GLSL, GLSL/WebGL, or HSLS. While approaches exist for using C# on the GPU, we are the first to exploit the cross-compile approach to bridge the gap between different GPU programming platforms and languages.

3 Conclusion

The initial results from our study described above show that a wide variety of different types of real-time 3D applications can be automatically cross-compiled to web browser applications without the need for manual correction of the result or significant drawbacks in terms of performance or functionality.

References

GOOGLE INC., 2001–2014. protobuf — Google Protocol Buffers: Google's data interchange format — Google's Project Hosting. Online: https://code.google.com/p/protobuf/.

MÜLLER, C., GÄRTNER, F., AND DIK, D. in press. Portable 3D-browser-applications using cross-compiled programming languages. In *Computational Science and Computational Intelligence, 2014 International Conference on*, IEEE, The American Council on Science and Education. Unpublished Paper.

THE KHRONOS GROUP, HERMAN, D., AND RUSSEL, K., 2013. Typed Array Specification — Editor's Draft (work in progress). Online: http://khronos.org/registry/typedarray/specs/latest/.