Towards Interoperability of Distributed Interactive Simulations through Node-based OpenGL Stream Processing

Maik Mory, Norbert Siegmund, Alexander Blankenburg, and Gunter Saake

Otto-von-Guericke-Universität
Universitätsplatz 2, D-39106 Magdeburg, Germany

Abstract. Coupling heterogeneous simulation environments is a requirement in the engineering domain. A novel approach exploits simulations' visualizations to setup distributed simulations: OpenGL stream processors intercept the dialogue between OpenGL clients and servers, and (distributedly) process the request-reply stream. As a first step towards an holisitic approach, we describe the Vanadium node-based OpenGL stream processing framework from the viewpoint of an integrated interoperability model. Results are a systematic examination of the framework's capabilities and limits; identification of open research questions; and an initial benchmark for architectural comparison with other OpenGL stream processing frameworks.

1 Introduction

Engineers use plenty of heterogeneous software. Thus, interoperability between various software products is an issue in the daily work of almost any engineer or sophisticated craftsman. We focus on distributed simulations. A simulation system primarily is software that supports engineers in decision making. In Digital Engineering, this definition includes digitally tracked mockups or prototypes. A simulation is distributed, if it uses at least two dedicated, heterogeneous software systems that interact by any means to implement the simulation required by the engineer.

There are plenty of file formats for almost any imaginable use case. Inherent latency renders file exchange unfeasible for interactive simulations and other use cases relying on bidirectional data flow. Nowadays, most engineering-software provides one or more application programming interfaces (API). Prominent examples are NX [1] and Matlab [2]. Coupling engineering-software via software-specific APIs requires a software developer with knowledge about the respective API, and it requires to establish interoperability between the software-specific API and the system landscape. Software-specific APIs allow for sophisticated integration solutions, but development might be too expensive for many scenarios.

Node-based OpenGL stream processing enables interactive distributed simulations at the convenience and cheapness of file-based methods. First, we describe the working principles of OpenGL stream processors. In Section 3, we present one selected node-based OpenGL stream processing framework along Manso et al.'s [3] integrated interoperability model with seven levels of interoperability. Finally, we discuss our results.

2 Background

Since the 1990s, OpenGL has been the dominant solution for visualization. OpenGL declares a client-server architecture. The OpenGL client requests configuration of the rendering pipeline, direct drawing operations, and geometric primitives. The server renders according to the client's requests to a framebuffer, which commonly is mapped to a display device.

In 1996, the concept of an OpenGL stream was established [4]. An OpenGL stream is the sequence of requests and replies between an OpenGL client and an OpenGL server. WireGL [5] was one of the first comprehensive node-based OpenGL processors, which targeted at distributed processing in commodity clusters. WireGL was open-sourced and renamed to Chromium [6]. The Chromium project drove development of many algorithms and technologies towards a comprehensive node-based OpenGL stream processing framework. For us, it is relevant that Chromium brought into focus, how to intercept the client-server dialogue with proprietary visualization software. Chromium's development stalled in 2008.

The German software company IC.IDO published an advertisement video about their software product IDO.Capture in 2007 [7]. The video illustrates an application scenario, which uses OpenGL stream processing for interactive distributed simulation. The technology leader in OpenGL stream processing TechViz advertises a similar solution called TechViz Fusion [8]. Apparently, both commercial competitors do not directly contribute to the academic discourse. Nevertheless they indicate that OpenGL stream processing is feasible in complex scenarios.

In previous work, we develop the Vanadium node-based OpenGL stream processing framework [9, 10]. This is inspired by Chromium's architecture, involves some lessons learned from other OpenGL stream processors (HijackGL [11] for example), and reconsiders some design decisions with respect to today's options. We describe details about Vanadium's architecture in Section 3 as necessary.

3 Vanadium's Interoperability Concept

We use Manso et al.'s interoperability model [3], because their broad survey defines a consistent and coherent vocabulary, which serves well in engineering-software scenarios. The authors declare seven levels of interoperability: technical, syntactic, semantic, pragmatic, dynamic, conceptual, and organizational. The remainder of this section will evaluate the Vanadium node-based OpenGL stream processing framework with respect to these levels.

Technical interoperability enables the interconnection of systems through common communication protocols allowing information exchange at a basic level [3]. We rely on two libraries for technical interoperability: the ØMQ library and the Detours library.

ØMQ [12] is a multi-platform messaging library. Its message primitive are bytestrings. The ØMQ library delivers messages inter-thread, inter-process, cluster-wide, and world-wide. ØMQ's minimalist programming interface supports request-reply, publish-subscribe, and push-pull messaging patterns. We emphasize ØMQ's support of reliable multicast protocols, which came in handy for us in distributed rendering scenarios [9].

Node-based OpenGL stream processing commonly assumes that stock software should serve as source for the OpenGL commands to be processed. However, special care has to be taken for proprietary software. Based on a survey [10], we select the binary interception technique together with the Detours library [13] as reference implementation. We successfully use the Detours library with Kuka's KukaSim, Bitmanagement's BSContact, and Siemens' NX 7.5, among others.

Syntactic interoperability ensures common data format or structure, language, logic, registers and files [3]. A robust syntax requires a complete formal specification. A special manifestation of OpenGL's formal specification are the so-called specfiles [14], which declare all of OpenGL's identifiers, symbols, and constants in a formal language again. Similar to Chromium, we rely on automated source-code generation. The OpenGL specfiles are input to our code generator, which creates mappings between the binary application interface as provided by the Detours library and the networking interface as encapsulated by the ØMQ library and vice versa. The entities introduced by OpenGL's specification are the symbols we talk about.

A shared, common vocabulary that avoids inaccuracies or mix-ups when interpreting the meaning of terms or symbols [3] enables **semantic interoperability**. Since we use OpenGL's formal specification for syntactic interoperability, it is obvious that OpenGL's documentation as provided with the OpenGL software development kit [15] serves as the semantic counterpart. OpenGL's informal specification is in English language, comprehensive, consentaneous, and mature.

Manso et al. [3] state that **pragmatic interoperability** is about interconnected systems knowing each other, so they are able to exploit application/services interfaces, to invoke methods or procedures, and handle the data they need to exchange with other systems. Due to OpenGL's architecture, the majority of visualizing software are either monolithic renderers or use tree-like scenegraphs. Monolithic renderers act as exclusive OpenGL clients, where invocations are scattered to the entire software application. In contrast, a scenegraph is a tree where the nodes are snippets of OpenGL invocations. The most popular and reference scenegraph is OpenSG [16].

One may wiretap a client-server dialogue and publish it for subscription. This converts OpenGL's bidirectional architecture into directed communication. It has been shown, that such a stream of OpenGL request-reply tuples is feasible for implementation of complex tasks [9] and well-suited for multicasting. Thus, we end up with a directed graph where the nodes are OpenGL stream processors and the edges are publish-subscribe relationships. We currently search for the limits of the proposed directed-graph architecture. We expect that the proposed architecture supports modularized virtual reality effects like they are motivated by Haringer and Beckhaus [17].

Dynamic interoperability allows the systems to monitor the running of the other systems and to respond to the changes in the transfer of information by taking advantage thereof [3]. We intentionally reduce interdependencies in our framework. Therefore, an OpenGL stream processor node in our framework does not even know its predecessors or successors; ideally, nodes faithfully process messages. Hence, we see dynamic interoperability as the ability for an OpenGL stream processor node to join or leave the graph at runtime.

In this consideration, it is important to know, that a published client-server dialogue implies (hidden) states of the OpenGL server between any request-reply tuple of the stream. Thus, a node may join the graph, if it is able to catch up the implied server state as required to work properly. A node may leave the graph, if its subscribed nodes are able to catch up the implied server state to whatever will replace the leaving node. If a node leaves without replacement, its successors break down too.

Knowing and reproducing the functioning of a system based on documentation, usually articulated in a format as used in engineering, yields **conceptual interoperability** [3]. Vanadium's exploited concept is geometric primitives with scanline rendering. Every engineering-software with an OpenGL-based visualization supports this concept. Every engineer should hopefully be able to interpret a geometric visualization. However, the wide range of framework participants is a trade-off towards sophisticated functions. If an OpenGL stream processor node receives a geometric primitive, there is no hint whether the primitive is a rigid body's surface, a fluid particle, an electrical field, or a user annotation. Therefore, Vanadium does not substitute tight couplings between distributed simulations. We expect to ease lightweight couplings between heterogeneous simulations.

Organizational interoperability is about business targets, process models, regulations and policies of access and use of data and services [3]. Many engineers and other stakeholders appreciate Vanadium's simple and low-barrier approach. However, we have to take into account other stakeholder's interests too. For example, there is the argument that OpenGL stream processors might "steal" models. OpenGL is an open client-server system. The Vanadium framework is just another OpenGL server, which might extend an engineering application's value.

4 Discussion

Based on a brief introduction to OpenGL stream processing, this paper describes the Vanadium node-based OpenGL stream processing framework. Our architecture is analyzed with respect to technical, syntactical, semantic, pragmatic, dynamic, conceptual, and organizational interoperability respectively in detail. This analysis enables a sound reasoning when to apply OpenGL stream processing as interoperability solution in engineering scenarios.

The systematic examination unveiles open research questions with respect to OpenGL stream processing. Dynamic interoperability has not been considered before. We could show that dynamic interoperability interferes with the pragmatic interoperability level, and we create a stub for further research.

Finally, this paper establishes a schema for an irrespective description of OpenGL stream processing frameworks. Thus, we enable qualitative comparability between frameworks. At the same time, this paper serves as initial benchmark for architectural comparison with other OpenGL stream processing frameworks. We encourage the research community to use similar comparison approaches.

Acknowledgments. Maik Mory and Norbert Siegmund are funded by German Ministry of Education and Science (BMBF) within the ViERforES project No. 01IM10002B.

References

1. Siemens PLM Software: NX Programming and Customization Fact Sheet, http://www.plm.automation.siemens.com/zh_cn/Images/4988_tcm78-4564.pdf (July 2010)
2. MATLAB®: C/C++ and Fortran API Reference, http://www.mathworks.de/help/techdoc/apiref/bqoqnz0.html (September 2012)
3. Manso, M., Wachowicz, M., Bernabé, M.: Towards an Integrated Model of Interoperability for Spatial Data Infrastructures. Transactions in GIS **13**(1) (2009) 43–67
4. Dunwoody, C.: The OpenGL Stream Codec – A Specification. Silicon Graphics, http://www.opengl.org/documentation/specs/gls/glsspec.txt. (1996)
5. Humphreys, G., Eldridge, M., Buck, I., Stoll, G., Everett, M., Hanrahan, P.: WireGL: A Scalable Graphics System for Clusters. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. SIGGRAPH '01, New York, NY, USA, ACM (2001) 129–140
6. Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P.D., Klosowski, J.T.: Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters. In: ACM SIGGRAPH ASIA 2008 courses. SIGGRAPH Asia '08, New York, NY, USA, ACM (2008) 43:1–43:10
7. ESI Software Germany GmbH: IC.IDO Virtual Reality: Software IDO.Capture, http://www.youtube.com/watch?v=1tymAyDmdNM (March 2007)
8. Boutin-Boila, E.: TechViz Fusion, http://www.techviz.net/wp-content/uploads/TechViz_Fusion_2011.pdf (March 2011)
9. Mory, M., Masik, S., Müller, R., Köppen, V.: Exposing Proprietary Virtual Reality Software to Nontraditional Displays. In: Proceedings of the 20th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision. WSCG Communication Proceedings, Union Agency (2012) 35–43
10. Mory, M., Pukall, M., Köppen, V., Saake, G.: Evaluation of Techniques for the Instrumentation and Extension of Proprietary OpenGL Applications. In: 2nd International ACM/GI Workshop on Digital Engineering (IWDE), Magdeburg, Germany (2011) 50–57
11. Mohr, A., Gleicher, M.: HijackGL: Reconstructing from Streams for Stylized Rendering. In: Proceedings of the 2nd International Symposium on Non-Photorealistic Animation and Rendering. NPAR '02, New York, NY, USA, ACM (2002) 13–20 and 154
12. Hintjens, P.: ØMQ – The Guide. iMatix Corporation, http://zguide.zeromq.org/
13. Hunt, G., Brubacher, D.: Detours: Binary Interception of Win32 Functions. In: Proceedings of the 3rd conference on USENIX Windows NT Symposium - Volume 3. WINSYM '99, Berkeley, CA, USA, USENIX Association (1999) 14
14. Khronos Group: Enumerant and Function Registry, http://www.opengl.org/registry/#specfiles
15. Khronos Group: OpenGL Software Development Kit Documentation, http://www.opengl.org/sdk/docs/
16. Reiners, D.: OpenSG: A Scene Graph System for Flexible and Efficient Realtime Rendering for Virtual and Augmented Reality Applications. PhD thesis, TU Darmstadt (2002)
17. Haringer, M., Beckhaus, S.: Dynamic Visual Effects for Virtual Environments. In: WSCG (Full Papers). (2010) 49–56