MegaMol™—for Fun and Profit

Sebastian Grottel*
Computer Graphics and Visualization, TU Dresden

Guido Reina†
VISUS
University of Stuttgart

Michael Krone‡
VISUS
University of Stuttgart

Christoph Müller§
VISUS
University of Stuttgart

Thomas Ertl¶
VISUS
University of Stuttgart

Abstract

Research in scientific visualization often utilizes small software prototypes for proof-of-concept implementations, performance evaluation, and image generation. In large-scale and particularly in long-running research projects, especially aiming at big data, these prototypes quickly reach their limitations, at least as soon as scientists from application domains start using the software. In this position paper we present our experience in creating MegaMol™ [3], an open-source visualization framework for large particle-based data. Having started as research prototype for high-performance computer graphics, the framework has grown over more than nine years and has become a software actively used by several Ph.D. students and collaboration partners from physics, bio-chemistry, thermodynamics and material science. We highlight aspects of the development process and long-term software architecture decision-making, which we identified to be critical to successfully deliver stable software releases. We discuss the increased work load required for defensive programming, error handling, testing, and software maintenance. MegaMol™ is developed at a German university by Ph.D. students and postdoctoral researchers, without the help of dedicated programmers.

Index Terms: D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement—Extensibility; K.6.1 [Management of Computing and Information Systems]: Project and People Management—Life Cycle; J.2 [Computer Applications]: Physical Sciences and Engineering—Chemistry/Physics

1 Introduction

The development of our open-source visualization system MegaMol™ started in 2006. It originates from a joint research project between biologists, physicists, material scientists and visualization experts working with large, particle-based data. Within this project, we faced the need for high-performance GPU-accelerated computer graphics to achieve diverse interactive visualizations for our project partners. We designed our software to be modular and to support rapid visualization prototype development, that is, to provide the necessary flexibility to visualization researchers. In addition, this approach allows central parts of the software to reach a mature and stable state. Such continuity enables the software to be usable by application domain scientists, which is usually not the case for typical research prototypes illustrating one single concept or algorithm. The figures in the appendix show various examples of visualizations and application domains that are supported by MegaMol™.

*e-mail: sebastian.grottel@tu-dresden.de
†e-mail: guido.reina@visus.uni-stuttgart.de
‡e-mail: michael.krone@visus.uni-stuttgart.de
§e-mail: christoph.mueller@visus.uni-stuttgart.de
¶e-mail: ertl@vis.uni-stuttgart.de

2 Development

MegaMol™ is written in C++, supporting Windows and Linux platforms, and primarily uses OpenGL and CUDA. The framework is strongly tailored towards fast rendering. As a consequence, the software itself only provides a thin layer above OpenGL avoiding overhead during rendering while exposing low-level functionality required for cutting-edge graphics programming. Thus, a central paradigm is that data management and layout on the CPU side has to fit the requirements of the GPU and strictly follows a zero-copy paradigm. Here MegaMol™ significantly differs from other visualization frameworks existing when we started the development.

The architecture supports different front ends. For instance, there are GUIs for specific application tasks using the MegaMol™ back end as engine. The software itself also supports synchronized execution on clusters (e.g., for rendering on large displays). All functionality in MegaMol™ is provided as so-called Modules. The Modules can be arranged in arbitrary call graphs, where control flow and data flow follow a pull paradigm. An in-depth description of the architecture can be found in [3].

Modularization The Modules, encapsulating functionality in MegaMol™, communicate via strongly typed channels dubbed Calls. Each Module can have an arbitrary number of typed user-adjustable parameters (e.g., numerical values, enumerations, strings, or file names). Developers explicitly specify via which Calls Modules communicate. This approach has several benefits: Task-specific functionality can be composed from an ever-growing set of operations and visualizations provided by existing Modules (c.f. Fig. 1). The separation into Modules establishes strong code ownership, thus minimizing the potential for conflicts. After defining the interfaces, Modules written by other developers can be treated as black boxes. Specifically, the strongly-typed channels allow for setup and runtime checks.

Further abstraction is provided by plugins, grouping related Modules and Calls. This allows for removing more specialized functionality or specific implementations from the framework's core library. For example, a dedicated plugin containing CUDA-accelerated functionality can be removed if a computer with a graphics card by AMD or Intel is used. This way, the core library retains as few dependencies as possible. Note that inter-dependencies between plugins are allowed.

Defensive Programming & Client-site Debugging Incomplete error handling might be the most severe issue of research prototypes. Reusable software requires defensive programming, handling error states and exceptions where appropriate to ensure consistent state and spot errors. MegaMol™ supports the developer with corresponding utilities as well. Developers can of course use debuggers and standard mechanisms like assertions, tracing or even low-level `printf` outputs to investigate and resolve issues.

However, this is neither feasible nor appropriate for an end user working with the software. As such, MegaMol™ is designed to not always run under the supervision of the developers in the controlled environment of our lab. Access to the debugging and reporting features are thus limited. To compensate, MegaMol™ provides a logging mechanism allowing non-expert users to provide detailed feedback to the developers in an error case. The verbosity level can

be adapted not to clutter logs during normal operations, e.g., output of info messages, warnings, or errors. Consequently, using this mechanism in our code allows for *client-site debugging* by making the log files available to the developer.

Run-time Configuration & Usability The graph of modules determining the actual functionality of MegaMol™ can be created using a graphical utility, the so-called MegaMol™ Configurator. This is a stand-alone C# software, fully supported on Linux using Mono. It queries an actual MegaMol™ installation about the available Modules and their interfaces. Using this information, the Configurator allows the user to only construct valid module graphs.

The result is a project XML file, encompassing the module graph as well as all parameter values provided for the Modules. All parameter values can be edited at runtime within MegaMol™, but can also be set to initial values using the Configurator. During execution, the user can save changes to project files. This ensures reproducibility and archiving of whole visualization states, even including low-level parameters like camera settings. This provides a basic level of provenance if states are stored in different files.

The default front end of MegaMol™ dynamically generates a GUI from the active modules. GUI controls are chosen based on the type of the parameters and are grouped hierarchically. The implementation uses AntTweakBar (http://anttweakbar.sourceforge.net). While this gives the user full control over all parameters, no task-based structure exists in this GUI. Therefore, we have created task-specific GUIs for certain applications that present only the relevant parameters to the user and offer higher-level widgets for more complex tasks and interactions (cf. Fig. 2).

Besides a GUI, documentation is an important factor for usability of software. The MegaMol™ project website provides a wiki and manuals including instructions for building and using MegaMol™, as well as tutorials and example files and projects.

3 Deployment

MegaMol™ is primarily distributed via it's project website (http://www.megamol.org). On this website, we provide packaged releases as well as direct access to the source code repositories.

Release Cycles vs. Continuous Integration MegaMol™ is under active development. Using continuous integration seems like an obvious choice. However, this makes supporting external users much harder, as they would use subtly different versions harboring vastly different issues. Additionally, forcing users to continuously update to not entirely tested software is not an option either.

Therefore, we opted for release cycles of tested and stable versions. For example, project partners and students always use the latest release. Of course, quick bug fixes can still easily applied based on these releases, although not officially included in the release itself. This way we reduce the number of different versions in active use, which also minimizes the effort for client-site debugging. Close project partners who require features under active development (e.g. for ongoing research projects) can still get direct access to the current software version not released yet.

Build System, Binary Releases, and Open Source Although MegaMol™ can run on Windows and Linux, the main development takes place on Windows using Microsoft Visual Studio. For Linux, we opted for the widely-used CMake build system (https://cmake.org/). Source code is available through our SVN repository. Especially for end users, we provide binaries of the latest release for Windows and Linux on our website. The Linux binaries are always compiled on the latest Ubuntu LTS distribution. Of course, since MegaMol™ is open source, our code can be re-used in other software projects. For example, Jurcik et al. [5] extended our Solvent Excluded Surface computation to support transparency, and Guo et al. [4] implemented a hierarchical ray casting of biomolecular data, extending functionality that was available in MegaMol™.

Maintenance & Long-term Commitment A large software framework like MegaMol™, aimed both at developers as well as at end users, requires a substantial amount of maintenance. We use bug tracking and feature request systems to organize tasks. The main developers of MegaMol™ are Ph.D. students and postdoctoral researchers, that is, scientists who are not intended to spend much time on system development and maintenance. However, not fixing bugs or adding new critical features will make a software project lose users rapidly, all of them in the worst case. For many of our visualization research projects, however, having users was hugely beneficial or even crucial in the past. We, therefore, have to carefully balance how much work to put into research and how much into maintenance. This can be seen as a long-term investment.

4 Conclusion

When we started developing MegaMol™ almost a decade ago, we clearly underestimated the huge workload this would cause. Repeatedly, the question was raised whether a Ph.D. student could afford this non-research-related workload. In hindsight, we believe it is worth to develop and maintain your own code base for problems not yet solved satisfactorily. Given the range and complexity of visualizations we can now create with MegaMol™, we find our choice justified. All participating institutions and people are today satisfied with the result, concerning both software and scientific publications. What remains difficult is the acceptance and appreciation of the scientific community. Most of the development work invested in a software system cannot be published. We believe the most important thing for a scientific software project is the continuous commitment of the individual developers against all odds.

Acknowledgments This work was partially funded by Deutsche Forschungsgemeinschaft (DFG) as part of SFB 716.

References

[1] S. Grottel, P. Beck, C. Müller, G. Reina, J. Roth, H.-R. Trebin, and T. Ertl. Visualization of Electrostatic Dipoles in Molecular Dynamics of Metal Oxides. *IEEE TVCG*, 18(12):2061–2068, 2012.

[2] S. Grottel, J. Heinrich, D. Weiskopf, and S. Gumhold. Visual analysis of trajectories in multi-dimensional state spaces. *CGF*, 33(6):310–321, 2014.

[3] S. Grottel, M. Krone, C. Müller, G. Reina, and T. Ertl. MegaMol - A prototyping framework for particle-based visualization. *IEEE TVCG*, 21(2):201–214, 2015.

[4] D. Guo, J. Nie, M. Liang, Y. Wang, Y. Wang, and Z. Hu. View-dependent level-of-detail abstraction for interactive atomistic visualization of biological structures. *Comput. Graph.*, 52:62–71, 2015.

[5] A. Jurcik, J. Parulek, J. Sochor, and B. Kozlikova. Accelerated visualization of transparent molecular surfaces in molecular dynamics. In *IEEE PacificVis*, pages 112–119, 2016.

[6] M. Krone, F. Frieß, K. Scharnowski, G. Reina, S. Fademrecht, T. Kulschewski, J. Pleiss, and T. Ertl. Molecular surface maps. *IEEE TVCG*, 23(1), 2017.

[7] M. Krone, M. Huber, K. Scharnowski, M. Hirschler, D. Kauker, G. Reina, U. Nieken, D. Weiskopf, and T. Ertl. Evaluation of visualizations for interface analysis of sph. In *EuroVis - Short Papers*, pages 109–113, 2014.

[8] C. Müller, M. Krone, K. Scharnowski, G. Reina, and T. Ertl. On the utility of large high-resolution displays for comparative scientific visualisation. In *VINCI*, volume 8. ACM, 2015.

[9] K. Scharnowski, M. Krone, G. Reina, T. Kulschewski, J. Pleiss, and T. Ertl. Comparative visualization of molecular surfaces using deformable models. *CGF*, 33(3):191–200, 2014.

[10] K. Scharnowski, M. Krone, F. Sadlo, P. Beck, J. Roth, H.-R. Trebin, and T. Ertl. 2012 IEEE visualization contest winner: Visualizing polarization domains in barium titanate. *IEEE CG&A*, 33(5):9–17, 2013.

[11] J. Staib, S. Grottel, and S. Gumhold. Visualization of particle-based data with transparency and ambient occlusion. *CGF*, 34(3):151–160, 2015.