University of Stuttgart
Visualization Research Center (VISUS)

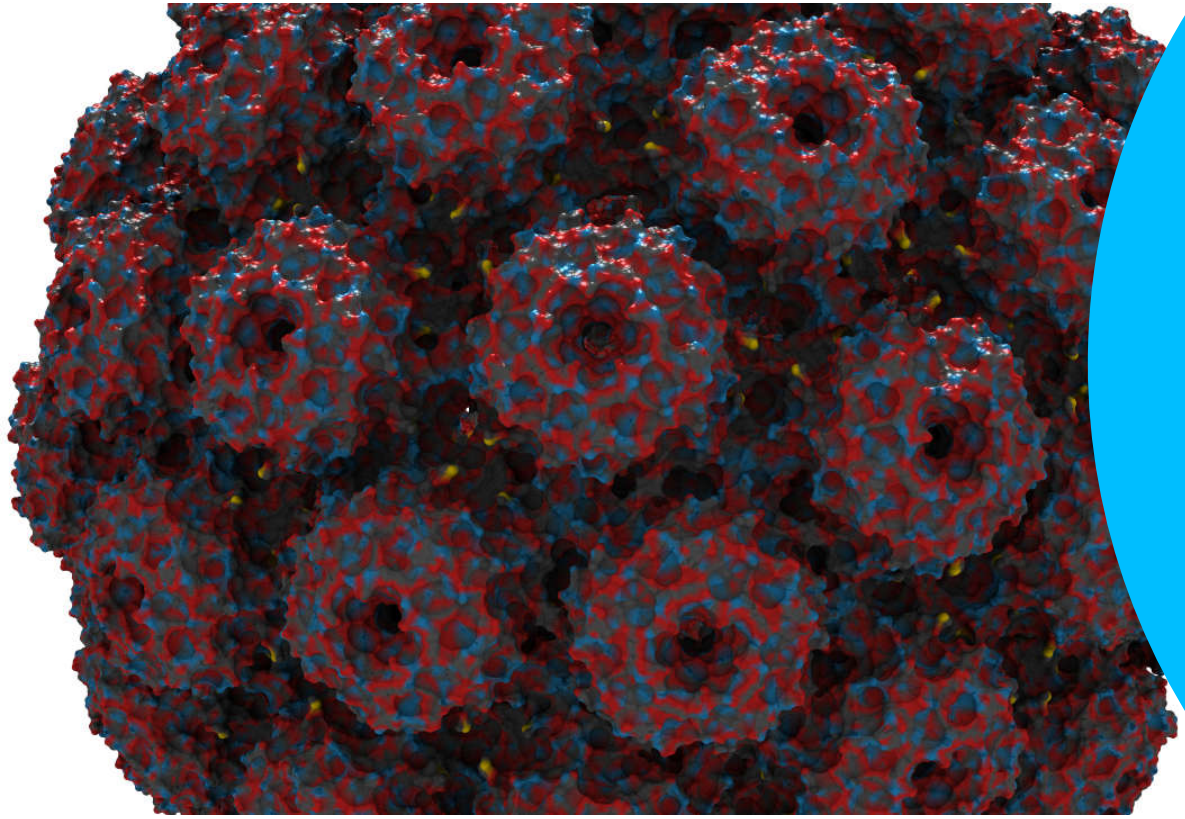


image source: MegaMol developer team

MegaMol – Sustainable Visualization Software for Research and Education

Patrick Gralka, Matthias Braun, Guido Reina, Thomas Ertl

Visualization Research Center
University of Stuttgart

What is MegaMol?

- Visualization Framework
  - Primarily for scientific visualization
  - Focus on extensibility and adaptability
- First version from 2007
  - In development ever since

Brief summary of DFG sustainability project follows

- Some details about revised decisions and lessons learned

image source: MegaMol developer team

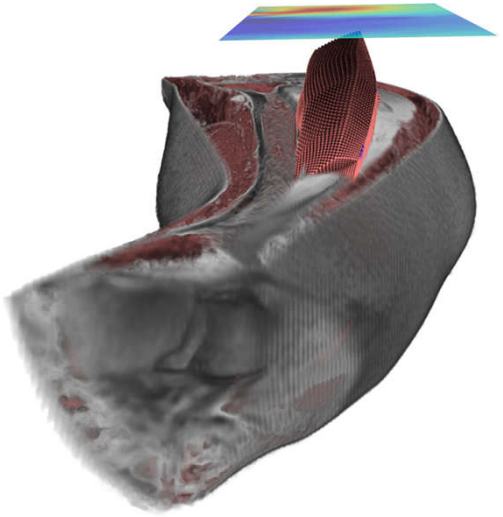


image source: MegaMol developer team

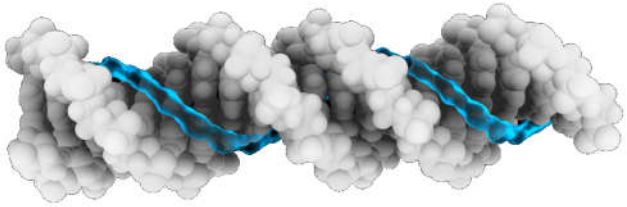
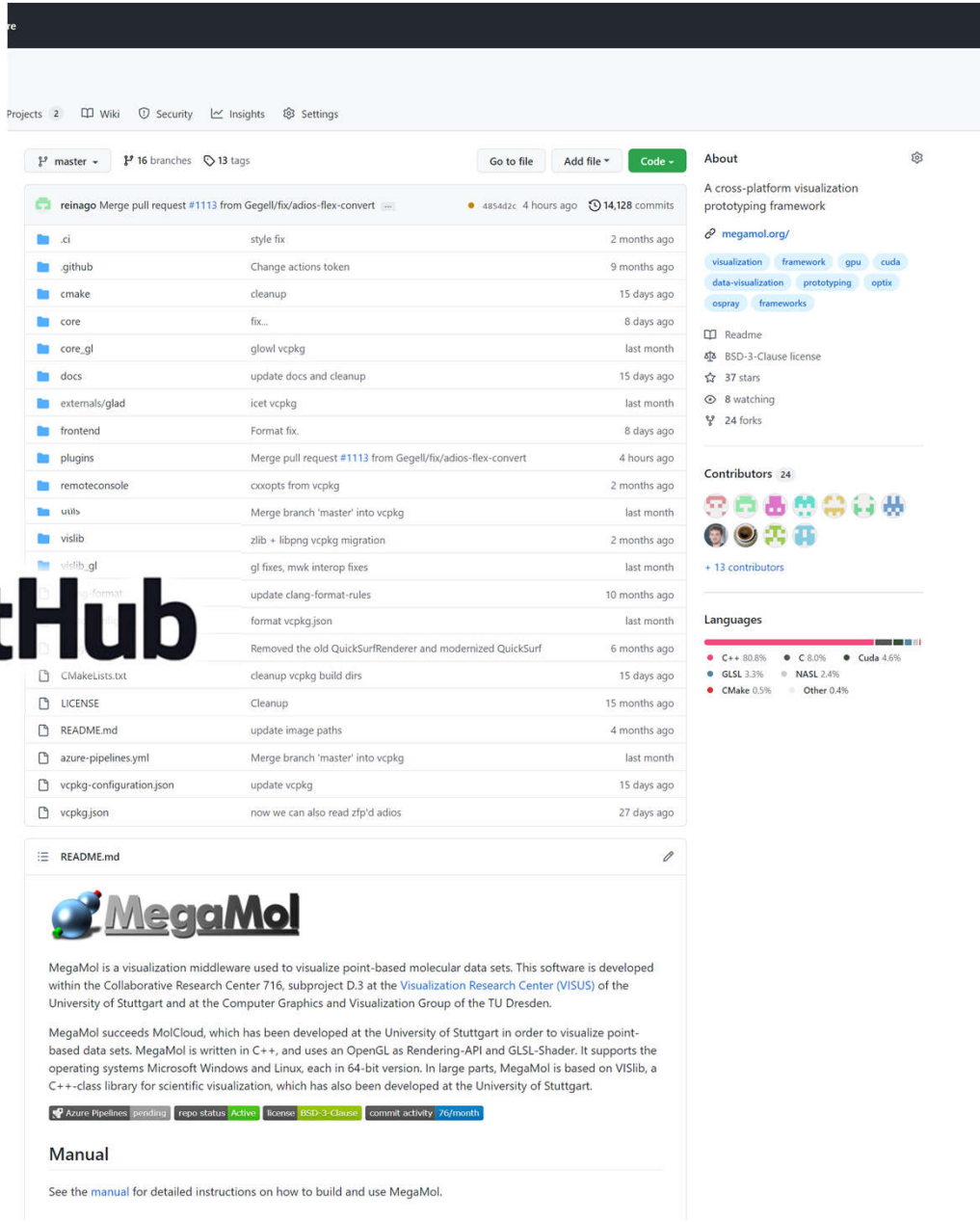


image source: MegaMol developer team

Open-Source Platform

- Status quo ante:
  - Core components, plugins, and support libraries in different, self-hosted SVN repositories
- Goals:
  - Easy access and dissemination
  - Modern development workflow
- Steps taken:
  - Merging of different code bases
  - Migration to git and *public* GitHub repository
- Result:
  - Software gained visibility, easier to contribute
  - “Statement” about openness
  - Pre-requisite for subsequent sustainability steps

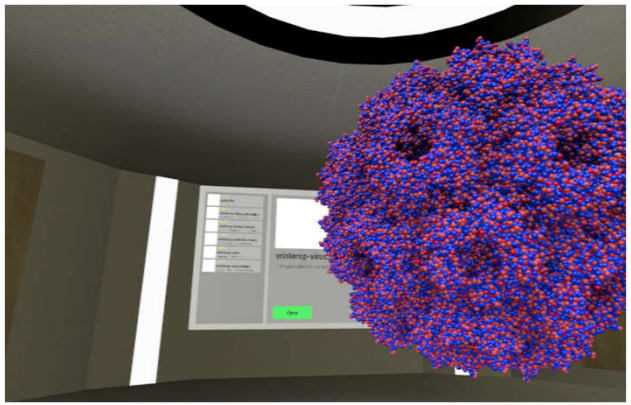
GitHub



Build Process


- Status quo ante:
  - Copy content of different repositories together
  - Specialized tools & scripts; platform-dependent build workflows
- Goals:
  - "One click" experience (collaborators!)
  - Use well-known tools for automation, easier integration of external libraries and maintenance
- Steps taken:
  - Completely new CMake-based build environment
  - Externals (43) via vcpkg (CMake & C++)
- Lessons learned/Open challenges:
  - Disregard minor issues and personal opinions about de-facto standards!
  - First from-scratch build very expensive, needs dependency caching


Portability



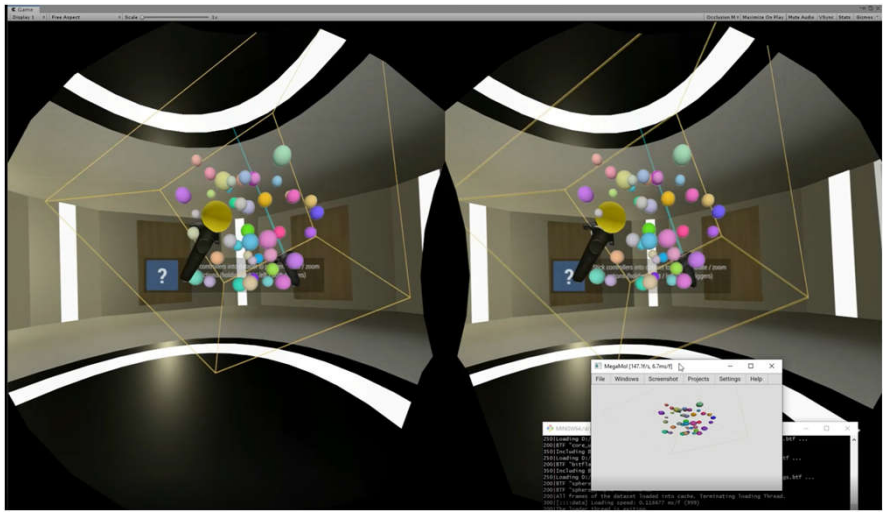


image source: MegaMol developer team

- Status quo ante:
  - Executable with different functionality depending on shipped (plugin-) DLLs (prone to pollution)
  - Narrow deployment options
  - "Out of the ordinary" builds can cause disruptions
- Goals:
  - More deployment options
  - Eliminating possibilities of failure
- Steps taken:
  - Refactoring of front end for simplicity & flexibility
  - A single, integrated executable; functionality dependent on build options
- Open challenges:
  - OS-independent solutions; Docker/Singularity; virtualization in general

Continuous Integration / Tests

- Status quo ante:
  - Local builds, manual releases, no regression
- Goals:
  - Always compiling and working main branch
  - Merge only compiling feature branches
- Steps taken:
  - CI on local resources via Azure DevOps
  - "Visual" regression tests
- Lessons learned/Open challenges:
  - Anything else is even more "expensive" for expanding code/user base
  - Thorough testing too expensive to do continuously (15’ for tiny coverage)
  - No GPU worker nodes yet – centralized resources desirable

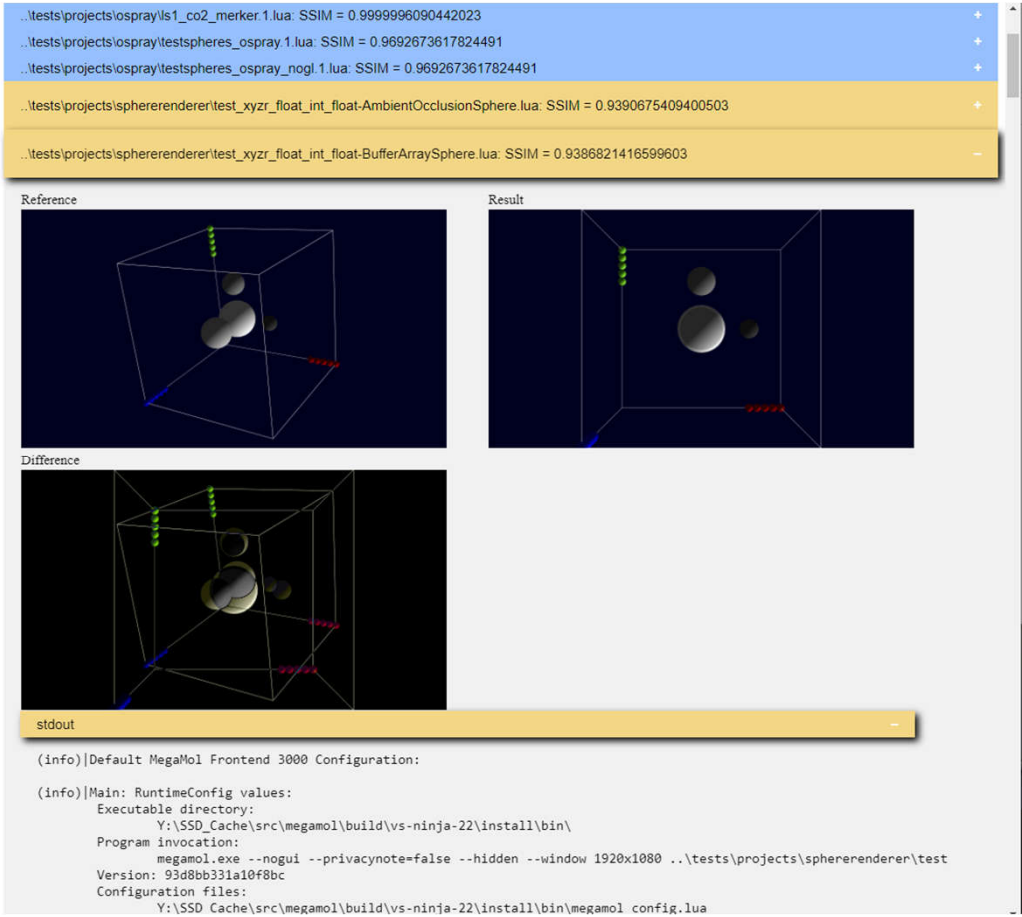


image source: MegaMol developer team

User Experience

- Status quo ante:
  - Separate executables to configure visualization pipeline and run it
- Goals:
  - Streamlined UX
  - Manipulation of visualization pipeline
- Steps taken:
  - Single executable
  - Complete rework of the user-facing front end
  - Better widgets plus direct manipulation
- Open challenges:
  - Portable (multi-modality) UI
  - Performance decoupling / remote UI

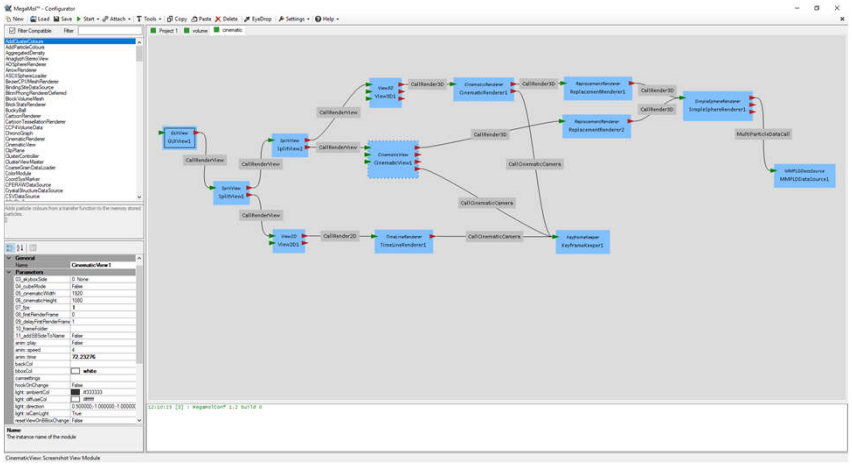


image source: MegaMol developer team

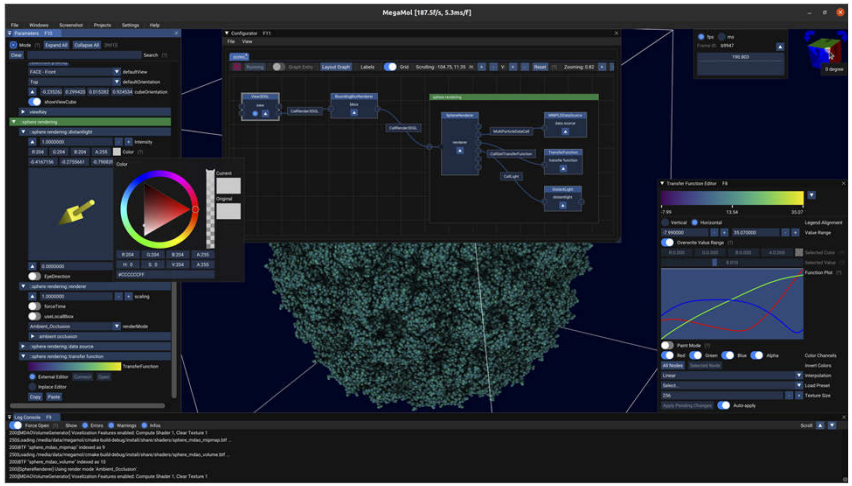


image source: MegaMol developer team

Developer Experience

- Status quo ante:
  - Pre-C++11 code; touched by many, understood by few
- Goals:
  - Better extensibility; Literal code
- Steps taken:
  - Removal of opaque implicit automatisms
  - Deduplication of code; consolidation of functionality
- Lessons learned:
  - Make sure a problem is solved at the correct abstraction level
    - Do not be afraid to introduce that abstraction level – you will do it in 2 years anyway
  - You *think* you are solving a different problem, so you write specialized code
    - Often you are not!
    - At least one person needs to see the big picture

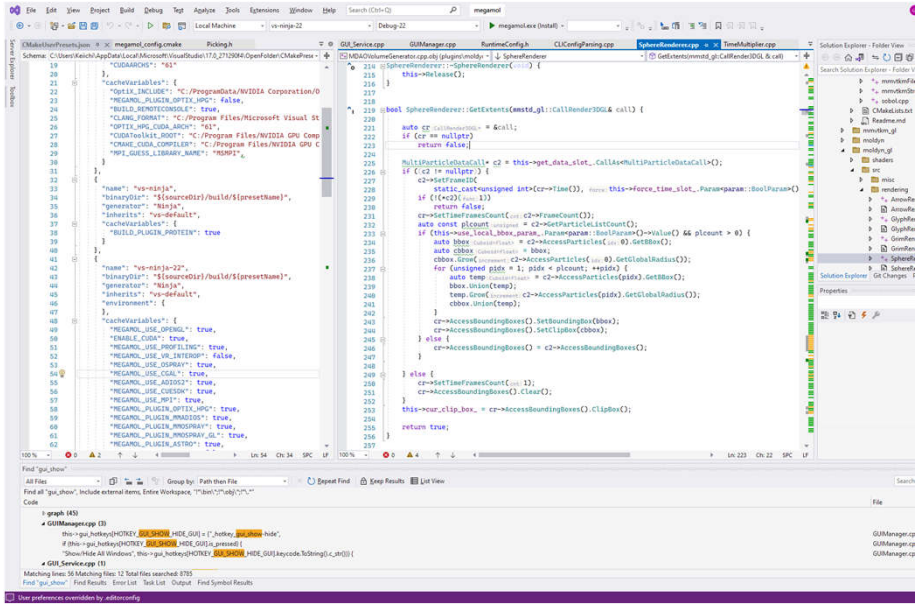


image source: MegaMol developer team

Community and Outreach

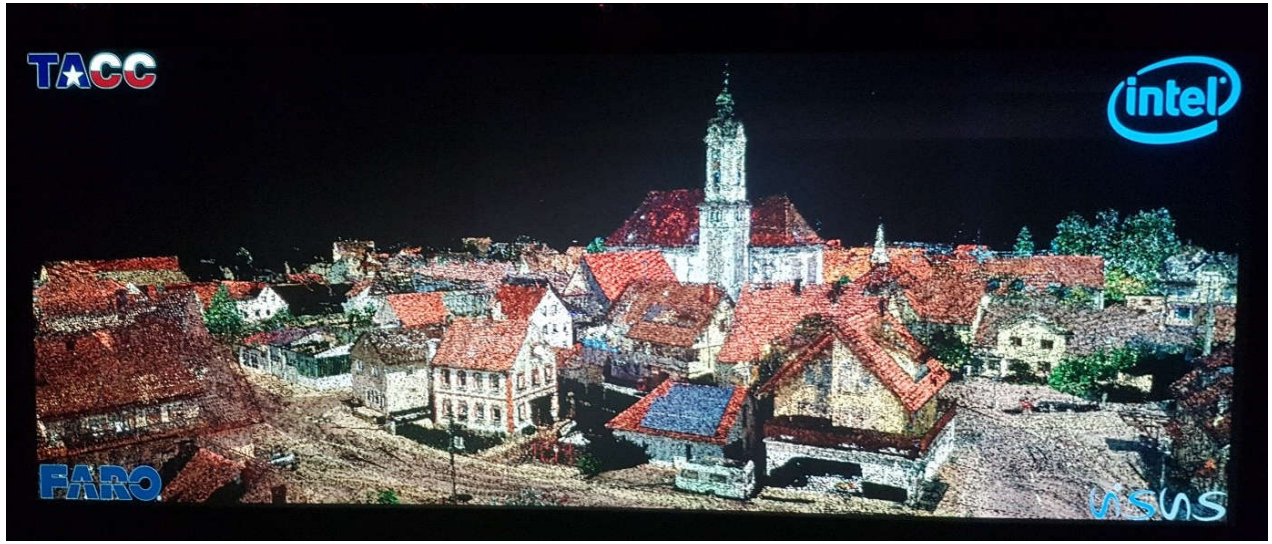


image source: MegaMol developer team

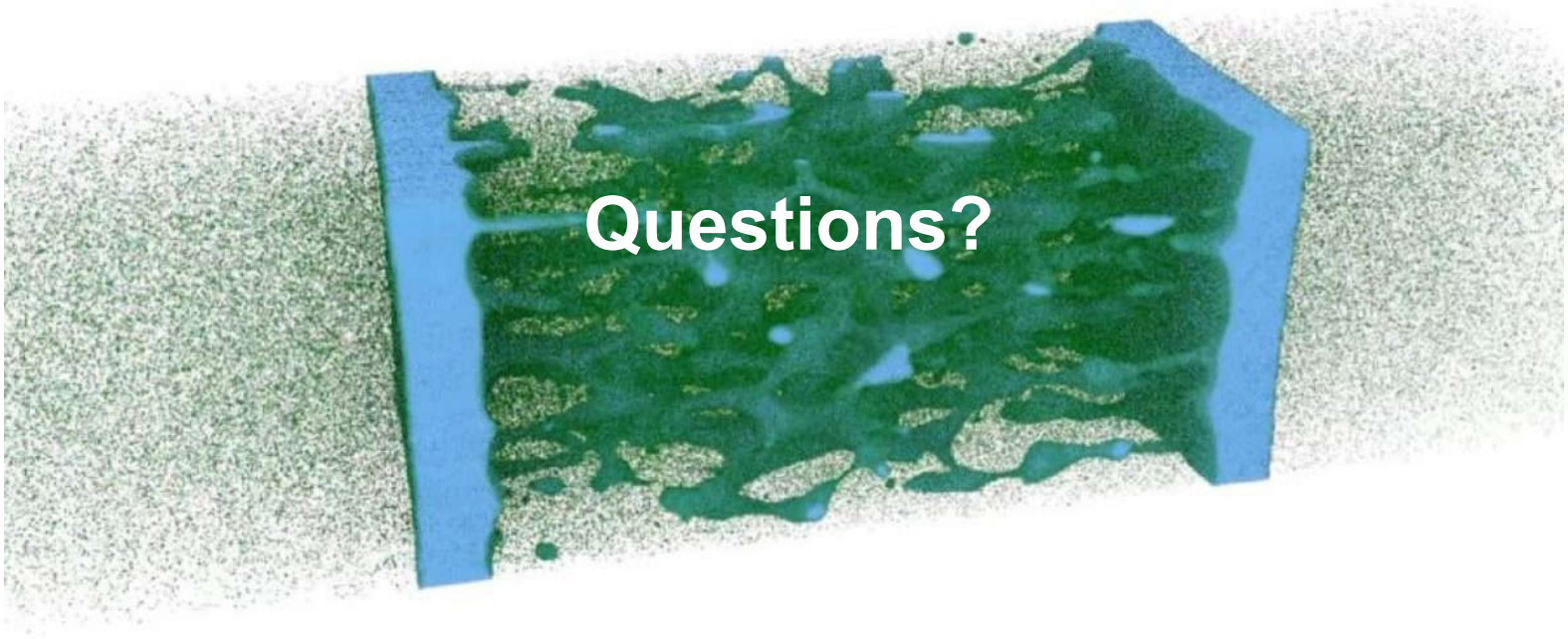
- Status quo ante:
  - No extensive dissemination
  - Hidden gem, invite only vibes
- Goals:
  - Attract new users
  - Attract broader support
- Steps taken:
  - Active engagement on GitHub, better documentation, …
  - Leveraging science/industry cooperation projects
- Lessons learned/Open challenges:
  - Entering wide spectrum between “in-house tool” and “actual product”
    - Puts strain and responsibility on people not paid, trained, rewarded for the tasks
    - Investigations on their own: Shonan / VisGap workshop series with impressive resonance

Conclusion

- MegaMol continuously strives to improve its FAIRness
  - Development is open & public
  - Improved build & usage experience
  - Improving platform support, improving data ingestion
  - Mission statement: replacing one-shot prototypes by reusable software

- Opaque implicit convenience for specific use cases is not desirable
  - Have an abstraction that allows for concise explicit formulation instead!
- Do not put off refactoring, especially when it reduces code (complexity)
  - Cascade effect (both ways!)

Thank you for your attention



Many thanks go to the main MegaMol contributors (by #commits)

Matthias Braun, Sebastian Grottel, Guido Reina, Tobias Rau, Patrick Gralka, Moritz Heinemann, Karsten Schatz, Christoph Müller, Sergej Geringer, Katrin Scharnowski, Michael Becher, Michael Krone, Alexander Straub, Christoph Schulz, Dominik Sellenthin, Bertram Thomaß, Daniel Kauker, Oliver Fernandes, Thomas Marmann, Florian Frieß, Michael Wörner, Alex Heller, Joachim Staib, Nina Dörr

Special thanks to the DFG for funding project 391302154

Backup & Details

How large is MegaMol?

- 2022-09-15 10:00 494 KLOC @ master
- 2017-09-15 10:00 455 KLOC @ master
- >300 modules
- 15 services

„Good“ commits nearly remove as much code as they add ;)

* cloc --include-lang=C++,"C/C++ Header",CMake,C,CUDA,GLSL,Python

Examples: You are not solving a different problem

- System state: XML
- Remote / Syncing: binary protocol
- Scripting for convenience and automation: Lua

→Everything is Lua, also correct abstraction for template/example visualizations per data type

- Sync state for remote
- Sync state from GUI
- Get state for reproducibility

→No graph walkers, change callbacks!

Some examples for implicity functionality

- Auto-loading of known formats via graphs deposited in the source code
- Self-configuring MegaMol based on available dlls