Appendix A
OpenGeoSys-6

by Dmitri Naumov, Lars Bilke, Thomas Fischer, Yonghui Huang, Christoph Lehmann, Xing-Yuan Miao, Thomas Nagel, Francesco Parisio, Karsten Rink, Haibing Shao, Wenqing Wang, Norihiro Watanabe, Tianyuan Zheng, Olaf Kolditz

A.1 OGS-6: Development and Challenges

A.1.1 Introduction

Current development of the general multi-physics simulation platform OpenGeoSys concentrates mainly on the new version 6 of the software. OGS-6 is a major rework of the previous version OGS-5 motivated by several improvement possibilities described in detail in the previous benchmark book edition Kolditz et al. (2016b) Appendix A.

The software development itself occurs on an open shared platform, making both amenable to scrutiny by the scientific community and public authorities. Besides describing development workflows that assure code quality, we highlight how code performance and accuracy are routinely tested in an automated test environment as well in this benchmarking initiative.

In the following sections are an overview of the current implementation, a description of the testing and continuious integration environment, and finally an overview of the high-performance-computing setups and their parallel performance.

A.1.2 Overview of Processes

Current overview of the capabilities is partially covered by benchmarks in this book. and includes following process implementations:

O. Kolditz et al. (eds.), *Thermo-Hydro-Mechanical-Chemical Processes in Fractured Porous Media: Modelling and Benchmarking*,
Terrestrial Environmental Sciences, https://doi.org/10.1007/978-3-319-68225-9

- variations of heat and component transport equations: ground water flow (elliptic steady-state equation), heat conduction, hydro-thermal process, pressure based liquid flow process, inert component-transport, Richards flow, two-phase flow (pressure-pressure and pressure-saturation formulations), temperature dependet two-phase flow (pressure-pressure formulation), (all of the parabolic kind),
- coupled mechanics processes: small deformation process and hydro-mechanical process, both with Lower-Interface-Element extensions, thermo-mechanical and thermo-hydro-mechanical processes, and a phase-field formulation,
- thermal energy storage process

A.1.3 Implementation of Workflows

For an implementation of a new process and working with the existing implementations a developer has several flexibilities in the choice of the non-linear solver, the linear solver availabilities, and the pre- and post-processing options.

There are two implementation possibilities for the processes: monolithic and staggered schemes. Independent of this choice, to resolve the non-linearities of the equations a fixed-point iteration and Newton-Raphson method are available. Most processes are implemented using the Newton-Raphson scheme and therefore are implemented monolithically.

The staggered scheme (together with fixed-point iterations) allow for quick working implementations of coupled multi-field processes, while choosing the Newton-Raphson scheme makes it possible to solve highly non-linear equations.

For the solution of the linear equation systems currently there are three major libraries supported: Eigen (Guennebaud et al. 2010), LIS (2017), and PETSc (Balay et al. 2016). These give a big variety of linear equations solvers, both direct and iterative, for the users to experiment. The Eigen library, furthermore, provides wrappers to external solvers from the PaStiX, SuiteSparse, SuperLU, and Intel MKL libraries.

For the input and output of the mesh geometries the Visualization Toolkit library (VTK) with its rich set of manipulation and visualization tools; primarily Paraview—a graphical front-end to the VTK libraries, and the Python bindings to the VTK giving powerful scripting options.

A.1.4 Transition from OGS-5

The benchmarking of the code is continued in the new version of OGS. Comparisons with the analytical solutions is one of the requirements for the newly implemented processes, where it is possible, and comparison to the OGS-5 (or other codes like FEFLOW FEFLOW 2017 or FEBio FEBio 2017) if same implementation is available. The OGS-6 development aims to introduce new processes and solution techniques, which were not possible in the former version.

A.1.5 Heterogeneous Computing

Another challenge is the adaptation to heterogeneous computing, where different kinds of processors are utilized to achieve higher efficiencies. Examples of heterogeneous systems are, for example, computer systems using CPU and GPU cores simultaneously.

A.2 Software Engineering and Continuous Integration

by Lars Bilke

OGS-6 is developed as an open-source project by using the GitHub platform for code hosting, code review, and bug and issue tracking (OGS 2017c). A Jenkins continuous integration (CI) server (OGS 2017b) builds and tests the software on every proposed change (pull request) on supported platforms (Windows, Linux, Mac Desktop systems and Linux HPC cluster systems). See Kolditz et al. (2015) Appendix-B for a general introduction.

The QA process is automated and defined in *Jenkins Pipeline Domain Specific Language* (DSL, Jenkins 2017a) scripts which are part of the source code to allow the concurrent testing of multiple development branches. We aim to identify reusable pipeline definitions, such as CI steps typical for a C++/CMake-based project (configuration, compilation, testing, artifact deploying), which can be shared with other software projects, and integrate those CI steps as a standalone *Jenkins Pipeline Shared Library* (Jenkins 2017b).

Submitting a proposed change (in the form of a GitHub pull request) triggers a CI-job which consist of several sub-jobs with several stages each. A sub-job is created for every supported platform. We test both on bare-metal hardware (e.g. a Linux cluster system) as well on containerized environments using Docker (2017). Docker container providing Linux environments with different compiler setups are defined via the Dockerfile DSL in code as well (OGS 2017a). Sub-job stages group complex steps into logical parts such as configuring, building and testing. After a CI-job finishes the developer gets immediate feedback which sub-job or even stage failed with its corresponding log output. Web-based automated code documentation (with Doxygen) and interactive benchmark descriptions with embedded 3D visualizations (with VTK 2017) are generated. Binaries for all supported platforms are generated and provided for further manual testing if required.

With this setup large parts of the whole software engineering infrastructure and processes are formalized and defined via DSLs in version controllable code-repositories allowing for easy contributing and peer-review on the code, infrastructure and process levels. For citing GitHub-based source code, digital object identifiers (DOIs) can be generated with the Zenodo platform.

A.3 High-Performance-Computing

by Thomas Fischer, Wenqing Wang, Christoph Lehmann, Dmitri Naumov

A.3.1 Why is High-Performance-Computing Necessary?

- For the solution of non-stationary models many time steps are necessary.
- For coupled approaches in cross-compartment topics many iterations between for the particular compartments specialized software may be necessary.
- Large scale domains or complex geological structures lead to meshes containing many cells. Also, in order to improve the accuracy of simulation results or to resolve small scale effects, a finer discretization of the domain is necessary. Doubling the model resolution for d-dimensional domains in each space direction results in 2^d times more mesh elements.
- Multi-physics with complex equations of state and the modeling of chemical reactions requires high numerical effort for both the assembly and for solving the system of linear equations.

Consequently, a high demand of computational resources (CPU time, large memory requirements and fast input/output operations) for the numerical analysis of large scale or complex problems arises.

A.3.2 Parallelization Approaches

Nowadays the computational power of a single core CPU hits its physical limits. Most of the growth of computational power is attributed to parallelization. As a consequence the usual way to tackle the huge computational effort for solving high resolution models is parallel computing. In principle there are two programming approaches for parallel computing, the shared memory approach and the distributed memory approach.

OpenMP (Open Multi-Processing) is a technique that makes use of the shared memory approach and accelerates the program mostly by parallelization of loops. This technique is relatively easy to use. OpenMP benefits from the multi-core architectures. Typically such architectures have a limited number of cores.

One representative of the distributed memory parallelization technique is MPI (Message-Passing Interface). Programs using MPI can utilize many compute nodes in one computation, where each compute node typically has multiple CPU cores. Efficient implementation of algorithms for distributed memory architectures (MPI) is more challenging than the implementations for shared memory (OpenMP) systems.

So far, OGS-6 is parallelized implementing the distributed memory approach, i.e., it uses MPI.

A.3.3 Results

A.3.3.1 Description of the Benchmark Example

As a test, a groundwater flow equation, an elliptic PDE, in a homogeneous medium is solved

$$\nabla \cdot (\kappa \nabla h) = 0, \quad \text{in } \Omega = [0, 1]^3,$$

with Dirichlet-type boundary conditions

$$h = 1, \quad \text{for } x = 0, \qquad \text{and} \qquad h = -1, \quad \text{for } x = 1.$$

The conductivity κ is set to 1. The analytical solution is $h(x, y, z) = 1 - 2x$.

For the numerical solution of the elliptic PDE the domain was discretized in different resolutions from 10^6 to 10^8 hexahedra, see Tables A.1 and A.2.

A.3.3.2 Run Times and Speedup for IO, Assembly and Linear Solver

All run time tests were conducted on the EVE cluster at the Helmholtz Centre for Environmental Research GmbH – UFZ. EVE is a Linux-based cluster with 65 compute nodes where each compute node has 20 compute cores. It has 11.2 TB RAM in total. The compute nodes are connected via a 40 Gbit Infiniband network. The computations are performed on the compute nodes that have 120 GB of RAM.

Since such a cluster is often used in parallel by multiple users, the measurement environment is not always in the same state. In order to avoid that other simulations on the cluster influence the computations, the test problem was partitioned so that always complete compute nodes are occupied. However, the total network traffic and the total IO to external storage on the cluster can not be controlled. For this reason, each run time test is repeated 10 times to exclude or at least minimize external environment influences.

Each compute core participating on the computations, outputs the time measurements for the particular tasks, i.e., mesh reading, assembling, linear solving, writing results. The maximum time for each particular task is chosen to average over the 10 simulations.

Strong scaling: Run times and speedup for IO, assembly, linear solver

The scaling is normally defined as $s(N) = \frac{t(1)}{t(N)}$, where $t(1)$ is the run time using one compute core and $t(N)$ is the run time using N cores. Because the problem is too large to fit in the memory associated with one compute core in Fig. A.1 the slightly changed definition $s_{20}(N) = \frac{t(20)}{t(N)}$ is depicted. The figure is based on the data from the Table A.1. As expected, when more compute cores are deployed the run times decrease for the assembly and for the linear solver.

Run times for reading the mesh drops down until the employment of 60 cores and stay around 2 s for higher number of cores. This could be an effect of the GPFS file

Table A.1 Fixed discretization $236 \times 236 \times 236$ ($= 13\,144\,256$) hexahedra, all run times in seconds

#cores	Reading mesh	Output time steps 0 and 1	Assembly	Linear solver
20	6.32	3.99 + 4.62	3.81	56.94
40	4.74	4.01 + 5.60	1.93	29.19
60	1.78	7.12 + 7.31	1.33	18.01
80	2.15	11.77 + 12.46	1.03	14.42
100	1.72	14.39 + 18.71	0.83	11.59
120	2.31	23.98 + 31.50	0.69	9.73
140	2.80	13.85 + 19.47	0.61	8.41

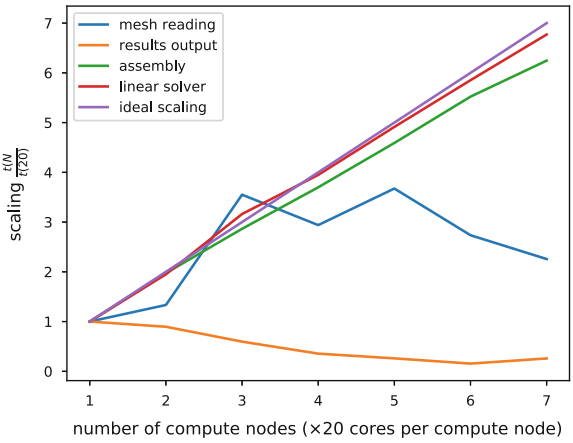

Fig. A.1 Scaling for different tasks of the simulation

system of the cluster that has to distribute smaller and smaller parts of the binary input files to more and more compute nodes. The overhead connected to this could be one reason why the reading of the mesh does not scale as expected.

While the input consists of a small number of binary files, the output of the simulation results is done in a different way at the moment. Each process opens its own file and writes its sub-domain specific data. For a large number of processes this results in opening a large number of files which is typically a time consuming operation on GPFS file systems. Figure A.1 indicates some potential for improvement, especially for the output of simulation results.

Weak scaling: Run times for IO, assembly, linear solver

To investigate the weak scaling of OGS-6 the discretization and the number of compute cores for the simulation is chosen such that the workload on a core is almost the same, as illustrated in the third column of Table A.2. The run time measurements are done with two different versions of OGS-6, one from February 2017 (last three rows in the table) and one from September 2017 (first three rows in the table).

Table A.2 All run times in seconds

Discretization #hexahedra	#cores	DOFs per core	Reading mesh	Output time steps 0 and 1	Assembly	Linear solver
184 × 184 × 184	20	311 475	3.55	2.51+2.53	1.86	21.59
232 × 232 × 232	40	312 179	4.14	4.89+5.91	1.90	27.34
292 × 292 × 292	80	311 214	4.80	10.71 + 12.43	1.89	33.58
292 × 292 × 292	80	311 214	3.84	3.98+4.42	2.39	37.39
368 × 368 × 368	160	311 475	5.01	4.74+5.48	2.39	46.21
465 × 465 × 465	320	314 202	6.73	5.52+6.41	2.38	69.19

In a preprocessing step the mesh is partitioned and stored in a small number of binary files. The time for reading of the mesh for the simulation slightly decreases. This could be caused by distribution of the data via the GPFS file system. There is a significant variation in the run times of the output which needs further investigation.

The run times for the assembly, being almost the same for different number of process, demonstrate that this part scales weakly very well. Furthermore, the jump in the assembly times between the two version (02/2017 vs. 09/2017) is caused by switching between dynamically allocated local matrices (02/2017) and statically allocated local matrices (09/2017).

The run times for solving the systems of linear equations shows that it becomes more difficult to solve the system the bigger it is, i.e., the number of iterations increases with the number of degrees of freedom.

A.4 Data Integration and Visualisation

by Karsten Rink, Carolin Helbig, Lars Bilke

In order to cope with increasing amounts of heterogeneous data from various sources, workflows combining data integration and data analytics methods become more and more important. Therefore, data integration and visual data analysis are important elements within the OpenGeoSys workflow concept. Figure A.2 shows a generic workflow for scientific visualisation of environmental data including categories and examples for functionality.

Some general challenges for data integration and visual data analysis for the field of environmental science are summarized in the following list and described in detail in Helbig et al. (2015):

- Multifaceted data: High dimensional parameter spaces
- Multimodal data: Correlation of observation and simulation
- Incorporation of multiple scales: Catchment scale vs sewage network

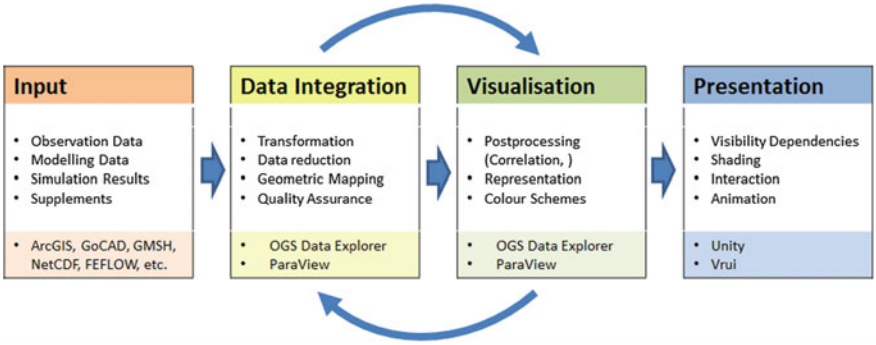

Fig. A.2 Elements of data integration and visual data analysis (Helbig et al. 2017)

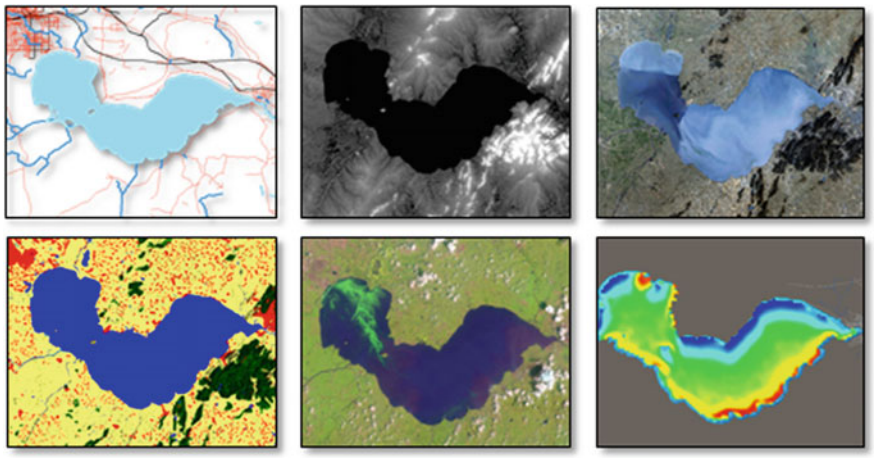

Fig. A.3 Example for data integration tasks: Environmental Information System Chaohu

- Multirun data: Ensembles or multiple scenarios
- Supplemental information: Adding Images/Videos/Diagrams/Papers/etc.

Typical tasks for data integration include, e.g.:

- Visualisation of input- and modelling data as well as simulation results within unified context,
- Data exploration to understand complex data, find correlations and dependencies,
- Detection of potential problems, inconsistencies or missing data.

Figure A.3 shows an example for heterogeneous data sets to be integrated for typical applications in environmental sciences. This includes urban infrastructures, digital elevation models (DEM), surface water quality, land use pattern and modeling results. Data integration is an essential prerequisite for building comprehensive Environmental Information Systems (EIS) (Rink et al. 2016).

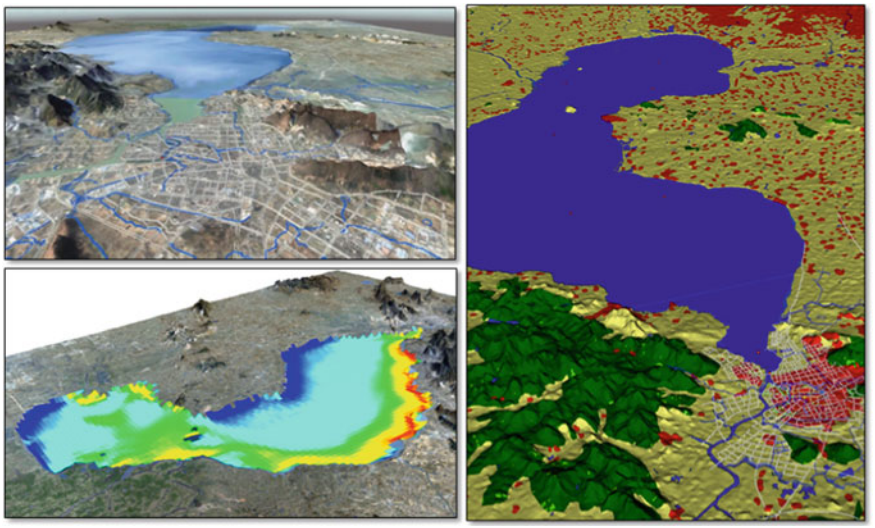

Fig. A.4 Environmental Information System - Chaohu combining urban infrastructures (left top), integrating model results (left bottom), land use information (right)

Figure A.4 depicts elements of the Environmental Information System – Chaohu combining observation data and modeling results within a unified geographical context. Municipal waste water and fertilizers from agricultural production are the main sources for the pollution of Lake Chaohu. This information is the basis for building predictive models concerning the water quality of Lake Chaohu and assessing the effect of remediation concepts.

Data integration and analysis requires appropriate tools for specific disciplines. The OpenGeoSys (OGS) DataExplorer is particularly suited for environmental applications (Fig. A.5). Typical requirements for data analysis are:

- Data Conversion
- Data Representation
- Mesh Generation
- Error Detection
- Transformations
- Detecting and resolving inconsistencies
- Mapping
- Scalar Information
- Visual Metaphors
- Export to Graphics Frameworks

The OGS DataExplorer provides numerous interfaces for data import and export functions mainly for visualization. Several interface to complementary modeling softwares (preferable open source products) are available (Fig. A.6).

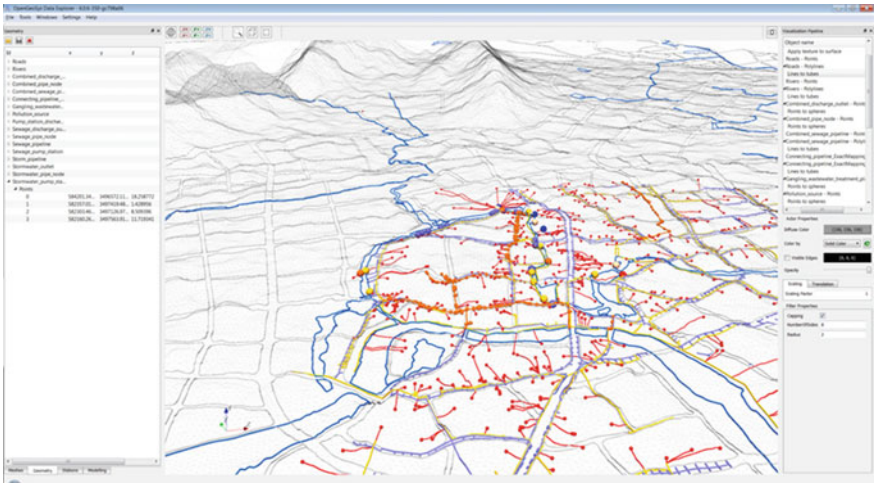

Fig. A.5 OpenGeoSys (OGS) DataExplorer

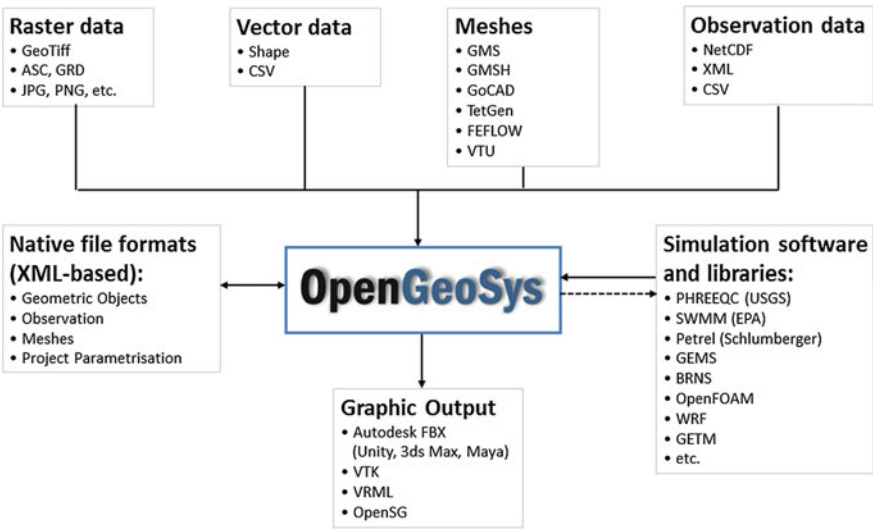

Fig. A.6 OGS Interfaces

Complex data collections for environmental applications benefit from embedding into Virtual Geographic Environments (VGE) (Fig. A.7) The Unity3D environment is used for interactive visualization applications providing a large variety of functionality, e.g.:

- Transfer functions
- Graphics shaders for visual appearance of data objects

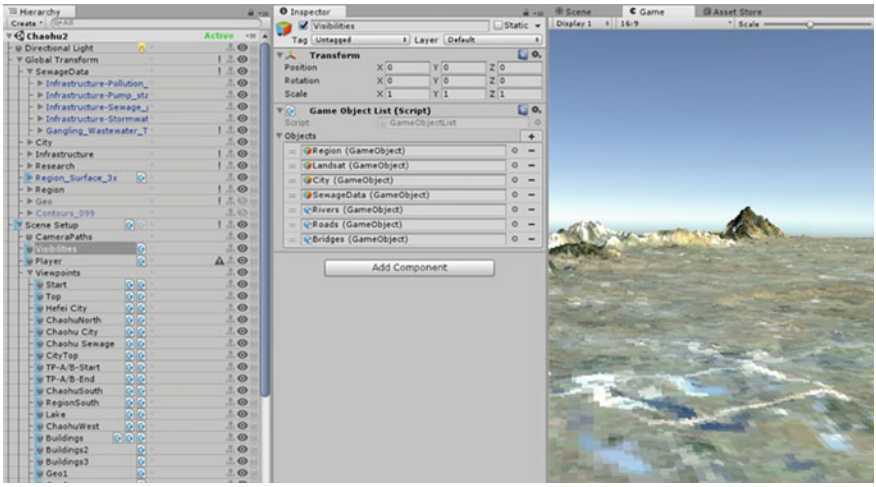

Fig. A.7 Unity3D environment for building Virtual Geographic Environments (VGE)

- Semantic grouping and visibility management
- Viewpoints showing aspects of interest
- Animations of time-dependent data
- View path animations
- Performance optimisation
- Tracking user interaction
- Adding supplemental information (images, videos)

Data integration and visual data analytics provide numerous added values for environmental research, such as:

Data integration within the geographical context:

- Integrated data frame for observation and simulation data
- Visualisation complex and heterogeneous data bases
- Error and inconsistency detection

Data exploration and knowledge transfer:

- Exploration and understanding of complex data sets
- Visualisation of multifaceted/multimodal/multirun data
- Supports discussions between scientists and communication with stakeholders or for public participation

Recent research and progress in Environmental Visualization are presented and discussed in a related workshop series "Visualization in Environmental Sciences (EnvirVis)" which is a regular part of the annual European visualization conference EUROVIS.