Archive of Numerical Software
vol. 3, no. 100, pages 1–8
© 2015

The deal.II Library, Version 8.2

Wolfgang Bangerth[1], Timo Heister[2], Luca Heltai[3], Guido Kanschat[4], Martin Kronbichler[5], Matthias Maier[6], Bruno Turcksin[7], and Toby D. Young[8]

[1]Department of Mathematics, Texas A&M University, College Station, TX 77843, USA, `bangerth@math.tamu.edu`
[2]Mathematical Sciences, O-110 Martin Hall. Clemson University. Clemson, SC 29634, USA, `heister@clemson.edu`
[3]SISSA - International School for Advanced Studies, Via Bonomea 265, 34136 Trieste, Italy, `luca.heltai@sissa.it`
[4]Interdisciplinary Center for Scientific Computing (IWR), Universität Heidelberg, Im Neuenheimer Feld 368, 69120 Heidelberg, Germany, `kanschat@uni-heidelberg.de`
[5]Institute for Computational Mechanics, Technische Universität München, Boltzmannstr. 15, 85748 Garching b. München, Germany, `kronbichler@lnm.mw.tum.de`
[6]Institute of Applied Mathematics, Heidelberg University, Im Neuenheimer Feld 293/294, 69120 Heidelberg, Germany, `matthias.maier@iwr.uni-heidelberg.de`
[7]Department of Mathematics, Texas A&M University, College Station, TX 77843, USA, `turcksin@math.tamu.edu`
[8]Institute of Fundamental Technological Research of the Polish Academy of Sciences, ul. Pawińskiego 5b, Warsaw 02-106, Poland, `tyoung@ippt.pan.pl`

Received: January 1st, 2015; **final revision:** January 16th, 2015; **published:** January 16th, 2015.

Abstract: This paper provides an overview of the new features of the finite element library `deal.II` version 8.2.

1 Overview

`deal.II` version 8.2 was released January 1, 2015. This paper provides an overview of the new features of this release and serves as a citable reference for the `deal.II` software library version 8.2. `deal.II` is an object-oriented finite element library used around the world in the development of finite element solvers. It is available for free under the GNU Lesser General Public License (LGPL) from the `deal.II` homepage at `http://www.dealii.org/`.

Version 8.2 contains, along with the usual set of new functions, bug fixes and documentation updates, the following noteworthy changes:

– Comprehensive support for geometries described by arbitrary manifolds and meshes that respect this description not only on the boundary but also internally

– Support for geometries imported from CAD using the OpenCASCADE library

- Three new tutorial programs on complex geometries, CAD geometries, and time stepping methods
- Support for users wanting to use C++11 features
- Improvements to multithreading support
- Vectorization of many vector operations using OpenMP SIMD directives

Some of these will be detailed in the following section. Information on how to cite `deal.II` is provided in Section 3.

2 Significant changes to the library

This release of deal.II contains a number of large and significant changes that will be discussed in the following sections. It of course also contains a vast number of smaller changes and added functionality; the details of these can be found in the file that lists all changes for this release and that is linked to from the web site of each release as well as the release announcement.

2.1 Supporting complex geometries through manifolds

Complex geometries can be described using the concept of a manifold, a topological space that resembles Euclidean space near each point. In mathematics, particularly topology, one describes a manifold using an atlas. An atlas consists of individual charts that, roughly speaking, describe individual regions of the manifold, by means of providing a local coordinate system near each point. (Such coordinate systems may not exist *globally*, as in the case of the surface of a sphere, but it must be possible to cover the entire manifold with charts that are valid in some local neighborhood of every point.) In `deal.II` these concepts have been formalized in the abstract base class `Manifold` that defines an interface by which the `Triangulation` and `Mapping` classes can query geometric information about the domain.

Whenever a new point is required (for example upon mesh refinement, or when integrating over curved manifolds using higher order mappings), new points are typically obtained by providing a local coordinate system on the manifold, identifying existing points in the local coordinate system (pulling them back using the local chart to obtain their local coordinates), finding the new point in the local coordinate system by weighted sums of the existing points, and transforming the point back to the real space (pushing it forward using the local chart).

A new identifier (the `manifold_id`) has been introduced to describe geometrical features of complex domains: every entity of a mesh (i.e., cells, faces, and edges) now have a `manifold_id` associated with them. This extends the existing behavior: Previous releases of the library allowed only boundaries to be curved, and used the `boundary_id` of the faces of a triangulation to identify how to add new points upon mesh refinement. In contrast, the new `manifold_id` also exists for cells and interior faces. Upon refinement, the `Triangulation` class queries each object (hex, quad or line) where new points should be placed. Attaching a custom `Manifold` to any such object ensures that new points are created consistently with the underlying geometry.

We provide a series of classes that implement this mechanism for commonly used geometries:

- `FlatManifold`: in the simplest case, the objects that make up the `Triangulation` are straight line segments, a bi-linear surface or a tri-linear volume. New vertices are then simply put into the middle of the old ones (where "middle" means a suitable average of the locations of the pre-existing vertices). This is the default manifold associated to each object of the `Triangulation`;

– `SphericalManifold`: you can use this manifold object to describe any sphere, circle, hypersphere or hyperdisc in two or three dimensions, both as a co-dimension one manifold descriptor or as co-dimension zero manifold descriptor;

– `CylindricalManifold`: in three dimensions, points are transformed using a cylindrical coordinate system along an arbitrarily oriented cylinder.

More general specialisations can be obtained by using the `ChartManifold` partial specialisation: this class is intended to directly implement the topological concept of a chart, by explicitly providing *pull back* and *push forward* expressions. Classes derived from `ChartManifold` only have to overload the `pull_back` and `push_forward` functions, while all other functions are then defined in terms of these two functions. `ChartManifold` assumes that the manifold can be represented by a single, globally valid chart. The class `FunctionManifold` further implements this using explicit `deal.II Function` classes, and the new tutorial `step-53` provides a concrete example on how to use these classes to work with complex geometries.

2.2 Interfacing with CAD geometries

If you have OpenCASCADE (`http://www.opencascade.org/`) installed[26], you can now use existing IGES or STEP CAD files to describe the boundary of your geometry to `deal.II`.

The `deal.II` library allows the user to specify the geometry of the analysis in one of two possible ways: i) by creating a coarse grid internally (for simple geometries, such as boxes, cylinders, shells, etc.) or ii) by reading a mesh input file in one of the supported formats via the `GridIn` class.

Along with the default boundary descriptor `StraightBoundary` which implements the standard behavior of `deal.II`, some elementary boundary descriptors are also included in the library to describe circles or spheres (`HyperBallBoundary`), cylinders (`CylinderBoundary`), half circles or half spheres (`HalfHyperBallBoundary`), and so on. In general, `deal.II` provides a curved boundary descriptor for each one of the elementary mesh that can be created from within the library itself.

Real world geometries, however, can rarely be described by straight lines, balls or cylinders, and much more complex structures are required if one wants to tackle industrial problems. The industry standard for the design of any geometry is based on Computer Aided Design (CAD) tools, which rely on Non Uniform Rational B-Splines (NURBS) descriptions of curves and surfaces. Virtually any object that surrounds us has been modeled using some NURBS patch in a CAD tool.

The namespace `dealii::OpenCASCADE` contains all utilities and wrappers that are needed to manipulate CAD files. Most of the functions in the `dealii::OpenCASCADE` namespace deal with `TopoDS_Shape` objects: the default topological object of the `OpenCASCADE` library, and provide interfaces to deal.II objects, like `Triangulation`, `Manifold` and `Boundary`.

The new step-54 tutorial program of the `deal.II` library shows an example application of the functions and classes in the `deal.II::OpenCASCADE` namespace. In step-54, a CAD shape and a coarse surface triangulation are imported from external files, and the resulting triangulation is then refined on the CAD surface using different projection strategies. A detailed description of these techniques is available in [15].

2.3 A new tutorial: Support for time stepping methods (step-52)

The goal of this tutorial is to show how to use the different Runge–Kutta methods of the new TimeStepping class. The Runge–Kutta methods used in this tutorial include explicit methods, implicit methods, and embedded methods. In step-52, the time dependent neutron diffusion equation is solved on a uniform mesh. To compare the different time stepping methods, the solution of the problem is chosen such that its spatial component can be exactly represented on the uniform mesh by quadratic continuous finite elements. Moreover, at the end of the last time step the exact solution is zero. Therefore, the different time discretization methods can be compared by simply looking at the discretized solutions.

2.4 Support for some C++11 features

The C++11 standard [27] provides many convenient features that can help make writing finite element codes simpler. `deal.II` does not use any of these features itself to ensure that it can be used with compilers that do not implement C++11 yet.

However, this does not prevent users from using C++11 features in their own codes. Among these, range-based for loops are particularly useful given that `deal.II` programs often contain many loops over all cells and that writing these loops is frequently quite verbose:

C++ code

```cpp
Triangulation<dim> triangulation;
...
typename Triangulation<dim>::active_cell_iterator
  cell = triangulation.begin_active(),
  endc = triangulation.end();
for (; cell!=endc; ++cell)
  cell->set_refine_flag();
```

Code such as this can be significantly simplified by combining C++11's range-based for loops and `auto`-typed variables. To support this, `deal.II`'s container classes (i.e., the triangulations and DoF handler classes) provide member functions that return objects that denote the entire set of cells, active cells, or cells on a particular level. This allows to write the same loop as above in the following, much shorter way:

C++ code

```cpp
Triangulation<dim> triangulation;
...
for (auto cell : triangulation.active_cell_iterators())
  cell->set_refine_flag();
```

The function `Triangulation::active_cell_iterators()`, and equivalents in the `DoFHandler` and `hp::DoFHandler` classes, were added in `deal.II` 8.2 to support this idiom. Other variants of these functions provide iterator ranges for all cells (not just the active ones) and for cells on individual levels. A new documentation module elaborates on support for C++11 in `deal.II`.

3 How to cite deal.II

In order to justify the work the developers of `deal.II` put into this software, we ask that papers using the library reference one of the `deal.II` papers. This helps us justify the effort we put into it.

There are various ways to reference `deal.II`. To acknowledge the use of a particular version of the library, reference the present document. For up to date information and bibtex snippets for this document see:

https://www.dealii.org/publications.html

The original deal.II paper containing an overview of its architecture is [7]. If you rely on specific features of the library, please consider citing any of the following:

- For geometric multigrid: [20, 19];
- For distributed parallel computing: [6];
- For *hp* adaptivity: [11];
- For matrix-free and fast assembly techniques: [22];
- For computations on lower-dimensional manifolds: [14];
- For integration with CAD files and tools: [15].

deal.II can interface with many other libraries:

- ARPACK [23]
- BLAS, LAPACK
- HDF5 [30]
- METIS [21]
- MUMPS [3, 1, 2, 24]
- muparser [25]
- NetCDF [29]
- OpenCASCADE [26]
- p4est [12]
- PETSc [4, 5]
- SLEPc [16]
- Threading Building Blocks [28]
- Trilinos [17, 18]
- UMFPACK [13]

Please consider citing the appropriate references if you use interfaces to these libraries.

Older releases of deal.II can be cited as [8, 9].

4 Acknowledgements

deal.II is a world-wide project with dozens of contributors around the globe. Other than the authors of this paper, the following people contributed code to this release: Alexander Grayver, Andrea Mola, Andrew Baker, Angela Klewinghaus, Arezou Ghesmati, Ben Thompson, Christoph Heiniger, Damien Lebrun-Grandie, Daniel Arndt, David Wells, Denis Davydov, Fahad Alrashed, Giorgos Kourakos, Jean-Paul Pelteret, Kainan Wang, Kevin Drzycimski, Krysztof Bzowski, Lukas Korous, Manuel Quezada de Luna, Markus Bürg, Martin Steigemann, Mayank Sabharwal, Michal Wichrowski, Mihai Alexe, Minh Do-Quang, Shiva Rudraraju, Uwe Köcher, Valentin Zingan, Their contributions are much appreciated!

deal.II and its developers are financially supported through a variety of funding sources. W. Bangerth and B. Turcksin were partially supported by the National Science Foundation under award OCI-1148116 as part of the Software Infrastructure for Sustained Innovation (SI2) program; by the Computational Infrastructure in Geodynamics initiative (CIG), through the National Science Foundation under Award No. EAR-0949446 and The University of California – Davis; and through Award No. KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST).

L. Heltai was partially supported by the project OpenViewSHIP, "Sviluppo di un ecosistema computazionale per la progettazione idrodinamica del sistema elica-carena", financed by Regione FVG - PAR FSC 2007-2013, Fondo per lo Sviluppo e la Coesione.

T. Heister was partially supported by the Computational Infrastructure in Geodynamics initiative (CIG), through the National Science Foundation under Award No. EAR-0949446 and The University of California – Davis and through Award No. KUS- C1-016-04, made by King Abdullah University of Science and Technology (KAUST).

The Interdisciplinary Center for Scientific Computing (IWR) at Heidelberg University has provided hosting services for the deal.II web page and the SVN archive.

References

[1] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[2] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[3] P.R. Amestoy, I.S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods in Appl. Mech. Eng.*, 184:501–520, 2000.

[4] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, K. Rupp, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.5, Argonne National Laboratory, 2014.

[5] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, K. Rupp, B. F. Smith, and H. Zhang. PETSc Web page. http://www.mcs.anl.gov/petsc, 2014.

[6] W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Trans. Math. Softw.*, 38:14/1–28, 2011.

[7] W. Bangerth, R. Hartmann, and G. Kanschat. deal.II — a general purpose object oriented finite element library. *ACM Trans. Math. Softw.*, 33(4), 2007.

[8] W. Bangerth, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, B. Turcksin, and T. D. Young. The `deal.II` library, version 8.0. *arXiv preprint `http://arxiv.org/abs/1312.2266v3`*, 2013.

[9] W. Bangerth, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, B. Turcksin, and T. D. Young. The `deal.II` library, version 8.1. *arXiv preprint `http://arxiv.org/abs/1312.2266v4`*, 2013.

[10] W. Bangerth and G. Kanschat. Concepts for object-oriented finite element software – the `deal.II` library. Preprint 1999-43, SFB 359, Heidelberg, 1999.

[11] W. Bangerth and O. Kayser-Herold. Data structures and requirements for *hp* finite element software. *ACM Trans. Math. Softw.*, 36(1):4/1–4/31, 2009.

[12] C. Burstedde, L. C. Wilcox, and O. Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM J. Sci. Comput.*, 33(3):1103–1133, 2011.

[13] T. A. Davis. Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30:196–199, 2004.

[14] A. DeSimone, L. Heltai, and C. Manigrasso. Tools for the solution of PDEs defined on curved manifolds with deal.II. Technical Report 42/2009/M, SISSA, 2009.

[15] L. Heltai and A. Mola. Towards the Integration of CAD and FEM using open source libraries: a Collection of deal.II Manifold Wrappers for the OpenCASCADE Library. Technical report, SISSA, 2015. Submitted.

[16] V. Hernandez, J. E. Roman, and V. Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31(3):351–362, 2005.

[17] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the Trilinos project. *ACM Trans. Math. Softw.*, 31:397–423, 2005.

[18] M. A. Heroux et al. Trilinos web page, 2014. http://trilinos.sandia.gov.

[19] B. Janssen and G. Kanschat. Adaptive multilevel methods with local smoothing for H^1- and H^{curl}-conforming high order finite element methods. *SIAM J. Sci. Comput.*, 33(4):2095–2114, 2011.

[20] G. Kanschat. Multi-level methods for discontinuous Galerkin FEM on locally refined meshes. *Comput. & Struct.*, 82(28):2437–2445, 2004.

[21] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[22] M. Kronbichler and K. Kormann. A generic interface for parallel cell-based finite element operator application. *Comput. Fluids*, 63:135–147, 2012.

[23] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*. SIAM, Philadelphia, 1998.

[24] MUMPS: a MUltifrontal Massively Parallel sparse direct Solver. `http://graal.ens-lyon.fr/MUMPS/`.

[25] muparser: Fast Math Parser Library. `http://muparser.beltoforion.de/`.

[26] OpenCASCADE: Open CASCADE Technology, 3D modeling & numerical simulation. `http://www.opencascade.org/`.

[27] International Standards Organization. ISO/IEC 14882:2011: The C++11 programming language standard. `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50372`, 2011.

[28] J. Reinders. *Intel Threading Building Blocks*. O'Reilly, 2007.

[29] R. Rew and G. Davis. NetCDF: an interface for scientific data access. *Computer Graphics and Applications, IEEE*, 10(4):76–82, 1990.

[30] The HDF Group. Hierarchical Data Format, version 5, 1997-NNNN. http://www.hdfgroup.org/HDF5/.