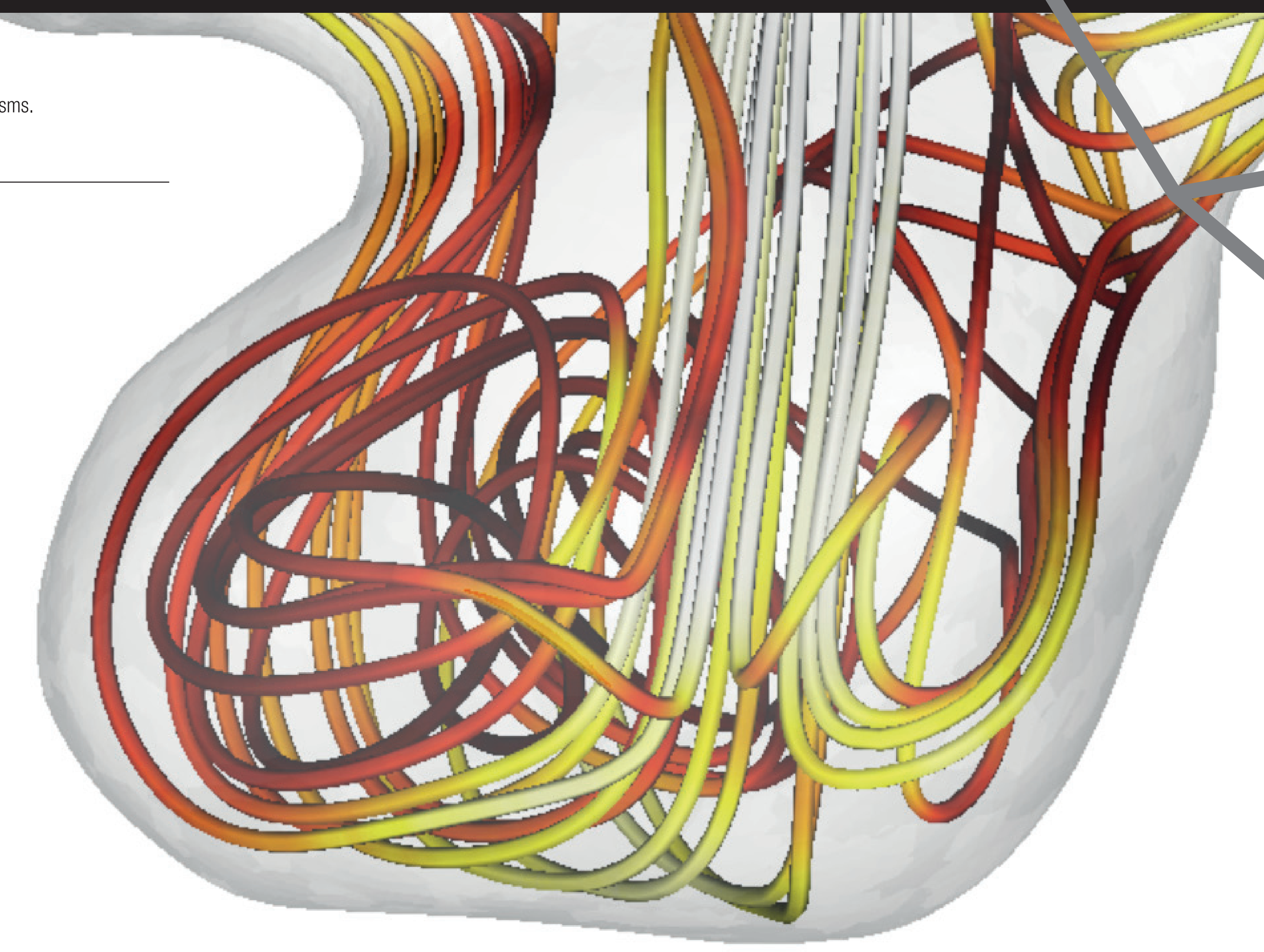
Authors: Martin Alnæs, Simula Research Laboratory | Jan Blechta, Charles University in Prague | Patrick Farrell, University of Oxford | Johan Hake, Simula Research Laboratory | Johan Hoffman, KTH | Johan Jansson, KTH | Niclas Jansson, KTH | August Johansson, Simula Research Laboratory | Claes Johnson, KTH | Benjamin Kehlet, Simula Research Laboratory | Robert C. Kirby, Baylor University | Matthew Knepley, University of Chicago | Miroslav Kuchta, University of Oslo | Hans Petter Langtangen, Simula Research Laboratory | Anders Logg, Chalmers University of Technology | Kent-Andre Mardal, University of Oslo and Simula Research Laboratory | Andre Massing, Simula Research Laboratory | Mikael Mortensen, University of Oslo and Simula Research Laboratory | Harish Narayanan, Simula Research Laboratory | Chris Richardson, University of Cambridge | Johannes Ring, Simula Research Laboratory | Marie Rognes, Simula Research Laboratory | Ridgway Scott, University of Chicago | Ola Skavhaug, Simula Research Laboratory | Andy Terrel, University of Chicago | Garth N. Wells, University of Cambridge | Kristian Ølgaard, Aalborg University |

FEniCS project



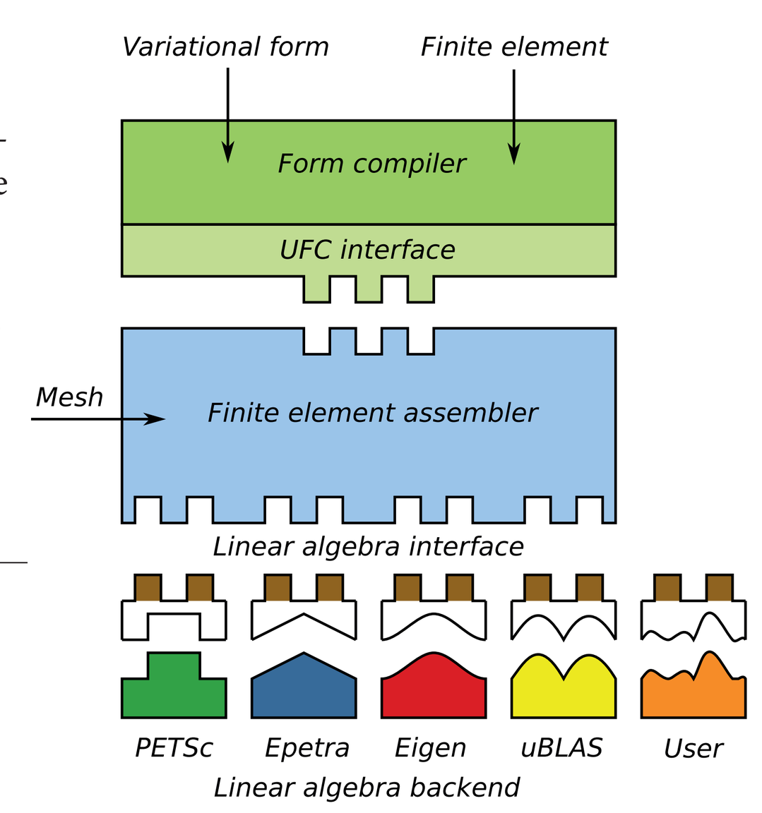
Simulation of hemodynamics in cerebral aneurysms.
[Evju, Valen-Sendstad, Mardal, 2013]

Overview

The FEniCS Project is a collaborative project for the development of innovative concepts and tools for automated scientific computing, with a particular focus on automated solution of differential equations by finite element methods.

FEniCS has an extensive list of features, including automated solution of finite element variational problems, automated error control and adaptivity, automated parallelization, a comprehensive library of finite elements, high performance linear algebra and many more.

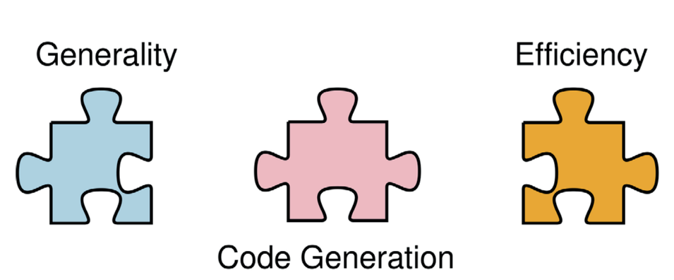
FEniCS is organized as a collection of interoperable components that together form the FEniCS Project. These components include the problem-solving environment DOLFIN, the form compiler FFC, the finite element tabulator FIAT, the just-in-time compiler Instant, the code generation interface UFC, the form language UFL and a range of additional components.



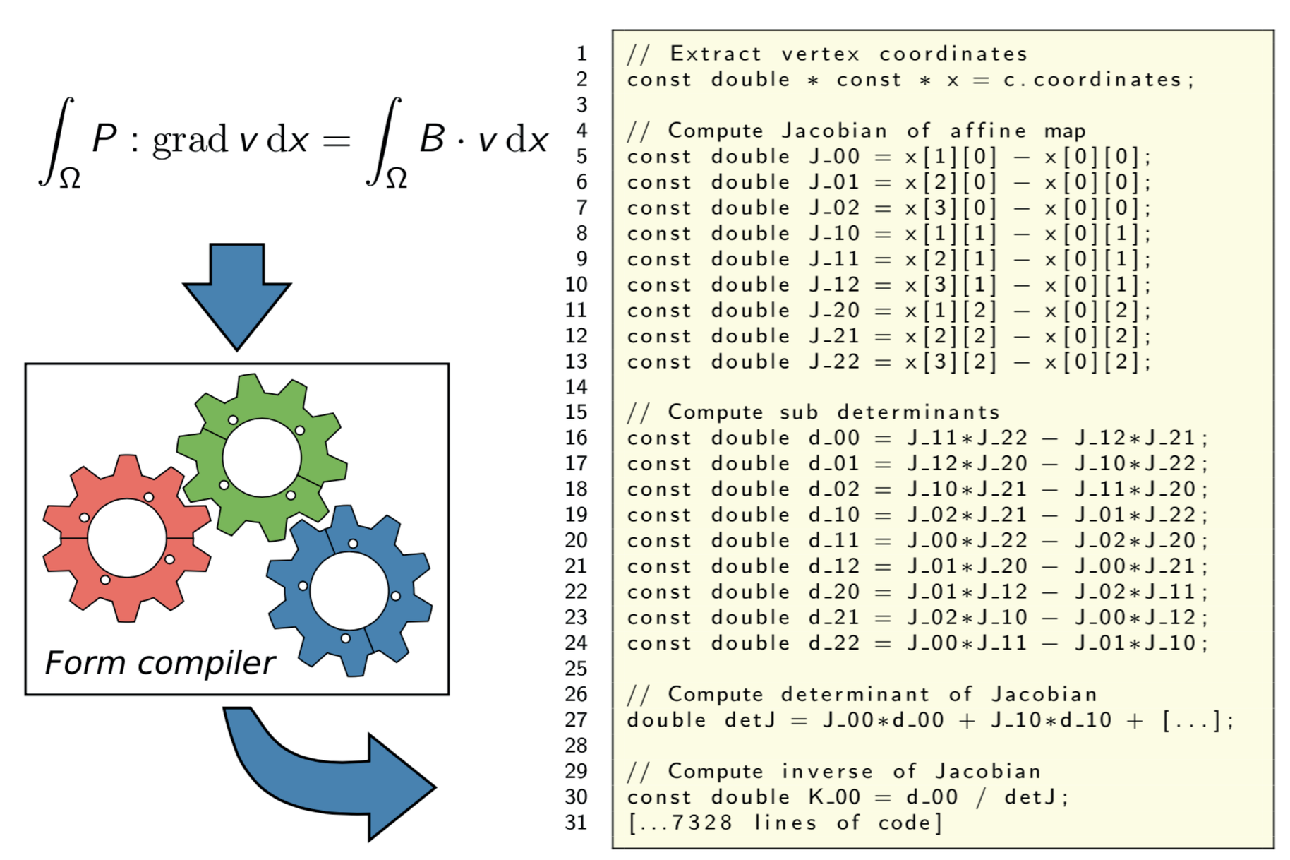
The figure illustrates the generic code generation and linear algebra interfaces of FEniCS.

Automatic code generation

A unique feature that makes FEniCS stand out among finite element libraries is its extensive use of automated code generation. Code generation is the key to a successful combination of generality and efficiency: allowing general systems of nonlinear partial differential equations to be expressed with ease in a language very close to mathematics, and to be solved with optimal performance on the most advanced supercomputers. For any given PDE, FEniCS generates application-specific highly optimized code targeted specifically to the given problem and the chosen discretization method.



Code generation is the key to combining generality (expressiveness) with efficiency.



$$\int_\Omega P : \mathrm{grad}\, v \,\mathrm{d}x = \int_\Omega B \cdot v \,\mathrm{d}x$$

```
// Extract vertex coordinates
const double * const * x = c.coordinates;

// Compute Jacobian of affine map
const double J_00 = x[1][0] - x[0][0];
const double J_01 = x[2][0] - x[0][0];
const double J_02 = x[3][0] - x[0][0];
const double J_10 = x[1][1] - x[0][1];
const double J_11 = x[2][1] - x[0][1];
const double J_12 = x[3][1] - x[0][1];
const double J_20 = x[1][2] - x[0][2];
const double J_21 = x[2][2] - x[0][2];
const double J_22 = x[3][2] - x[0][2];

// Compute sub determinants
const double d_00 = J_11*J_22 - J_12*J_21;
const double d_01 = J_12*J_20 - J_10*J_22;
const double d_02 = J_10*J_21 - J_11*J_20;
const double d_10 = J_02*J_21 - J_01*J_22;
const double d_11 = J_00*J_22 - J_02*J_20;
const double d_12 = J_01*J_20 - J_00*J_21;
const double d_20 = J_01*J_12 - J_02*J_11;
const double d_21 = J_02*J_10 - J_00*J_12;
const double d_22 = J_00*J_11 - J_01*J_10;

// Compute determinant of Jacobian
double detJ = J_00*d_00 + J_10*d_10 + [...];

// Compute inverse of Jacobian
const double K_00 = d_00 / detJ;
[...7328 lines of code]
```

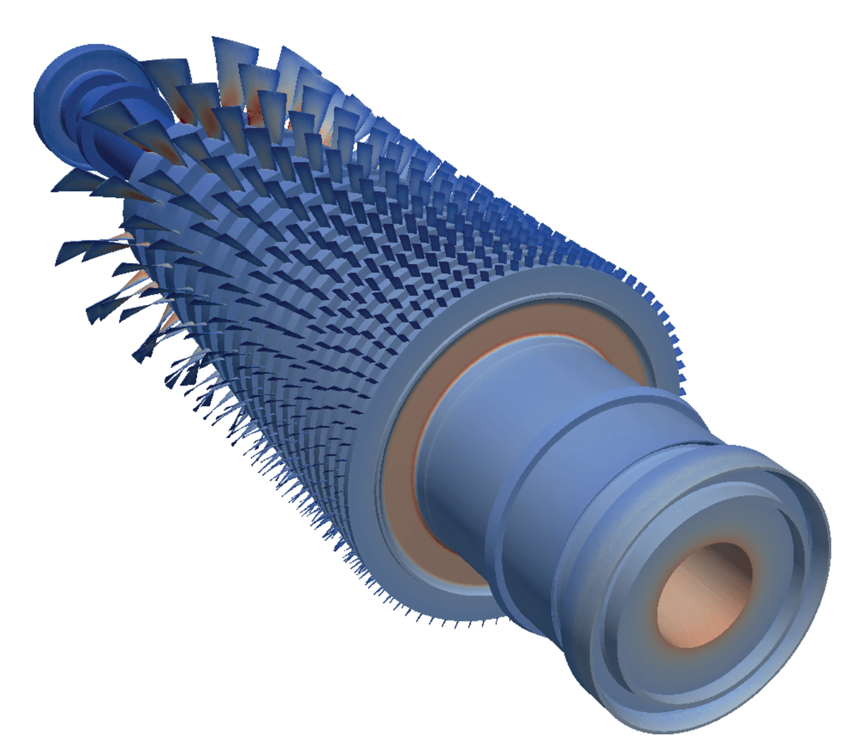
The figure illustrates the code generation process for a hyperelastic model problem (actual generated code).

History

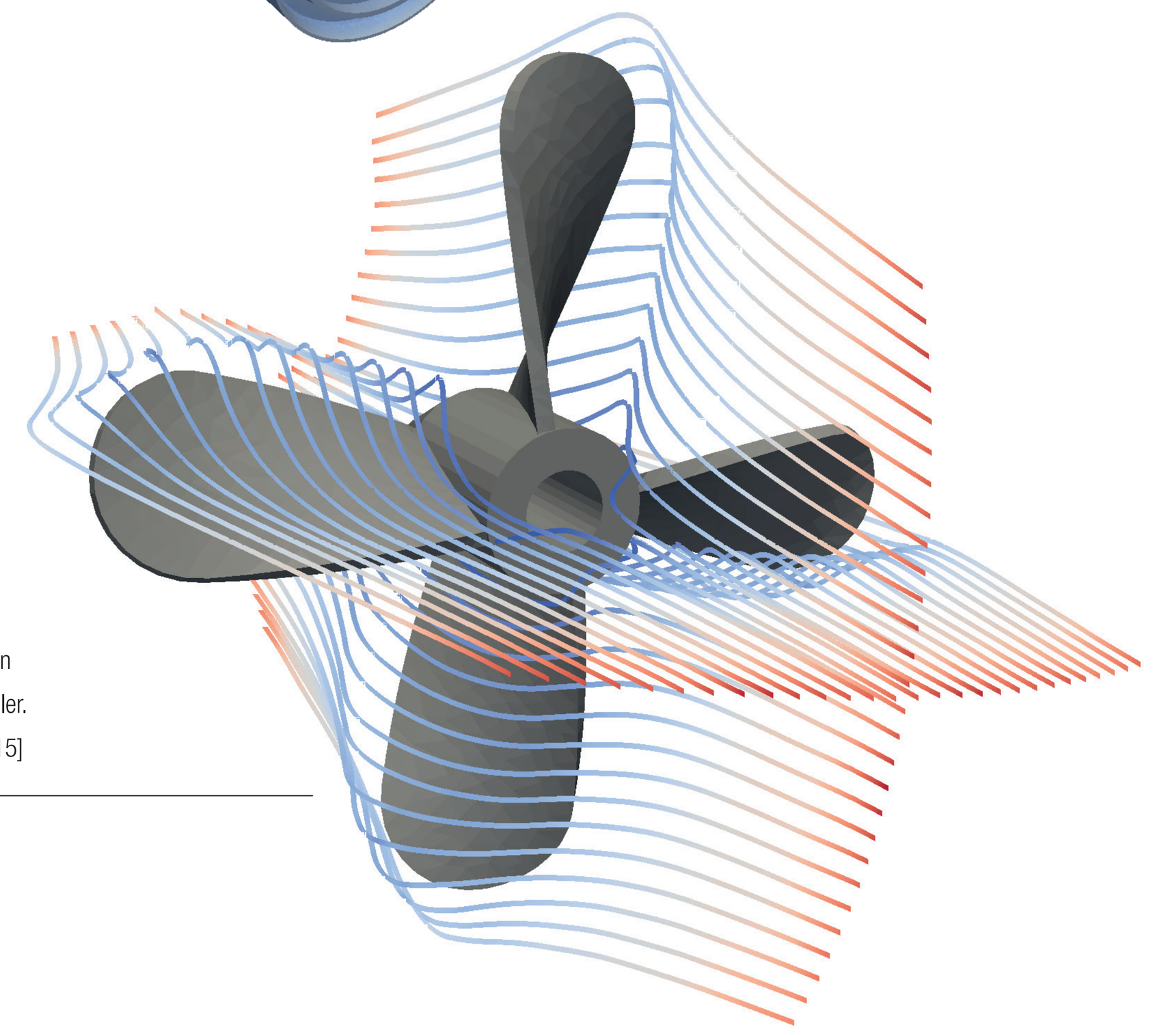
The starting point for the FEniCS Project was the combination of the C++ finite element library DOLFIN and the FIAT Python module for tabulation of finite element basis functions. In 2003, the developers of DOLFIN and FIAT joined forces in the creation of the FEniCS Project, which lead to a rapid development of novel technologies for automated finite element code generation, realized in the FEniCS components FFC, UFL and UFC. In 2011, version 1.0 was released and accompanied by the 700-page FEniCS book - one of Springer's most downloaded and most cited books in the mathematics category in recent years. The release of 1.0 marked the first stable version of FEniCS and presented to users a well-designed, intuitive and effective user interface for solution of general nonlinear systems of partial differential equations. Since the release of 1.0, most efforts have been directed towards improving the parallel scaling of FEniCS. With the release of 1.5 in early 2015, FEniCS has matured and offers cutting edge parallel performance on a range of hardware.

Examples

Limiting streamlines in a simulation of the flow past a rudimentary landing gear.
[Vilela De Abreu, Jansson, Hoffman, 2014]



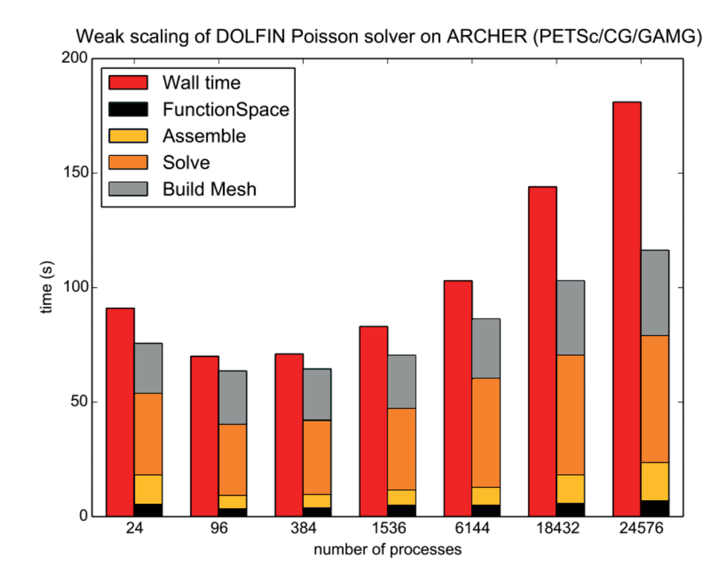
Elastic simulation of a gas turbine rotor.
[Wells and Sishaan, 2013]



Multimesh / cut FEM simulation of Stokes flow around a propeller.
[Johansson, Larson, Logg, 2015]

Recent and ongoing developments

Parallel computing
FEniCS supports high performance, distributed parallel simulations for large-scale applications, based on efficient implementation of scalable distributed meshes, refinement, parallel IO and access to high performance parallel linear algebra backends, such as PETSc. Simulations of elliptic problems with over 12 billion degrees of freedom have been performed.



The figure shows the weak scaling of a DOLFIN finite element Poisson solver on a unit cube mesh, with tetrahedral linear elements. Simulations were performed on ARCHER, the UK national supercomputer.

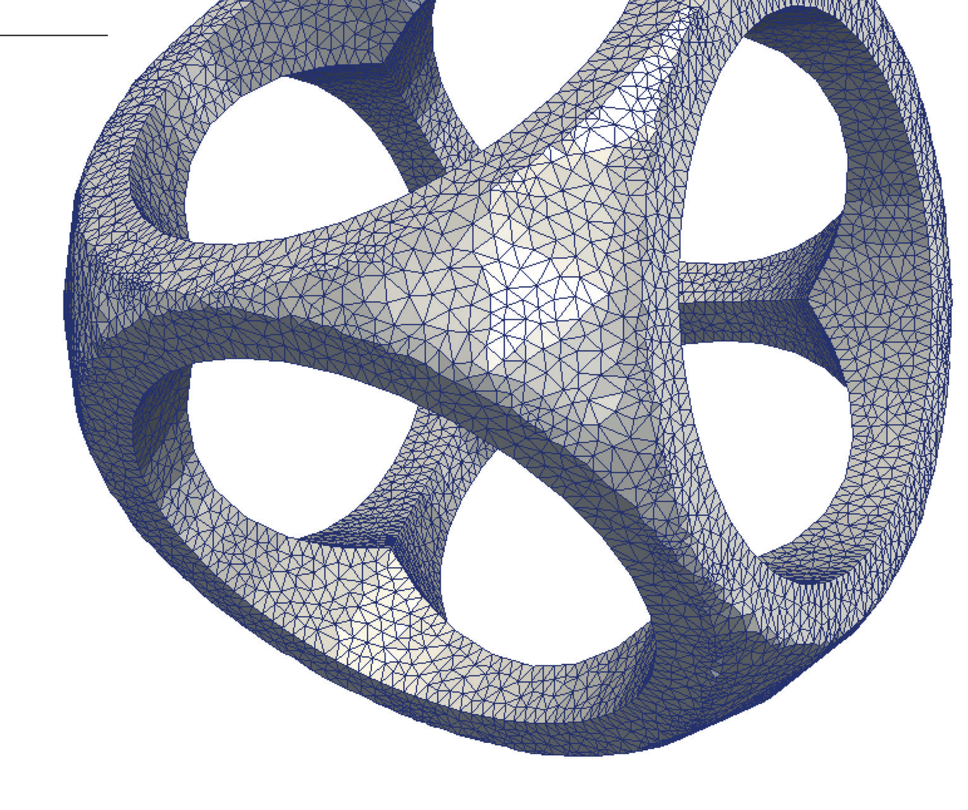
Efficient code generation
Driven by the needs of users applying FEniCS to ever more advanced models, the performance of the symbolic framework in UFL and code generation process in FFC has received a targeted profiling effort. The new FEniCS component UFLACS released as part of FEniCS 1.5 brings dramatic improvements to the performance and memory usage of the symbolic framework and code generation machinery. In addition, a new approach to code generation has been introduced in the first release of the UFLACS project. In UFLACS, a clever value numbering scheme is applied to efficiently deal with large tensor algebra expression trees produced by the Automatic Differentiation of complicated PDEs such as highly nonlinear solid mechanics models or fully coupled fluid-structure interaction problems. UFLACS also adds support for iso-parametric elements to FEniCS.

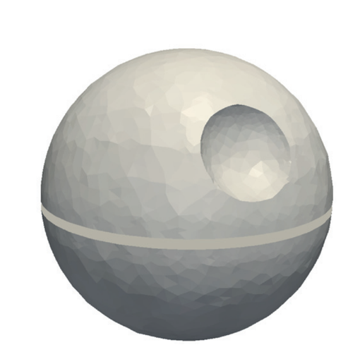
Parametrized geometries
The release of FEniCS 1.5 introduces the new package mshr, which provides user-friendly mesh generation in 2D and 3D based on constructive solid geometry (CSG). Generating a mesh is as easy as

```
csg  = Sphere(Point(0,0,0), 1) * Sphere(Point(1,1,1), 1)
mesh = generate_mesh(csg, resolution=10)
```

I addition to the mesh generation functionality of mshr, FEniCS provides a script for converting from a number of file formats to the native FEniCS file format.

Tetrahedral mesh generated from a simple CSG description with FEniCS mshr.





Mesh of the Deathstar generated with FEniCS mshr.

Multimesh finite element methods
With version 1.5 of FEniCS, initial support has been added for the formulation of multimesh finite element methods. These are finite element methods formulated on two or more distinct, nonmatching, and potentially overlapping meshes. The multimesh framework allows for the formulation of efficient finite element methods on complex and dynamic geometries, by removing the need for costly mesh generation and regeneration. Multimesh support in FEniCS includes special integration measures for expressing stabilized finite element methods involving integrals on cut cells, interfaces and overlaps involving many meshes.

Installation and packaging

Binary packages for Debian, Ubuntu and Mac
FEniCS is available directly from the official repositories of Debian and Ubuntu GNU/Linux. For Mac users, a standard point-and-click installation package is provided.

Building from source
While FEniCS can be built from source using standard build systems (CMake, distutils), compiling the whole FEniCS software stack including dependencies can be a daunting task, both for new users and developers. For this reason, HashDist has been adopted as a simple, reliable, efficient, and reproducible tool for building FEniCS software stacks from source. A simple script `[fenics-install.sh]` allows users to automatically build a FEniCS software stack via a one-line command:

```
curl -s http://fenicsproject.org/fenics-install.sh | bash
```

This one-liner will download HashDist, instruct HashDist to install FEniCS and finally set up a custom Unix environment for easy access to the FEniCS HashDist installation.

Docker
FEniCS provides various Docker containers as a simple way to get a customized FEniCS installation. Using Docker, everyone in a team can run an identical image for development before seamlessly transferring the environment to an HPC or a cloud computing environment such as Amazon AWS for full production runs. Because of Docker's lightweight container technology, nearly 100% of the underlying performance of the hardware is retained, in contrast with traditional virtualization approaches. Using Docker, users may install the latest version of FEniCS via a one-line command:

```
docker run -t -i fenicsproject/stable-ppa:latest /bin/bash
```

Even more options
As if that were not enough, FEniCS can also be installed using virtualization techniques such as Vagrant and Virtualbox, and via package managers such as Conda and MacPorts. With a multitude of options available, ranging from assisted source installation to package managers to virtualization, FEniCS can be installed with ease on most platforms, including the latest supercomputers.

References: R. C. Kirby. Algorithm 839: *FIAT, a new paradigm for computing finite element basis functions*, ACM Transactions on Mathematical Software (2004). R. C. Kirby and A. Logg. *A compiler for variational forms*, ACM Transactions on Mathematical Software (2006). M. S. Alnæs, A. Logg, K.-A. Mardal, O. Skavhaug and H. P. Langtangen. *Unified framework for finite element assembly*, International Journal of Computational Science and Engineering (2009). A. Logg and G. N. Wells. *DOLFIN: Automated finite element computing*, ACM Transactions on Mathematical Software (2010). K. B. Ølgaard and G. N. Wells. *Optimisations for quadrature representations of finite element tensors through automated code generation*, ACM Transactions on Mathematical Software (2010). A. Logg, K.-A. Mardal, G. N. Wells et al. *Automated Solution of Differential Equations by the Finite Element Method*, Springer (2012). M. S. Alnæs, A. Logg, K. B. Ølgaard, M. E. Rognes, G. N. Wells. *Unified Form Language: A domain-specific language for weak formulations of partial differential equations*, ACM Transactions on Mathematical Software (2014). **Acknowledgement:** We gratefully acknowledge the following grants that have supported the development of FEniCS: Center of Excellence grant awarded to the Center for Biomedical Computing at Simula Research Laboratory by the Research Council of Norway, grant no. 179578/F30. Patient-Specific Mathematical Modeling with Applications to Clinical Medicine: Stroke and Syringomyelia, Research Council of Norway, grant no. 209951. Outstanding Young Investigator Grant: Automation of Error Control with Application to Fluid-Structure Interaction in Biomedicine, Research Council of Norway, grant no. 180450.

FEniCS project
fenicsproject.org