D1.5
Polygonal discretization in deal.II

Project Title	dealii-X: an Exascale Framework for Digital Twins of the Human Body
Project Number	101172493
Funding Program	European High-Performance Computing Joint Undertaking
Project start date	1 October 2024
Duration	27 months

dealii-X has received funding from the European High-Performance Computing Joint Undertaking Programme under grant agreement Nº 101172493

Deliverable title	Polygonal discretization in deal.II
Deliverable number	D1.5
Deliverable version	v1
Date of delivery	August 31, 2025
Actual date of delivery	August 29, 2025
Nature of deliverable	DEM - Demonstrator, pilot, prototype
Dissemination level	Public
Work Package	WP1
Partner responsible	SISSA

Abstract	This document describes a tutorial program showcasing the usage of polytopal meshes building on top of the deal.II library. library.
Keywords	Polygonal meshes, Discontinuous Galerkin, Tutorial program

Document Control Information

Version	Date	Author	Changes Made
0.1	13/08/2025	Marco Feder	Initial draft
1.0	29/08/2025	Martin Kronbichler	Review and final version

Approval Details

Approved by: Martin Kronbichler
Approval Date: August 29, 2025

Distribution List

- Project Coordinators (PCs)
- Work Package Leaders (WPLs)
- Steering Committee (SC)
- European Commission (EC)

Disclaimer: This project has received funding from the European Union. The views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or the European High-Performance Computing Joint Undertaking (the “granting authority”). Neither the European Union nor the granting authority can be held responsible for them.

Contents

1 Introduction

This objective of the dealii-X Work Package 1.5 is dedicated to the introduction, development, and validation of an experimental module for polygonal discretizations within the deal.II library. The deal.II library currently supports a wide range of finite element methods, defined on standard simplicial and hexahedral meshes, but does not support polygonal and polyhedral (*polytopic*) meshes. Polytopal methods gained popularity in recent years due to their ability to handle complex geometries and adapt to varying mesh sizes. These methods enable enhanced accuracy while reducing the number of degrees of freedom. In particular, discontinuous Galerkin methods formulated on polytopal meshes (PolyDG) (Cangiani et al., 2014; Antonietti et al., 2013) naturally accommodate complex geometries, benefitting from the inherent flexibility of agglomerated grid structures, i.e., by grouping together several cells of an initial fine mesh.

Recently, several agglomeration strategies have been proposed within the community: among them, we mention the `R3MG` (Feder et al., 2025) approach, which allows the generation of *sequences* of nested polytopal meshes and can be used also in the context of multilevel methods. This is the approach described in this Work Package.

1.1 Purpose of the Document

The objective of this document is to provide a comprehensive description of the usage of the new polygonal discretization features, enabling interested users to effectively exploit these capabilities. This new library, named `Polydeal`, builds on top of deal.II and provides additional functionality for working with polytopal meshes. It is available at `https://github.com/fdrmrc/Polydeal`. The module is designed to be integrated into the deal.II library and is up to date with the latest deal.II master branch. As it will be clear from the forthcoming tutorial program, the new interface is not intrusive and does not require significant changes to existing user codes, thanks to the extensive re-usage of deal.II's core components.

2 Tutorial program

In order to demonstrate the new polygonal discretization capabilities in deal.II, we present a tutorial program that shows the new features and how to use them in practice.

We consider the following elliptic problem, posed on a domain $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, discretized with a polytopal mesh $\mathcal{T}_h$:

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega, \\ u &= g_D \quad \text{on } \partial\Omega, \end{aligned} \tag{1}$$

where $f \in L^2(\Omega)$ is the right-hand side, and g_D is a Dirichlet boundary condition. We consider meshes made of polygonal or polyhedral cells generated through agglomeration. This process can be performed using classical graph-partitioners such as `METIS` (Karypis and Kumar, 2005).

2.1 The tutorial program

This tutorial is available in the Polydeal repository at /examples/poisson.cc. It follows the structure of a standard deal.II tutorial, using the same underlying concepts and naming conventions. In this section, we will describe the main steps needed to set up a polytopal mesh, and to solve the elliptic problem in (1) using a polytopal Discontinuous Galerkin discretization. In particular, the main differences compared to classical deal.II tutorials will be highlighted in order to introduce the new classes and functions.

2.1.1 Setup and mesh generation

In addition to classical include files needed by deal.II, the program requires the inclusion of the following header files which contain declarations of the new classes and functions designed to handle polygonal meshes and discontinuous Galerkin discretizations:

Listing 1: New header files

```
1 #include <agglomeration_handler.h>
```

```
2 #include <poly_utils.h>
```

First, a standard mesh made by usual shapes is created. This can be done using the built-in deal.II mesh generation capabilities, or by importing an external mesh file through the `GridIn` class. In this example, we assume to use the latter approach.

Listing 2: Importing an external mesh

```
1 GridIn<dim> grid_in;
2 Triangulation<dim> tria;
3 grid_in.attach_triangulation(tria);
4  // unstructured quad mesh
5 std::ifstream gmsh_file(MESH_DIR "/unstructured.msh");
6 grid_in.read_msh(gmsh_file);
```

The number of polytopes to be created is defined by the user, and it is set to `n_subdomains` in the following snippet. It is equal to the number of partitions that will be queried from the graph partitioner. `AgglomerationHandler` is a new class which plays the pivotal role of handling *general* polytopal shapes, as well as the management of their properties. Each polytope is hence defined as a collection of cells, which are here grouped together based on an integer index. After that, the `AgglomerationHandler` is informed about the definition of each agglomerate through the `define_agglomerate()` method.

Listing 3: Creation of polytopal elements

```
1
2 agglo_handler = std::make_unique<AgglomerationHandler<dim>>(*original_tria);
3 if (partitioner_type == PartitionerType::metis)
4 {
5     // Partition the triangulation with graph partitioner.
6     GridTools::partition_triangulation(n_subdomains,
7     tria,
8     SparsityTools::Partitioner::metis);
9
10    // A polytope is defined as a collection of cells
11    using Polytope = Vector<typename Triangulation<dim>::cell_iterator>;
12    Vector<Polytope> polytopes;
13    // For every subdomain, agglomerate elements together
14    for (std::size_t i = 0; i < n_subdomains; ++i)
15        agglo_handler->define_agglomerate(polytopes[i]);
16 }
```

After this setup step, all the topological and geometrical information needed to define a polytopic partition $\mathcal{T}_h$ of the domain Ω is available. In particular, polytopes within the mesh are now enumerated, and for each one it is possible to ask the number of faces, their neighbors, the volume, and other geometric properties needed by

the numerical scheme at hand.

2.1.2 Discontinuous Galerkin discretization

On the polytopal mesh $\mathcal{T}_h$, it is possible to define a discontinuous Galerkin (DG) discretization. This involves the use of local polynomial spaces defined on the *bounding box* on each polytope (Cangiani et al., 2014). To construct a nodal DG space on the polytopal mesh, we can use the `FE_DGQ` 7class provided by deal.II. This will define, on each bounding box, a nodal polynomial space of degree p in each coordinate direction. It is then possible to enumerate the degrees of freedom (DoFs) on the polytopal mesh, and to create the sparsity pattern associated with the discretization space, necessary to initialize the pattern of nonzero entries in the sparse matrix associated with the problem. This is achieved again through the `AgglomerationHandler` class, which provides the necessary methods `distribute_agglomerated_dofs()` and `create_agglomeration_sparsity_pattern()`.

Listing 4: Distribution of DoFs on a polytopal mesh

```
FE_DGQ<dim> dg_fe(1); //linear elements
SparsityPattern                 sparsity_pattern;

//agglo_handler pointer to an instance of AgglomerationHandler
agglo_handler->distribute_agglomerated_dofs(dg_fe);
agglo_handler->create_agglomeration_sparsity_pattern(sparsity_pattern)

// Allocate entries for stiffness matrix
system_matrix.reinit(sparsity_pattern);
```

The assembly phase consists of looping over all polytopes in $\mathcal{T}_h$, and scattering the local contributions to the global stiffness matrix and right-hand side vector in the usual way as in deal.II tutorials. To this aim, suitable iterators and accessors on mesh entities have been developed. This allows to hide all the implementation details and the underlying data structures in user codes, in the same fashion as the deal.II library does for standard meshes.

The local contributions are then scattered into the global stiffness matrix and right-hand side vector using the deal.II class `AffineConstraints`. All in all, after the execution of this loop, the global linear system of equations

$$Au = b$$

for the nodal values $\{u_i\}_{i=1}^{N}$ (where N is the total number of DoFs) is ready to be solved. Such solution step is completely orthogonal to the new polytopal interface, and suitable solution strategies for handling the linear system can be exploited.

Listing 5: Assembly of stiffness matrix and rhs vector

```
   // fill local matrix and rhs
for (const auto &polytope : agglo_handler->polytope_iterators()) {
    local_cell_matrix = 0.;
    local_cell_rhs = 0.;
    polytope->get_dof_indices(local_dof_indices);
    for (unsigned int q_point : agglo_values.quadrature_points()) {
        for (unsigned int i = 0; i < dofs_per_cell; ++i) {
            for (unsigned int j = 0; j < dofs_per_cell; ++j) {
                // assemble local cell matrix and local cell rhs
                }
            }
        }
        for (unsigned int f = 0; f < polytope->n_faces(); ++f) {
            // add contributions from the face to the local cell matrix
            // and local cell rhs
        }
    // distribute DoFs
    constraints.distribute_local_to_global(cell_matrix, cell_rhs, local_dof_indices,
                                           stiffness_matrix, system_rhs);
  }
```

The listing above follows verbatim the structure of a standard deal.II assembly phase, with the only difference that the iterators and accessors used to loop over the polytopal mesh are provided by the `AgglomerationHandler` class, rather than by the well-known `DoFHandler` class provided by deal.II.

2.2 Output and visualization

Once the linear system of equations for u has been solved, post-processing and visualization can be performed building on top of the new polytopal infrastructure and the standard deal.II visualization capabilities. Among the desired features, computation of errors and visualization of the solution are supported. Error norms with respect to the exact solution are easily computed by using the utility functions defined in the included header file `poly_utils.h`. Finally, the solution computed on the polytopal mesh u_h is interpolated onto the original mesh, and visualized using the `DataOut` class provided by deal.II.

In this specific program, the analytical solution is set to be

$$u(x, y) = \sin(\pi x) \sin(\pi y).$$

In Figure 1, we show the polytopic solution u_h interpolated onto the original mesh, and the polytopal grid over which the discretization has been performed. The original mesh is an unstructured quadrilateral mesh, displayed in Figure 2.

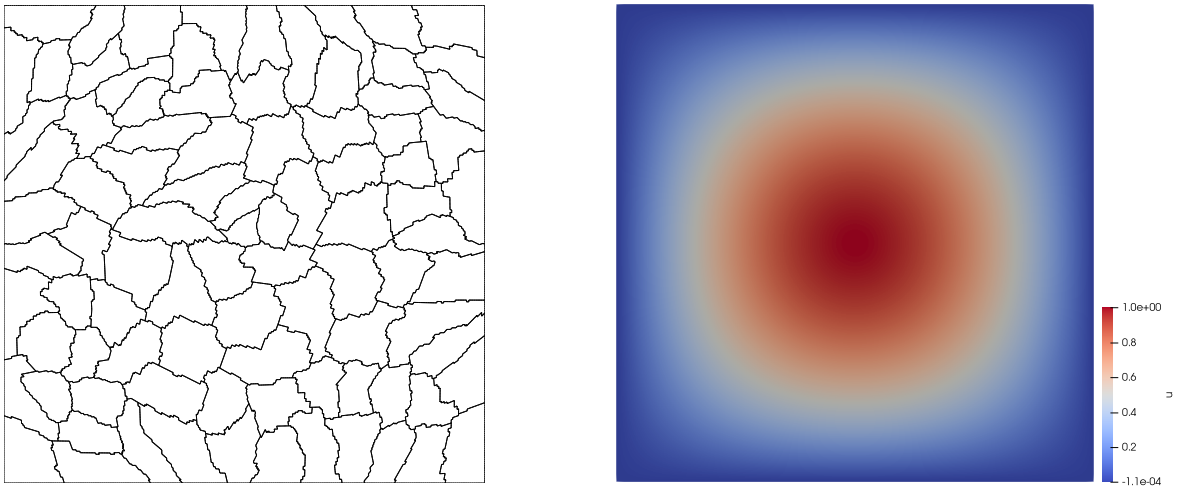

Figure 1: Left: polytopal mesh generated by agglomeration. Right: solution u_h interpolated onto the original mesh.

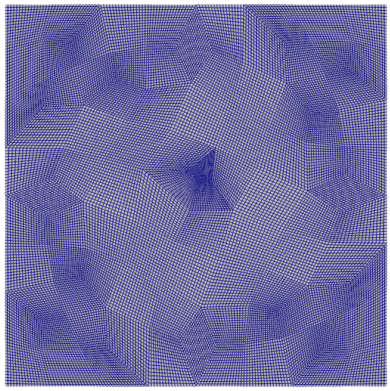

Figure 2: Unstructured quadrilateral mesh used as the starting point for agglomeration.

3 Conclusion

The presented tutorial program demonstrates the use of the polytopal discretization module of deal.II. The future code development will be based on the code in `https://github.com/fdrmrc/Polydeal` and extended to additional equations and approaches.

References

Andrea Cangiani, Emmanuil Georgoulis and Paul Houston, ”hp-Version discontinuous Galerkin methods on polygonal and polyhedral meshes,” Mathematical Models and Methods in Applied Sciences, vol. 24, no. 4, pp. 2009-2041, 2014.

Paola Antonietti, Stefano Giani, and Paul Houston, ”hp-Version Composite Discontinuous Galerkin Methods for Elliptic Problems on Complicated Domains”, SIAM Journal on Scientific Computing, vol. 35, A1417-A1439, 2013.

Marco Feder, Andrea Cangiani and Luca Heltai, ”R3MG: R-tree based agglomeration of polytopal grids with applications to multilevel methods”, Journal of Computational Physics, vol. 526, 113773, 2025.

George Karypis, and Vipin Kumar, ”A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs”, SIAM Journal on Scientific Computing, vol. 20, 359-392, 1998.