CSMA 2024
16ème Colloque National en Calcul des Structures
13-17 Mai 2024, Presqu'île de Giens (Var)

Muscat : Mesh manipulation and finite element engine for engineering and science

F. Bordeu[1], M. de Buhan[1], F. Casenave[1], J. Cortial[1], R. Messahel[1]

[1] *Safran Tech, Digital Sciences & and Technologies Department*
Rue des Jeunes Bois, Châteaufort 78114 Magny-Les-Hameaux, France

Résumé — Numerical simulations of physical phenomena can be computed by many (commercial/free) software packages, but despite the apparent variety, all of them rely on a relatively small set of operations during the preparation, exploitation, and post-processing of these simulations, e.g., handling and modifying meshes and fields. Muscat is a Python library designed to address these supporting tasks. It features an efficient data model for meshes and field objects, as well as input/output routines compatible with various formats. A finite element engine allows to assemble abstract variational formulations and integrate fields on volumes and surfaces.

Muscat is actively used in artificial intelligence and model order reduction [1, 2, 3, 4], topology optimization [5], and material sciences [6] projects. This paper reproduce the content of the original publication for BasicTools (Muscat version 1.0) of [7].
Mots clés — Python, C++, mesh, fields, finite elements, pre-processing, post-treatment.

1 Statement of need

Industrial design tasks often rely on numerical simulation workflows involving different software packages, each providing its own specific post-processing tools. Common tasks like transferring computed fields from one tool to another must be routinely implemented, with subtle variations. This limits interoperability and increases complexity.

Muscat is a solution to these concerns. It introduces a data model for meshes and related physical fields that can be populated using different readers and exported using various writers : no new mesh or solution format is forced upon the user. The data-oriented design of Muscat allows high performance operations using a high-level language (Python with NumPy). Muscat allows users to convert meshes to other "in-memory" formats (VTK [8], PyVista [9], MeshIO [10], CGNS [11], and Gmsh [12]), enabling mixing (and reusing) the various treatments available in other frameworks. Other features available in Muscat include various mesh handling routines, field transfer operators, and a flexible finite element engine. Other features available in Muscat include various mesh handling routines, multi-threaded field transfer operators computation (oneTBB [13]), and a flexible multi-threaded finite element engine.

2 State of the field

In the computational fluid dynamics community, the CFD General Notation System (CGNS) [11] format is a de-facto standard. However, to the authors' knowledge, no such standard exists for solid mechanics. One may consider VTK and MeshIO for mesh manipulation and file format conversion, respectively, but the post-processing of integration point data, a key requirement in solid mechanics, would not be possible. Most available tools implement the simple, but potentially dangerous, approach of extrapolating the integration point values to the nodes of the mesh or averaging in every cell. This can lead to a misinterpretation of the solution and incorrect engineering decisions. Also, only a few finite element engines allow assembling abstract variational formulations on arbitrary geometries, like FreeFem++ [14] or FEniCS [15], amongst others.

3 Overview

The main features of Muscat are :

- Mesh (in the ‘Containers‘ module) : The Mesh class implements an efficient unstructured mesh data model : elements are stored using only one array for each element type. The mesh can feature nodes and elements tags. Many functions are available for creating, cleaning and modifying meshes (e.g., field transfer and mesh morphing).
- Filters (in the ‘Containers‘ module) : Various types of ‘ElementFilter‘s and ‘NodeFilter‘s allow to handle subparts of meshes by selecting element- and node-sets using threshold functions, tags, element types, element dimensionality, and masks Filters can be combined using Boolean operations (union, complementary, ...).
- A finite element engine (in the ‘FE‘ module) : A general weak formulation engine able to integrate fields over parts of the meshes is available. The ‘FETools‘ submodule contains specific functions for Lagrange P1 finite elements, including the computation of stiffness and mass matrices. The domain of integration is defined using ‘ElementFilter‘s, making the integration domain flexible. P0 and P2 Lagrange finite element spaces are implemented and tested. The framework is non-isoparametric : the user can write weak formulations mixing P0, P1, and P2 fields on P1 or P2 meshes.
- Input/Output functions (in the ‘IO‘ module) : Various readers (respectively, writers) for importing (respectively, exporting) meshes and solution fields from (respectively, to) Muscat’ internal data model are available. Supported formats include, among others, geo/geof (Z-set [16]), VTK, XDMF, SAMCEF, ABAQUS, LS-DYNA, and a bridge with MeshIO is provided. Readers for the ABAQUS and SAMCEF proprietary formats are also enabled when properly licensed software is available locally. See the Muscat documentation [1] for more details.
- Implicit geometry engine (in the ‘ImplicitGeometry‘ module) : Arbitrary subdomains can be defined using implicit geometries (via level-set functions). Basic shapes (spheres, half-spaces, cylinders, cubes), transformations (symmetry, translation, rotation) and binary operators (union, difference, and intersection) can be used to construct complex shapes. In turn these shapes can be used to select elements or points (using ‘ElementFilter‘ of ‘NodeFilter‘), or be evaluated on point clouds to explicitly construct iso-zero surfaces.
- Linear algebra functions (in the ‘LinAlg‘ module) : Some common operations on linear systems for finite elements are implemented : penalization, elimination, Lagrange multipliers, and Ainsworths [17] method to impose essential boundary conditions or linear multi-point constraints. The submodule ‘LinearSolver‘ offers an abstraction layer for sparse linear solvers, including : Cholesky of the ‘sksparse‘ package ; factorized, CG, LSQR, GMRES, LGMRES of the ‘scipy.sparse.linalg‘ module ; CG, LU, BiCGSTAB, SPQR of the C++ Eigen library ; Pardiso solver of the MKL library ; and the AMG solver of ‘pyamg‘ package.

The large majority of functions are illustrated in the same file where they are defined, in ‘CheckIntegrity‘ functions.

4 Examples

We present two examples ; see Muscat documentation [2] for more details.

4.1 Pre/post deep learning

Convolution-based deep learning algorithms generally rely on structured data. Muscat can be used to transfer a field computed on an unstructured mesh using finite elements to a structured grid and vice versa. To validate the operation, the error on the final field is evaluated with respect to the original field.

1. https://muscat.readthedocs.io/en/latest/_source/Muscat.IO.html
2. https://muscat.readthedocs.io/en/latest/Examples.html

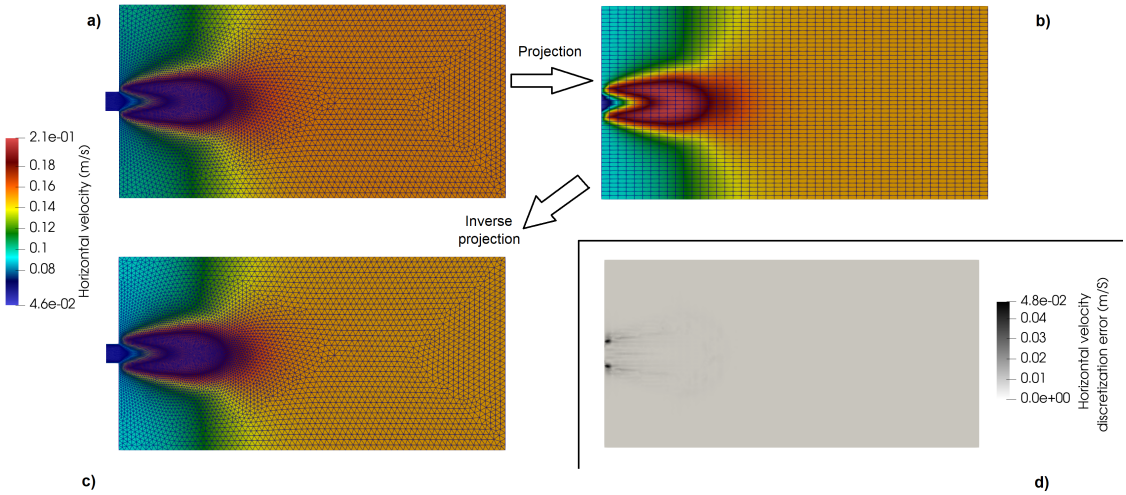

FIGURE 1 – Deep learning workflow coupled to finite element simulator a) Initial field on unstructured mesh, b) transferred field into regular grid (projection step), c) inverse transfer into original unstructured mesh, d) transfer error on unstructured mesh.

4.2 Mechanical analysis : Thick plate with two inclusions

Consider a thick plate with two inclusions, one softer and the other stiffer than the base material. The plate is clamped on the left side with a constant traction applied on the right side. We compute the strain energy on only one inclusion. The linear elasticity problem is solved using P1 Lagrange finite elements on an unstructured mesh.

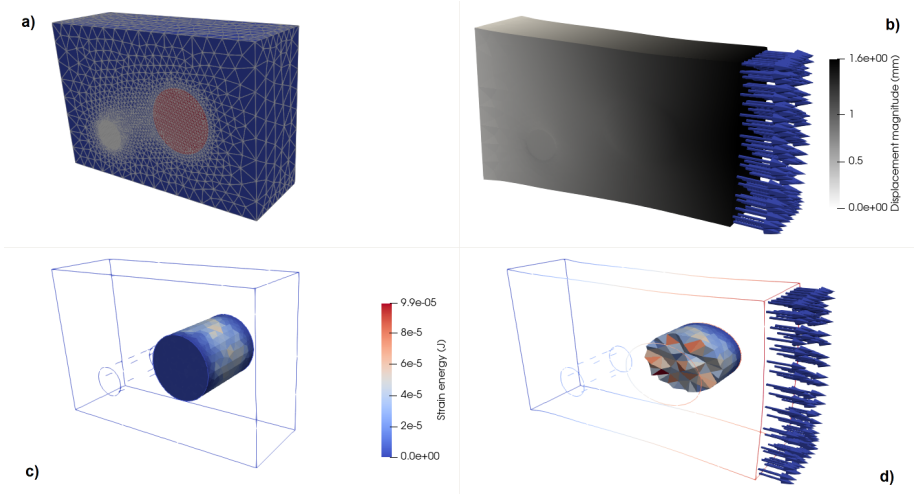

FIGURE 2 – Analysis of a mechanical thick plate with two inclusions a) illustration of the mesh with highlighting of the two inclusions, b) magnitude of the displacement solution on the deformed mesh (also showing the applied traction), c) strain energy in the large inclusion, d) cutaway view of the strain energy in the large inclusion (also showing the applied traction).

Références

[1] T. Daniel, F. Casenave, N. Akkari, and D. Ryckelynck. Model order reduction assisted by deep neural networks (ROM-net). *Adv. Model. and Simul. in Eng. Sci.*, 7(16), 2020.

[2] T. Daniel, F. Casenave, N. Akkari, and D. Ryckelynck. Data augmentation and feature selection for automatic model recommendation in computational physics. *Math. Comput. Appl.*, 26(1), 2021.

[3] T. Daniel, F. Casenave, N. Akkari, D. Ryckelynck, and C. Rey. Uncertainty quantification for industrial numerical simulation using dictionaries of reduced order models. *Mechanics & Industry*, 23 :3, 2022.

[4] N. Akkari, F. Casenave, T. Daniel, and D. Ryckelynck. Data-targeted prior distribution for variational autoencoder. *Fluids*, 6(10), 2021.

[5] C. Nardoni, D. Danan, C. Mang, F. Bordeu, and J. Cortial. *A R&D Software Platform for Shape and Topology Optimization Using Body-Fitted Meshes*, pages 23–39. Springer International Publishing, Cham, 2022.

[6] H. Proudhon. pymicro, 2013-present. https ://github.com/heprom/pymicro.

[7] F. Bordeu, F. Casenave, and F. Cortial. Basictools : a numerical simulation toolbox. *Journal of Open Source Software*, 8(86) :5142, 2023.

[8] W. Schroeder, K. Martin, and B. Lorensen. *Visualization Toolkit : An object-oriented approach to 3D graphics*. Kitware, Inc., fourth edition, 2006.

[9] C. Bane Sullivan and A. Kaszynski. PyVista : 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). *Journal of Open Source Software*, 4(37) :1450, may 2019.

[10] N. Schlömer. meshio : Tools for mesh files, 2015-present. https ://github.com/nschloe/meshio.

[11] CGNS contributors. Cfd general notation system, 1994-present. http ://cgns.github.io/.

[12] C. Geuzaine and J-F. Remacle. Gmsh : A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11) :1309–1331, 2009.

[13] James Reinders. Intel threading building blocks - outfitting c++ for multi-core processor parallelism. 2007.

[14] F. Hecht. New development in FreeFem++. *Journal of Numerical Mathematics*, 20(3-4) :251–266, 2012.

[15] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M.E. Rognes, and G.N. Wells. The FEniCS project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

[16] Mines ParisTech and ONERA the French aerospace lab. Zset : nonlinear material & structure analysis suite, 1981-present. http ://www.zset-software.com.

[17] Mark Ainsworth. Essential boundary conditions and multi-point constraints in finite element analysis. *Computer Methods in Applied Mechanics and Engineering*, 190(48) :6323–6339, 2001.