The Journal of Open Source Software

VTUFileHandler: A VTU library in the Julia language that implements an algebra for basic mathematical operations on VTU data

Maximilian Bittens[1]

1 Federal Institute for Geosciences and Natural Resources (BGR)

DOI: 10.21105/joss.04300

Software

- Review
- Repository
- Archive

Editor: Mehmet Hakan Satman

Reviewers:

- @dmbates
- @mkitti

Submitted: 01 March 2022
Published: 24 May 2022

Abstract

With increasing computing resources, investigating uncertainties in simulation results is becoming an increasingly important factor. Hereby, a deterministic simulation is computed several times with different deviations of the input parameters to produce a variety of outputs of the same model to analyze those effects. The relevant stochastic or parametric output variables, such as mean (expected value) and variance, are often calculated and visualized only at selected individual points of the whole domain. This project aims to provide a simple way to perform stochastic or parametric post-processing of simulations results on entire domains using the VTK unstructured grid (VTU) file system (Schroeder et al., 2006) and the Julia language (Bezanson et al., 2012) as an example. The VTU file format is primarily used in conjunction with Paraview, an open-source, multi-platform data analysis and visualization tool, to display results of, e.g., structural or fluid mechanics simulations.

Statement of need

To the authors knowledge, there is no library available, neither for the VTU result file-format nor any other simulation result file-format, which standardizes stochastic/parametric post-processing. To this date, this kind of *meta* post-processing seems to be done by purely proprietary means. With this novel approach, stochastic properties can be displayed on the whole domain using well established visualization software.

State of the field

There are other approaches to writing and reading VTU, or more generally VTK, files available in the Julia community. WriteVTK.jl is a package for the creation of VTK XML files from Julia out of data already available in system memory. ReadVTK.jl is a project primarily dedicated to read data written by WriteVTK.jl. Neither of those packages explicitly addresses reading, writing and manipulating generic VTU files. However, an advanced VTK Python Wrapper does exist.

Introduction

The Visualization Toolkit (VTK) is an open source software project for manipulating and displaying scientific data. It supports a variety of visualization algorithms and advanced modeling techniques such as implicit modeling and mesh smoothing. It defines three types of file formats: a legacy format, an XML format and an HDF file format. Since the HDF file format is fairly new, the XML file format is the most used so far and will be also used here.

VTK datasets are classified into one of two categories: structured (tensor grids, image data) and unstructured (meshes). We will restrict ourselves to unstructed datasets in the following since modern simulation results are often times performed on complex geometries. Simulation results saved as VTU files can be displayed and investigated with the Paraview application.

Julia is a fast and dynamic programming language which enables fast prototyping as well as efficiently implemented software solutions. With its in-built features for numerical mathematics and distributed computing, it is very well suited for implementing computational physics. However, since Julia is a relatively newer programming language, it has often time lacks the native connections to other software projects or industry standards that exist in Python, for example.

The software presented here should make it possible to read in, manipulate and write out existing VTU files.

Motivation

Consider a discrete computational model $\mathcal{M}$, providing a generic output $\mathbf{Y}$ for a given set of inputs $\mathbf{X}$:

$$\mathbf{Y} = \mathcal{M}(\mathbf{X}) \,. \tag{1}$$

For example, the output $\mathbf{Y}$ can be a scalar, a vector, a matrix, or a finite-element post-processing result. In this case, we consider the output to be a VTU file. The input parameters are considered a set of scalars $\mathbf{X} = \{X_1, ..., X_N\}$, and for simplicity, the set is reduced to a *singleton* $(N = 1)$. Equation (1) is called the *deterministic case*. As a next step, we introduce a parametric variation $\mathbf{X} := \mathbf{X}(\boldsymbol{\xi})$, where $\boldsymbol{\xi}$ maps the inputs from a minimum to a maximum value. We refer to this problem formulation as the *parametric* (or if ξ_i, $i \in 1, ..., N$ is a random variable with a probability density function, *stochastic*) *case*:

$$\mathbf{Y}(\boldsymbol{\xi}) = \mathcal{M}(\mathbf{X}(\boldsymbol{\xi})) \,. \tag{2}$$

Since $\mathcal{M}(\mathbf{X}(\boldsymbol{\xi}))$ is no longer deterministic, further methods are required to discretize the *sample space* and to post-process and visualize the results. Different methods for uncertainty quantification can be found in Gates & Bittens (2015) or Sudret et al. (2017), for example. The most prominent method for computing the expected value of the problem described in Equation (2) is the Monte-Carlo method:

$$\mathbb{E}[\mathbf{Y}(\boldsymbol{\xi})] \approx \tilde{\mathbb{E}}[\mathcal{M}(\mathbf{X}(\boldsymbol{\xi}))] = \frac{1}{M}\sum_{i=1}^{M} \mathcal{M}(\mathbf{X}(\tilde{\boldsymbol{\xi}}_i)) \,, \quad \tilde{\xi}_{ij} \sim \mathcal{U}(0,1) \,. \tag{3}$$

From (3) we can conclude that if $\mathbf{Y}(\tilde{\boldsymbol{\xi}}_i) = \mathcal{M}(\mathbf{X}(\tilde{\boldsymbol{\xi}}_i))$ is a deterministic VTU result file at position $\tilde{\boldsymbol{\xi}}_i$ in the sample space, it is sufficient to implement the operators `+(::VTUFile,::VTUFile)` and `/(::VTUFile,::Number)` to compute the expected value on the entire domain by help of the Monte-Carlo method.

Definition of a VTUFile algebra

First we define an abstract VTUFile which represents the simulation results stored in memory as a series of coefficients of supporting points and can be thought of as a vector or matrix. As a next step, let $V = (\text{VTUFile}, +, *)$ be a field and $A \subseteq R^n$ a vector space over V. Then V is an algebra if for all $x, y, z \in A$ and $a, b \in V$ the following holds:

$$(x + y) * z = x * z + y * z \tag{4}$$
$$z * (x + y) = z * x + z * y \tag{5}$$
$$(ax) * (bx) = (ab)(x * y) \tag{6}$$

The above holds in general, if the $(*)$-operator acts scalar-wise:

$$(x * y)_i := x_i * y_i \quad \text{for all} \;\; x, y \in A \; . \tag{7}$$

Preliminaries

The VTUFileHandler will eventually be used to perform stochastic post-processing on large VTU result files. Therefore, the following assumptions have to be fulfilled for the software to work correctly:

1. The VTU file must be in binary format and, in addition, can be Zlib compressed.
2. Operators can only be applied to VTU files that share the same topology. The user must ensure that this condition is met.
3. The data type of numerical fields of the VTU file, for which operators should be applied, has to be `Float64`.

Features

The VTUFileHandler implements a basic VTU reader and writer through the functions:

```julia
function VTUFile(file::String) ... end
function Base.write(vtu::VTUFile, add_timestamp=true) ... end
```

By default, a timestamp is added if VTU files are written to disk not to overwrite existing files. Only data fields that are registered by the function

```julia
function set_uncompress_keywords(uk::Vector{String}) ... end
```

before reading the VTU file are uncompressed and can be altered. For applying math operators onto a data field, the associated field has to be registered by the function

```julia
function set_interpolation_keywords(ik::Vector{String}) ... end
```

The following math operators acting point-wise on nodal results (point data) are implemented:

```julia
+(::VTUFile, ::VTUFile),+(::VTUFile, ::Number),
-(::VTUFile, ::VTUFile),-(::VTUFile, ::Number),
*(::VTUFile, ::VTUFile),*(::VTUFile, ::Number),
/(::VTUFile, ::VTUFile),/(::VTUFile, ::Number),
^(::VTUFile, ::Number)
```

In-place variations of the operators above are implemented as well.

Example

A three-dimensional cube with dimension (x, y, z) with $0 \leq x, y, z \leq 2$ discretized by quadratic hexahedral elements with 27 points and 8 cells named `vox8.vtu` with a linear ramp in x-direction ($f(x = 0, y, z) = 0,\ f(x = 2, y, z) = 0.8$) as a result field termed `xramp` will be used as an example (see Figure 1). The following set of instructions transforms the result field from a linear ramp to a quadratic function in x-direction (displayed as a piece-wise linear field due to the discretization):

```julia
set_uncompress_keywords(["xRamp"]) # uncrompress data field xramp
set_interpolation_keywords(["xRamp"]) # apply math operators to xramp
vtu = VTUFile("vox8.vtu"); # read the vtu
vtu += vtu/4; # [0.0,...,0.8] -> [0.0,...,1.0]
vtu *= 4.0; # [0,...,1.0] -> [0.0,...,4.0]
vtu -= 2.0; # [0,...,4.0] -> [-2.0,...,2.0]
vtu ^= 2.0; # [-2.0,...,2.0] -> [4.0,...,0.0,...,4.0]
rename!(vtu, "vox8_1.vtu")
write(vtu)
```

Both, the initial (`vox8.vtu`) and the manipulated file (`vox8_1.vtu`) can be loaded and displayed with Paraview. The result is depicted in figure Figure 1.

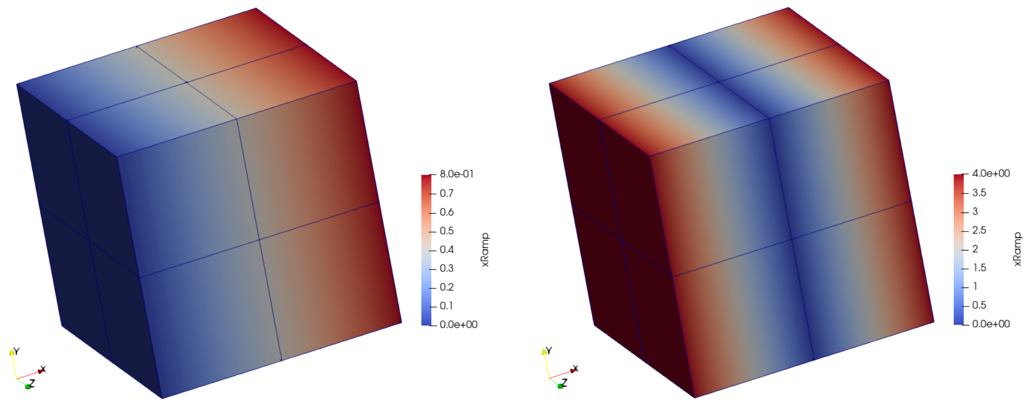

Figure 1: Cube with initial result field (left). Cube with manipulated result field (right). Rendered with Paraview.

Conclusion

A basic VTU library was implemented, which does not claim completeness in terms of VTU features. However, the implemented math operators constitute a complete feature set sufficient to compute a complete parametric or stochastic post-processing of VTU files. This implementation can readily be used for this purpose or can be utilized as a template for extending a different VTU library. The quantification of uncertainties in coupled thermo-hydro-mechanical simulations can serve as an example of an application where this tool together with ogs6py and OpenGeoSys (Buchwald et al., 2021) can be used to fully automate stochastic computations.

References

Bezanson, J., Karpinski, S., Shah, V. B., & Edelman, A. (2012). Julia: A fast dynamic language for technical computing. *arXiv Preprint arXiv:1209.5145*.

Buchwald, J., Kolditz, O., & Nagel, T. (2021). ogs6py and VTUinterface: Streamlining OpenGeoSys workflows in python. *Journal of Open Source Software*, *6*(67), 3673. https://doi.org/10.21105/joss.03673

Gates, R. L., & Bittens, M. R. (2015). A multilevel adaptive sparse grid stochastic collocation approach to the non-smooth forward propagation of uncertainty in discretized problems. *arXiv Preprint arXiv:1509.01462*.

Schroeder, W., Martin, K., & Lorensen, B. (2006). The visualization toolkit, 4th edn. kitware. *New York*. ISBN: 978-1-930934-19-1

Sudret, B., Marelli, S., & Wiart, J. (2017). Surrogate models for uncertainty quantification: An overview. *2017 11th European Conference on Antennas and Propagation (EUCAP)*, 793–797. https://doi.org/10.23919/EuCAP.2017.7928679