




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

Maximilian Bittens¹

¹ Federal Institute for Geosciences and Natural Resources (BGR)

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Mehmet Hakan Satman](#)



Reviewers:

- [@dmbates](#)
- [@mkitti](#)

Submitted: 01 March 2022

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Abstract

With increasing computing resources, investigating uncertainties in simulation results is becoming an increasingly important factor. A discrete numerical simulation is computed several times with different deviations of the input parameters to produce different 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/parametric post-processing of numerical simulations on entire domains using the VTK unstructured grid (VTU) file system and the Julia language as an example.

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.

Introduction

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

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

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 ([Schroeder et al., 2000](#)). The input parameters are considered a set of scalars $\mathbf{X} = \{X_1, \dots, X_N\}$, 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 $\mathbf{X} := \mathbf{X}(\xi)$, where ξ maps the inputs from a minimum to a maximum value. We refer to this problem formulation as the *parametric* (or if ξ_i , $i \in 1, \dots, N$ is a random variable with a probability density function, *stochastic*) case:

$$\mathbf{Y}(\xi) = \mathcal{M}(\mathbf{X}(\xi)) . \quad (2)$$

Since $\mathcal{M}(\mathbf{X}(\xi))$ 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:

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

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 $+(\text{:VTUFile}, \text{:VTUFile})$ and $/(\text{:VTUFile}, \text{:Number})$ to compute the expected value on the entire domain by help of the Monte-Carlo method.

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, have to be `Float64`.

Features

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

```
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

```
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

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

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

```
+(::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

62 linear ramp to a quadratic function in x -direction (displayed as a piecewise linear field due to
63 the discretization):

```
set_uncompress_keywords(["xRamp"]) # uncompress 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]
```

64 The initial field and the resultant field of the above operations are displayed in figure Figure 1.

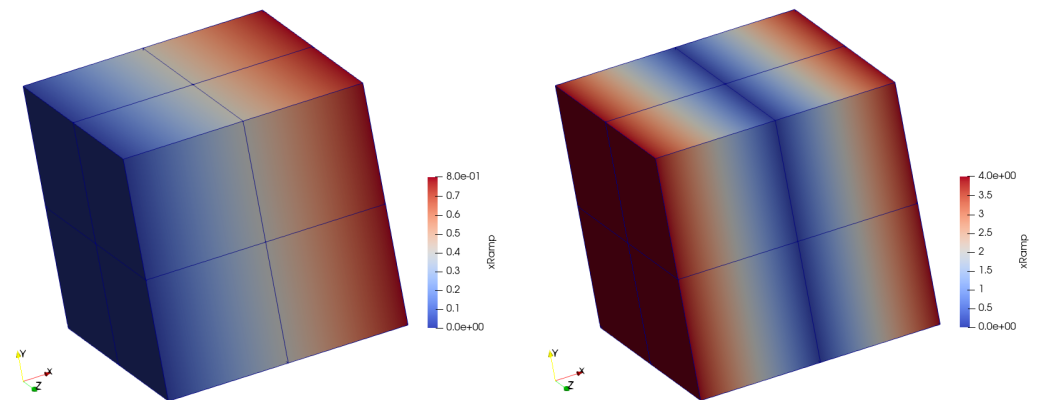


Figure 1: Cube with initial result field (left). Cube with manipulated result field (right).

65 Conclusion

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

74 References

- 75 Buchwald, J., Kolditz, O., & Nagel, T. (2021). *ogs6py* and VTUinterface: Streamlining
76 OpenGeoSys workflows in python. *Journal of Open Source Software*, 6(67), 3673. <https://doi.org/10.21105/joss.03673>
77
78 Gates, R. L., & Bittens, M. R. (2015). A multilevel adaptive sparse grid stochastic collocation
79 approach to the non-smooth forward propagation of uncertainty in discretized problems.
80 *arXiv Preprint arXiv:1509.01462*.
81 Schroeder, W. J., Avila, L. S., & Hoffman, W. (2000). Visualizing with VTK: A tutorial. *IEEE*
82 *Computer Graphics and Applications*, 20(5), 20–27. <https://doi.org/10.1109/38.865875>
83 Sudret, B., Marelli, S., & Wiart, J. (2017). Surrogate models for uncertainty quantification:
84 An overview. *2017 11th European Conference on Antennas and Propagation (EUCAP)*,

DRAFT