Aerospace Science and Technology 45 (2015) 272–283

Contents lists available at ScienceDirect

Aerospace Science and Technology

www.elsevier.com/locate/aescte

Cassiopee: A CFD pre- and post-processing tool

Christophe Benoit *, Stéphanie Péron *, Sâm Landier *

ONERA – The French Aerospace Lab, F-92322 Châtillon, France

ARTICLE INFO

Article history:
Received 22 January 2015
Received in revised form 29 May 2015
Accepted 30 May 2015
Available online 3 June 2015

Keywords:
Computational Fluid Dynamics
Pre-processing
Post-processing
CGNS
Open source software

ABSTRACT

This paper presents an overview of the capabilities of a new open-source pre- and post-processing tool for Computational Fluid Dynamics simulations, called Cassiopee. Its architecture, which is basically a set of Python modules, and the handled data, which is based on CGNS standard, are described. Some examples of workflows that can be built with Cassiopee functions are provided. Finally, some applications of Cassiopee functions to realistic CFD configurations are briefly presented.

1. Introduction

In 2008, ONERA started a project to gather pre- and post-processing tools in a single software, called Cassiopee [1], following other initiatives as Salome [2] or gmsh [3]. At that time, pre- and post-processing tools were spread among scientists, each one generally having its file formats and data representations. Thus, it seemed clear that capitalizing all the knowledge in a single software environment and making it interoperable was worth the effort. However, this effort pays off only if the common data representation is well accepted and easily accessible to users.

For this reason, the Python language was chosen as a high level interface, since it is very fast to start with, easy to use and is spread in the scientific community (see for example: SciPy [4], matplotlib [5], Paraview [6], FieldView [7] or Tecplot [8]).

To simplify the use of Cassiopee, two interfaces are provided: the first one, that we call the “array” interface, is directly based on numpy arrays. In this way, Cassiopee functions can be easily mixed with functions using the numpy library.

The second interface is based on the Python representation of the CFD General Notation System (CGNS) of data [9]. The data handled by Cassiopee functions is then an imbricated set of lists describing a computation tree, compliant with the CGNS standard, as described by Poinot [10]. We call this interface the “pyTree” interface.

Functions are classified in thematic Python modules, according to their features. Each Python module is independent and can be compiled and installed separately from the others.

Currently, among all the available functionalities, one can for example perform minor mesh modifications, mesh improvements (e.g. mesh smoothing), preprocessing of a CFD computation (e.g. connectivity computation or mesh splitting), code coupling or solution post-processing.

Unlike other mesh generators dedicated to CFD simulations, such as Pointwise [11] or ANSYS ICEM CFD [12], Cassiopee does not provide a full solution to mesh generation. It is much more used as a supplementary tool for specific mesh generation or local modifications (refinement, local smoothing...). It is also used to quickly develop prototype algorithms of mesh generation.

As compared to other overset grid assembly tools, such as Overture [13], Suggar++ [14] or Chimera Grid Tools [15], the software gathers the algorithms developed at ONERA [16,17].

Concerning post-processing, Cassiopee has a very limited ambition and does not position itself within other post-processors, such as VisIt [18], Paraview, Ensight [19] or FieldView. So far, implemented algorithms distributed as open-source are classic ones. Nevertheless, post-processing algorithms can be used in the same Python script as mesh generation functions, enabling to tackle some specific problems.

Today, these modules are commonly used for CFD and CAA applications at ONERA and some European aerospace industries. For instance, let us cite some rotorcraft, CROR and aircraft configurations that use Cassiopee mesh generation functions [20–22], overset grid assembly functions [23–25] and post-processing functions [26]. In 2013, Cassiopee modules were released under the

* Corresponding authors.
E-mail addresses: christophe.benoit@onera.fr (C. Benoit), stephanie.peron@onera.fr (S. Péron), sam.landier@onera.fr (S. Landier).

http://dx.doi.org/10.1016/j.ast.2015.05.023
1270-9638/

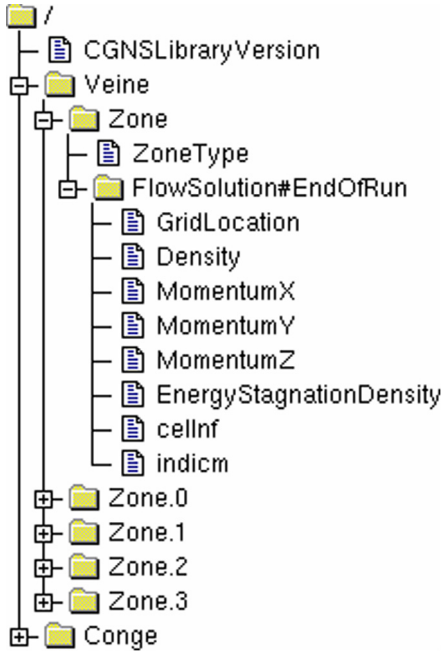

Fig. 1. Example of a CGNS/Python (or pyTree) representation.

GNU General Public License GPL3, in the hope that it will be useful to the CFD community.

2. Design choices

As previously said, the pyTree is one of the two choices of data representations available in Cassiopee. From now on, we will focus on this interface only. A pyTree is a hierarchical data set where each node is a Python list of this type:

```
['name', v, [], 'Type']
```
(1)

The string `'name'` is simply the name of the node, `v` is a numpy array defining the value of the node (for instance, for the first component of the grid coordinates, say 'CoordinateX', the numpy array stores all the x-coordinate values for the corresponding zone), the string `'Type'` is a CGNS-compliant name and describes the node type (e.g. `'Zone_t'` for a zone node). The third element of the Python list is [] and define the list of nodes that are the current node's children.

This structure enables to store the mesh coordinates (in a node named 'GridCoordinates'), the flow solution (located at nodes and at centers), boundary conditions (in a node of name 'ZoneBC' for each zone) or the grid connectivity. An example of a representation of a pyTree is displayed in Fig. 1.

This data structure is common to all the modules of Cassiopee. In addition, the philosophy of Cassiopee modules is purely functional, hence a Cassiopee function can be written as:

$$b = f(a) \text{ or } \_f(a) \tag{2}$$

where a and b are pyTrees. Function f returns a copy b of the pyTree, whereas function prefixed with a "_" results in an in-place modification of the pyTree (the given pyTree is directly modified by the function $\_f$). For the sake of generality, we try (as far as possible) to make all the Cassiopee functions deal with all mesh types: structured meshes, unstructured meshes with basic elements, and polyhedral meshes, as displayed in Fig. 2. Element names are also defined according to the CGNS standard: for instance an 'NGON/NFACES'-type mesh describes a general polyhedral mesh.

Another original feature of Cassiopee functions is that boundary conditions and solutions (defined both at nodes and centers) are also modified by the functions wherever this is relevant.

For example, consider a simple function that subzones a given zone. It can be applied either on structured grids (using min–max indices in the three directions of the zone) or on unstructured grids (given a list of element indices). When the subzone is created, not only the grid coordinates are extracted, but also the solutions at nodes and cell centers and the boundary conditions as well.

3. Content of Cassiopee modules

The software is made of independent Python modules. Each module can be compiled and installed separately. The release R-3 of Cassiopee contains thirteen modules [1], including `Converter`, `Geom`, `Generator`, `Transform`, `Connector`, `Post`, described briefly in the following.

3.1. Converter module

`Converter` module enables input/output between the in-memory data representation and various file formats of discrete data, such as Tecplot, plot3d, mesh, STL, OBJ... and of course CGNS/ADF and CGNS/HDF5 formats. `Converter` module also provides low-level functions to handle pyTrees, such as pyTree node

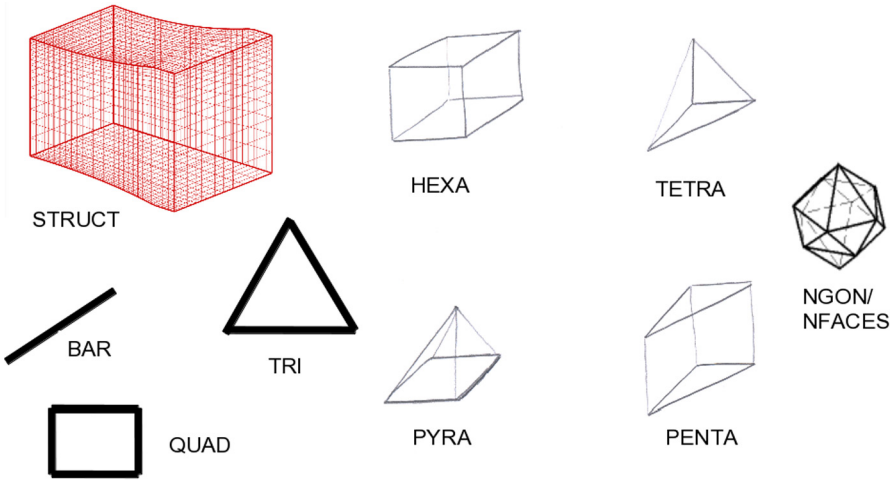

Fig. 2. Mesh types compliant with Cassiopee functions.