NUREG/IA-0539

International
Agreement Report

New Functionality of TRACE: The 3D Post-Processing for the VESSEL Component in SALOME Platform

Prepared by:
Kanglong Zhang; Victor Hugo Sanchez-Espinoza
Institute for Neutron Physics and Reactor Technology (INR), Karlsruhe Institute of Technology (KIT) Hermann-von-Helmholtz-Platz 1
Eggenstein-Leopoldshafen, Baden-Württemberg, 76344, Germany

K.Tien, NRC Project Manager

Division of Systems Analysis
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

Manuscript Completed: July 2022
Date Published: May 2024

Prepared as part of
The Agreement on Research Participation and Technical Exchange
Under the Thermal-Hydraulic Code Applications and Maintenance Program (CAMP)

Published by
U.S. Nuclear Regulatory Commission

NUREG/IA-0539

International Agreement Report

New Functionality of TRACE: The 3D Post-Processing for the VESSEL Component in SALOME Platform

Prepared by:
Kanglong Zhang; Victor Hugo Sanchez-Espinoza
Institute for Neutron Physics and Reactor Technology (INR), Karlsruhe Institute of Technology (KIT) Hermann-von-Helmholtz-Platz 1
Eggenstein-Leopoldshafen, Baden-Württemberg, 76344, Germany

K.Tien, NRC Project Manager

Division of Systems Analysis
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

Manuscript Completed: July 2022
Date Published: May 2024

Prepared as part of
The Agreement on Research Participation and Technical Exchange
Under the Thermal-Hydraulic Code Applications and Maintenance Program (CAMP)

Published by
U.S. Nuclear Regulatory Commission

ABSTRACT

The researchers from the group of Reactor Physics and Dynamics (RPD) which is under the Institute of Institute for Neutron Physics and Reactor Technology (INR) - Karlsruhe Institute of Technology (KIT) - Germany, have developed a new post-processing functionality for the U.S. NRC system thermal-hydraulic code - TRACE. Now, the 3D VESSEL component of TRACE and the calculated physical fields stored in the component can be visualized in a pre- and post-processing open-source platform - SALOME with the help of a powerful data-processing library - MEDCoupling.

The researchers develop new Fortran and C++ routines to automatically identify and select the geometrical data set from TRACE input files. This data set is processed by the MEDCoupling library to generate a polyhedron mesh and a surface mesh. They are both 3D objects. The former store cell-centered fields e.g., the coolant temperature and the pressure while the latter store face-located variables e.g., the coolant velocity and the pressure drop.

Twenty-one kinds of fields can be post-processed associated with the two VESSEL meshes at present including the coolant density, the void fraction, etc. Users can conveniently access the VESSEL meshes and the fields in SALOME with plenty of operations e.g., cutting, data filtering. Scaling up the processed fields is now in planning.

This functionality was tested by processing the results gained from a VVER-1000 coolant mixing simulation.

FOREWORD

This report describes the development and demonstration of the post-processing of the TRACE 3D VESSEL component. This new functionality is a tremendous supplementary to the classical post-processing methods known as the 1D curves plotting by AptPlot and 2D/3D chromatic graphs displaying by SNAP. A polyhedron mesh and a surface mesh are developed for the VESSEL component and twenty-one kinds of fields can be write in, with the MEDCoupling library. They can be post-processed in the SALOME platform.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

The scope of this report is to present the development of the 3D visualized post-processing ability for the Cylinder and Cartesian VESSEL components of TRACE. This is based on the MEDCoupling open-source library and the SALOME platform. Currently, twenty-one fields in TRACE can be visualized in SALOME platform. Those fields can be processed with various operations e.g., cut, sliced, and filtered.

ABBREVIATIONS AND ACRONYMS

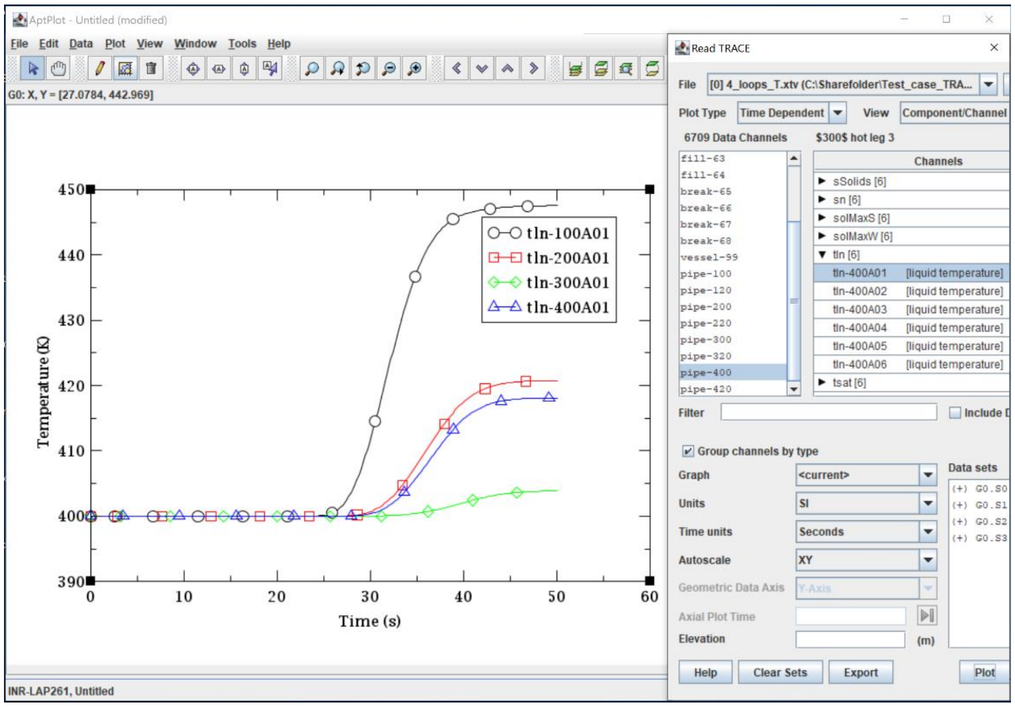
CAFEAN	Common Application Framework for Engineering Analysis
LWR	Light Water Reactor
NPP	Nuclear Power Plant
NRC	U.S. Nuclear Regulatory Commission
PWR	Pressurized light-Water Reactor
RPV	Nuclear Pressure Vessel
SNAP	Symbolic Nuclear Analysis Package

1 INTRODUCTION

1.1 Overview of TRACE

TRACE is the reference best-estimate thermal-hydraulic system code of the U.S. Nuclear Regulatory Commission (NRC) for Light Water Reactors (LWR). A system of six balance equations in the two-fluid formulation plus additional equations to describe the transport of boron dissolved in the liquid phase and of non-condensable in the gas phase is solved for one-dimensional and three-dimensional components used to represent a nuclear power plant [1]. Besides, correlations for heat transfer in all relevant heat transfer modes of vertical and horizontal flow regimes are implemented together with a heat conduction solver for structures with and without a heat source. TRACE contains different components such as PIPE, VALVE, VESSEL, HTSTR, CHAN, POWER, CONTAN to represent various parts or systems of a Nuclear Power Plant (NPP). Among the various components, VESSEL is the special 3D item that can model the Reactor Pressure Vessel (RPV), RWST, and other components in which 3D phenomena take place. By taking advantage of VESSEL, TRACE can simulate a 3D (x, y, z) Cartesian- and/or (r, θ, z) cylindrical-geometry flow calculation. Dedicated models are also available for the description of critical flow, thermal stratification, counter-current flow, etc.

1.2 Current Post-Processing of TRACE



File: [0] 4_loops_T.xtv (C:\Sharefolder\Test_case_TRA...
Plot Type: Time Dependent   View: Component/Channel
6709 Data Channels 300 hot leg 3

fill-63, fill-64, break-65, break-66, break-67, break-68, vessel-99, pipe-100, pipe-120, pipe-200, pipe-220, pipe-300, pipe-320, pipe-400, pipe-420

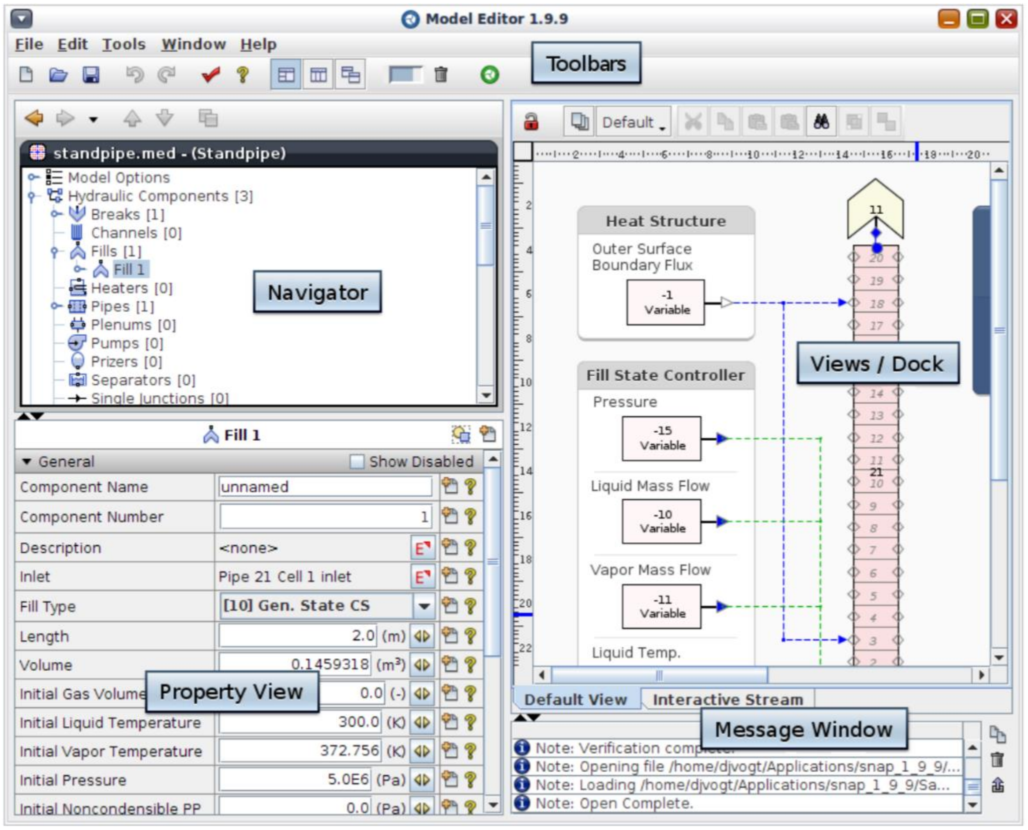
Channels: sSolids [6], sn [6], solMaxS [6], solMaxW [6], tln [6]: tln-400A01 [liquid temperature], tln-400A02 [liquid temperature], tln-400A03 [liquid temperature], tln-400A04 [liquid temperature], tln-400A05 [liquid temperature], tln-400A06 [liquid temperature]; tsat [6]

Filter: Include
[✓] Group channels by type
Graph: <current>
Units: SI
Time units: Seconds
Autoscale: XY
Geometric Data Axis: Y-Axis
Axial Plot Time:
Elevation: (m)
Data sets: (+) G0.S0, (+) G0.S1, (+) G0.S2, (+) G0.S3

Figure 1-1 The General Appearance of AptPlot

When TRACE is launched under a command window, it will read in the input file and generates several output files including the XTV, TPR, OUT, and other files. The XTV file contains the plotting data for all components simulated in the model and it can be imported by the AptPlot tool which is a simple scientific drawing program. AptPlot is a free WYSIWYG 2D plotting tool designed for creating production quality plots of numerical data and performing data analysis. AptPlot contains extensive scripting and GUI support for the manipulation and analysis of data sets. AptPlot is intended to be a pure-Java drop-in replacement for the Motif X-Window based Grace plotting package maintained by the Grace Team and coordinated by Evgeny Stambulchik. It can be accessed from https://www.appliedprog.com/aptplot/index.jsp .

The appearance of AptPlot is illustrated in Figure 1-1. The figure displays the main interface and the sub-interface for data selection. More sub-interfaces e.g., the curve editing interface, the legend editing interface, the coordination editing interface could be launched as required. AptPlot is a good tool for careful quantitative analysis and the time evolution of fields. However, it can only plot 1D curves and one plot should not contain too many curves to avoid displaying lots of information in a mess. This limitation is the inherent drawback for qualitative analysis e.g., the fields' distribution on the full vessel scale. Figuring out the critical information from a bunch of curves is difficult and sometimes impossible.



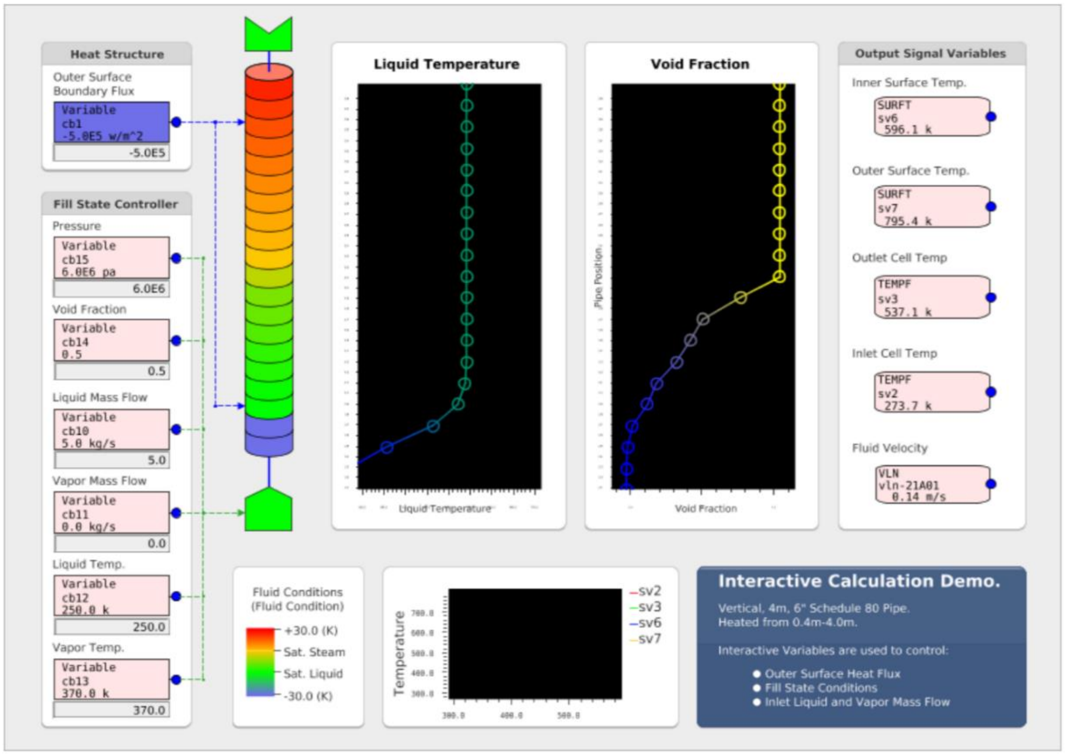
Component Name	unnamed
Component Number	1
Description	<none>
Inlet	Pipe 21 Cell 1 inlet
Fill Type	[10] Gen. State CS
Length	2.0 (m)
Volume	0.1459318 (m³)
Initial Gas Volume	0.0 (-)
Initial Liquid Temperature	300.0 (K)
Initial Vapor Temperature	372.756 (K)
Initial Pressure	5.0E6 (Pa)
Initial Noncondensible PP	0.0 (Pa)

```
Note: Verification comp
Note: Opening file /home/djvogt/Applications/snap_1_9_9/...
Note: Loading /home/djvogt/Applications/snap_1_9_9/Sa...
Note: Open Complete.
```

Figure 1-2 The Main Panel of the Model Editor of SNAP

The other classical tool to post-process the TRACE result is the Symbolic Nuclear Analysis Package (SNAP) with the animation plug-in. SNAP consists of a suite of integrated applications designed to simplify the process of performing engineering analysis. SNAP is built on the Common Application Framework for Engineering Analysis (CAFEAN) which provides a highly flexible framework for creating and editing input for engineering analysis codes as well as extensive functionality for submitting, monitoring, and interacting with the codes. For the nuclear community, SNAP provides researchers a powerful GUI simplifying the input files composing for

TRACE, RELAP, PARCS, etc. Moreover, SNAP can cover the complete simulation procedures from model composing, computation launching, and results post-processing. SNAP provides researchers an advanced option to manipulate and control simulations at a high level. SNAP has several main components known as the model editor, configuration tool, etc. The model editor is the dominant component whose main panel is exhibited in Figure 1-2.



Variable cb1 -5.0E5 w/m^2 | -5.0E5

Variable cb15 6.0E6 pa | 6.0E6

Variable cb14 0.5 | 0.5

Variable cb10 5.0 kg/s | 5.0

Variable cb11 0.0 kg/s | 0.0

Variable cb12 250.0 k | 250.0

Variable cb13 370.0 k | 370.0

SURFT sv6 596.1 k

SURFT sv7 795.4 k

TEMPF sv3 537.1 k

TEMPF sv2 273.7 k

VLN vln-21A01 0.14 m/s

Interactive Calculation Demo.

Vertical, 4m, 6" Schedule 80 Pipe.
Heated from 0.4m-4.0m.

Interactive Variables are used to control:

- Outer Surface Heat Flux
- Fill State Conditions
- Inlet Liquid and Vapor Mass Flow

Figure 1-3 The Window Displaying the Animation of TRACE Results

Particularly for the post-processing, when the simulation terminates, SNAP can directly call AptPlot internally and process the XTV file. Besides, SNAP has interactive post-processing capabilities displaying 2D or 3D colorful animations assisted by the animation plug-in. Within such a display, the results of a calculation may be animated in a variety of ways. Figure 1-3 exhibits one standard animation window displaying the TRACE results. An animation display retrieves data from a Calculation Server and represents it visually in some fashion. This data can be from actively running calculations, completed calculations, imported EXTData files, etc.

SNAP usually uses some objects called display beans to represent the various components in the color animations. Most of the beans are 2D viewed object e.g., the PIPE, VALVE, etc. Besides, a special 3D display bean corresponds to the 3D VESSEL component, which can display the whole vessel geometry as well as the fields inside and also the vessel rings, see Figure 1-4.

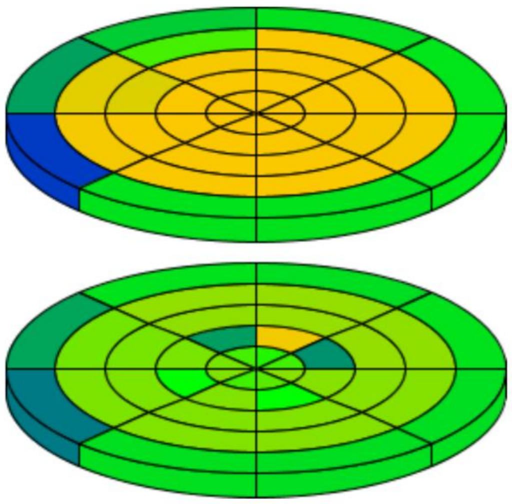


Figure 1-4 The Vessel Ring Display Bean for 3D VESSEL Component of TRACE

Compared with AptPlot, the animation plug-in in SNAP makes the qualitative analysis possible, especially in the complex VESSEL component where lots of cells stack up. However, the careful inspection of the fields in this bean is not that feasible, especially when the inner cells are interested. Also, the methods to manipulate the fields or geometrical cells of the VESSEL are not satisfactory. There are not plentiful filters helping to process the 3D display bean of VESSEL, which weakens the quantitative analyzing capability. Another potential limitation of the SNAP 3D animation is that the fields and the "mesh" are mixed. This feature does not follow the tendency of modern programs whose mesh and fields normally separate.

1.3 Objectives of the Report

To take advantage of modern powerful post-processing toolkits, an open-source data-processing library – MEDCoupling was adopted to generate two 3D meshes for the VESSEL component of TRACE. A polyhedron mesh and a surface mesh are explicitly defined and written as mesh files. The physical fields naturally separate from the meshes. All the meshes and fields are in MED format according to the MEDCoupling library. They can be processed by the open-source SALOME platform with various, plentiful, and powerful filters. Moreover, the newly generated meshes and fields can be applied to codes' coupling issues in a quite straightforward manner thanks to the coupling-oriented features of the MEDCoupling library.

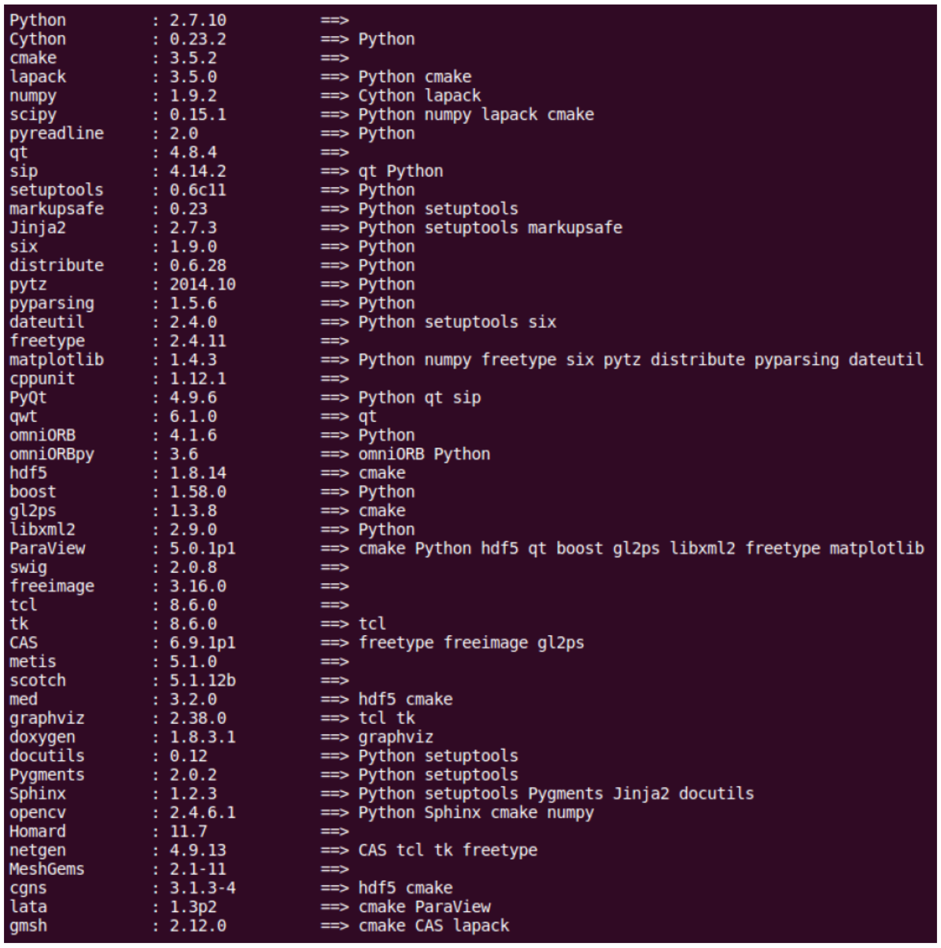
1.4 Scope of the Report

This report is subdivided into six chapters. The first chapter mainly talks about the current post-processing of TRACE. The second chapter describes some basic knowledge of the SALOME platform and the MEDCoupling library. The third chapter details the mesh generation for the 3D VESSEL component of TRACE and the logics of the enhanced TRACE package. A demonstration with a VVER-1000 coolant mixing case is exhibited in Chapter 4. Finally, the main conclusions and outlook are summarized in Chapters 5 and 6.

2 THE SALOME PLATFORM AND MEDCOUPLING LIBRARY

The new 3D meshes for the TRACE 3D VESSEL component as well as the fields are constructed based on the MEDCoupling library and they are processed in the SALOME platform. Thus some general knowledge of the platform and library are key items to be delivered in this chapter.

2.1 Overview of SALOME

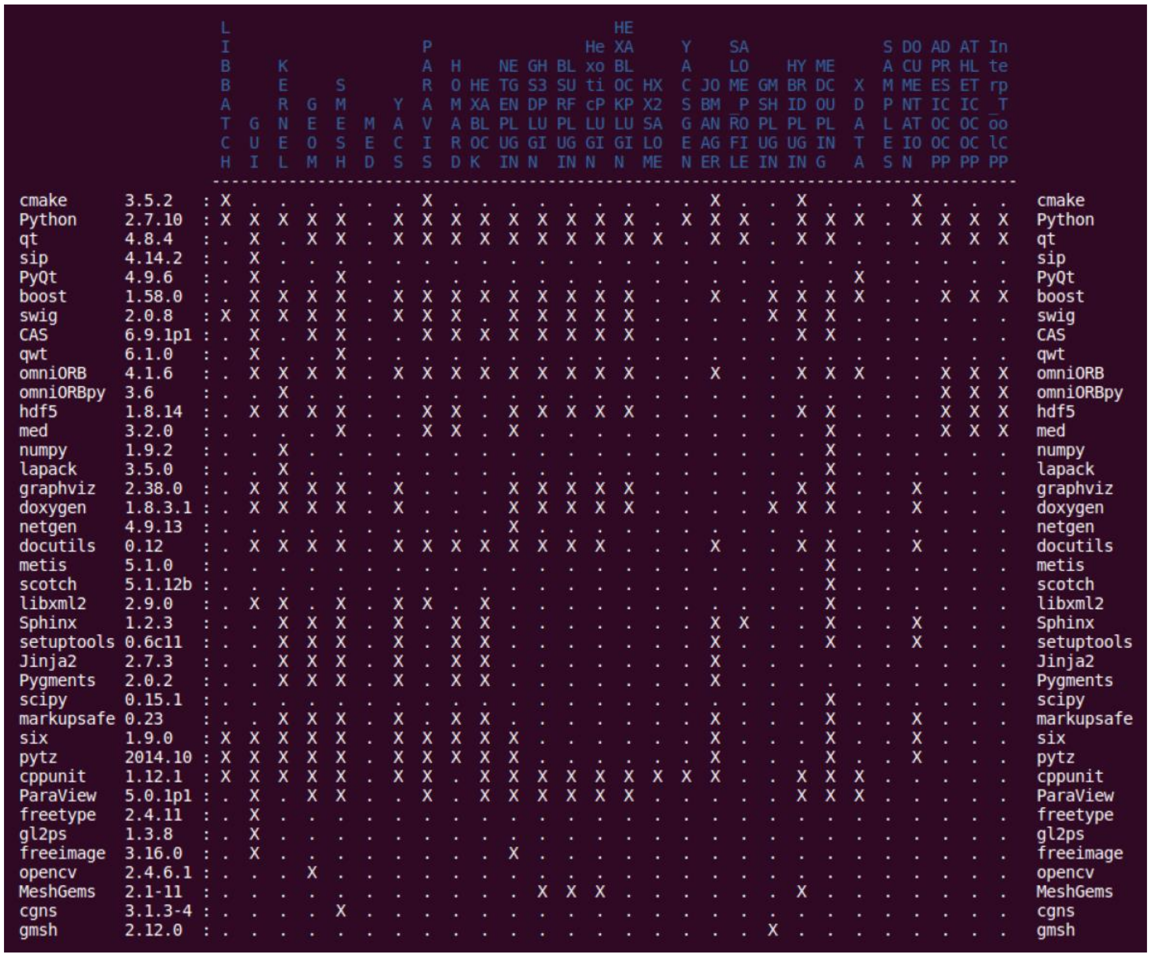


```
Python          : 2.7.10          ==>
Cython          : 0.23.2          ==> Python
cmake           : 3.5.2           ==>
lapack          : 3.5.0           ==> Python cmake
numpy           : 1.9.2           ==> Cython lapack
scipy           : 0.15.1          ==> Python numpy lapack cmake
pyreadline      : 2.0             ==> Python
qt              : 4.8.4           ==>
sip             : 4.14.2          ==> qt Python
setuptools      : 0.6c11          ==> Python
markupsafe      : 0.23            ==> Python setuptools
Jinja2          : 2.7.3           ==> Python setuptools markupsafe
six             : 1.9.0           ==> Python
distribute      : 0.6.28          ==> Python
pytz            : 2014.10         ==> Python
pyparsing       : 1.5.6           ==> Python
dateutil        : 2.4.0           ==> Python setuptools six
freetype        : 2.4.11          ==>
matplotlib      : 1.4.3           ==> Python numpy freetype six pytz distribute pyparsing dateutil
cppunit         : 1.12.1          ==>
PyQt            : 4.9.6           ==> Python qt sip
qwt             : 6.1.0           ==> qt
omniORB         : 4.1.6           ==> Python
omniORBpy       : 3.6             ==> omniORB Python
hdf5            : 1.8.14          ==> cmake
boost           : 1.58.0          ==> Python
gl2ps           : 1.3.8           ==> cmake
libxml2         : 2.9.0           ==> Python
ParaView        : 5.0.1p1         ==> cmake Python hdf5 qt boost gl2ps libxml2 freetype matplotlib
swig            : 2.0.8           ==>
freeimage       : 3.16.0          ==>
tcl             : 8.6.0           ==>
tk              : 8.6.0           ==> tcl
CAS             : 6.9.1p1         ==> freetype freeimage gl2ps
metis           : 5.1.0           ==>
scotch          : 5.1.12b         ==>
med             : 3.2.0           ==> hdf5 cmake
graphviz        : 2.38.0          ==> tcl tk
doxygen         : 1.8.3.1         ==> graphviz
docutils        : 0.12            ==> Python setuptools
Pygments        : 2.0.2           ==> Python setuptools
Sphinx          : 1.2.3           ==> Python setuptools Pygments Jinja2 docutils
opencv          : 2.4.6.1         ==> Python Sphinx cmake numpy
Homard          : 11.7            ==>
netgen          : 4.9.13          ==> CAS tcl tk freetype
MeshGems        : 2.1-11          ==>
cgns            : 3.1.3-4         ==> hdf5 cmake
lata            : 1.3p2           ==> cmake ParaView
gmsh            : 2.12.0          ==> cmake CAS lapack
```

Figure 2-1 The Dependent 3rd Party Packages of SALOME

The SALOME platform is an open-source software framework for numerical pre- and post-processing and integration of numerical solvers in various scientific domains. It is supported by CEA, EDF, and OPEN CASCADE. SALOME has already been employed to perform a wide range of simulations, which are typically related to industrial equipment in power plants (nuclear power plants, wind turbines, dams).

SALOME is developed and built based on dozens of 3rd party packages or toolkits e.g., Python, QT, gmsh. The most right column in Figure 2-1 lists the complete set of those packages whose version information is print in the second column. The other columns in the figure tell the interdependence of those packages. Please note that the specific versioned packages are for SALOME 7.8.0 only. Different versions of SALOME normally depend on different versions of those packages. They are called prerequisites in the SALOME platform and concreate the very low-level cornerstones for all other super-structures of SALOME.



```
                     L                                          HE
                     I                    P                 He XA    Y     SA             S  DO AD AT In
                     B     K              A  H     NE GH BL xo BL    A     LO    HY ME    A  CU PR HL te
                     B     E     S        R  O  HE TG S3 SU ti OC HX C  JO ME GM BR DC X  M  ME ES ET rp
                     A     R  G  M     Y  A  M  XA EN DP RF cP KP X2 S  BM _P SH ID OU D  P  NT IC IC _T
                     T  G  N  E  E  M  A  V  A  BL PL LU PL LU LU SA G  AN RO PL PL PL A  L  AT OC OC oo
                     C  U  E  O  S  E  C  I  R  OC UG GI UG GI GI LO E  AG FI UG UG IN T  E  IO OC OC lC
                     H  I  L  M  H  D  S  S  D  K  IN N  IN N  N  ME N  ER LE IN IN G  A  S  N  PP PP PP
                    -------------------------------------------------------------------------------------
cmake      3.5.2   : X  .  .  .  .  .  .  X  .  .  .  .  .  .  .  .  .  X  .  .  X  .  .  .  X  .  .  .    cmake
Python     2.7.10  : X  X  X  X  X  .  X  X  X  X  X  X  X  X  X  .  X  X  X  .  X  X  X  .  X  X  X  X    Python
qt         4.8.4   : .  X  .  X  X  .  X  X  X  X  X  X  X  X  X  X  .  X  X  .  X  X  .  .  .  X  X  X    qt
sip        4.14.2  : .  X  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .    sip
PyQt       4.9.6   : .  X  .  .  X  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  X  .  .  .  .  .    PyQt
boost      1.58.0  : .  X  X  X  X  .  X  X  X  X  X  X  X  X  X  .  .  X  .  X  X  X  X  .  .  X  X  X    boost
swig       2.0.8   : X  X  X  X  X  .  X  X  X  .  X  X  X  X  X  .  .  .  .  X  X  X  .  .  .  .  .  .    swig
CAS        6.9.1p1 : .  X  .  X  X  .  .  X  X  X  X  X  X  X  X  .  .  .  .  .  X  X  .  .  .  .  .  .    CAS
qwt        6.1.0   : .  X  .  .  X  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .    qwt
omniORB    4.1.6   : .  X  X  X  X  .  X  X  X  X  X  X  X  X  X  .  .  X  .  .  X  X  X  .  .  X  X  X    omniORB
omniORBpy  3.6     : .  .  X  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  X  X  X    omniORBpy
hdf5       1.8.14  : .  X  X  X  X  .  .  X  X  .  X  X  X  X  X  .  .  .  .  .  X  X  .  .  .  X  X  X    hdf5
med        3.2.0   : .  .  .  .  X  .  .  X  X  .  X  .  .  .  .  .  .  .  .  .  .  X  .  .  .  X  X  X    med
numpy      1.9.2   : .  .  X  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  X  .  .  .  .  .  .    numpy
lapack     3.5.0   : .  .  X  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  X  .  .  .  .  .  .    lapack
graphviz   2.38.0  : .  X  X  X  X  .  X  .  .  .  X  X  X  X  X  .  .  .  .  .  X  X  .  .  X  .  .  .    graphviz
doxygen    1.8.3.1 : .  X  X  X  X  .  X  .  .  .  X  X  X  X  X  .  .  .  .  X  X  X  .  .  X  .  .  .    doxygen
netgen     4.9.13  : .  .  .  .  .  .  .  .  .  .  X  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .    netgen
docutils   0.12    : .  X  X  X  X  .  X  X  X  X  X  X  X  X  .  .  .  X  .  .  X  X  .  .  X  .  .  .    docutils
metis      5.1.0   : .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  X  .  .  .  .  .  .    metis
scotch     5.1.12b : .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  X  .  .  .  .  .  .    scotch
libxml2    2.9.0   : .  X  X  .  X  .  X  X  .  X  .  .  .  .  .  .  .  .  .  .  .  X  .  .  .  .  .  .    libxml2
Sphinx     1.2.3   : .  .  X  X  X  .  X  .  X  X  .  .  .  .  .  .  .  X  X  .  .  X  .  .  X  .  .  .    Sphinx
setuptools 0.6c11  : .  .  X  X  X  .  X  .  X  X  .  .  .  .  .  .  .  X  .  .  .  X  .  .  X  .  .  .    setuptools
Jinja2     2.7.3   : .  .  X  X  X  .  X  .  X  X  .  .  .  .  .  .  .  X  .  .  .  .  .  .  .  .  .  .    Jinja2
Pygments   2.0.2   : .  .  X  X  X  .  X  .  X  X  .  .  .  .  .  .  .  X  .  .  .  .  .  .  .  .  .  .    Pygments
scipy      0.15.1  : .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  X  .  .  .  .  .  .    scipy
markupsafe 0.23    : .  .  X  X  X  .  X  .  X  X  .  .  .  .  .  .  .  X  .  .  .  X  .  .  X  .  .  .    markupsafe
six        1.9.0   : X  X  X  X  X  .  X  X  X  X  X  .  .  .  .  .  .  X  .  .  .  X  .  .  X  .  .  .    six
pytz       2014.10 : X  X  X  X  X  .  X  X  X  X  X  .  .  .  .  .  .  X  .  .  .  X  .  .  X  .  .  .    pytz
cppunit    1.12.1  : X  X  X  X  X  .  X  X  .  X  X  X  X  X  X  X  X  X  .  .  X  X  X  .  .  .  .  .    cppunit
ParaView   5.0.1p1 : .  X  .  X  X  .  .  X  .  X  X  X  X  X  X  .  .  .  .  .  X  X  X  .  .  .  .  .    ParaView
freetype   2.4.11  : .  X  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .    freetype
gl2ps      1.3.8   : .  X  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .    gl2ps
freeimage  3.16.0  : .  X  .  .  .  .  .  .  .  .  X  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .    freeimage
opencv     2.4.6.1 : .  .  .  X  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .    opencv
MeshGems   2.1-11  : .  .  .  .  .  .  .  .  .  .  .  X  X  X  .  .  .  .  .  .  X  .  .  .  .  .  .  .    MeshGems
cgns       3.1.3-4 : .  .  .  .  X  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .    cgns
gmsh       2.12.0  : .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  X  .  .  .  .  .  .  .  .    gmsh
```

Figure 2-2 The Dependency of SALOME Modules to SALOME Prerequisites

Figure 2-2 prints the dependency of SALOME modules to SALOME prerequisites. The rightmost two columns are the names and versions of the SALOME prerequisites. The leftmost column is a direct copy of the prerequisite names. The dark blue texts on the top of the figure are the high-level modules in SALOME. The "X" in the central table tells that the module in the corresponding column depends on the prerequisite on the corresponding row.

Among the various components, eight modules constitute the main trunk and branches of SALOME. The program outline is exhibited in Figure 2-3. There, KERNEL and GUI provide the core functionalities of SALOME, GEOM is for CAD usage while MESH is in charge of generating computational grids, PARAVIS is nothing but ParaView which is post-processing professional, MED contains lots of mesh interpolation tools and also supplies a universal data format standard for all other modules, YACS is used to organize and monitor calculation chains, the last stands for the user-developed modules. In total, KERNEL and GUI are the base, GEOM, MESH, and

PARAVIS are for pre- and post-processing, MED, YACS, and user-module are closely related to coupling issues. Coupling is another significant functionality of SALOME, which potentially benefits the coupling development involving TRACE.

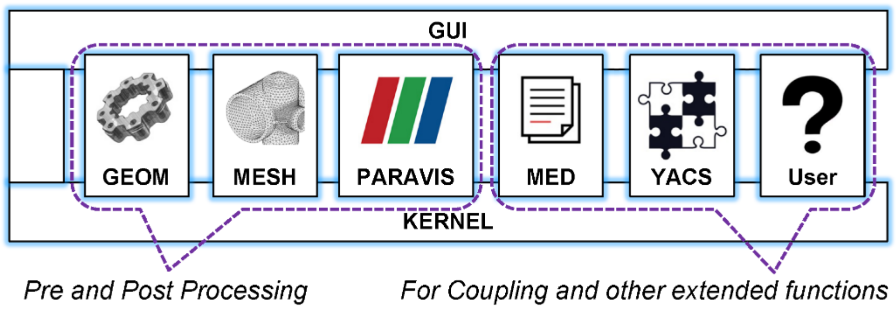


Figure 2-3 The General Structure of the SALOME Platform

PARAVIS is the module for post-processing. Its main interface is displayed in Figure 2-4.

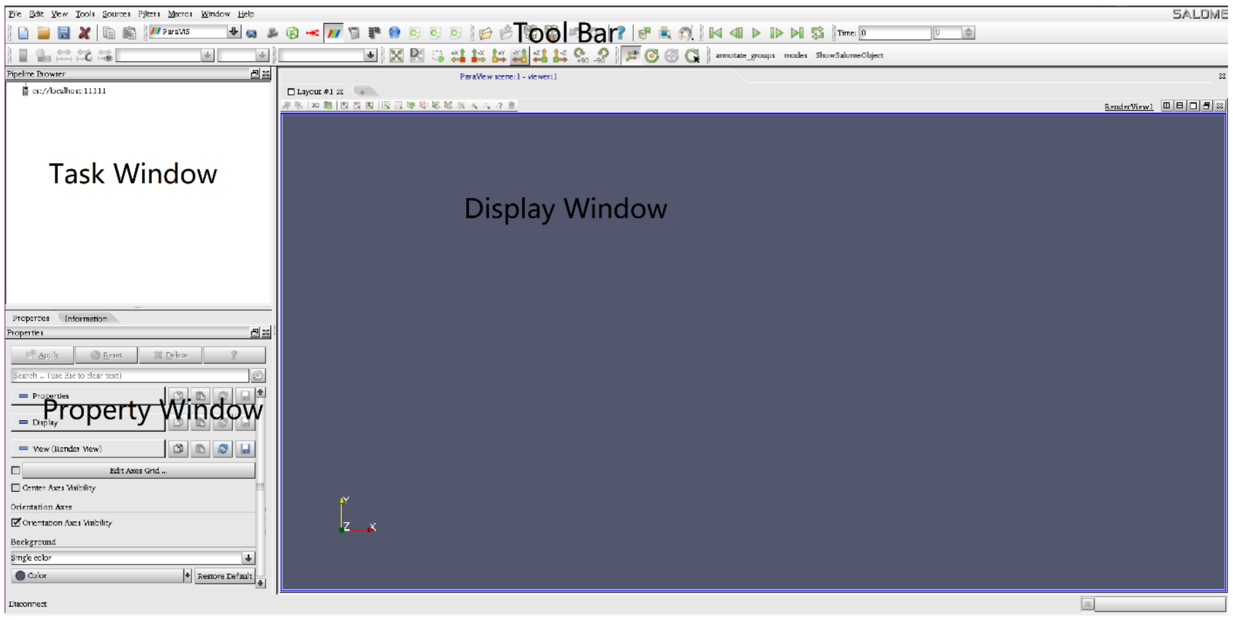


Figure 2-4 The PARAVIS Main Interface in SALOME

The windows layout is quite similar to ParaView. Precisely, PARAVIS is a special Paraview with some additional functions in SALOME. The demonstration in Chapter 4 plays in this window.

2.2 The MEDCoupling Library

MEDCoupling is a data-processing library in general including format definition and plenty of methods. It gathers several powerful functionalities around the input and output data of field-physics-oriented simulation codes. Data manipulated by MEDCoupling are objects relative to fields for simulation codes. The functionalities are accessible through python modules and are split into 4 categories:

1) Data movement: read/write from/to file, reduce, extract, duplicate, aggregate, compare, exchange data memory to memory across process (image of multifile to file);
2) Data analysis: extract/gather information in data to transform it into a condensate workable data (not necessarily field linked) for further use;
3) Data conversion: interpolate, project, repair, decimate, format conversion to make discuss simulation codes with each other;
4) Data optimization for simulation code: renumbering, partition for multiprocessor codes.

The emphasis of MEDCoupling is mesh and field which are both in MED format. They are the elemental unit to be processed in SALOME. MEDCoupling supports twenty-one kinds of meshes. Here below we only list the available key words of the mesh types. Readers can refer to the on-line documentation for details, please go https://docs.salome-platform.org/latest/dev/MEDCoupling/ developer/index.html.

1) 1D lines: NORM_POINT1, NORM_SEG2, NORM_SEG3, NORM_SEG4, NORM_POLYL;
2) 2D surfaces: NORM_TRI3, NORM_QUAD4, NORM_POLYGON, NORM_TRI6, NORM_TRI7, NORM_QUAD8, NORM_QUAD9, NORM_QPOLYG;
3) 3D volumes: NORM_TETRA4, NORM_PYRA5, NORM_PENTA6, NORM_HEXA8, NORM_TETRA10, NORM_HEXGP12, NORM_PYRA13, NORM_PENTA15, NORM_HEXA20, NORM_HEXA27, NORM_POLYHED.

Because of the particularity of the VESSEL component, all the cells are 3D volumes. Hereby they should be explained in detail. There are 11 kinds of 3D mesh supported by MEDCoupling, in six general types. The meshes and the corresponding keywords are exhibited in Figure 2-5. The MED format meshes are not only geometrical objects but have inherent built-in attributes. The following list posts some of those attributions. For the complete attribution list and the explanations, please go to the online documents.

1) Splitting policy
   - Type describing the different ways in which the hexahedron can be split into tetrahedrons.
   - Options: PLANAR_FACE_5, PLANAR_FACE_6, GENERAL_24, GENERAL_48.
2) Intersection type
   - How the mesh is intersected.
   - Options: Triangulation, Convex, Geometric2D, PointLocator, Barycentric, BarycentricGeo2D.
3) Numbering Policy
   - Define the data arrangement in the memory.
   - Options: ALL_C_MODE, ALL_FORTRAN_MODE.

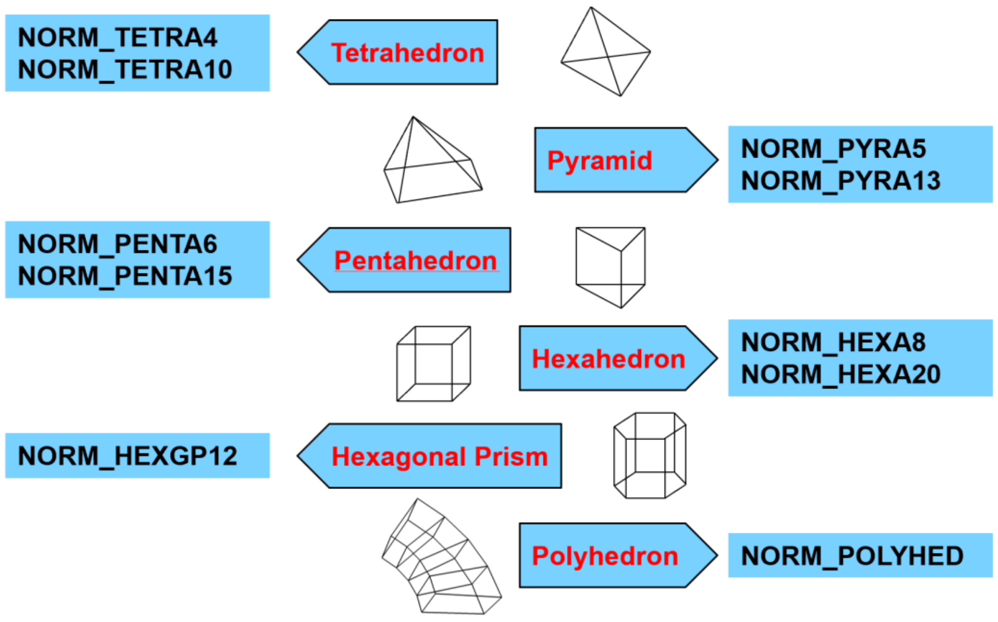


Figure 2-5 The 3D Volumetric Cell Types Supported by MEDCoupling

MEDCoupling proceeds the following steps to build a mesh.

1) Define the numbering roles of the points composing the mesh. This is the base of everything. You must define the order of those pints.
2) Set or calculate the coordination of those points. The coordination is stored in an array according to the points order defined in step one.
3) Define the connectivity of those points. The connectivity is an array containing a set of those points numbers. It declares which point should be connected to another.
4) Declare a special MED array to receive the points coordination. Allocate appropriate space for the array to receive the points coordination.
5) Set the components for the MED array storing points coordination. No actual operation here, just a declaration of an object.
6) Declare the MED mesh. Define the mesh dimension. Tell how many cells are included in the mesh. Set the coordination MED array to the mesh. Use the connectivity array to insert the cell to the mesh.

Several cases for the practice of building simple meshes can be found in Appendix A. All the six mesh types except the polyhedron mesh shown in Figure 2-5 are involved together with the source code. Furthermore, two more practices demonstrating the construction of a PWR type core and a VVER type are given in Appendix B and Appendix C.

Now we turn to the other key element – field. A field in the MEDCoupling point of view is a structure that allows storing a discretization of physical value on a defined discretized spatial and possibly temporal support. It is an array with couples of attributions e.g., name, time, mesh. Figure 2-6 gives an example of the fields. It can be inferred that the base of a field is an array. The field must have a name and must be assigned to a mesh. Some other attributes e.g., time are also possible for a field. Please note that different fields may share the same mesh.

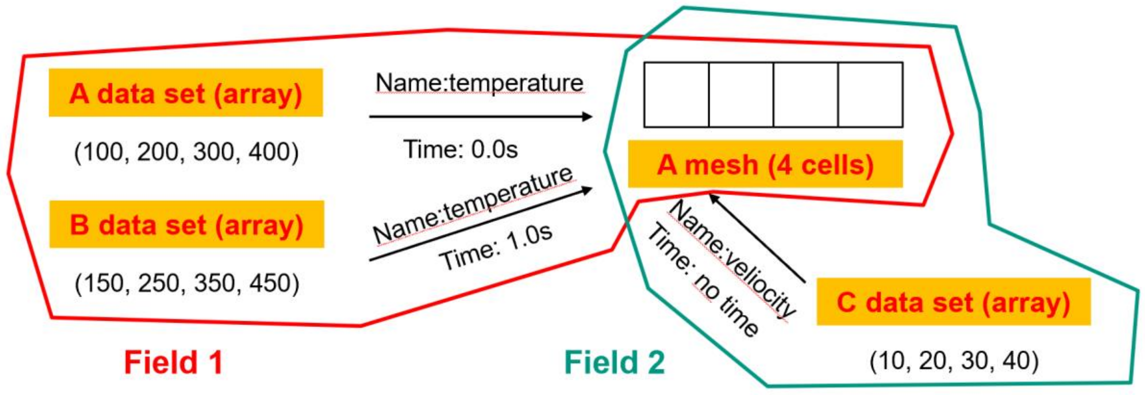


Figure 2-6 An Example of Fields with Attributions

Some further features of a MED field are listed below and the details are well explained in the online documentation.

1) Type of fields: ON_CELLS, ON_NODES, ON_GAUSS_PT, ON_GAUSS_NE, ON_NODES_KR;
2) Time discretization: NO_TIME, ONE_TIME, LINEAR_TIME, CONST_ON_TIME_INTERVAL;
3) Field nature: ConservativeVolumic, Integral, IntegralGlobConstraint, RevIntegral.

Similar to the MED mesh, couples of functions are available to manipulate the MED field. Several common functions are described in the following list and the complete set could be found in the online documentation.

1) setNature: Four field natures available: ConservativeVolumic, Integral, IntegralGlobConstraint, RevIntegral;
2) setArray: Fill a MED array to the field. Solve the problem of what;
3) setName: State the name of the field e.g., temperature, pressure. Solve the problem of who;
4) setMesh: Assign a mesh to the field. Solve the problem of where;
5) setTime: Assign a problem time to the field. Solve the problem of when;
6) getIJ: Get a value from a field on a specified location;
7) getNumberOfComponents, getNumberOfTuples, getNumberOfValues: Get a static information form a field;
8) getMesh: Get the mesh of the field;
9) getArray: Get the array of the field;
10) checkCoherency: Checks if this field is correctly defined, else an exception is thrown.

The typical procedures to build a MED field are:

1) Declare the MED field.
   - ParaMEDMEM::MEDCouplingFieldDouble *field_tetra = ParaMEDMEM:: MEDCouplingFieldDouble::New(ON_CELLS, ONE_TIME);
2) Set the field’s nature.
   - field_tetra->setNature(ConservativeVolumic);
3) Define the field’s name.
   - field_tetra->setName("V_nanhua1");
4) Assign a mesh to the field.
   - field_tetra->setMesh(nanhua1_tetra);
5) Declare a MED array, allocate appropriate space to store the actual field data.
   - DataArrayDouble *ar_tetra = DataArrayDouble::New();
   - ar_tetra->alloc(1, 3);
6) Set the MED array component.
   - ar_tetra->setInfoOnComponent(0, "X");
7) Project the MED array to simple C++ array for convenient data inserting.
   - double *a_final_tetra = ar_tetra->getPointer();
8) Fill the C++ array with data.
9) Assign the MED array to the field.
   - field_tetra->setArray(ar_tetra);

Appendix D Gives two examples for MED field construction depend on the PWR and VVER mesh developed in Appendix B and C.

3 DEVELOPMENT OF THE TRACE POST-PROCESSING FUNCTIONALITY

This chapter will extend the explanation of MED meshes and fields from general cell types to the specific TRACE 3D VESSEL component. Besides, the code structure of the updated TRACE program is discussed as well, by which the code developers might find useful information.

3.1 Construct Mesh and Field in MED Format for the 3D VESSEL Component

A mesh is built following a sequence from point to surface to volume. The procedures are shown in Figure 3-1.

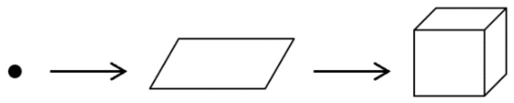


Figure 3-1 A Cell is Built from Point to Surface to Volume.

As it was told in section 2.2, the MEDCoupling library supports five cell types: tetrahedron, triangular prism, hexahedron, hexagonal prism, and polyhedron. The first four cells are displayed in Figure 3-2a. But they can't be applied to represent a TRACE cell. This is because the cells in TRACE are either fan-shaped or annular (Figure 3-2b). So, the only possible option is the polyhedron.

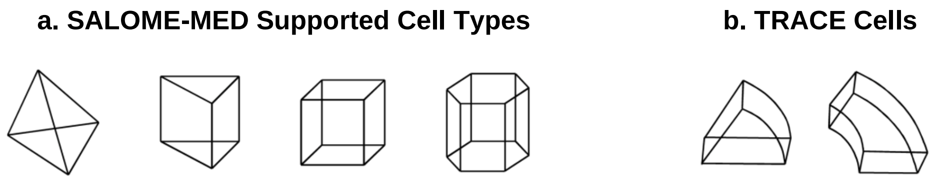


Figure 3-2 Cells Supported by MEDCoupling Library and the Typical TRACE Cells.

Normally, only eight points are used to define a TRACE annular cell because TRACE uses the cylindrical-coordinate system to model the VESSEL component. However, since the MEDCoupling library applies only Cartesian coordinates, the derived eight points from the TRACE input file for one single cell are not sufficient to form cells that contain space curves under Cartesian coordinates. The direct result is a cubic that could not represent the real TRACE annular cell, see the left plot of Figure 3-3. To complete the cell, some assistant points must be inserted (the red points in the right plot of Figure 3-3). Those points could be treated as an interpolation between two original TRACE points. The interpolation is automatically done based on the user-defined resolution (the curves amount to approximate a circle). However, there arises another challenge known as the creation of the spatial faces with those points.

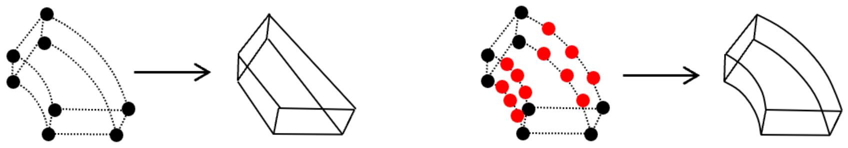


Figure 3-3 Construction of a Typical TRACE Cell

MEDCoupling has special logic to determine the coplanarity of a set of points. If those points are to form a spatial 3D surface, the non-coplanar points would lead to closing difficulties failing the surface definition. While this kind of spatial surfaces is common in the VESSEL component of TRACE see Figure 3-4.

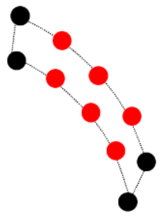


Figure 3-4 The Typical Spatial Surfaces to be Defined for the VESSEL Component of TRACE

To precisely define the spatial surfaces, we use a couple of 2D quadrangles to approximate them. The possible approaches are presented in Figure 3-5, following which are the explanations.

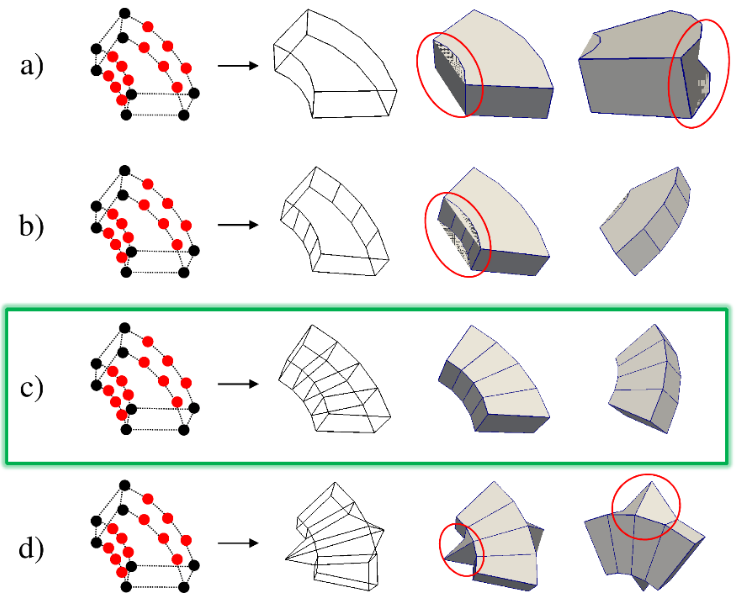


Figure 3-5 Simulation of a Typical TRACE Cell Using Different Approximation Methods a) Direct Point Use, b) Quadrangles Approximation for Spatial Faces, c) Quadrangles Approximation for Spatial and Multi-Points (Upper and Bottom) Faces, d) Use of Non-Coplanar Points.

a) Directly use the points to form the spatial faces. Only six faces are involved in the annular volume. But this approach causes misshapen cells highlighted by the red circles.
b) Use several sets of quadrangles to approximate the two spatial faces. Now the volume is properly defined. However, there are some display problems for the two planes containing more than four points highlighted by the red circles.
c) Use several sets of quadrangles to approximate both the spatial faces and the multi-points planes. Now the annular volume is perfectly defined and displayed and considered to be adequate for further use.
d) This is exactly not an option for the cell definition but disclaims that the co-planarity of the points is the crucial reason affecting the cells building.

Up to this point, a TRACE mesh could be explicitly built by the third approach. No additional definitions except the TRACE input file are required. Now, it is ready to construct the MED fields from TRACE memory and write them into the meshes. In total, twenty-one kinds of data can be derived to form MED fields. They are summarized in Table 3-1.

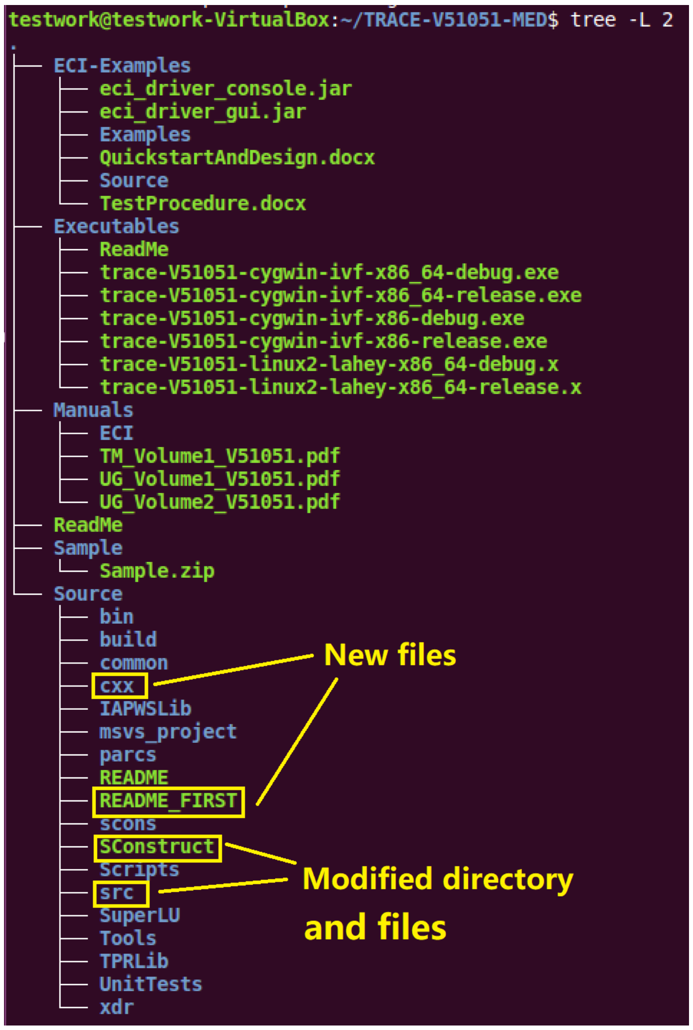
Table 3-1 The Physical Data Which Can be Derived as MED Fields

ALPN	void fraction
PN	pressure
TLN	liquid temperature
TVN	gas temperature
VLNXR	liquid velocity on x or r direction
VLNYT	liquid velocity on y or t direction
VLNZ	liquid velocity on z-direction
VVNXR	gas velocity on x or r direction
VVNYT	gas velocity on y or t direction
VVNZ	gas velocity on z-direction
MMFLXR	mass flow rate velocity on x or r direction
MMFLYT	mass flow rate velocity on y or t direction
MMFLZ	mass flow rate velocity on z-direction
AM	non-consen gas mass
PAN	non-consen gas pressure
ROLN	liquid density
ROVN	gas density
ROAN	non-consen gas density
ROM	mixture density
CONCWN	solute mass ratio
CONCSN	solute mass fraction

The volumetric polyhedron mesh discussed in the earlier part of this section can properly store the cell-centered fields e.g., the liquid temperature and liquid density. However, the velocities or mass flow rates of the coolant are defined at the interfaces between the volumetric cells. The polyhedron mesh could not handle those data. A new 3D surface mesh was developed based on the polyhedron mesh but only contains the interfaces between the cells. The velocities and mass flow rates are processed with this mesh in SALOME. An AP-1000 testing case will demonstrate the meshes and fields vividly in the next Chapter.

3.2 <u>The Low-Level Logic of the Enhanced TRACE Package</u>

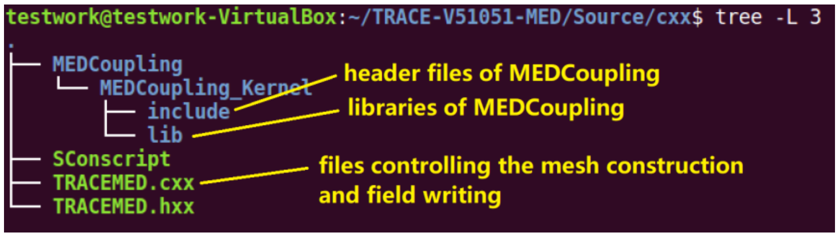
This section gives some clues to follow the updates of the new TRACE package for code developers.



```
testwork@testwork-VirtualBox:~/TRACE-V51051-MED$ tree -L 2
.
├── ECI-Examples
│   ├── eci_driver_console.jar
│   ├── eci_driver_gui.jar
│   ├── Examples
│   ├── QuickstartAndDesign.docx
│   ├── Source
│   └── TestProcedure.docx
├── Executables
│   ├── ReadMe
│   ├── trace-V51051-cygwin-ivf-x86_64-debug.exe
│   ├── trace-V51051-cygwin-ivf-x86_64-release.exe
│   ├── trace-V51051-cygwin-ivf-x86-debug.exe
│   ├── trace-V51051-cygwin-ivf-x86-release.exe
│   ├── trace-V51051-linux2-lahey-x86_64-debug.x
│   └── trace-V51051-linux2-lahey-x86_64-release.x
├── Manuals
│   ├── ECI
│   ├── TM_Volume1_V51051.pdf
│   ├── UG_Volume1_V51051.pdf
│   └── UG_Volume2_V51051.pdf
├── ReadMe
├── Sample
│   └── Sample.zip
└── Source
    ├── bin
    ├── build
    ├── common
    ├── cxx
    ├── IAPWSLib
    ├── msvs_project
    ├── parcs
    ├── README
    ├── README_FIRST
    ├── scons
    ├── SConstruct
    ├── Scripts
    ├── src
    ├── SuperLU
    ├── Tools
    ├── TPRLib
    ├── UnitTests
    └── xdr
```

Figure 3-6 The File Structure of the Updated TRACE Package

Figure 3-6 presents the file structure of the updated TRACE package with enhanced post-processing functionality. It is inferred that the trunk and main branches of TRACE are not significantly changed. Only some stuff in the “Source” folder is modified or added. The “README_FIRST” is nothing but a quick instruction on how to build and run the executable. The root compiling-control file for Scons – “SConstruct” is modified to properly handle the new files and directories. The new directory “cxx” is inspected in Figure 3-7.



```
testwork@testwork-VirtualBox:~/TRACE-V51051-MED/Source/cxx$ tree -L 3
.
├── MEDCoupling
│   └── MEDCoupling_Kernel
│       ├── include
│       └── lib
├── SConscript
├── TRACEMED.cxx
└── TRACEMED.hxx
```

header files of MEDCoupling
libraries of MEDCoupling
files controlling the mesh construction and field writing

Figure 3-7 The File Structure of CXX

MEDCoupling is developed mainly in cxx. Hybrid programming mixed with Fortran and cxx is necessary. To fully take advantage of the functionalities of MEDCoupling, new cxx files must be developed as the wrapper. Besides, the cxx file must contain the logic to construct the meshes and to write the fields. To modulize the source, all of the cxx-related kinds of stuff are put in one folder.

As to the “src” folder, there are five new files added to the Fortran source. They are:

1) med_dataM.f90;
2) med_funcM.f90;
3) med_TprVesselM.f90;
4) med_trans.f90;
5) med_VessInputM.f90.

The first two files are new scripts as the interface on the Fortran side connecting the cxx routines. The other three files are derived from the original TRACE files. Note that the file names are the same with TRACE files except the uniform prefix. Only limited changes are made to those files. The “Sconscript” in this folder is also modified to reflect the changes.

Now, the updated TRACE package can be compiled in two ways.

1) python python scons/scons.py compiler=intel;
2) python python scons/scons.py compiler=intel icoco_med=med.

The first method will build the TRACE executable that is 100 percent the same as the originally published TRACE. The second method will build the executable with the enhanced post-processing functionalities. Switching the two modes is controlled only by adding a new compiling keyword. This is from the idea of keeping both TRACE source structure and building manners as much the same as the originally published TRACE package.

Still, one more operation has to be performed before launching the new TRACE executable. That is, the liking root of the MEDCoupling library has to be explicitly declared because TRACE will follow this definition to find the libraries to construct the meshes and fields. Users can either add the path to “LD_LIBRARY_PATH” in the current launching shell or permanently add the definition to “bashrc” under the home.

4 DEMONSTRATION OF THE POST-PROCESSING FUNCTIONALITY OF TRACE

The testing of the new post-processing capability is done by an AP-1000 transient case. Selected fields and data processing methods are exhibited.

4.1 The AP-1000 Transient Case

The AP1000 plant is a two loop pressurized water reactor (PWR), as shown in Figure 4-1 of Generation III+ that uses a simplified innovative, and effective approach to safety.

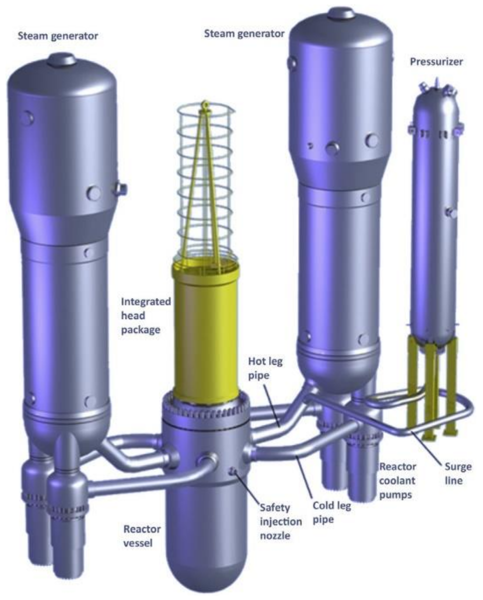


Figure 4-1 Configuration of the Main Components of the AP1000 Reactor

The main characteristics of this reactor are summarized in Table 4-1.

Table 4-1 Main Plant Thermal-Hydraulic Data of AP1000

Parameter	Value
Core power $[MWth]$	3400
Reactor coolant system mass flow rate $[kg/s]$	15187.4
System pressure $[Bar]$	155
Core inlet temperature $[K]$	553.817
Core outlet temperature $[K]$	596.483
Core mass flow rate $[kg/s]$	14275.56

The model of AP1000 in SNAP is shown in Figure 4-2. The vessel inlets and outlets are modeled by the FILL and BREAK components. Note there are two loops including four cold legs and two hot legs. Each hot leg corresponds to two cold legs. A schematic diagram is given in Figure 4-3 to clearly illustrate the vessel configuration.

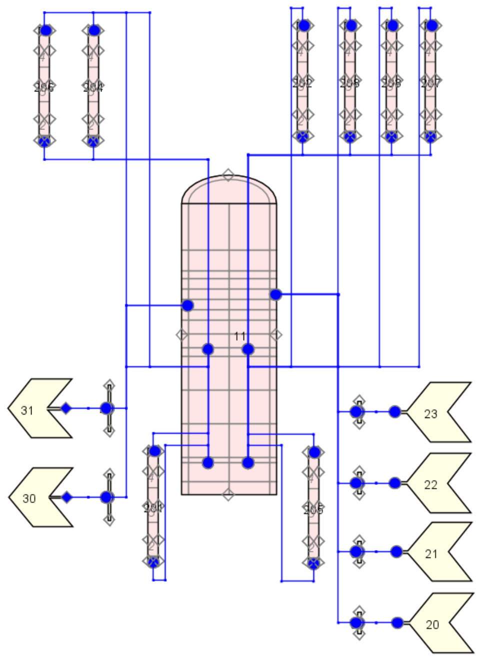


Figure 4-2 The AP1000 Model in SNAP

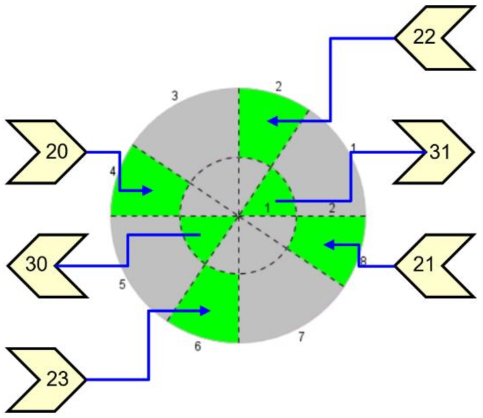


Figure 4-3 The Configuration of the Hot Legs and Cold Legs of the Vessel

The transient for the AP1000 reactor, which may occur as a consequence of the rupture of one steam line in a heat exchanger that leads to an asymmetric coolant temperature drop in the inlet of the core. Here, in this case, the power field is not involved for the sake of modeling simplicity.

The transient evolution is:

1) t = 0 s, temp (Fill 20, 21, 22, 23) = 553.7 K
2) t = 5 s, temp (Fill 20, 21, 22, 23) = 553.7 K
3) t = 15 s, temp (Fill 20, 23) = 503.7 K

4) t = 20 s, temp (Fill 20, 23) = 503.7 K

Figure 4-4 represents the temperature in the four lines during the transient time.

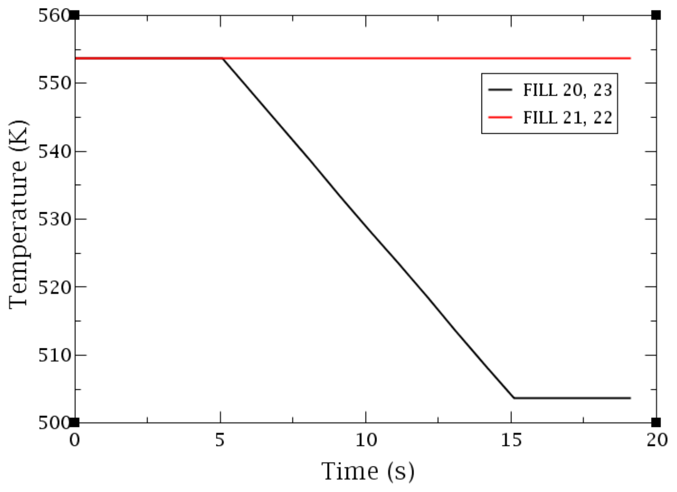
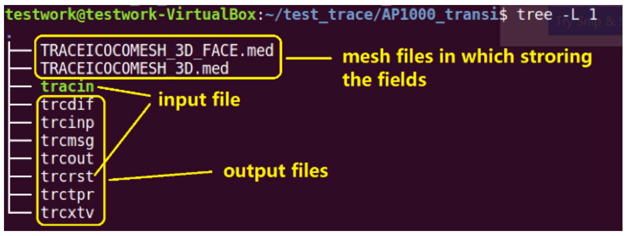


Figure 4-4 Transient Evolution

4.2 Demonstration of the Meshes and Fields in SALOME

Once we got the new TRACE executable, it can be launched as normal TRACE. For fresh running, it demands an input file – tracin. For restart running, it needs the modified input file – tracin and the restart file – trcrst. When the calculation finishes normally, several files are generated, see Figure 4-5.



```
testwork@testwork-VirtualBox:~/test_trace/AP1000_transi$ tree -L 1
.
├── TRACEICOCOMESH_3D_FACE.med
├── TRACEICOCOMESH_3D.med
├── tracin
├── trcdif
├── trcinp
├── trcmsg
├── trcout
├── trcrst
├── trctpr
└── trcxtv
```

Figure 4-5 The Input and Output Files of the New TRACE

Note that this is a restart case. So the input files are tracin and trcrst. The other regular output files are the same as normal TRACE. The highlighted files are the two MED files. “TRACEICOCOMESH_3D.med” is the polyhedron mesh storing the cell-centered field e.g., the coolant temperature. “TRACEICOCOMESH_3D_FACE.med” is the surface mesh storing the edge field e.g., the coolant velocity. Now we can launch SALOME and import the two .med files. The window would have such an appearance as displayed in Figure 4-6.

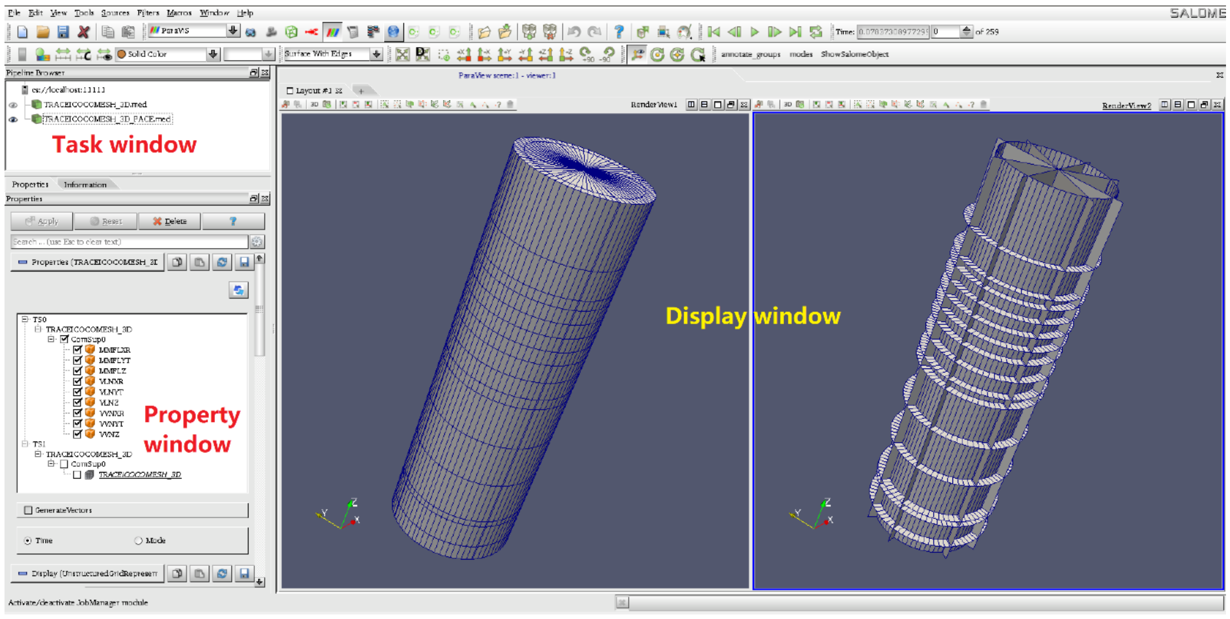


Figure 4-6 The Appearance of SALOME after Importing the Two MED Files

The display window presents the two meshes. The left one is the polyhedron mesh and the right one is the surface mesh. The zoomed-in task window listing the two med field windows are exhibited in Figure 4-7.

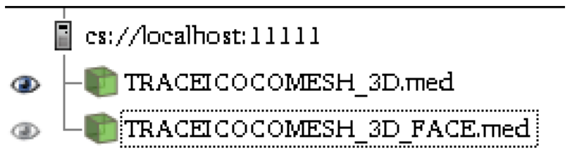
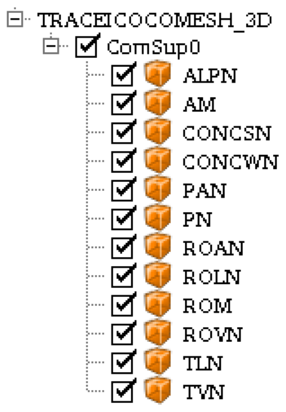


Figure 4-7 The Zoomed-in Task Window Displaying the Two MED Files

When one med file was selected in the task window, the property window will list the available fields storing in that file and mesh. Figure 4-8 and Figure 4-9 exhibit the zoomed-in property windows when the deferent med file is selected. The total field number is twenty-one as told in section 3.1.


TRACEICOCOMESH_3D
ComSup0
ALPN
AM
CONCSN
CONCWN
PAN
PN
ROAN
ROLN
ROM
ROVN
TLN
TVN

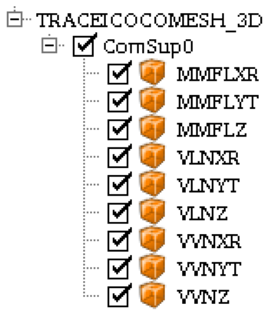

TRACEICOCOMESH_3D
ComSup0
MMFLXR
MMFLYT
MMFLZ
VLNXR
VLNYT
VLNZ
VVNXR
VVNYT
VVNZ

Figure 4-8 The Zoomed-in Property Window Displaying the Fields Storing in the Polyhedron Mesh

Figure 4-9 The Zoomed-in Property Window Displaying the Fields Storing in the Surface Mesh

Because the testing case is a temperature distributing transient, the coolant temperature field is selected to demonstrate the post-processing of the fields locating in the polyhedron mesh, see Figure 4-10. From left to right, the first plot displays the overall coolant temperature distribution in the vessel, the second plot gives the temperature profile inside the vessel by clipping the vessel one half out, the third plot selects the cells having a temperature higher than 540K and present them, the fourth plot selects the cells whose temperature is lower than 540K and present them.

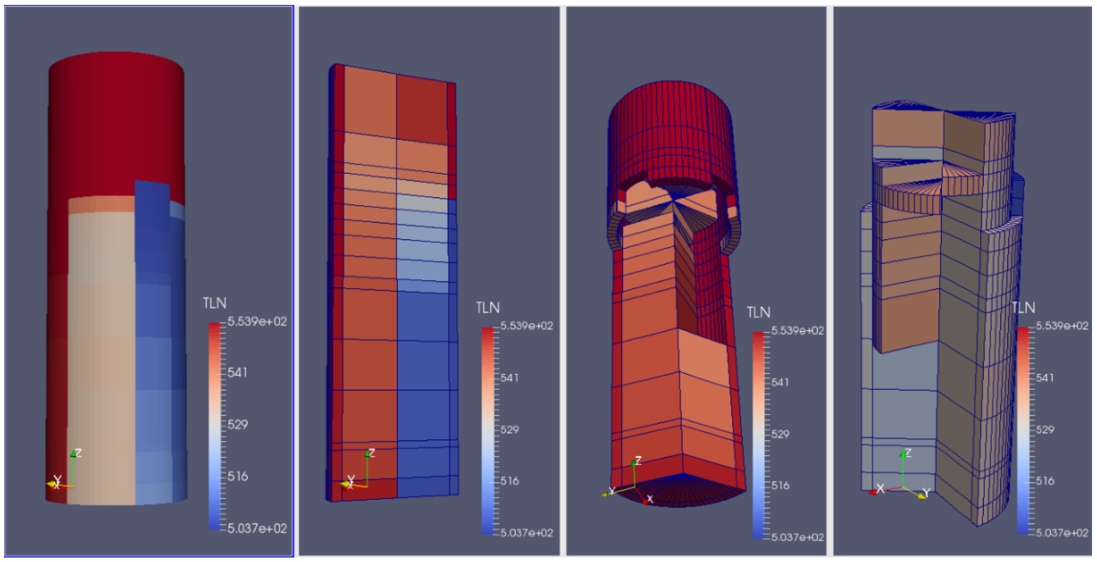


Figure 4-10 The Coolant Temperature Field in the Polyhedron Mesh Processed in SALOME

We can see an obvious cold part and hot part of the vessel. The coolant temperature evolution along the flow path is also clear. From the vessel inlet to the downcomer to the core inlet to the core outlet to the vessel outlet, the coolant in the colder part is getting hotter and hotter. While the coolant in the hotter part is getting colder and colder. The coolant mixing plays a key role in this phenomenon.

As to the fields in the surface mesh, since the mesh has only one radial surface on XR direction which is the wall isolating the downcomer and core area, the XR surfaces all have zero velocities. Thus only the velocities or mass flow rate on Z (axial) and YT (azimuthal) directions are demonstrated, see Figure 4-11.

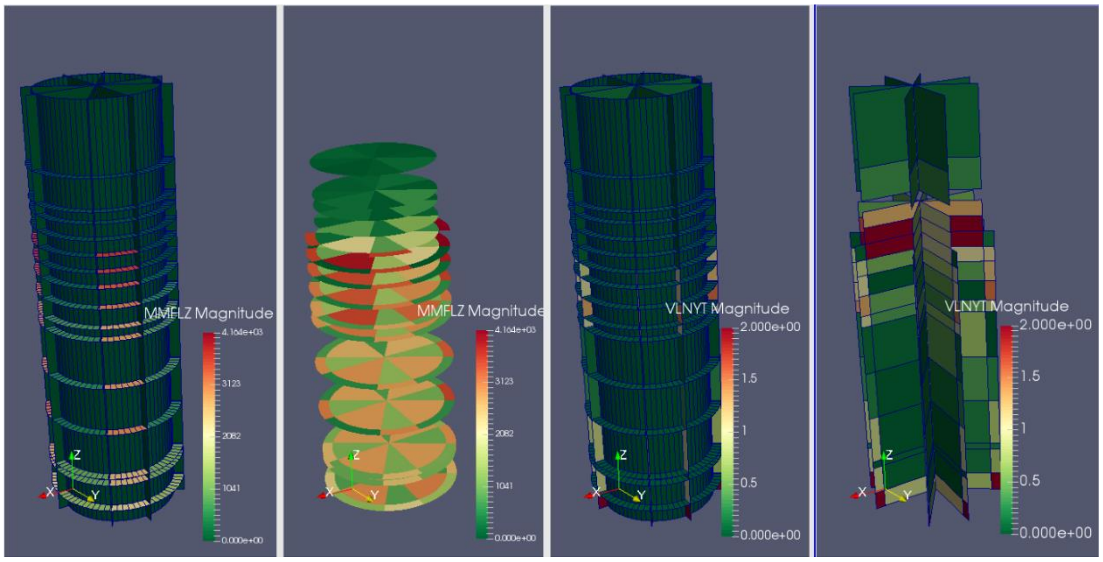


Figure 4-11 The Mass Flow Rate on Z Direction and the Coolant Velocity on YT Direction in the Surface Mesh Processed in SALOME

From left to right, the first plot is the Z-directional coolant mass flow rate on the full surface mesh. Note that only the axial surfaces have positive values while the values on all the radial and azimuthal surfaces are zero. The second plot shows only the non-zero axial surfaces so that both the inner faces and outer faces are visible. The third plot is the azimuthal or YT-directional coolant velocity on the full surface mesh. Also, note that only the azimuthal surfaces have positive values while the values on all the axial and radial surfaces are zero. The fourth plot shows only the non-zero azimuthal surfaces so that both the inner faces and outer faces are visible.

The axial mass flow rates differ significantly in magnitude when on the surface near the vessel inlet. Along the flow path in the downcomer, the difference is getting flattered because the coolant flows from the cells of the high mass flow rate to the cells of the low mass flow rate. Oppositely, along the flow path from the core inlet to the vessel outlet, the flat mass flow difference is getting significant again because only two cells are corresponding to the two vessel outlet. All of the coolants flow from all other cells to the two cells and finally flow out of the vessel.

The axial mass flow difference evolution along the significant vessel inlet to the flat lower plenum to the significant vessel outlet could be inferred from the azimuthal coolant velocity distributions plotted in the rightmost figure. Along the flow path in the downcomer, the azimuthal velocity is getting larger and larger indicating the inter-flow between azimuthal cells is getting stronger. Thus the axial flow rate difference is getting smaller. Similarly, from the lower plenum to the vessel outlet, the azimuthal velocity is getting larger indicating the inter-flow between azimuthal cells is becoming larger. Thus the axial flow rate difference is getting larger again.

The above figures demonstrate the outstanding capability of the MED files for qualitative analysis of the fields in the VESSEL component of TRACE. Besides, the plentiful of quantitative analysis can also be performed in SALOME. Take the coolant temperature and velocity for instance, in Figure 4-12, from left to right, the first figure plots the cell in the highest and lowest temperature together with their locations in the full vessel mesh. The second figure plots the single cell in the highest cell and plots the evolution over the transient time. The third figure plots the surface having the largest azimuthal coolant velocity together with its location in the full mesh. The four mesh plots this single cell and plots the evolution over the transient time.

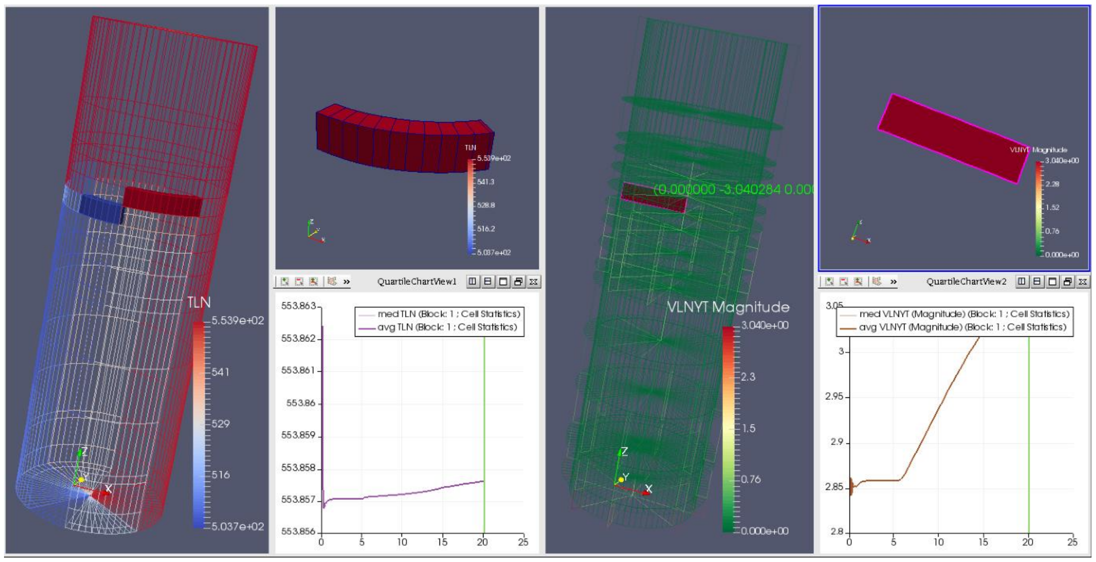

TLN
5.539e+02
541
529
516
5.037e+02
QuartileChartView1
med TLN (Block: 1 : Cell Statistics)
avg TLN (Block: 1 : Cell Statistics)
VLNYT Magnitude
3.040e+00
2.3
1.5
0.76
0.000e+00
QuartileChartView2
med VLNYT (Magnitude) (Block: 1 : Cell Statistics)
avg VLNYT (Magnitude) (Block: 1 : Cell Statistics)

Figure 4-12 The Data Analysis of Specific Part or Cells

It is to note that the data processing methods presented in this report are very basic and only for demonstration purposes. More powerful methods can be found in the online documentation.

5 CONCLUSION

Based on the open-source data-processing MEDCoupling library, a polyhedron mesh and a surface mesh for the 3D VESSEL component of TRACE are constructed. Twenty-one kinds of calculated fields including the cell-centered field e.g., the coolant temperature and the edge field e.g., the coolant velocity could be written from TRACE memory to the two meshes respectively. The meshes and fields now could be imported into the open-source SALOME platform for post-processing. SALOME could perform both qualitative analysis and quantitative analysis. With the new enhanced post-processing functionality to TRACE, the physical field in the VESSEL component could be inspected in a more flexible and apparent manner.

6 OUTLOOK

MEDCoupling is more than a post-processing library. It can also handle the field translation problems between different computational domains. This is an outstanding capability for code coupling problems, which is one of the main development objectives of the library. To fully take advantage of this capability, TRACE might extend its MED fields to adapt coupling issues. That is, we can develop a multi-scale or multi-physics coupling system involving TRACE with the enhanced post-processing functionalities from MEDCoupling.

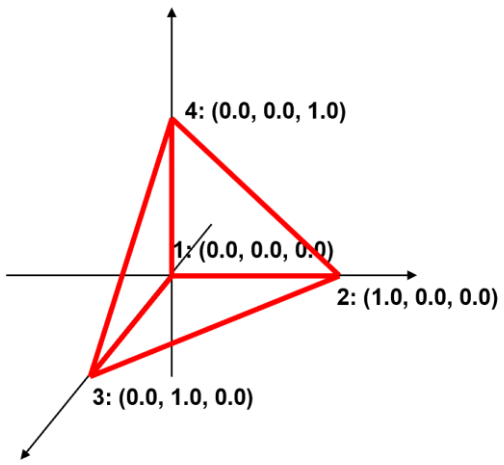
7 REFERENCES

[1] TRACE "TRACE V5.1051 theory manual," U.S. Nuclear Regulatory Commission, Washington, DC, U.S..

APPENDIX A PRACTICE OF SINGLE MESH CONSTRUCTION

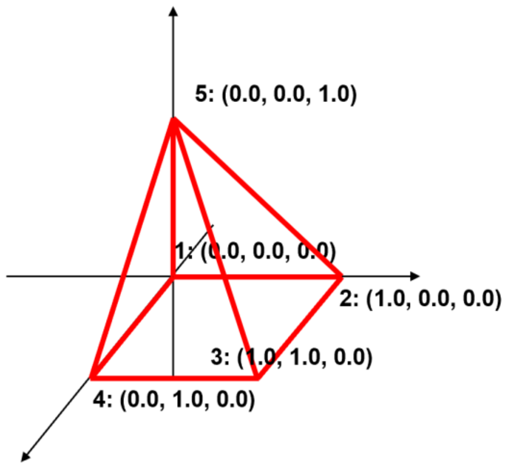
Six simple cases are demonstrating the mesh generation process and the source code is also available.

(1)
tetrahedron



1) Define the order of the points;
2) double coor[12]= {0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0};
3) double conn[4]= {0, 1, 2, 3};
4) DataArrayDouble *c_3d = DataArrayDouble::New(); c_3d->alloc(4, 3); std::copy(coor, coor + 4*3, c_3d->getPointer());
5) c_3d->setInfoOnComponent(0, "X"); c_3d->setInfoOnComponent(1, "Y"); c_3d->setInfoOnComponent(2, "Z");
6) ParaMEDMEM::MEDCouplingUMesh *m_3d =ParaMEDMEM::MEDCouplingUMesh::New("mesh_tetra", 3); m_3d->setCoords(c_3d);m_3d->allocateCells(1); m_3d->insertNextCell(INTERP_KERNEL::NORM_TETRA4, 4, conn).

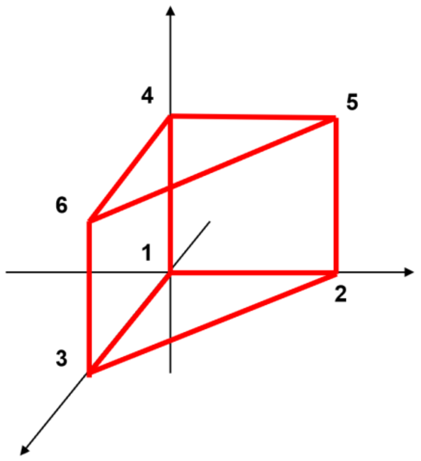
(2)
pyramid



1) Define the order of the points;

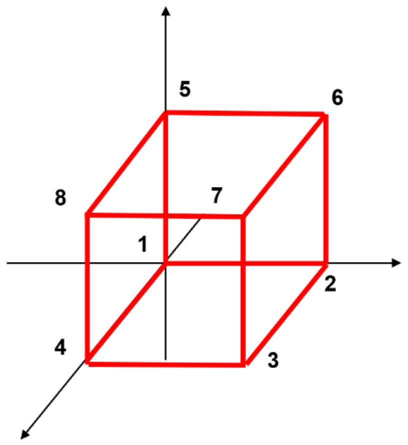
2) double coor[15]= {0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0};
3) double conn[5]= {0, 1, 2, 3, 4};
4) DataArrayDouble *c_3d = DataArrayDouble::New(); c_3d->alloc(5, 3); std::copy(coor, coor + 5*3, c_3d->getPointer());
5) c_3d->setInfoOnComponent(0, "X"); c_3d->setInfoOnComponent(1, "Y"); c_3d->setInfoOnComponent(2, "Z");
6) ParaMEDMEM::MEDCouplingUMesh *m_3d =ParaMEDMEM::MEDCouplingUMesh::New("mesh_pyra", 3); m_3d->setCoords(c_3d);m_3d->allocateCells(1); m_3d->insertNextCell(INTERP_KERNEL:: NORM_PYRA5, 5, conn).

(3)
pentahedron



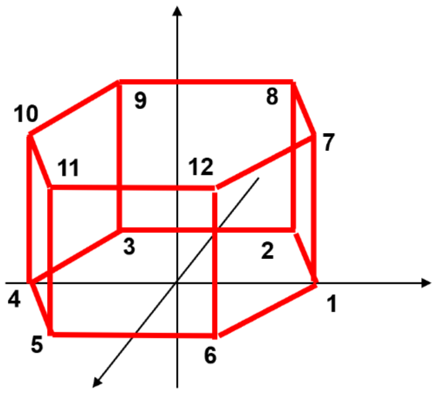
1) Define the order of the points;
2) double coor[18]= {0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0};
3) double conn[6]= {0, 1, 2, 3, 4, 5};
4) DataArrayDouble *c_3d = DataArrayDouble::New(); c_3d->alloc(6, 3); std::copy(coor, coor + 6*3, c_3d->getPointer());
5) c_3d->setInfoOnComponent(0, "X"); c_3d->setInfoOnComponent(1, "Y"); c_3d->setInfoOnComponent(2, "Z");
6) ParaMEDMEM::MEDCouplingUMesh *m_3d =ParaMEDMEM::MEDCouplingUMesh::New("mesh_pent", 3); m_3d->setCoords(c_3d);m_3d->allocateCells(1); m_3d->insertNextCell(INTERP_KERNEL:: NORM_PENTA6, 6, conn).

(4)
hexahedron



1) Define the order of the points;
2) double coor[24]= {0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0};
3) double conn[8]= {0, 1, 2, 3, 4, 5, 6, 7};
4) DataArrayDouble *c_3d = DataArrayDouble::New(); c_3d->alloc(8, 3); std::copy(coor, coor + 8*3, c_3d->getPointer());
5) c_3d->setInfoOnComponent(0, "X"); c_3d->setInfoOnComponent(1, "Y"); c_3d->setInfoOnComponent(2, "Z");
6) ParaMEDMEM::MEDCouplingUMesh *m_3d =ParaMEDMEM::MEDCouplingUMesh::New("mesh_hexa", 3); m_3d->setCoords(c_3d);m_3d->allocateCells(1); m_3d->insertNextCell(INTERP_KERNEL:: NORM_HEXA8, 8, conn).

(5)
hexagonal prism



1) Define the order of the points;
2) double coor[36]= {1.0, 0.0, 0.0, 0.5, 0.866, 0.0, -0.5, 0.866, 0.0, -1.0, 0.0, 0.0, -0.5, -0.866, 0.0, 0.5, -0.866, 0.0, 1.0, 0.0, 1.0, 0.5, 0.866, 1.0, -0.5, 0.866, 1.0, -1.0, 0.0, 1.0, -0.5, -0.866, 1.0, 0.5, -0.866, 1.0};
3) double conn[12]= {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
4) DataArrayDouble *c_3d = DataArrayDouble::New(); c_3d->alloc(12, 3); std::copy(coor, coor + 12*3, c_3d->getPointer());
5) c_3d->setInfoOnComponent(0, "X"); c_3d->setInfoOnComponent(1, "Y"); c_3d-

>setInfoOnComponent(2, "Z");

6) ParaMEDMEM::MEDCouplingUMesh *m_3d =ParaMEDMEM::MEDCouplingUMesh::New("mesh_hexp", 3); m_3d->setCoords(c_3d);m_3d->allocateCells(1); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXGP12, 12, conn).

APPENDIX B PRACTICE OF PWR TYPE MESH CONSTRUCTION

This case extends the mesh set from a single cell to multi-cell. It is only for demonstration thus contains only nine cells instead of representing the full PWR core. The final desired mesh is shown in Figure B-1. The meshes are hexagonal.

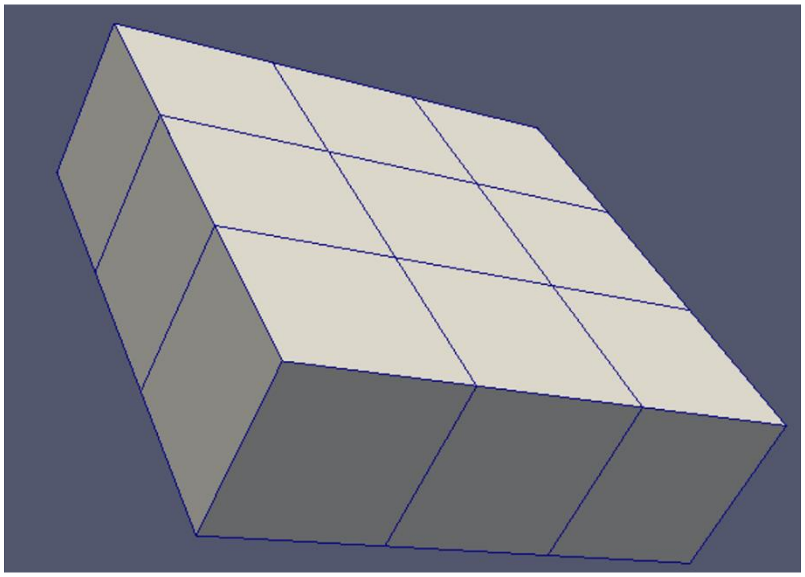


Figure B-1 The Desired Core Mesh Type for PWR

1) The first step is setting the order of the points, Figure B-2.

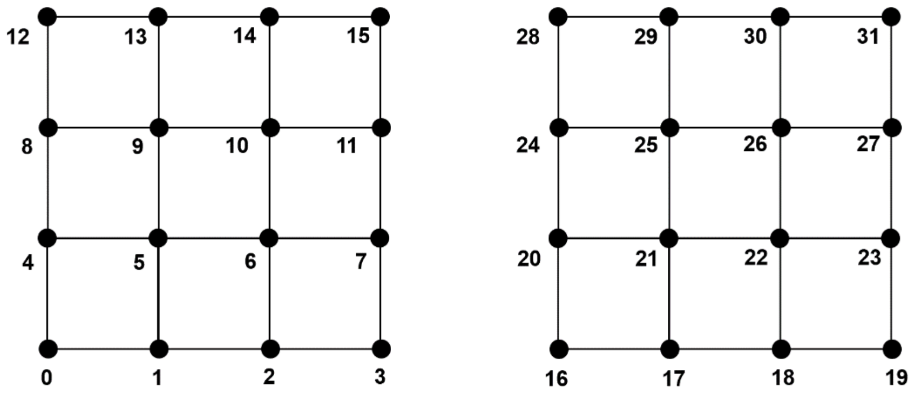


Figure B-2 The Order of the Points of PWR Core Mesh

2) Then, define the coordination of the points. double coor[96]= {0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 2.0, 0.0, 0.0, 3.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 2.0, 1.0, 0.0, 3.0, 1.0, 0.0, 0.0, 2.0, 0.0, 1.0, 2.0, 0.0, 2.0, 2.0, 0.0, 3.0, 2.0, 0.0, 0.0, 3.0, 0.0, 1.0, 3.0, 0.0, 2.0, 3.0, 0.0, 3.0,

3.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 2.0, 0.0, 1.0, 3.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 3.0, 1.0, 1.0, 0.0, 2.0, 1.0, 1.0, 2.0, 1.0, 2.0, 2.0, 1.0, 3.0, 2.0, 1.0, 0.0, 3.0, 1.0, 1.0, 3.0, 1.0, 2.0, 3.0, 1.0, 3.0, 3.0, 1.0}.

3) Build the connectivity array. int conn[72]= {0,1,5,4,16,17,21,20, 1,2,6,5,17,18,22,21, 2,3,7,6,18,19,23,22, 4,5,9,8,20,21,25,24, 5,6,10,9,21,22,26,25, 6,7,11,10,22,23,27,26, 8,9,13,12,24,25,29,28, 9,10,14,13,25,26,30,29, 10,11,15,14,26,27,31,30}.
4) DataArrayDouble *c_3d = DataArrayDouble::New(); c_3d->alloc(32, 3); std::copy(coor, coor + 32*3, c_3d->getPointer()).
5) c_3d->setInfoOnComponent(0, "X"); c_3d->setInfoOnComponent(1, "Y"); c_3d->setInfoOnComponent(2, "Z").
6) ParaMEDMEM::MEDCouplingUMesh *m_3d = ParaMEDMEM::MEDCouplingUMesh::New("mesh_PWR", 3);
7) m_3d->setCoords(c_3d); m_3d->allocateCells(9); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXA8, 8, conn); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXA8, 8, conn+8); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXA8, 8, conn+16); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXA8, 8, conn+24); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXA8, 8, conn+32); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXA8, 8, conn+40); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXA8, 8, conn+48); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXA8, 8, conn+56); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXA8, 8, conn+64).

APPENDIX C   PRACTICE OF VVER TYPE MESH CONSTRUCTION

This case extends the mesh set from a single cell to multi-cell. It is only for demonstration thus contains only nine cells instead of representing the full PWR core. The final desired mesh is shown in Figure C-3. The meshes are hexagonal.

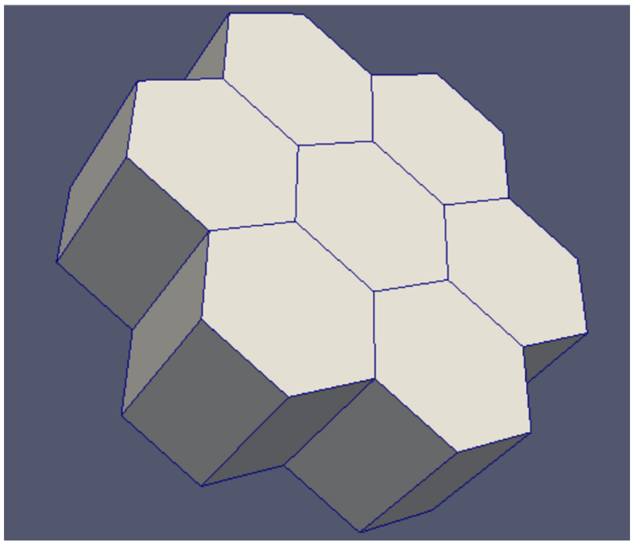


Figure C-3 The Desired Core Mesh Type for VVER

1) The first step is setting the order of the points, Figure C-4.

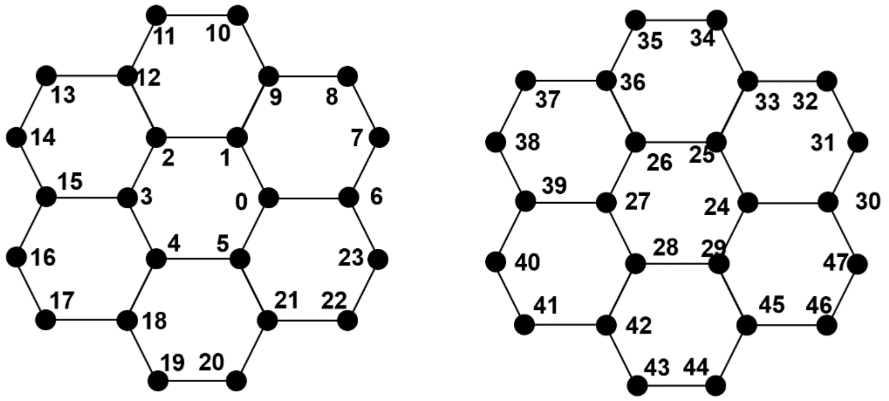


Figure C-4 The Order of the Points of VVER Core Mesh

2) Then, define the coordination of the points. double coor[144]= {2.0774, 1.5, 0.0, 1.7887, 2.0, 0.0, 1.2113, 2.0, 0.0, 0.9225, 1.5, 0.0, 1.2113, 1.0, 0.0, 1.7887, 1.0, 0.0, 2.6550, 1.5, 0.0, 2.9437, 2.0, 0.0, 2.6550, 2.5, 0.0, 2.0774, 2.5, 0.0, 1.7887, 3.0, 0.0, 1.2113, 3.0, 0.0,

0.9225, 2.5, 0.0, 0.3450, 2.5, 0.0, 0.0563, 2.0, 0.0, 0.3450, 1.5, 0.0, 0.0563, 1.0, 0.0, 0.3450, 0.5, 0.0,0.9225, 0.5, 0.0, 1.2113, 0.0, 0.0, 1.7887, 0.0, 0.0, 2.0774, 0.5, 0.0, 2.6550, 0.5, 0.0, 2.9437, 1.0, 0.0, 2.0774, 1.5, 1.0, 1.7887, 2.0, 1.0, 1.2113, 2.0, 1.0, 0.9225, 1.5, 1.0, 1.2113, 1.0, 1.0, 1.7887, 1.0, 1.0, 2.6550, 1.5, 1.0, 2.9437, 2.0, 1.0, 2.6550, 2.5, 1.0, 2.0774, 2.5, 1.0, 1.7887, 3.0, 1.0, 1.2113, 3.0, 1.0, 0.9225, 2.5, 1.0, 0.3450, 2.5, 1.0, 0.0563, 2.0, 1.0, 0.3450, 1.5, 1.0, 0.0563, 1.0, 1.0, 0.3450, 0.5, 1.0, 0.9225, 0.5, 1.0, 1.2113, 0.0, 1.0, 1.7887, 0.0, 1.0, 2.0774, 0.5, 1.0, 2.6550, 0.5, 1.0, 2.9437, 1.0, 1.0}.

3) Build the connectivity array. int conn[84]= {0,1,2,3,4,5,24,25,26,27,28,29, 0,6,7,8,9,1,24,30,31,32,33,25, 1,9,10,11,12,2,25,33,34,35,36,26, 2,12,13,14,15,3,26,36,37,38,39,27, 3,15,16,17,18,4,27,39,40,41,42,28, 4,18,19,20,21,5,28,42,43,44,45,29, 5,21,22,23,6,0,29,45,46,47,30,24}.
4) DataArrayDouble *c_3d = DataArrayDouble::New(); c_3d->alloc(48, 3); std::copy(coor, coor + 48*3, c_3d->getPointer()).
5) c_3d->setInfoOnComponent(0, "X"); c_3d->setInfoOnComponent(1, "Y"); c_3d->setInfoOnComponent(2, "Z").
6) ParaMEDMEM::MEDCouplingUMesh *m_3d = ParaMEDMEM::MEDCouplingUMesh::New("mesh_VVER", 3);
7) m_3d->setCoords(c_3d); m_3d->allocateCells(7); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXGP12, 12, conn); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXGP12, 12, conn+12); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXGP12, 12, conn+24); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXGP12, 12, conn+36); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXGP12, 12, conn+48); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXGP12, 12, conn+60); m_3d->insertNextCell(INTERP_KERNEL::NORM_HEXGP12, 12, conn+72).

APPENDIX D PRACTICE OF MED FIELD CONSTRUCTION

a) A field for PWR type meshes in Appendix B

1) ParaMEDMEM::MEDCouplingFieldDouble *field_PWR = ParaMEDMEM::
MEDCouplingFieldDouble::New(ON_CELLS, ONE_TIME);
2) field_PWR->setNature(ConservativeVolumic);
3) field_PWR->setName(“PWR_source_T");
4) field_PWR->setMesh(mesh_PWR);
5) field_PWR->setTime(problem_time, problem_step, -1);
6) DataArrayDouble *ar_PWR = DataArrayDouble::New();
7) ar_PWR->alloc(1, 3);
8) ar_PWR->setInfoOnComponent(0, "X");
9) int nb_data_set = ar_PWR->getNumberOfTuples();
10) int nb_component = ar_PWR->getNumberOfComponents();
11) double *a_final_PWR = ar_PWR->getPointer();
12) field_PWR->setArray(ar_PWR);
13) MEDLoader::WriteFieldUsingAlreadyWrittenMesh("mesh_PWR.med", field_PWR).

b) A field for VVER type meshes in Appendix C

1) ParaMEDMEM::MEDCouplingFieldDouble *field_VVER = ParaMEDMEM::
MEDCouplingFieldDouble::New(ON_CELLS, ONE_TIME);
2) field_PWR->setNature(ConservativeVolumic);
3) field_PWR->setName(“VVER_source_T");
4) field_PWR->setMesh(mesh_VVER);
5) field_PWR->setTime(problem_time, problem_step, -1);
6) DataArrayDouble *ar_VVER = DataArrayDouble::New();
7) ar_VVER->alloc(1, 3);
8) ar_VVER->setInfoOnComponent(0, "X");
9) int nb_data_set = ar_VVER->getNumberOfTuples();
10) int nb_component = ar_VVER->getNumberOfComponents();
11) double *a_final_VVER = ar_VVER->getPointer();
12) field_VVER->setArray(ar_VVER);
13) MEDLoader::WriteFieldUsingAlreadyWrittenMesh("mesh_VVER.med",
field_VVER).

NRC FORM 335 (12-2010) NRCMD 3.7 **U.S. NUCLEAR REGULATORY COMMISSION** **BIBLIOGRAPHIC DATA SHEET** *(See instructions on the reverse)*	1. REPORT NUMBER (Assigned by NRC, Add Vol., Supp., Rev., and Addendum Numbers, if any.) **NUREG/IA-0539**
2. TITLE AND SUBTITLE **New Functionality of TRACE: The 3D Post-Processing for the VESSEL Component in SALOME Platform**	3. DATE REPORT PUBLISHED MONTH **May** YEAR **2024**
	4. FIN OR GRANT NUMBER
5. AUTHOR(S) Kanglong Zhang; Victor Hugo Sanchez-Espinoza	6. TYPE OF REPORT Technical
	7. PERIOD COVERED (Inclusive Dates)

8. PERFORMING ORGANIZATION - NAME AND ADDRESS (If NRC, provide Division, Office or Region, U. S. Nuclear Regulatory Commission, and mailing address; if contractor, provide name and mailing address.)

Institute for Neutron Physics and Reactor Technology (INR), Karlsruhe Institute of Technology (KIT)

Hermann-von-Helmholtz-Platz 1

9. SPONSORING ORGANIZATION - NAME AND ADDRESS (If NRC, type "Same as above", if contractor, provide NRC Division, Office or Region, U. S. Nuclear Regulatory Commission, and mailing address.)

Division of System Analysis
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, D.C. 20555-0001

10. SUPPLEMENTARY NOTES

K. Tien, NRC Project Manager

11. ABSTRACT (200 words or less)

The researchers from the group of Reactor Physics and Dynamics (RPD) which is under the Institute of Institute for Neutron Physics and Reactor Technology (INR) - Karlsruhe Institute of Technology (KIT) - Germany, have developed a new post-processing functionality for the U.S. NRC system thermal-hydraulic code -TRACE. Now, the 3D VESSEL component of TRACE and the calculated physical fields stored in the component can be visualized in a pre- and post-processing open-source platform - SALOME with the help of a powerful data-processing library - MEDCoupling.

The researchers develop new Fortran and C++ routines to automatically identify and select the geometrical data set from TRACE input files. This data set is processed by the MEDCoupling library to generate a polyhedron mesh and a surface mesh. They are both 3D objects. The former store cell-centered fields e.g., the coolant temperature and the pressure while the latter store face-located variables e.g., the coolant velocity and the pressure drop.

21 kinds of fields can be post-processed associated with the two VESSEL meshes at present including the coolant density, the void fraction, etc. Users can conveniently access the VESSEL meshes and the fields in SALOME with plenty of operations e.g., cutting, data filtering. Scaling up the processed fields is now in planning.

This functionality was tested by processing the results gained from a VVER-1000 coolant mixing simulation.

12. KEY WORDS/DESCRIPTORS (List words or phrases that will assist researchers in locating the report.) TRACE, SALOME; MED; Post-processing	13. AVAILABILITY STATEMENT unlimited
	14. SECURITY CLASSIFICATION *(This Page)* unclassified *(This Report)* unclassified
	15. NUMBER OF PAGES
	16. PRICE

NUREG/IA-0539 New Functionality of TRACE: The 3D Post Processing for the VESSEL Component in SALOME Platform May 2024