SimPhoNy-Mayavi Documentation

Release 0.2.0

SimPhoNy FP7 Collaboration

January 13, 2016

Contents

A plugin-library for the Simphony framework (http://www.simphony-project.eu/) to provide visualization support (using http://www.paraview.org/) of the CUDS highlevel components.

CHAPTER 1

Repository

Simphony-paraview is hosted on github: https://github.com/simphony/simphony-paraview

CHAPTER 2

Requirements

- paraview >= 3.14.1
- simphony >= 0.2.1

`simphony-paraview` is known to work with paraview 3.14.1 (official) and paraview 4.1.0 (paraviewopenfoam) on Ubuntu 12.04 (precise). Installation instructions are provided below.

2.1 Paraview 3.14.1

```
sudo apt-get install paraview
```

2.2 ParaviewOpenFoam 4.1.0

```
sudo sh -c "echo deb http://www.openfoam.org/download/ubuntu precise main > /etc/apt/sources.list.d/o
sudo apt-get update
sudo apt-get install paraviewopenfoam410
```

2.3 Optional requirements

To support the documentation built you need the following packages:

- sphinx >= 1.2.3
- sectiondoc https://github.com/enthought/sectiondoc
- mock

Alternative running **`pip install -r doc_requirements.txt`** should install the minimum necessary components for the documentation built.

CHAPTER 3

Installation

The package requires python 2.7.x, installation is based on setuptools:

```
# build and install
python setup.py install
```

or:

```
# build for in-place development
python setup.py develop
```

CHAPTER 4

Testing

To run the full test-suite run:

```
python -m unittest discover
```

CHAPTER 5

Documentation

To build the documentation in the doc/build directory run:

```
python setup.py build_sphinx
```

Note:

- One can use the –help option with a setup.py command to see all available options.
- The documentation will be saved in the `./build` directory.

CHAPTER 6

Usage

After installation the user should be able to import the `paraview` visualization plugin module by:

```
from simphony.visualization import paraview_tools
paraview_tools.show(cuds)
```

CHAPTER 7

FAQ

- Paraview contains a separate python runtime called **pvpython**. which python should we use?

  simphony-paraview is tested and developed using the system python on Ubuntu 12.04. In theory one could install simphony and simphony-paraview on any other python 2.7.x runtime like *pvpython*, but you will need to build all dependencies against the pvpython runtime environment.

- When using paraviewopenfoam and the system simphony-paraview does not work, whats wrong?

  Openfoam paraview does not make the provided python packages available to the system python thus in order to use the simphony-paraview plugin from the system python one needs to change the following environment variables:

```
export PYTHONPATH=${PYTHONPATH}:/opt/paraviewopenfoam410/lib/paraview-4.1/site-packages/:/opt/pa
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/opt/paraviewopenfoam410/lib/paraview-4.1
```

CHAPTER 8

Known Issues

- Intermittent segfault when running the test-suite (#22)
- Pressing `a` while interacting with a view causes a segfault (#23)
- An Empty window appears when using the snapshot function (#24)

CHAPTER 9

Directory structure

- simphony-paraview – Main package code.
  - core – Utilities and basic conversion tools.
- examples – Holds examples of visualizing simphony objects with simphony-paraview.
- doc – Documentation related files:
  - source – Sphinx rst source files
  - build – Documentation build directory, if documentation has been generated using the `make` script in the `doc` directory.

CHAPTER 10

User Manual

10.1 SimPhoNy

Paraview tools are available in the simphony library through the visualisation plug-in named `paraview_tools`.

e.g:

```
from simphony.visualisation import paraview_tools
```

10.1.1 Visualizing CUDS

The `show()` function is available to visualise any top level CUDS container. The function will open a window containing a 3D view of the dataset. Interaction is supported using the mouse and keyboard:

keyboard

- `j` / `t`: toggle between joystick (position sensitive) and trackball (motion sensitive) styles. In joystick style, motion occurs continuously as long as a mouse button is pressed. In trackball style, motion occurs when the mouse button is pressed and the mouse pointer moves.
- `3`: toggle the render window into and out of stereo mode. By default, red-blue stereo pairs are created.
- `e`: exit the application.
- `f`: fly to the picked point
- `p`: perform a pick operation.
- `r`: reset the camera view along the current view direction. Centers the actors and moves the camera so that all actors are visible.
- `s`: modify the representation of all actors so that they are surfaces.
- `w`: modify the representation of all actors so that they are wireframe.

mouse

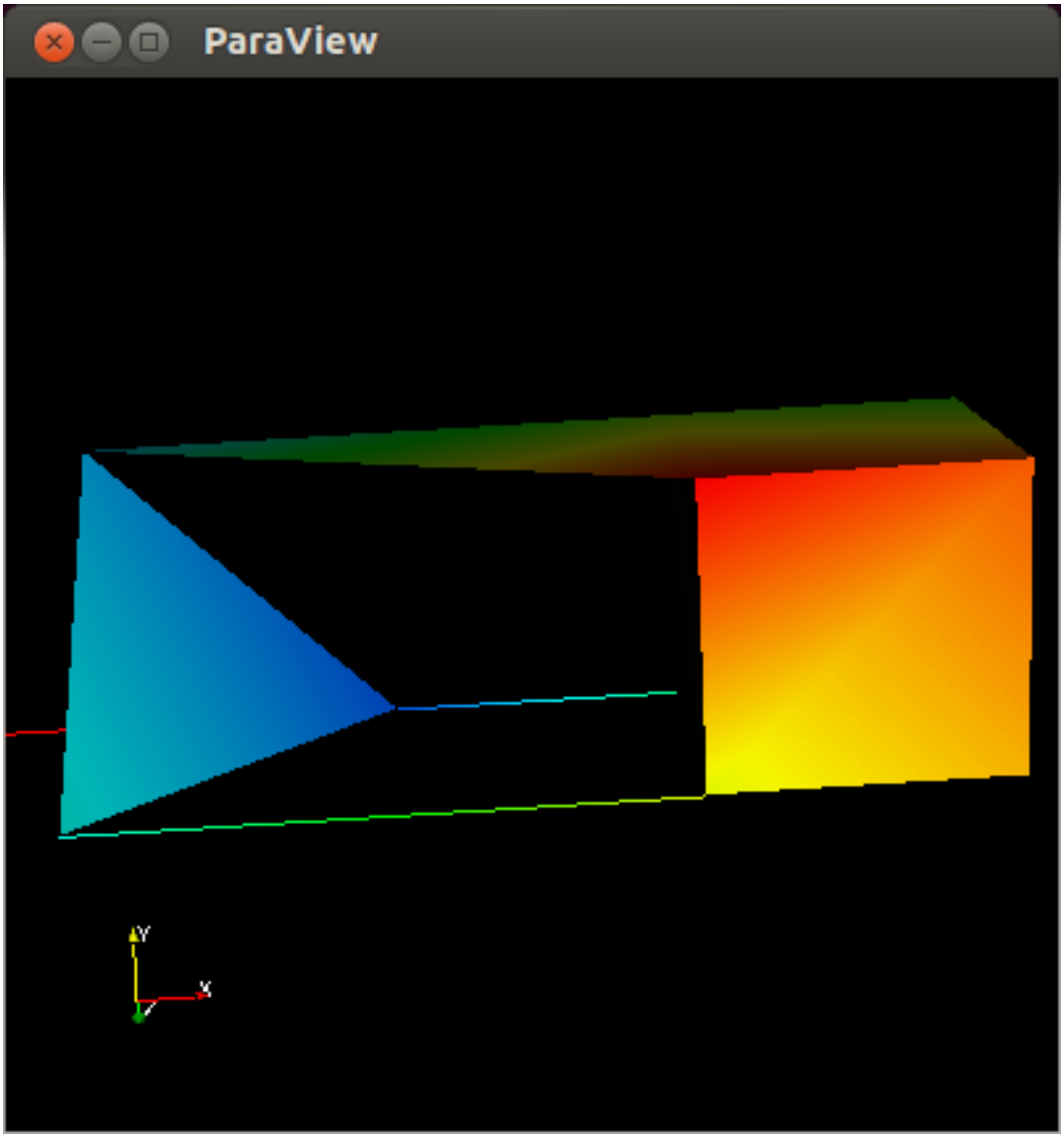
- **Button 1**: rotate the camera around its focal point (if camera mode) or rotate the actor around its origin (if actor mode). The rotation is in the direction defined from the center of the renderer's viewport towards the mouse position. In joystick mode, the magnitude of the rotation is determined by the distance the mouse is from the center of the render window.
- **Button 2**: pan the camera (if camera mode) or translate the actor (if actor mode). In joystick mode, the direction of pan or translation is from the center of the viewport towards the mouse

position. In trackball mode, the direction of motion is the direction the mouse moves. (Note: with 2-button mice, pan is defined as <Shift>-Button 1.)

- **Button 3**: zoom the camera (if camera mode) or scale the actor (if actor mode). Zoom in/increase scale if the mouse position is in the top half of the viewport; zoom out/decrease scale if the mouse position is in the bottom half. In joystick mode, the amount of zoom is controlled by the distance of the mouse pointer from the horizontal centerline of the window.

Mesh example

```
from numpy import array

from simphony.cuds import Mesh, Point, Cell, Edge, Face
from simphony.core.data_container import DataContainer
from simphony.core.cuba import CUBA


points = array([
    [0, 0, 0], [1, 0, 0], [0, 1, 0], [0, 0, 1],
    [2, 0, 0], [3, 0, 0], [3, 1, 0], [2, 1, 0],
    [2, 0, 1], [3, 0, 1], [3, 1, 1], [2, 1, 1]],
    'f')

cells = [
    [0, 1, 2, 3],  # tetra
    [4, 5, 6, 7, 8, 9, 10, 11]]  # hex

faces = [[2, 7, 11]]
edges = [[1, 4], [3, 8]]

mesh = Mesh('example')

# add points
uids = mesh.add_points(
    Point(coordinates=point, data=DataContainer(TEMPERATURE=index))
    for index, point in enumerate(points))

# add edges
edge_uids = mesh.add_edges(
    Edge(points=[uids[index] for index in element])
    for index, element in enumerate(edges))

# add faces
face_uids = mesh.add_faces(
    Face(points=[uids[index] for index in element])
    for index, element in enumerate(faces))

# add cells
cell_uids = mesh.add_cells(
    Cell(points=[uids[index] for index in element])
    for index, element in enumerate(cells))

if __name__ == '__main__':
    from simphony.visualisation import paraview_tools

    # Visualise the Mesh object
    paraview_tools.show(mesh, select=(CUBA.TEMPERATURE, 'points'))
```

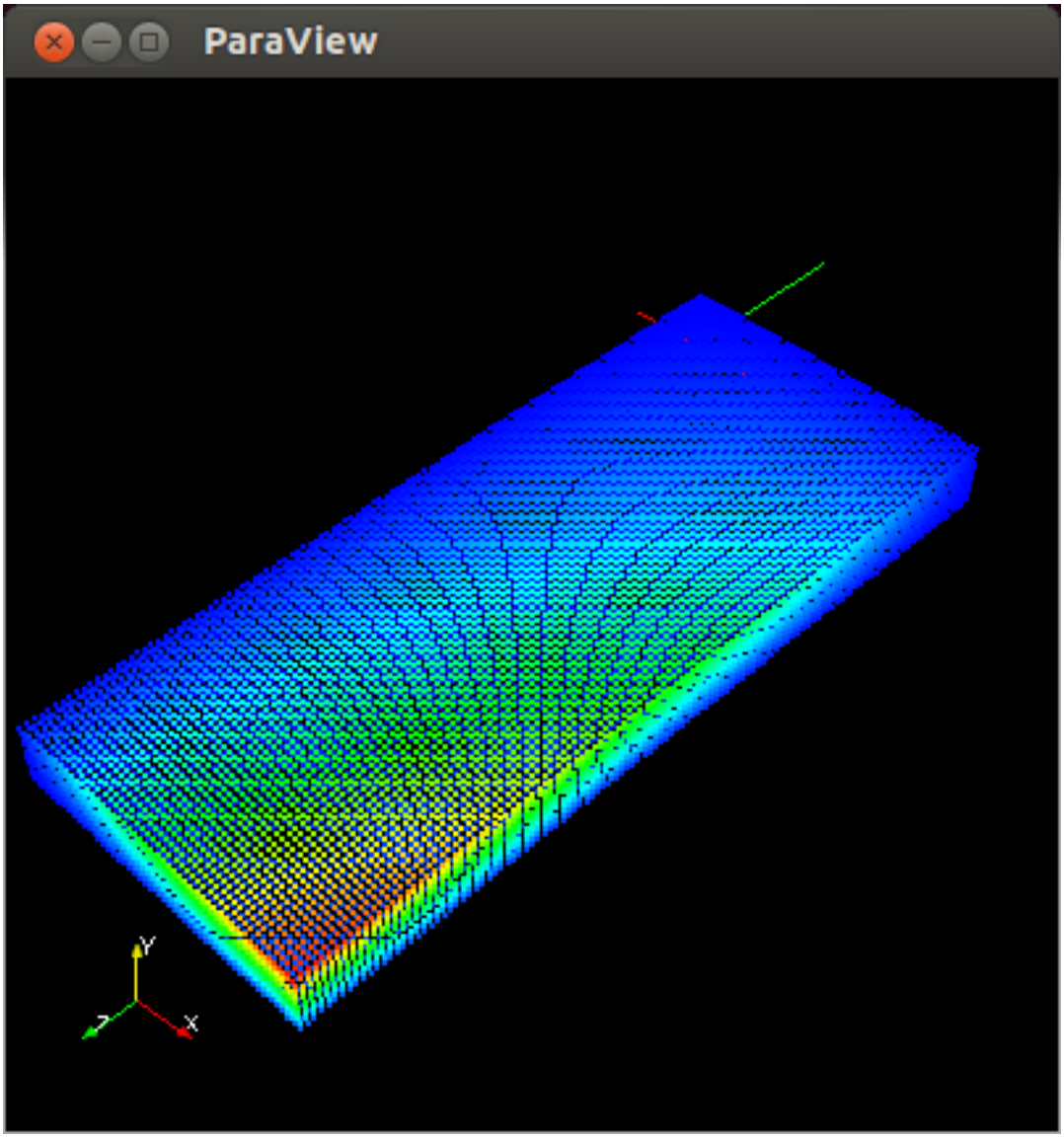

ParaView
Y
X

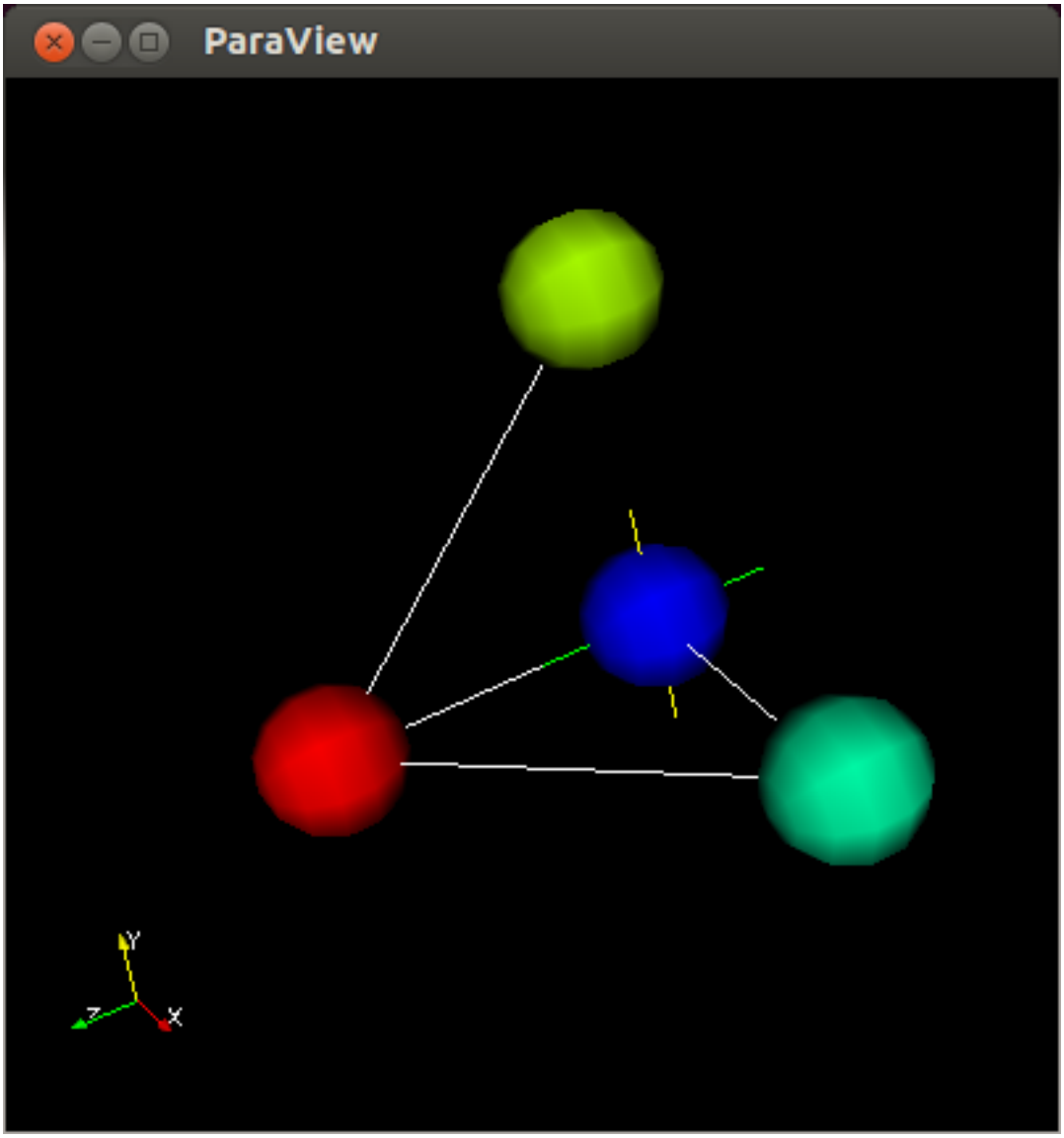
Lattice example

```
import numpy

from simphony.cuds.lattice import make_cubic_lattice
from simphony.core.cuba import CUBA

lattice = make_cubic_lattice('test', 0.1, (50, 10, 120))


def set_temperature(nodes):
    for node in nodes:
        index = numpy.array(node.index) + 1.0
        node.data[CUBA.TEMPERATURE] = numpy.prod(index)
        yield node

lattice.update_nodes(set_temperature(lattice.iter_nodes()))


if __name__ == '__main__':
    from simphony.visualisation import paraview_tools

    # Visualise the Lattice object
    paraview_tools.show(lattice, select=(CUBA.TEMPERATURE, 'nodes'))
```

Particles example

```
from numpy import array

from simphony.cuds import Particles, Particle, Bond
from simphony.core.data_container import DataContainer
from simphony.core.cuba import CUBA


points = array([[0, 0, 0], [1, 0, 0], [0, 1, 0], [0, 0, 1]], 'f')
bonds = array([[0, 1], [0, 3], [1, 3, 2]])
temperature = array([10., 20., 30., 40.])

particles = Particles('test')
uids = particles.add_particles(
    Particle(
        coordinates=point,
        data=DataContainer(TEMPERATURE=temperature[index]))
    for index, point in enumerate(points))

particles.add_bonds(
    Bond(particles=[uids[index] for index in indices])
    for indices in bonds)


if __name__ == '__main__':
    from simphony.visualisation import paraview_tools

    # Visualise the Particles object
    paraview_tools.show(particles, select=(CUBA.TEMPERATURE, 'particles'))
```


ParaView
Y
Z
X


ParaView
Y
Z
X

CHAPTER 11

API Reference

11.1 Plugin module

This module `simphony_paraview.plugin` provides a set of tools to visualize CUDS objects. The tools are also available as a visualisation plug-in to the simphony library.

11.2 Core module

A module containing core tools and wrappers for paraview data containers used in simphony_paraview.

Classes

`cuba_data_accumulator.CUBADataAccumulator`

Functions

`supported_cuba`()	Return a set of currently supported CUBA keys.
`default_cuba_value`(cuba)	Return the default value of the CUBA key as a scalar or numpy array.
`cuds2vtk.cuds2vtk`	

Mappings

`constants.points2edge`
`constants.points2face`
`constants.points2cell`
`constants.dataset2writer`
`constants.cuba_value_types`

11.2.1 Description

simphony_paraview.core.cuba_utils.**supported_cuba**()
Return a set of currently supported CUBA keys.

simphony_paraview.core.cuba_utils.**default_cuba_value**(*cuba*)
Return the default value of the CUBA key as a scalar or numpy array.

Int type values have `-1` as default, while float type values have `numpy.nan`.

D

S