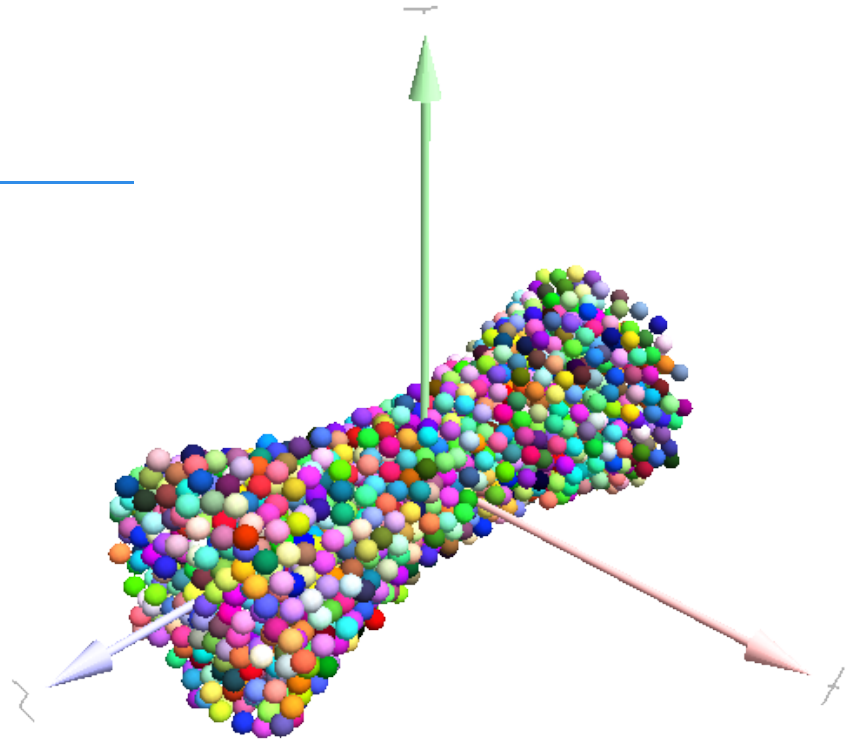
THM short-course: Day 1

The architecture of Yade

Robert Caulk[1], Bruno Chareyre[1]

June 20th, 2022

[1]Univ. Grenoble Alpes
Grenoble INP, 3SR

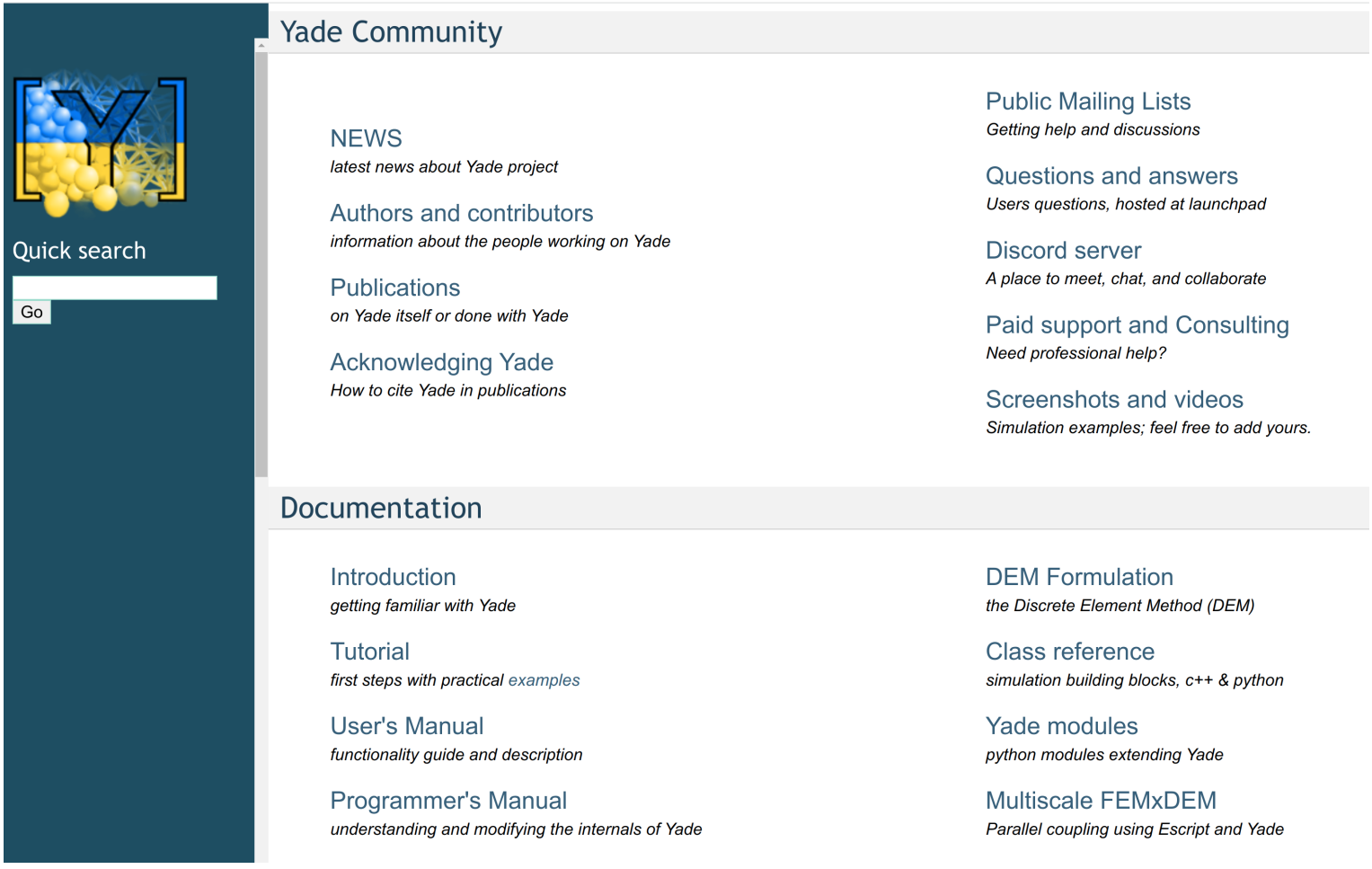


Resources, resources, resources

The Yade website

Just one keyword search away from knowing exactly what a command does:

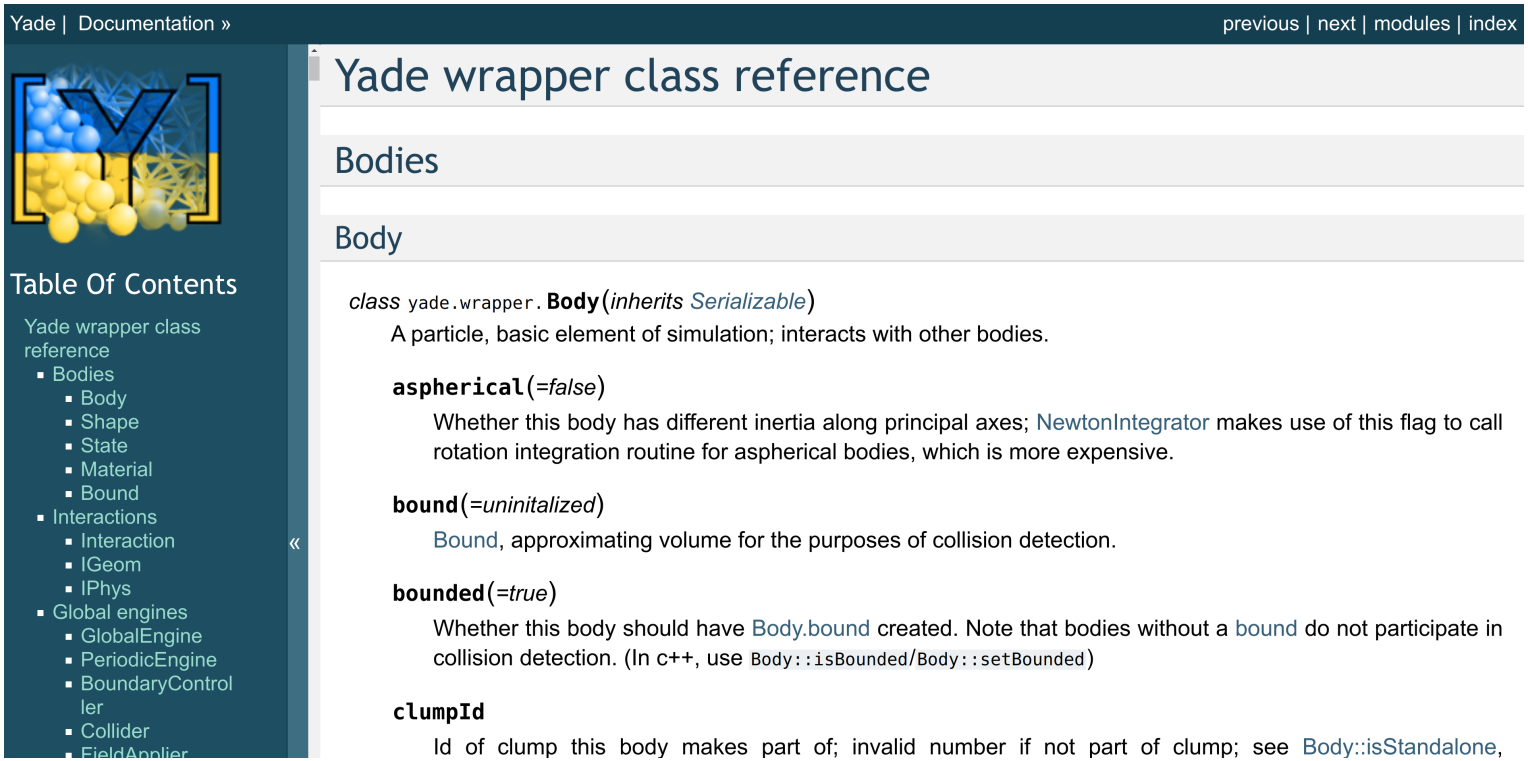
www.yade-dem.org



Class Reference - a full index

A full description of thousands of user accessible parameters

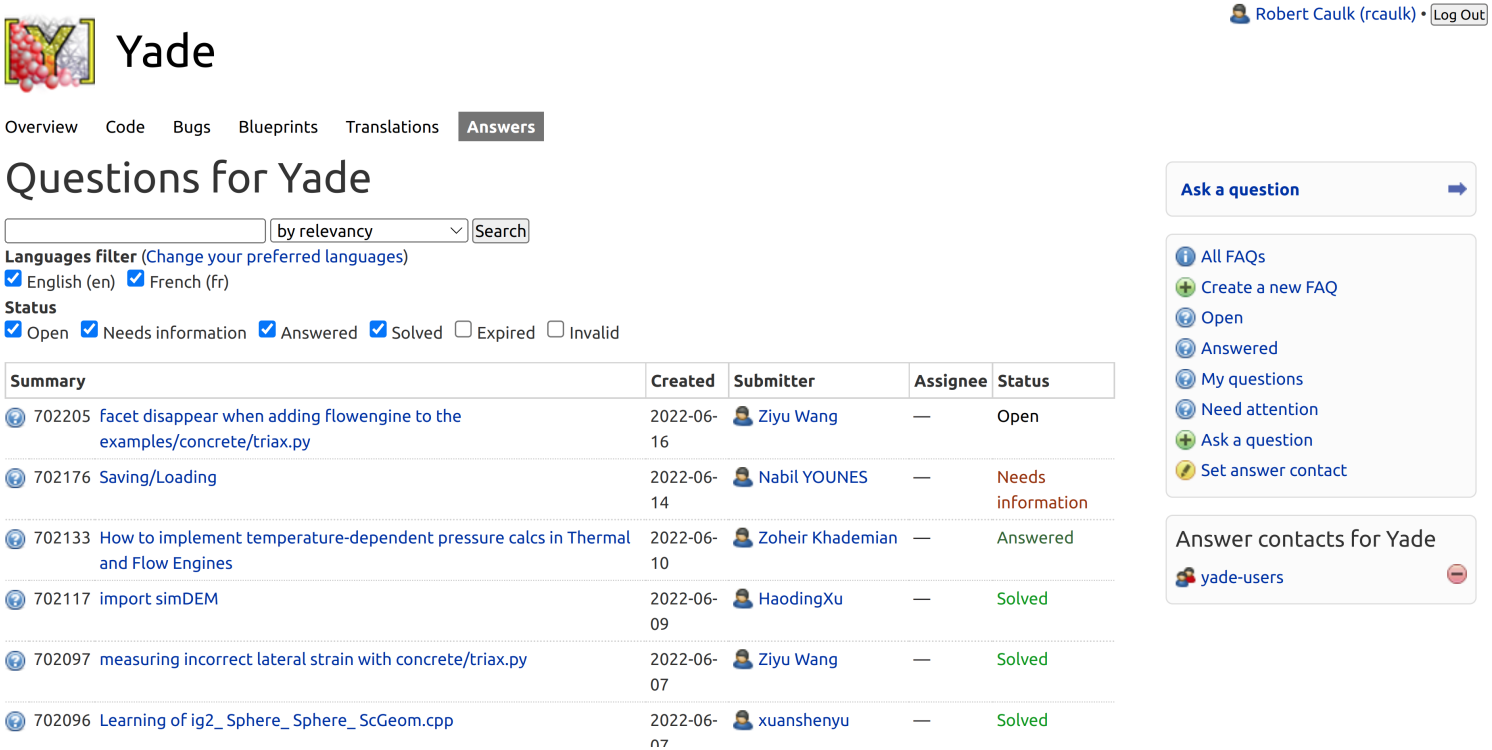
https://yade-dem.org/doc/yade.wrapper.html



Launchpad - **the community forum**

Fully searchable forum filled with common errors, and other users solving similar problems

https://answers.launchpad.net/yade



Google

Python problems, Linux problems, Yade problems!

https://www.google.com

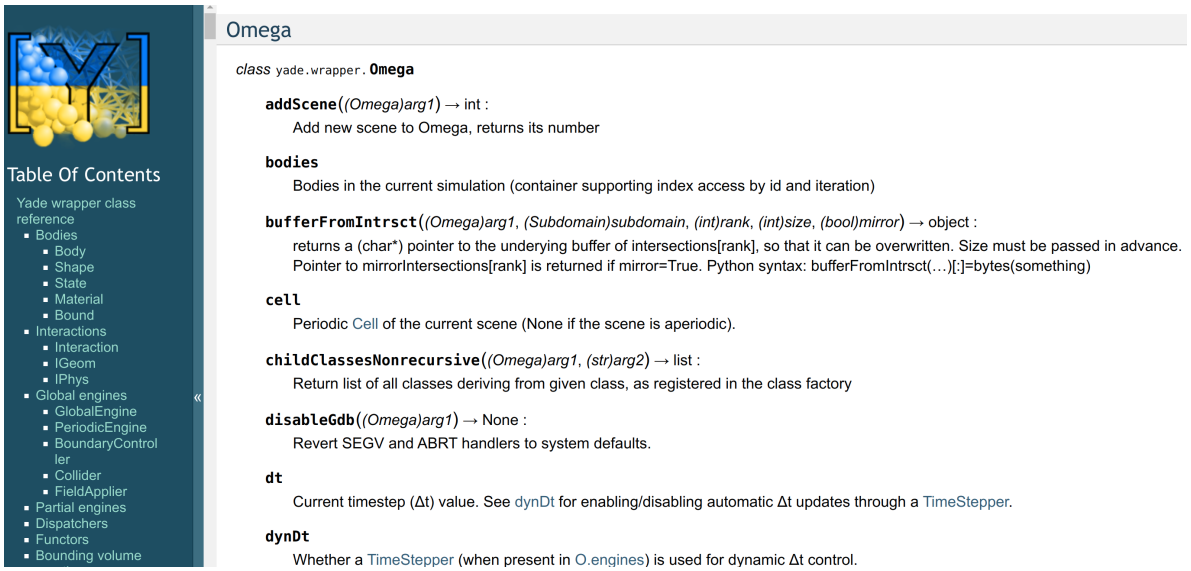
Architectural Overview

The scene

Yade is an object-oriented code:

> Everything that the user interacts with is an object
> The largest object in Yade is referred to as the **scene**

The **scene** is assigned to the variable `Omega` (O) in the user input script and it contains all the building blocks for our creation of our simulation. For example:



https://yade-dem.org/doc/yade.wrapper.html#omega

The scene

Yade is an object-oriented code:

- Everything that the user interacts with is an object
- The largest object in Yade is referred to as the **scene**

The **scene** is assigned to the variable `Omega` (`O`) in the user input script and it contains all the building blocks for our creation of our simulation. For example:

```
O.materials  # all user defined materials
O.bodies  # all user defined bodies
O.engines  # all user defined engines
```

Constructing a scene - materials

The user creates and appends objects to the **scene** before running any simulations. A user can create a new `FrictMat()` material and then append it to the scene:

```
sphere_material = FrictMat(young=1e6,
                           poisson=0.5,
                           frictionAngle=radians(18),
                           density=2500,
                           label='spheres')
O.materials.append(sphere_material)
```

Constructing a scene - bodies

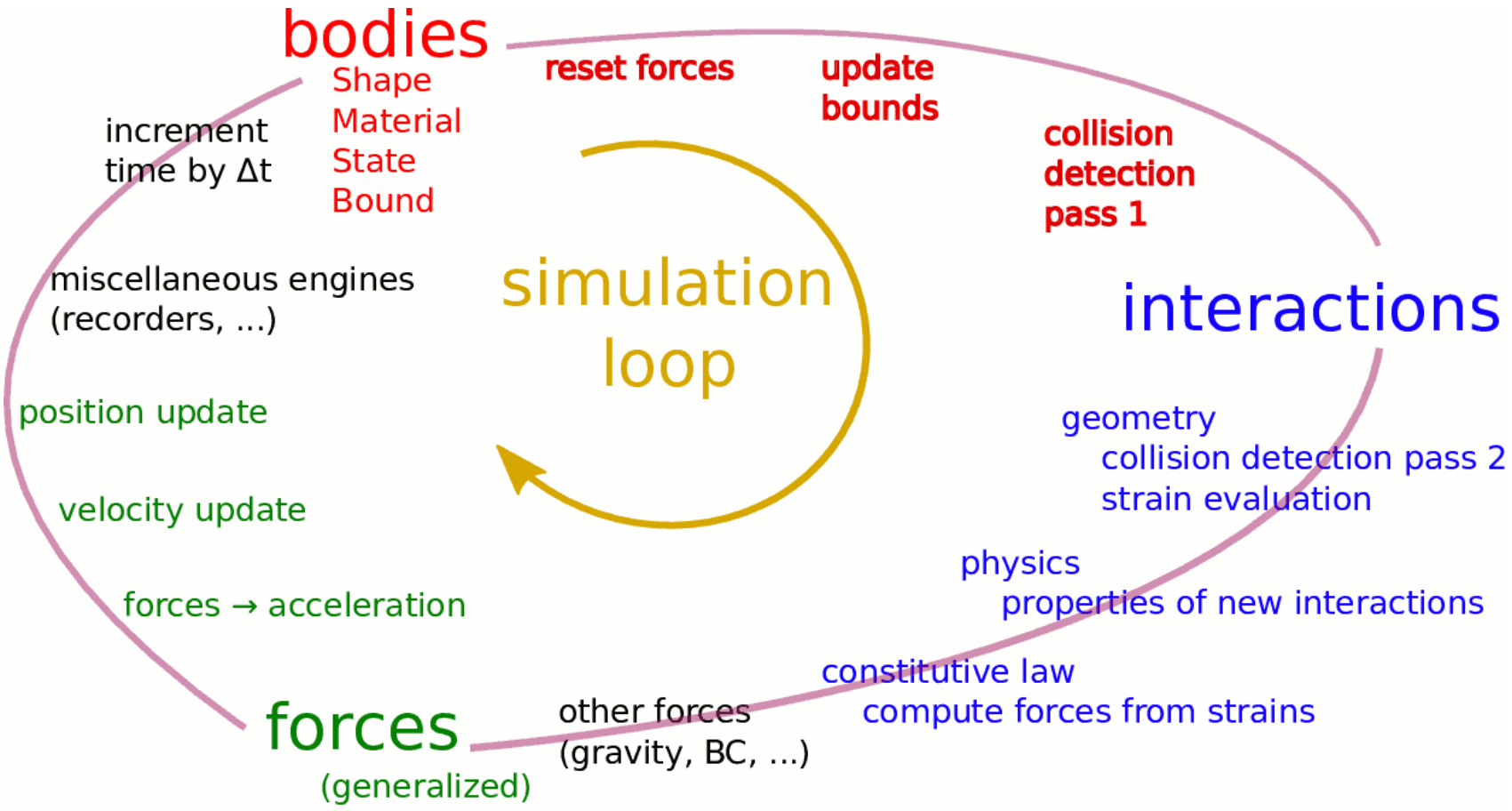
Yade is filled with tools to assist with the creation of all types of bodies in various geometries. Look in our documentation for all the options: https://yade-dem.org/doc/yade.pack.html#module-yade.pack

A user may create a `sphere()` object and append it to the simulation:

```
sphere = sphere(center=(0, 0, 0),
                radius=.5,
                fixed=True)
O.bodies.append(sphere)
```

Constructing a scene - engines

The Yade DEM `O.engine` list is the list of algorithms that Yade will execute for each time-step:



Constructing a scene - engines

The Yade DEM `O.engine` list is the list of algorithms that Yade will execute for each time-step:

```
O.engines = [
        ForceResetter(),
        InsertionSortCollider([Bo1_Sphere_Aabb()]),
        InteractionLoop(
        [Ig2_Sphere_Sphere_ScGeom()],  # geometry
        [Ip2_FrictMat_FrictMat_FrictPhys()],  # physics
        [Law2_ScGeom_FrictPhys_CundallStrack()]  # contact
        ↪  law
        ),
        NewtonIntegrator(gravity=(0, 0, -9.81), damping=0.1)
]
```

The user may add, edit, and remove many of these different engines. Some are necessary and others are optional.

PyRunner - **infinite flexibility**

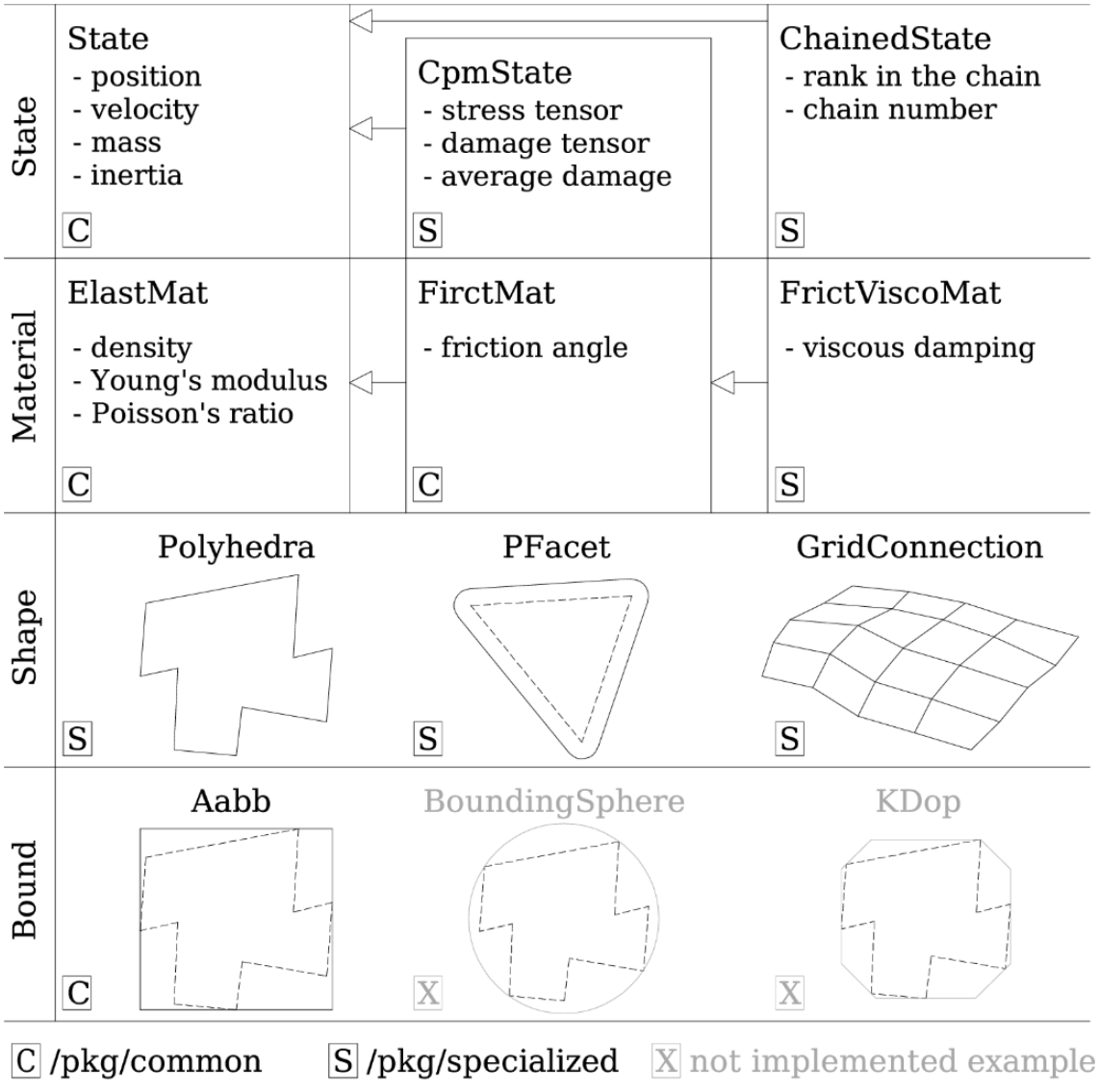
Users can ask Yade to execute custom functions in the `O.engines` list by adding a `PyRunner()`:

```
O.engines = [
ForceResetter(),
InsertionSortCollider([Bo1_Sphere_Aabb()]),
InteractionLoop(
[Ig2_Sphere_Sphere_ScGeom()],  # geometry
[Ip2_FrictMat_FrictMat_FrictPhys()],  # physics
[Law2_ScGeom_FrictPhys_CundallStrack()]  # contact law
),
PyRunner(command="print('kinetic energy,
↪  kineticEnergy())",realPeriod=5)
NewtonIntegrator(gravity=(0, 0, -9.81), damping=0.1)
]
```

Where is the data stored??

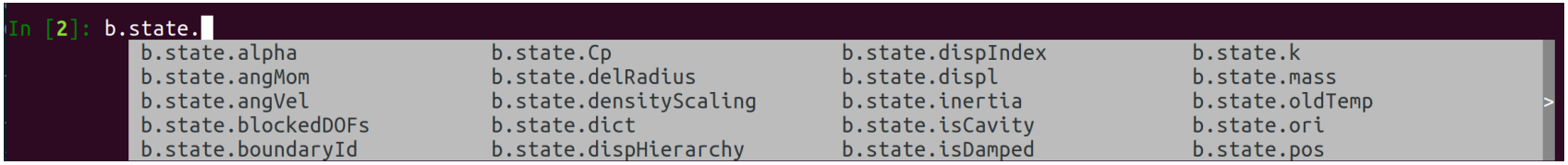
Accessing scene data

Information can be gathered from the various objects:



Accessing scene data

Information can be gathered from the various objects. For example, the body state contains a plethora of information:



Note: this visual was generated by running:

```
yadedaily example.py
```

and then typing into the ipython prompt:

```
b = O.bodies[0]  # indexing into the bodies container
b.state.  # then hit the `tab` key to reveal all
 ↪  options
```

This interactive method of investigating available Yade variables can be used for any object and any method.

Accessing scene data

Collecting state variables can be achieved in numerous ways. Primarily, using the `saveDataText()` function:

```
def history():
        plot.saveDataTxt('data_collect.txt',
                         vars=('t','i','pos_x'))

# then we add our trusty PyRunner
O.engines=O.engines+[PyRunner(iterPeriod=500,
                         command='history()',
                         label='recorder')]
```

Note: Yade has a plethora of export tools at your disposal available here:

https://yade-dem.org/doc/yade.export.html#module-yade.export

Plotting scene data

Plotting in Yade is as simple as employing the `plot.addData()` function followed by a `plot.plot()`:

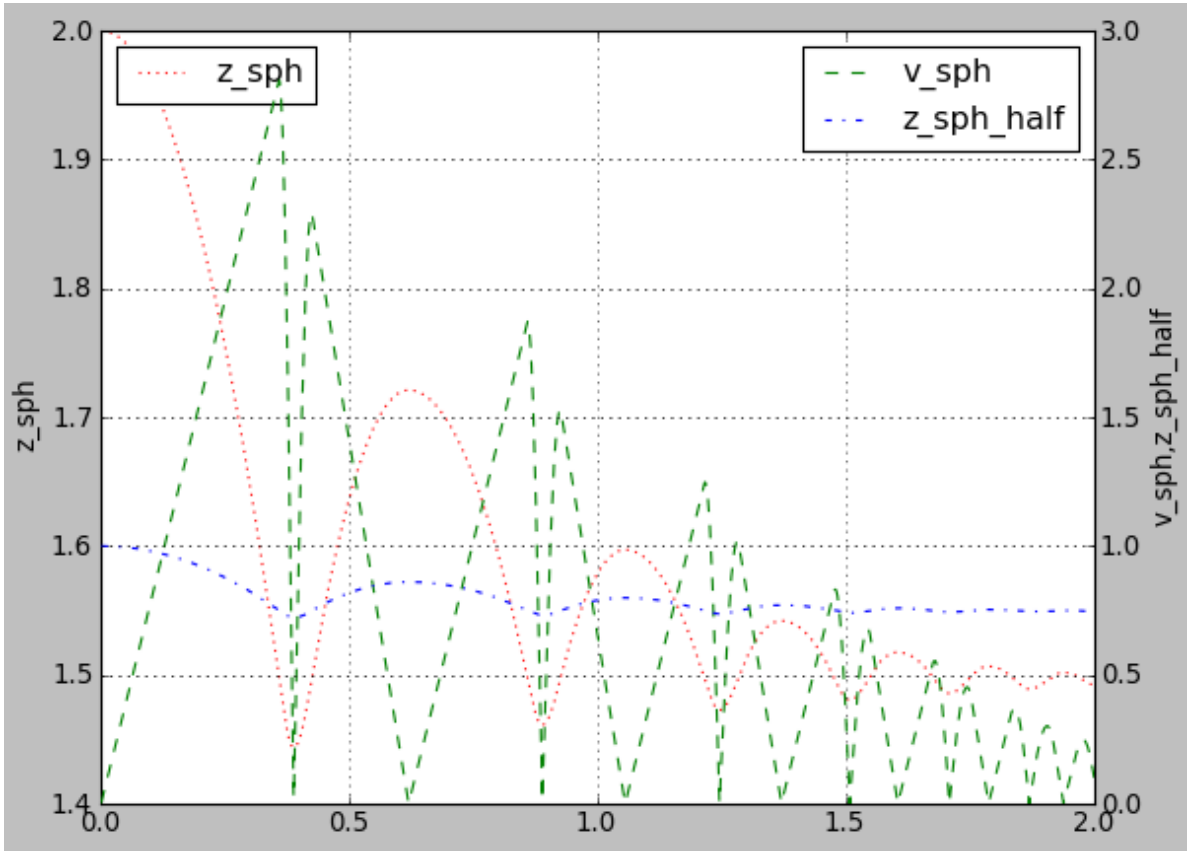
```
def history():
        plot.addData(t = O.time, i = O.iter, pos_x =
        ↪  O.bodies[10].pos[0])
        plot.saveDataTxt('data_collect.txt',
        ↪  vars=('t','i','pos_x'))
O.engines=O.engines+[PyRunner(iterPeriod=500,
↪  command='history()', label='recorder')]
from yade import plot
plot.plots = {'t': ('pos_x', 'b--')}
plot.plot()
```

Note: Yade has a plethora of plotting tools at your disposal available here:

https://yade-dem.org/doc/yade.plot.html

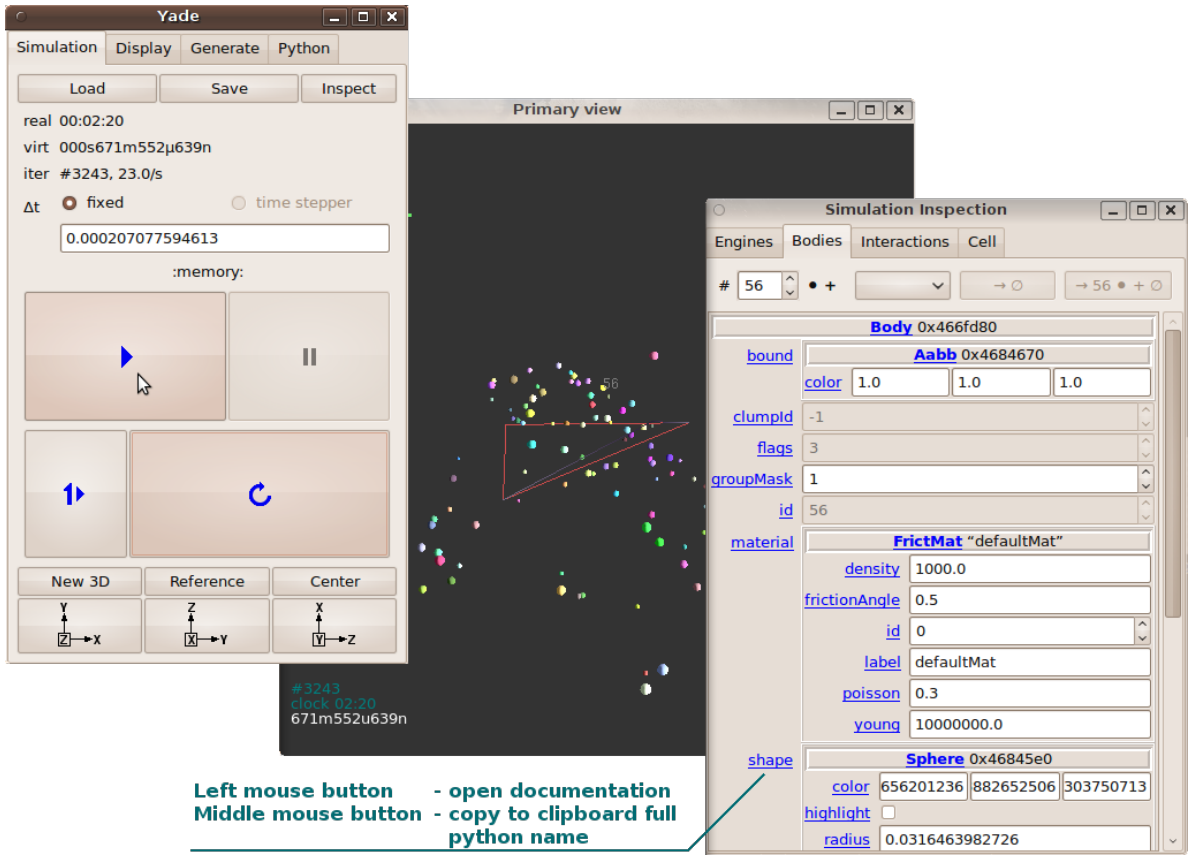
Plotting scene data

The `plot` module enables full flexibility for data plotting such as multiple lines per axis, double y-axes, controlling line type/color etc.



Visualizing the scene - Qt

Yade has it's own GUI feature for rapid scene validation which is invoked by `yade.qt.Controller()` or pressing `F12`.



Visualizing the scene - Paraview

Yade exporter can export VTK files for reopening in Paraview (open-source). This software enables deep data analysis and visualization. The user simply adds `VTKRecorder()` to their `O.engines` list:

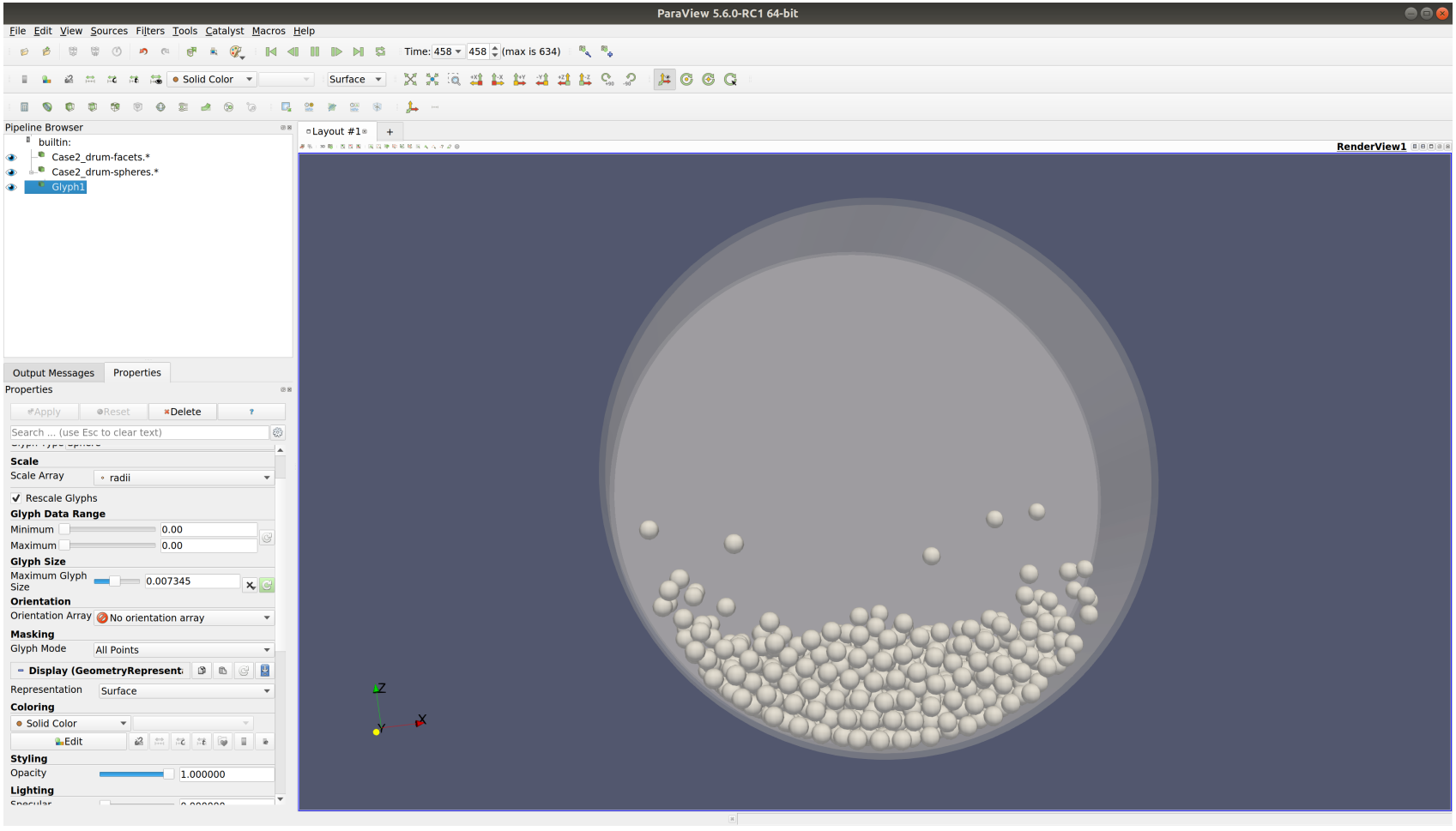
```
O.engines = [
        ...
        VTKRecorder(iterPeriod=1000,
                                fileName='uniqeId',
                                recorders=['spheres','facets'],
                                label='vtkrecorder')
]
```

Users can explore all the deep functionality of the `VTKRecorder()` by going to the documentation:

https://yade-dem.org/doc/yade.wrapper.html#yade.wrapper.VTKRecorder

Visualizing the scene - Paraview

Paraview has powerful post-processing tools



Controlling the simulation!

O.run()

Users can run their simulation (read start iterating on the O.engines list) by running the command O.run().

```
run((Omega)arg1[, (int)nSteps=-1[, (bool)wait=False]])
 ↪   | None :|
         Run the simulation. nSteps how many steps to
          ↪  run, then stop (if positive); wait will
          ↪  cause not returning to python until
          ↪  simulation will have stopped.
```

O.stop()

Users can stop their simulation (read start iterating on the `O.engines` list) by running the command `O.stop()`.

step(*(Omega)arg1*) → None :

Advance the simulation by one step. Returns after the step will have finished.

stopAtIter

Get/set number of iteration after which the simulation will stop.

stopAtTime

Get/set time after which the simulation will stop.

Endless functionalities to discover

Yade can also:

- Import meshes from stl files
- Save and load simulations
- Create dense packings
- Triaxial/uniaxial tests
- Couplings (FEMDEM, CFD, PFV)
- MPI
- Custom contact laws
- Variable dependent properties