

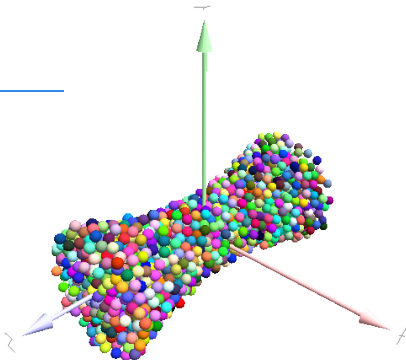
THM short-course: **Day 1**

The architecture of Yade

Robert Caulk¹, Bruno Chareyre¹

June 20th, 2022

¹Univ. Grenoble Alpes
Grenoble INP, 3SR

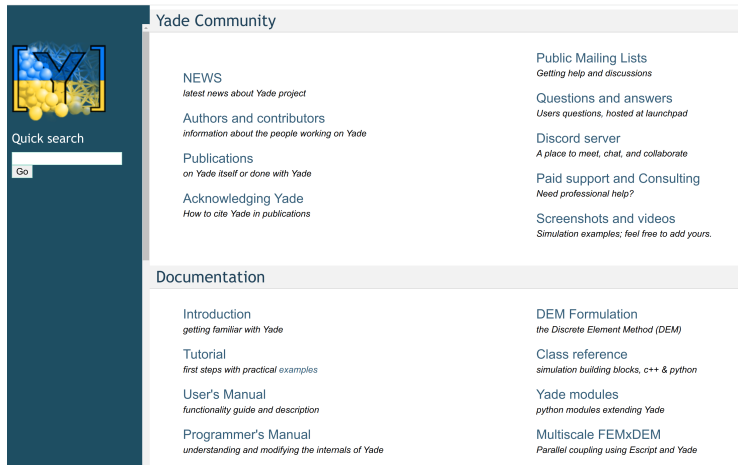


Resources, resources, resources

The Yade website

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

www.yade-dem.org



The image shows a vertical navigation menu on the left side of the Yade website. It features a large blue 'Y' logo with a 3D particle simulation background. Below the logo is a 'Quick search' section with a text input field and a 'Go' button. The main navigation area is divided into two horizontal sections: 'Yade Community' and 'Documentation'. Each section contains a list of links with brief descriptions.

Yade Community

- NEWS**
latest news about Yade project
- Authors and contributors**
information about the people working on Yade
- Publications**
on Yade itself or done with Yade
- Acknowledging Yade**
How to cite Yade in publications
- Public Mailing Lists**
Getting help and discussions
- Questions and answers**
Users questions, hosted at launchpad
- Discord server**
A place to meet, chat, and collaborate
- Paid support and Consulting**
Need professional help?
- Screenshots and videos**
Simulation examples; feel free to add yours.

Documentation

- Introduction**
getting familiar with Yade
- Tutorial**
first steps with practical examples
- User's Manual**
functionality guide and description
- Programmer's Manual**
understanding and modifying the internals of Yade
- DEM Formulation**
the Discrete Element Method (DEM)
- Class reference**
simulation building blocks, c++ & python
- Yade modules**
python modules extending Yade
- Multiscale FEMxDEM**
Parallel coupling using Escript and Yade

Class Reference - a full index

A full description of thousands of user accessible parameters

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

Yade | Documentation » previous | next | modules | index

Yade wrapper class reference

Bodies

Body

class yade.wrapper.Body (inherits *Serializable*)
A particle, basic element of simulation; interacts with other bodies.

aspherical (=false)
Whether this body has different inertia along principal axes; *NewtonIntegrator* makes use of this flag to call rotation integration routine for aspherical bodies, which is more expensive.

bound (=uninitialized)
Bound, approximating volume for the purposes of collision detection.

bounded (=true)
Whether this body should have *Body.bound* created. Note that bodies without a *bound* do not participate in collision detection. (In c++, use *Body::isBounded/Body::setBounded*)

clumpId
Id of clump this body makes part of; invalid number if not part of clump; see *Body::isStandalone*,

Launchpad - the community forum

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

<https://answers.launchpad.net/yade>



Yade

Overview Code Bugs Blueprints Translations **Answers**

Questions for Yade

by relevancy

Languages filter (Change your preferred languages)

English (en) French (fr)

Status

Open Needs information Answered Solved Expired Invalid

Summary	Created	Submitter	Assignee	Status
702205 Facet disappear when adding flowengine to the examples/concrete/triax.py	2022-06-16	Ziyu Wang	—	Open
702176 Saving/Loading	2022-06-14	Nabil YOUNES	—	Needs information
702133 How to implement temperature-dependent pressure calcs in Thermal and Flow Engines	2022-06-10	Zohair Khademan	—	Answered
702117 import simDEM	2022-06-09	HaodingXu	—	Solved
702097 measuring incorrect lateral strain with concrete/triax.py	2022-06-07	Ziyu Wang	—	Solved
702096 Learning of ig2_Sphere_Sphere_ScGeom.cpp	2022-06-07	xuanshenyu	—	Solved

[Robert Caulk \(rcaulk\)](#) [Log Out](#)

[Ask a question](#)

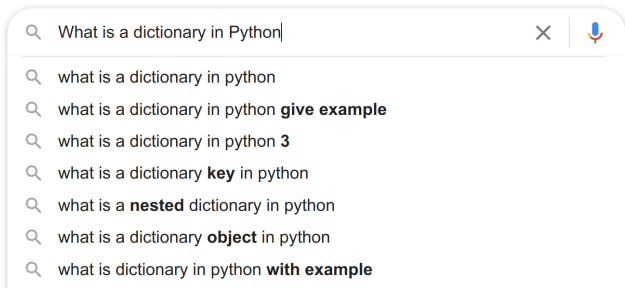
- [All FAQs](#)
- [Create a new FAQ](#)
- [Open](#)
- [Answered](#)
- [My questions](#)
- [Need attention](#)
- [Ask a question](#)
- [Set answer contact](#)

Answer contacts for Yade

[yade-users](#)

Python problems, Linux problems, Yade problems!

<https://www.google.com>



Architectural Overview

The scene

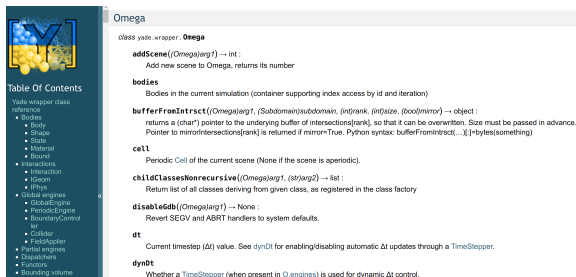
Yade is an object-oriented code:

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:



Omega

```
class yade.wrapper: Omega

    addScene((Omega)arg1) → int :
        Add new scene to Omega, returns its number

    bodies
        Bodies in the current simulation (container supporting index access by id and iteration)

    bufferFromIntrct((Omega)arg1, (Subdomain)subdomain, (int)rank, (int)size, (bool)mirror) → object :
        returns a (char*) pointer to the underlying buffer of intersections[rank], so that it can be overwritten. Size must be passed in advance.
        Pointer to mirrorIntersections[rank] is returned if mirror=True. Python syntax: bufferFromIntrct(...);|-bytes(something)

    cell
        Periodic: Cell of the current scene (None if the scene is aperiodic).

    childClassesNonrecursive((Omega)arg1, (str)arg2) → list :
        Return list of all classes deriving from given class, as registered in the class factory

    disableGdb((Omega)arg1) → None :
        Revert SEGV and ABRT handlers to system defaults.

    dt
        Current timestep (Δt) value. See dyndt for enabling/disabling automatic Δt updates through a TimeStepper.

    dyndt
        Whether a TimeStepper (when present in O.engines) is used for dynamic Δt control.
```

<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:

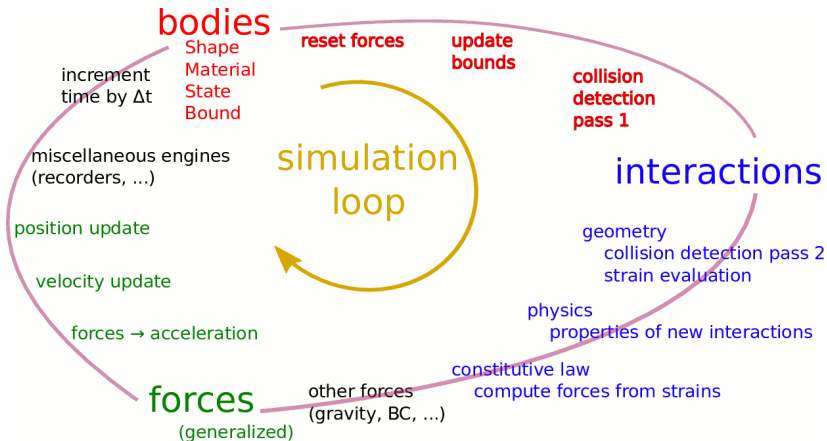
<https://yade-dem.org/doc/yade.pack.html#module-yade.pack>

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 0.engine list is the list of algorithms that Yade will execute for each time-step:



Constructing a scene - engines

The Yade DEM `O.engines` 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.


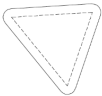
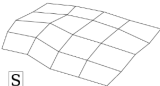
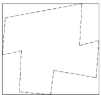


Users can ask Yade to execute custom functions in the `O.engines` list by adding a `PyRunner()`:

```
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:

State	State - position - velocity - mass - inertia C	CpmState - stress tensor - damage tensor - average damage S	ChainedState - rank in the chain - chain number S
Material	ElastMat - density - Young's modulus - Poisson's ratio C	FrictMat - friction angle C	FrictViscoMat - viscous damping S
Shape	Polyhedra  S	PFacet  S	GridConnection  S
Bound	Aabb  C	BoundingSphere  X	KDop  X

C /pkg/common S /pkg/specialized X not implemented example

Accessing scene data

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

```
In [2]: b.state.  
b.state.alpha          b.state.Cp          b.state.dispIndex    b.state.k  
b.state.angMom         b.state.delRadius   b.state.displ        b.state.mass  
b.state.angVel        b.state.densityScaling b.state.inertia      b.state.oldTemp  
b.state.blockedDOFs   b.state.dict        b.state.isCavity     b.state.orl  
b.state.boundaryId   b.state.dispHlerarchy b.state.isDamped     b.state.pos
```

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

[//yade-dem.org/doc/yade.export.html#module-yade.export](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()`:

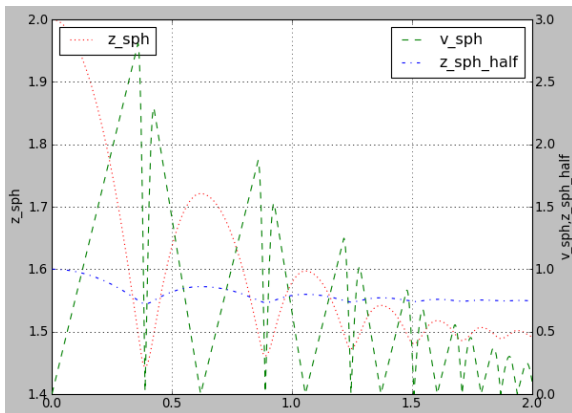
```
def history():
    plot.addData(t = O.time, i = O.iter, pos_x =
        ↪ O.bodies[10].pos[0])
    plot.saveDataTxt('data_collect.txt',
        ↪ vars=('t', 'i', 'pos_x'))
O.engines=O.engines+[PyRunner(iterPeriod=500,
    ↪ command='history()', label='recorder')]
from yade import plot
plot.plots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

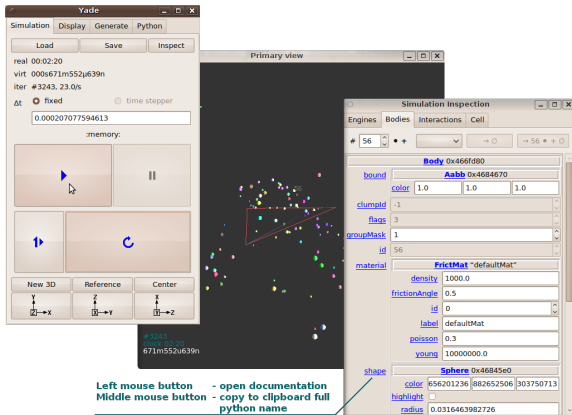
Plotting scene data

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:

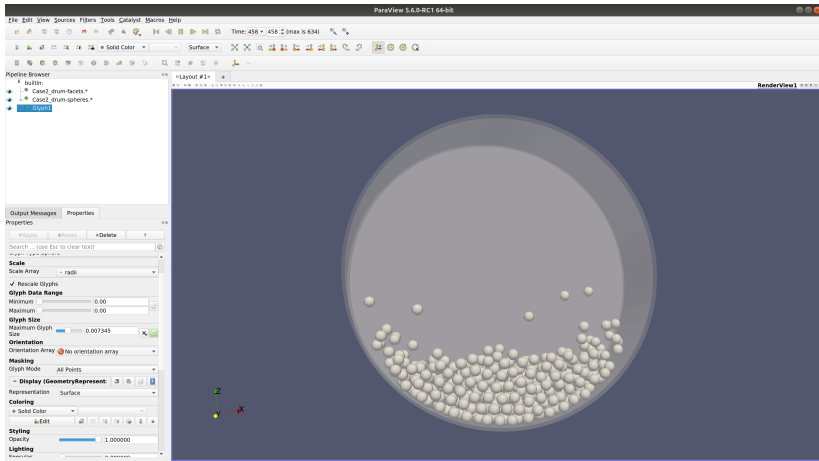
```
O.engines = [  
    ...  
    VTKRecorder(iterPeriod=1000,  
                fileName='uniqueId',  
                recorders=['spheres', 'facets'],  
                label='vtkrecorder')  
]
```

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!

Users can run their simulation (read start iterating on the 0.engines list) by running the command 0.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.

0.stop()

Users can stop their simulation (read start iterating on the `0.engines` list) by running the command `0.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.

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