Yade: Past, Present, Future

Václav Šmilauer

CTU Prague & UJF Grenoble

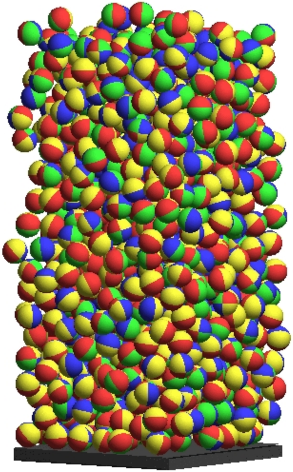
12 March 2010 (updated 3 January 2011)

marks advanced topics

contains clickable hyperlinks to **documentation** and **websites**

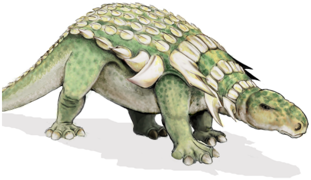
Outline

DEM & (Pre)history



- DEM: explicit dynamics of particles
- Simple discontinuum models
- Cundall 1979: nondeformable discs, 2d, explicit dynamics, penalty contact function
- Frédéric Donzé: Spherical Discrete Element Code
- Yade starts in 2004, “flexible platform” (J. Kozicki, O. Galizzi)

DEM & (Pre)history



- DEM: explicit dynamics of particles
- Simple discontinuum models
- Cundall 1979: nondeformable discs, 2d, explicit dynamics, penalty contact function
- Frédéric Donzé: Spherical Discrete Element Code
- Yade starts in 2004, “flexible platform” (J. Kozicki, O. Galizzi)

DEM & (Pre)history

- DEM: explicit dynamics of particles
- Simple discontinuum models
- Cundall 1979: nondeformable discs, 2d, explicit dynamics, penalty contact function
- Frédéric Donzé: Spherical Discrete Element Code
- Yade starts in 2004, “flexible platform” (J. Kozicki, O. Galizzi)

DEM & (Pre)history

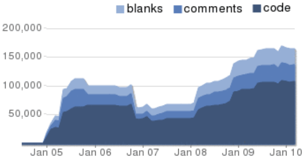
- DEM: explicit dynamics of particles
- Simple discontinuum models
- Cundall 1979: nondeformable discs, 2d, explicit dynamics, penalty contact function
- Frédéric Donzé: Spherical Discrete Element Code
- Yade starts in 2004, “flexible platform” (J. Kozicki, O. Galizzi)

Yade beginnings

www.yade-dem.org
launchpad.net/yade

- Written in c++, running on Linux/Unix
- Proof-of-concept implementations: DEM, FEM, mass-spring, lattice
- No documentation
- Sometimes functionally questionable
- Demanding on programming skills for "users"
- Object-oriented design

Sanitization period (2007-2010)



- Motivated by our development of concrete model
- Removing bad code
- Enforcement of consistent names
- Parallel computation
- Documentation
- Python scripting

Outline

Python

- Scripting (non-compiled) object-oriented language
- Large documented standard library
- Easy to interface with fortran/c/c++
- Language of choice for many scientific projects (similar to matlab)
- ☞

Python in Yade

- c++ classes mirrored in python, with full attribute access
- scripts efficient for simulation setup, postprocessing
- compatible over many internal changes
- runtime control & debugging from the command line

Data components

Body (particle)

Shape **Sphere**, **Facet**, ...

Material **ElastMat**, **FrictMat**, ...

State position, orientation, velocity, ...

Bound for approximate collision detection (**Aabb**)

Generalized forces

Interaction of 2 bodies

IGeom different for **Sphere+Sphere**, **Facet+Sphere**, ...

IPhys internal state of interaction (plasticity variables, damage, history)

Functional components

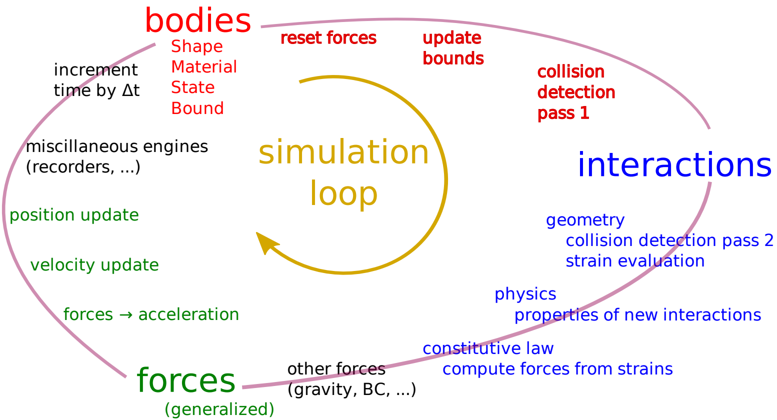
Engines

- GlobalEngine act on all bodies/interactions
- PartialEngine act on some bodies/interactions
- **Dispatchers** calls functions based on classes of arguments: e.g. **Facet+Sphere** needs different function than **Sphere+Sphere** collision

Functors

Callable function-like objects. Accept only certain classes and are called by **Dispatchers**.

Simulation structure



What it looks like in python I.

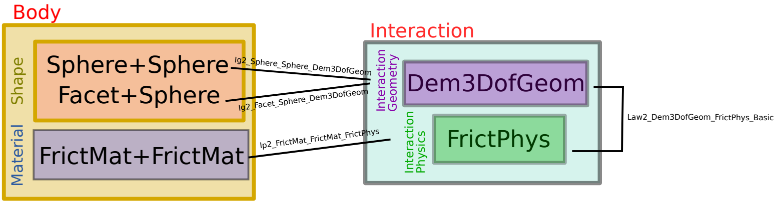
Simulation loop in code

```
O.engines=[
   ForceResetter(),
   InsertionSortCollider([Bo1_Sphere_Aabb(),Bo1_Facet_Aabb()]),
   InteractionLoop(
      [Ig2_Sphere_Sphere_L3Geom(), Ig2_Facet_Sphere_L3Geom()],
      [Ip2_FrictMat_FrictMat_FrictPhys()],
      [Law2_L3Geom_FrictPhys_ElPerfPl()],
   ),
   GravityEngine(gravity=(0,0,-9.81)),
   NewtonIntegrator()
]
```

Functor names explained

Ig2_Facet_Sphere_L3Geom

- Ig2 2-ary functor creating **IGeom**
- Facet accepting a **Facet** as first argument
- Sphere and **Sphere** as second argument
- L3Geom returning **L3Geom** instance



4 types of functors

BoundFunctor (Bo1)

approximate volume representation (hidden inside **Collider**)

IGeomFunctor (Ig2)

resolves geometry of interaction (e.g. displacement, shear), based on **Shapes** of bodies

IPhysFunctor (Ip2)

derives properties of interaction, i.e. creates **IPhys** for given particles' **Materials**

LawFunctor (Law2)

resolves forces on particles, using **IGeom** and **IPhys** of some types, created by previous functors.

What it looks like in python II.

Simulation data in code

```
O.materials.append(
   FrictMat(young=30e9,poisson=.3,density=3000,frictionAngle=.5)
)
O.bodies.append([
   utils.sphere((0,0,3),radius=1),
   utils.facet([(-1,-1,0),(1,0,0),(0,1,0)])
])
O.dt=.5*utils.PWaveTimeStep()
```

Running simulation

```
O.run(10000); O.wait()     # Basic simulation control
O.save('/tmp/a.yade.gz')
print O.bodies[3].state.vel     # inspection of (c++) data
print O.interactions[0,2].geom.normal
print O.materials[0].young
quit()
```

"Meshing" volumes with spheres

See horse (surface import), mill ("by hand")

Volume representation

- Boundary: triangulated surface; imported (STL, GTS, gmsh) / created "by hand" (possibly parametric)
- Volume: constructive solid geometry, boolean composition

Sphere packing generators (decoupled from volume)

- Import packing (text, LSMGenGeo)
- Dynamic: triax compression/decompression, gravity
- Geometric: from tetrahedron mesh (SpherePadder), from boundary specification (LSMGenGeo)

"Meshing" volumes with spheres (2)

Solid representation

predicate=**pack.inSphere**((0,0,0),1)

Boundary representation

predicate=**pack.inGtsSurface**(gts.read(open('horse.coarse.gts')))

Boolean composition (intersection &, union |, difference -)

predicate=**pack.inSphere**((0,0,0),1) & **pack.inCylinder**((.5,0,-1),(.5,0,1),.5)

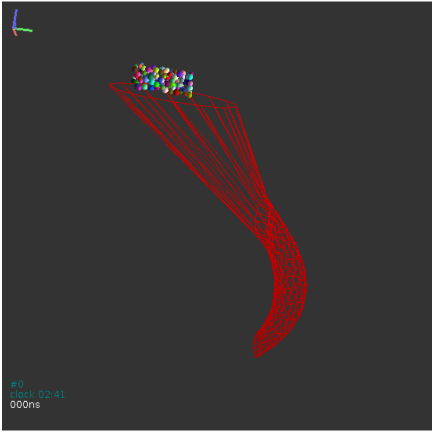
Call packing generator with arbitrary predicate

spheres=**pack.randomDensePack**(**pack.inHyperboloid**(
(0,0,-.1),(0,0,.1),.05,.085),spheresInCell=2000,radius=3.5e-3)

O.bodies.append(spheres)

Sphere falling through funnel

source script, movie



Running, controlling, collecting

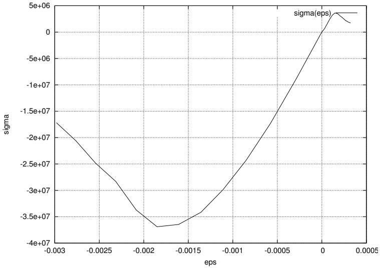
Collecting data

```
O.engines=[...,
   PyRunner(command='addPlotData()',iterPeriod=100),
   PyRunner(command='checkPostpeak()',realPeriod=3),
]
def addPlotData():
    plot.addData(eps=strainer.strain,sigma=strainer.avgStress)
plot.plots={'eps':('sigma',)}      # define what to plot
```

Controlling simulation from within the loop

```
def checkPostpeak():
   maxSigma=max(maxSigma,strainer.sigma)
   if strainer.sigma<.5*maxSigma:     # check some condition
      print "Damaged, exiting. Peak stress was", maxSigma
      plot.saveGnuplot('damaged')     # save curves for postprocessing
      import sys; sys.exit(0)
```

Postprocessing

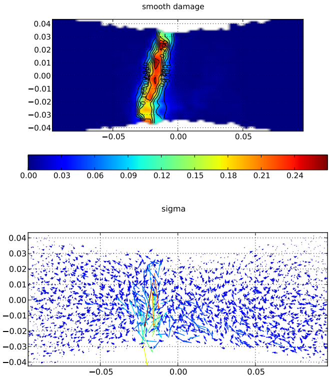


1d **yade.plot** module: **matplotlib**; **Gnuplot**

2d **yade.post2d** module

3d built-in OpenGL view; **VTKRecorder**, with **Paraview** (slices, movie export, . . .)

Postprocessing

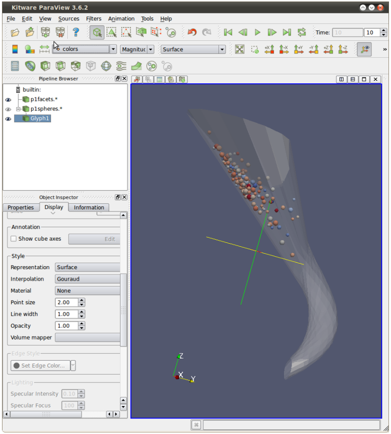


1d **yade.plot** module: **matplotlib**; **Gnuplot**

2d **yade.post2d** module

3d built-in OpenGL view; **VTKRecorder**, with **Paraview** (slices, movie export, . . .)

Postprocessing



- 1d **yade.plot** module: **matplotlib**; **Gnuplot**
- 2d **yade.post2d** module
- 3d built-in OpenGL view; **VTKRecorder**, with **Paraview** (slices, movie export, . . .)

Yade landscape

Community

- website, wiki, bugs and specifications tracking
- responsive mailing lists for users and developers, ≈ 10 messages/day
- used at multiple institutes, mostly research

Code

- central code repository with history
- documented code structure (in progress)
- documentation of c++/python classes and python modules
- Linux/Unix only

Generalities

Performance

- Shared-memory parallelism using OpenMP; speedup depending on scenario & machine, $\approx 5\times$ on 8 cores.
- Profiling tools (**yade.timing**)

Usability

- Batch scheduling and execution (parametric studies)
- Remote watching and control over http and telnet
- Debugging tools (**yade.log**), embedded debugger
- Embedded ipython shell

Engines

Loading control

TriaxialStressController (stress/strain rate), **PeriTriaxEngine** (periodic boundary conditions), **UniaxialStrainer** (strain control), **PeriIsoCompressor** (periodic iso-stress).

Applying conditions

GravityEngine (constant gravity field), **ForceEngine**, **RotationEngine**, **TranslationEngine**, ...

Algorithms

InsertionSortCollider (collision detection), **NewtonIntegrator** (2^{nd} order central-differences explicit integration scheme), **GlobalStiffnessTimeStepper** (adjust timestep based on packing stiffness)

Particles and interactions

Shapes

Sphere, **Facet**, **Wall**, **Box**. (**Tetra**, polyhedral grains, . . .).

Handling collisions (**IGeom**)

Handling collisions of 2×**Sphere**, **Facet+Sphere**, **Box+Sphere**, **Wall+Sphere**.

Contact laws

Dry friction (classical DEM), **Mindlin's contact**, **Plassiard's formulation**, **Cohesive-frictional** model, **rock** model, **concrete** model, **capillary effects** between grains. (more outside source tree or undocumented)

Coupling

OpenFOAM, Comsol, fluids.

What a contact law looks like


```cpp
void Law2_L3Geom_FrictPhys_ElPerfPl::go(shared_ptr<IGeom>& ig, shared_ptr<IPhys>& ip, Interaction* I){
	L3Geom& geom=ig->cast<L3Geom>(); FrictPhys& phys=ip->cast<FrictPhys>();

	// compute local force
	Vector3r F=geom.relU().cwise()*Vector3r(phys.kn,phys.ks,phys.ks);

	// break if necessary
	if (localF[0]>0) scene->interactions->requestErase(I->getId1(),I->getId2()); return;

	// plastic limit
	Real maxFs=-F[0]*phys->tangensOfFrictionAngle;
	Eigen::Map<Vector2r> Fs(&localF[1]);
	if (Fs.squaredNorm()>maxFs*maxFs){
		Real ratio=sqrt(maxFs*maxFs/Fs.squaredNorm());
		geom.u0+=(1-ratio)*Vector3r(0,geom.relU()[1],geom.relU()[2]); // increment plastic displacement
		Fs*=ratio;                                                     // decrement shear force value;
	}

	// apply force to particles (converts to global coords first)
	geom.applyLocalForce(F,I,scene,static_cast<FrictPhys*>(ip.get()));
}
```

Outline

Future

Continue maintenance

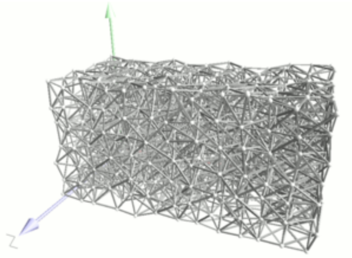
- Documentation
- Code cleanup
- Improve performance

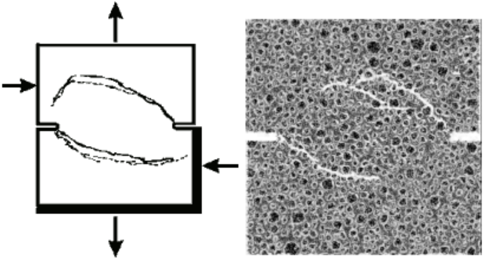
Becoming reference platform for discrete models

- Reusable common functionality (e.g. deformation computation, collision detection, integrator, . . .)
- Encourage cooperation via python (numpy).
- Integrate couplings with external software (OpenFOAM, . . .).

Outline

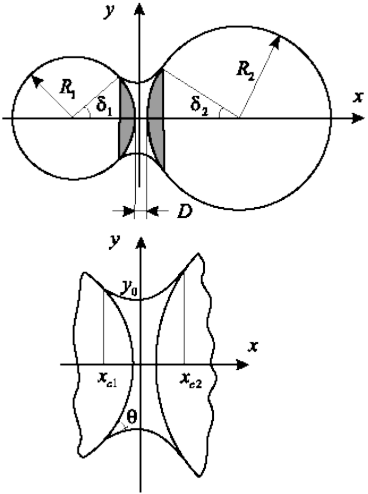
Jan Kozicki, Grenoble





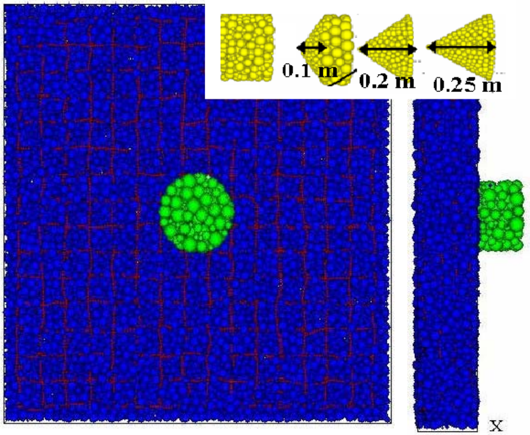
3d lattice model of tensile concrete fracture.

Luc Scholtès, Grenoble



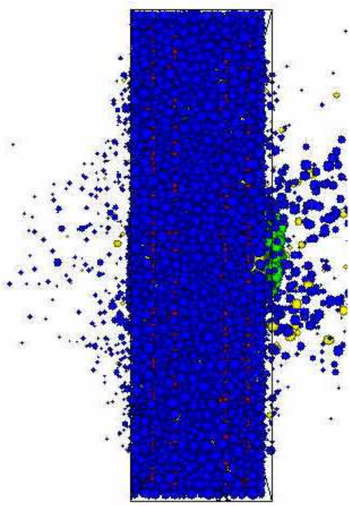
Behavior of granular media with capillary effects between grains.

Wenjie Shiu, Grenoble



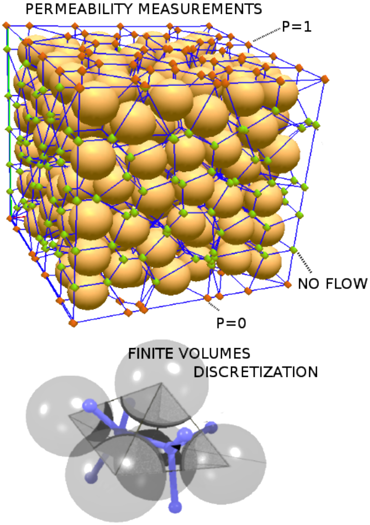
Missile impact on concrete structures.

Wenjie Shiu, Grenoble



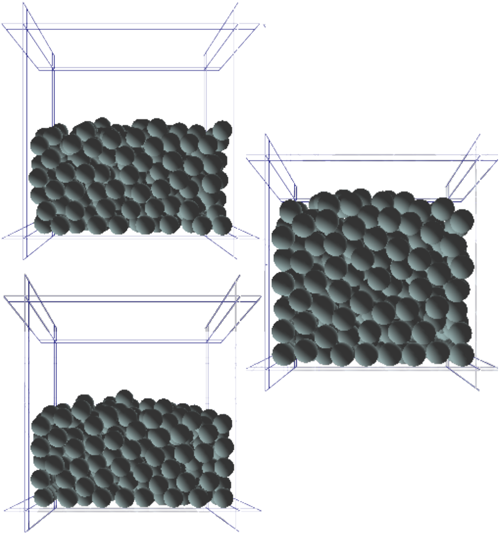
Missile impact on concrete structures.

Emanuele Catalano, Grenoble



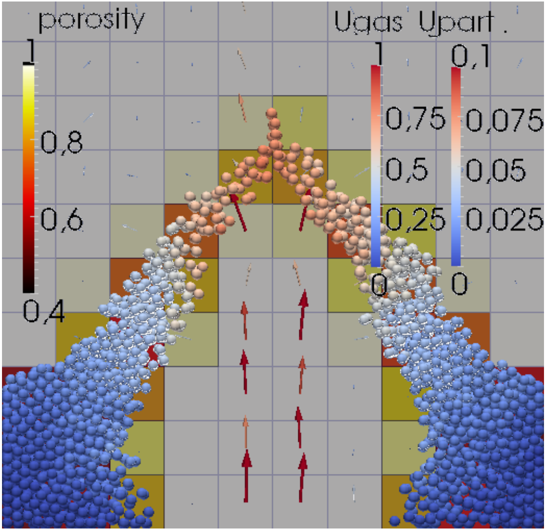
A finite volumes-DEM coupled formulation for fluid-solid interactions in granular media.

Benoît Charlas, Grenoble



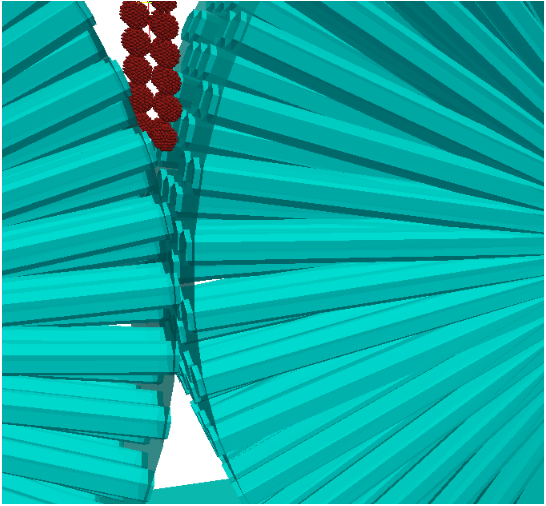
Hydride metal powders in hydrogen storage tanks — swelling & shrinking due to chemical reactions with hydrogen, creating mechanical effects.

Sergei Dorofeenko, Moscow



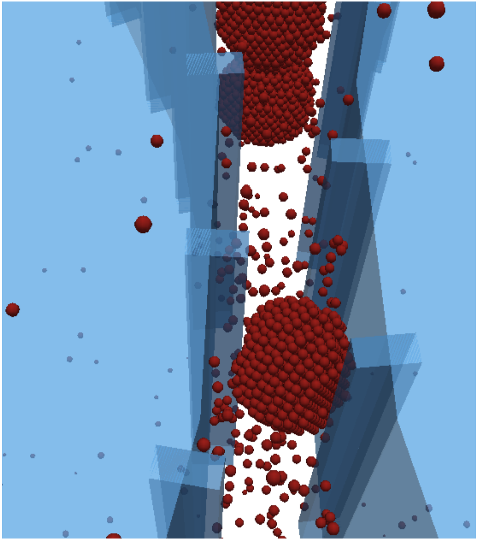
Coupling Computational Flow Dynamics (CFD) and DEM — OpenFOAM and Yade.

Anton Gladky, Freiberg



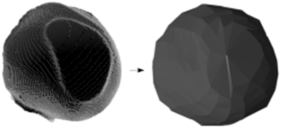
Mineral processing — analyzing rock destruction in the machine.

Anton Gladky, Freiberg



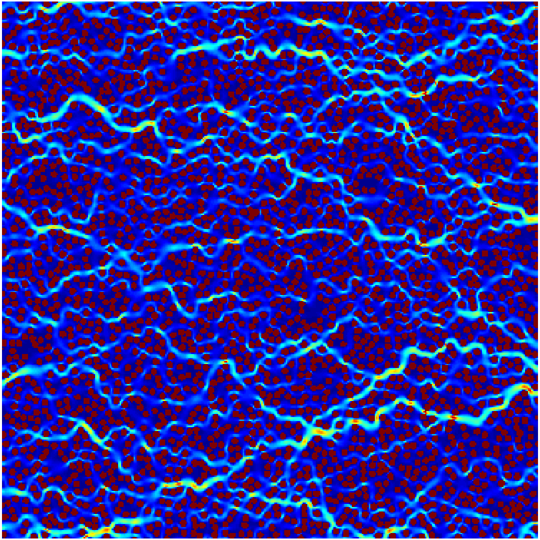
Mineral processing — analyzing rock destruction in the machine.

Alsidqi Hasan, Grenoble



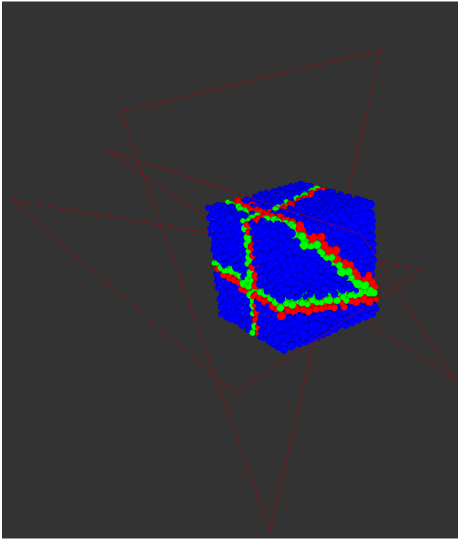
Modeling snow grains based on CT scans, as polyhedra which can deform along crystallographic planes.

Franck Lominé, Nantes



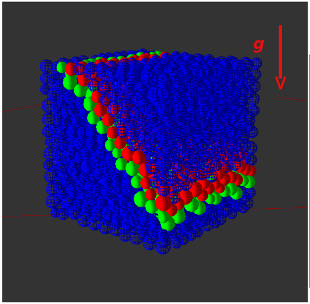
Interaction between DEM-modeled solid and Lattice Boltzmann Method (LBM) modeled fluid. (Started by Luc Scholtès)

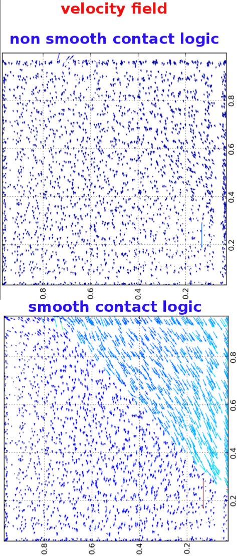
Luc Scholtès, Brisbane



Fractured rock mass with smooth contact discontinuities; discontinuities can be imported from Discrete Fracture Network Modelers.

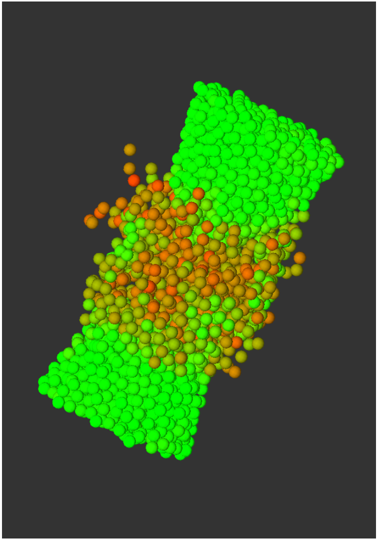
Luc Scholtès, Brisbane





Fractured rock mass with smooth contact discontinuities; discontinuities can be imported from Discrete Fracture Network Modelers.

Václav Šmilauer, Prague/Grenoble



Particle model of concrete, based on continuous formulation (plasticity, rate-dependence, damage).

Thanks for attention

Got questions

?

Ask them at
answers.launchpad.net/yade/