**DEM model for NPH simulation**

**Geometry**

I have started using a simplified geometry for the brain.

I have generated a sphere (10 cm radius) and an .stl file using Gmsh.

I converted the .stl file into a .gts file and then imported into my code as a predicate.

I believe it is possible to create a more realistic model (for example 2 concentric sphere, where the inner one represents the ventricles) or even import a patient specific mesh.

It is possible to add particles to the model knowing radius and position:

s=utils.sphere((0,0,0), radius=1)

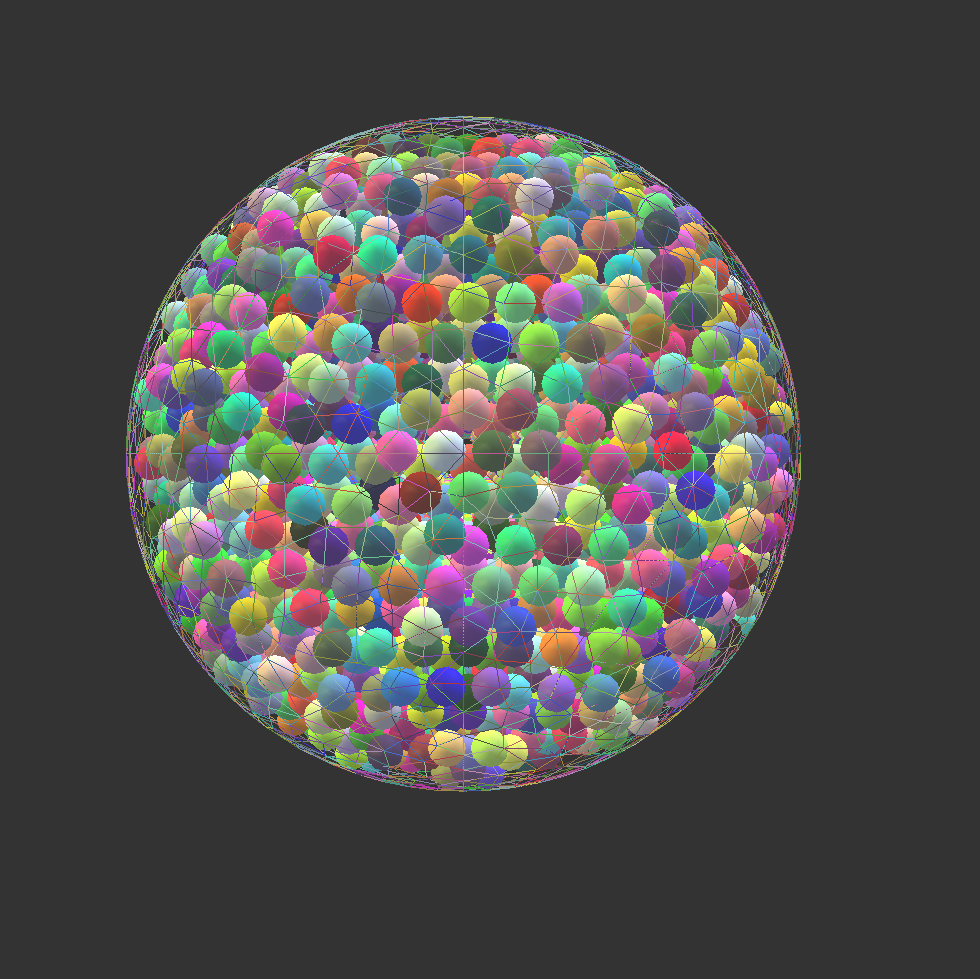
O.bodies.append(s)

**Packing**

The predicate has been filled with spheres using the *regularHexa* function that return set of spheres in regular hexagonal grid,.

The spheres have a radius of 0.50 cm and there is no gap between the spheres.

These values can be easily adjusted.

  
Illustration 1: Snapshot of the model.

**Material properties**

Unfortunately, there is not much agreement on the material properties of brain tissue. According to Li Et Al Influences of brain tissue poroelastic constants on intracranial pressure (ICP) during constant-rate infusion the Young modulus is around 10000 Pa. According to Tully and Ventikos Cerebral water transport using multiple-networkporoelastic theory: application to normalpressure hydrocephalus instead the Young modulus is closer to 500Pa.

As concerns the Poisson ratio this varies between 0.35 and 0.48.

It is possible to specify the material as well for each sphere:

O.bodies[id].material = ElastMat(),

and then to define the properties such that

O.bodies[id].material.young = 1000

O.bodies[id].material.poisson = 0.35

O.bodies[id].material.density = 1.0

…

There are several models implemented in Yade for the material. Depending on the material we choose, we can choose the best model for computing the interactions between spheres.

**Spheres’ properties**

It is possible to access each sphere’s properties using a foor loop:

for b in O.bodies:

if isinstance(b.shape, Sphere):

print "id = ", b.id

print "position = ", b.state.pos

print "radius = ", b.shape.radius

print "mass = ", b.state.mass

**Simulation loop**

From Yade’s manual:

*In a typical DEM simulation, the following sequence is run repeatedly:*

*• reset forces on bodies from previous step*

*• approximate collision detection (pass 1)*

*• detect exact collisions of bodies, update interactions as necessary*

*• solve interactions, applying forces on bodies*

*• apply other external conditions (gravity, for instance).*

*• change position of bodies based on forces, by integrating motion equations.*

*Each of these actions is represented by an Engine, functional element of simulation. The sequence of engines is called simulation loop.*

We can use this setup. Eventually also a FlowEngine can be incorporated.

O.engines=[

ForceResetter(),

InsertionSortCollider([Bo1\_Sphere\_Aabb(),Bo1\_Facet\_Aabb()],label='collider'),

InteractionLoop(

[Ig2\_Sphere\_Sphere\_ScGeom(),Ig2\_Facet\_Sphere\_ScGeom()],

[Ip2\_FrictMat\_FrictMat\_FrictPhys()],

[Law2\_ScGeom\_FrictPhys\_CundallStrack()],

),

NewtonIntegrator(damping=0.1,gravity=[0,0,0]),

# FlowEngine(label="flow"), commented at the moment

]

**ForceResetter():** itresets forces at each timestep

**InsertionSortCollider():** it manages collision between particles and facets. The functors Bo1\_Sphere\_Aabb() and Bo1\_Facet\_Aabb() are bound functors and define the Axis-aligned bounding boxes (Aabb) for spheres and for facets. At this stage we only have spheres and facets so it should be enough.

There is the aabbEnlargeFactor that should be passed to the Bo1\_ functor that is used to create distant interaction, but is only meaningful with an IGeomFunctor which will not simply discard such interactions: Ig2\_Sphere\_Sphere\_ScGeom::interactionDetectionFactor should have the same value as aabbEnlargeFactor.

At the moment I am not able to understand if this is something we actually need.

**InteractionLoop()**: in this loop the interaction between the spheres. It needs 3 functors: Ig2, Ip2, Law functors respectively.

The functor prefixed with Ig2 will handle collisions between different entities. For example, Ig2\_Sphere\_Sphere\_ScGeom() handles interaction for Sphere + Sphere, Ig2\_Facet\_Sphere\_ScGeom() handles the interaction between Facet and Sphere.

The Ip2 and Law2 functors need to be chosen carefully such that they are compatible.

There is a wide variety of possibilities.

Ip2 functor accepts the given combination of Material types between the particles and Law2 describes the constitutive law of the model.

**NewtonIntegrator**

It performs the integration of the equations. Damping is 0.2 by default.

**Fluid Engine**

It is possible to incorporate the fluid dynamics with the Flow engines.

It seems possible to impose the pressure in one point like this:

flow.imposePressure(Vector3((0.0,0.0,0.0)), 10.0)

Unfortunately, it is not straightforward to impose boundary conditions.