USER MANUAL

for

CAVIAR

Version 1.0

Prepared by M. Biagooi

IASBS September 3, 2019

Contents

1 Introduction

1.1 What is CAVIAR?

CAVIAR is a simulation software purposed to perform molecular dynamics (MD) of soft matter physics. Here we mention some of its simulation abilities.

- Simple MD with Lennard-Jones (LJ), Dissipative particle dynamics (DPD), Electrostatic.
- Parallel simulation using Massage Passing Interface (MPI) with domain decomposition.
- Ewald sum for electrostatics in 1D and 3D. Slab correction for 3D ewald sums (ELC).
- Molecules with soft bonds and angles.
- Molecules with hard bonds (Shake, M-shake and Rattle)
- NVE, Langevin NVT and some other ensembles.
- The ability to simulate electrostatic of charged particles near conductive boundaries. These boundaries are made of geometrical elements. This is one of the unique features of CAVIAR.
- Other features such as gravity, Hookean granular force which make the package a little more general.

This guide surely does not cover all of CAVIAR-1.0.0 abilities and power. We hope to make a detailed version of the user documentation available as soon as possible.

1.2 License

We have chosen GNU Lesser General Public License (LGPL) version 3.0 for the whole CAVIAR package. Anyone can use it according to this license which can be found in the main directory of CAVIAR, or at the URL below.
`https://www.gnu.org/licenses/lgpl-3.0.en.html`

1.3 Disclaimer

CAVIAR has been designed and developed meticulously. But we wont claim it to be error-less or bug-less. Before using any new simulation or situation, always try to simulate some standard tests that you know the answer to it. It is highly recommended that the user has enough knowledge of the algorithms that it is being used.

In addition to that, we have provided (and will provide more) some terminal warning outputs in cases the user doing something that may make unphysical simulation results if one didnt pay enough attention. Always check the terminal output of the program.

1.4 Authors, developers, contributors and history

CAVIAR is an acronym for Finite Elements for Charged Particle Package. CAVIAR package started and developed as a part of Morad Biagooi PhD project, advised by Ehsan Nedaaee Oskoee at Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan, Iran. It was the continuation of Mohammad Samanipour master thesis idea of simulating charged particles near conductive boundaries using a new algorithm. It continued to be developed as a MD code. The package core design has rewritten many times to take the current shape. The main authors and developers are Morad Biagooi and Ehsan Nedaaee Oskoee. We have to mention S. Alireza Ghasemi, assistant professor at IASBS as a contributor to CAVIAR project, specially about developing and handling electrostatic algorithms. The last contributor is Ashkan Shahmoradi, a physics student at IASBS, responsible for developing an interface for CAVIAR in other kinds of software, editing this user guide, and also the implementations of Shake, M-Shake and Rattle constraints algorithms.

1.5 How to cite CAVIAR?

Right now (in the process of editing this user guide) the introduction paper of CAVIAR-1.0.0 is in the process of submission. So here we cannot give you the exact reference. If you have published anything while using CAVIAR, please contact package authors and give us the information. We will add it to the package website.

2 Installing CAVIAR

2.1 System requirements

CAVIAR version 1.0.0 needs a g++ compiler that supports C++11 standard completely and a CMake with a version higher than 2.8.8. If one wants to have plt_dealii force_field as well, a pre-installed deal.II library is necessary. If you are not a deal.II user, please install it as we instruct in another subsection.

Running CAVIAR on parallel processors is also possible. This version has an interface with Massage Passing Interface (MPI). For using it, you have to install OpenMPI or MPICH on the platform.

CAVIAR does not design geometry and does not create mesh by itself. For that purpose, we recommend using SALOME-PLATFORM which is an open-source software. Using Salome needs a little practice and understanding of 3D design. We teach you how to make simple geometries and meshes in another part.

CAVIAR does not visualize your outputs. We recommend use Ovito which is available open-source in order to visualize xyz formats in addition to vtk geometries. To see the output of finite-element calculations which deal.II outputs as vtk format, ParaView is a good choice.

You can find more details about these kinds of software and libraries at the Bibliography section.

2.2 CAVIAR library and simulator

Installation of CAVIAR is completely easy and its like an ordinary program that uses the CMake and it can be done by a few commands. At first, if you have installed the requirements, go to the directory where you have extracted CAVIAR then create a build directory. Open a terminal and go to build directory with cd command:

```
cd ${BUILD_DIRECTORY}
```

where ${BUILD_DIRECTORY} is addressed to the directory named build. After that, you need to execute the following sequence of commands:

```
cmake ${PATH_TO_CAVIAR}
```

Where ${PATH_TO_CAVIAR} is the address of CAVIAR directory. If you made the build directory inside the CAVIAR directory for that address you just need put ../ and it will points to that directory.

If it goes smoothly, you may see these terminal outputs,

```
-- CAVIAR_DEBUG_VERSION : OFF
-- CMAKE_BUILD_TYPE : Release
-- CAVIAR_WITH_MPI : OFF
-- CAVIAR_SINGLE_MPI_MD_DOMAIN : OFF
-- CAVIAR_WITH_DEALII : OFF
-- CAVIAR_WITH_DEALII_MPI : OFF
```

This shows the CAVIAR current configurations. If you want to use plt_dealii force, you need to re-configue the code like this,

```
cmake ${PATH_TO_CAVIAR} -DCAVIAR_WITH_DEALII=ON
```

And if you want to configure in MPI,

```
cmake ${PATH_TO_CAVIAR} \
-DCAVIAR_WITH_DEALII=ON \
-DCAVIAR_SINGLE_MPI_MD_DOMAIN=ON \
-DCAVIAR_WITH_DEALII_MPI=ON
```

For a CAVIAR without any deal.II linkage and with MPI, execute,

```
cmake ${PATH_TO_CAVIAR} -DFCAVIAR_WITH_MPI=ON
```

This should have illustrated the way to configure the code. After it, just execute,

```
make
```

It may take a few minutes to completely generate and compile the package. After that, You should see a file with the name CAVIAR at the build directory. You can run it interactively. Also, to run a simple example, do it this way.

```
./CAVIAR < ${PATH_TO_CAVIAR}/examples/e1-two-atoms/e1-two-atoms.casl
```

You can see a o_xyz.xyz file which can be visualized with Ovito software.

2.3 CAVIAR Mesh Modifier

Mesh modifier configuration system is CMake. It does not need much requirements. Go inside the directory. Then just do it like how you compile and run the code:

```
cd build
cmake ..
make
./MeshModifier
```

2.4 CAVIAR post processing tools

We have provided some simple C++ codes for post processing outputs of force_field:: Plt_dealii class. In each directory, you can find a shell file named compile.sh which contains a simple compilation command using g++ compiler. After the execution, an executable binary is available inside the directory. Run it according to the related documentation.

2.5 Deal.II library

One of the requirements of CAVIAR for simulating charged particles near conductive boundaries is deal.II library which is a C++ library for finite element calculations. We have used deal.II version 8.5.1 to develop CAVIAR 1.0.0. We are not sure if any higher or lower version is compatible with our code. Deal.II can be compiled with an interface with many libraries which CAVIAR-1.0.0 does not need. What CAVIAR may uses, in addition to deal.II basic solvers, is MPI solvers. In that case, having Trilinos library, and/or PETSc library installed alongside deal.II would complete CAVIAR-1.0.0 requirements. To install deal.II, first download dealii-8.5.1.tar.gz, open a terminal and go to the directory you extracted deal.II. Then follow this shell commands:

```
tar -xvzf dealii-8.5.1.tar.gz
mkdir build
cd build
cmake \
-DCMAKE_INSTALL_PREFIX=/opt/deal.II_release/  \
-DCMAKE_BUILD_TYPE=Release \
-DDEAL_II_WITH_MPI=ON \
-DDEAL_II_WITH_PETSC=ON \
-DDEAL_II_PETSC_WITH_COMPLEX=OFF \
-DDEAL_II_WITH_TRILINOS=ON -DTRILINOS_DIR=/opt/trilinos \
../dealii-8.5.1/
```

This configuration assumes that you have installed Trilinos and PETSc beforehand. These are libraries that allow you to use plt_dealii_mpi which is MPI version of the force_field. If you want to run the code in one process, you can turn them off or dont mention them at all in the CMake command. The correct way to install PETSc or Trilinos is described in deal.II documentation.

3 CAVIAR package structure

3.1 What is inside CAVIAR Directory?

Here we talk about what one sees in the CAVIAR directoy.

- cmake: Some CMake script files to configure and build the package.
- doc: Documentations of CAVIAR. It is not much at this version.
- examples: There are some examples to learn and reference to. It is recommended to start your new scripts by changing and developing this examples.
- include: C++ header files. We will talk about them in another guide.
- scripts: Some simple shell scripts used for developing and also some for the users.
- src: C++ source files.
- tests: Some tests for the package. They are not complicated at this version.
- tools: These are some preprocessing and post-processing tools. We will talk about them.
- CMakeLists.txt: Main CMake script file. It will call all other CMake files that are exist in the package.
- Other files: Some simple text files about the package.

3.2 Interpreter and objects

If you open src/caviar or include/caviar directory, you will find out that it is divided into interpreter and objects. Interpreter contains all the classes responsible for reading scripts, handling objects, error reporting, creating and calling CAVIAR objects. The users and developers usually do not even need to open the interpreter directory.

Objects are the physical and mathematical stuff of CAVIAR. If you open the directory, you will see that it is divided into atom_data, constraint, domain, force_field, integrator, md_simulator, neighborlist, shape, unique and writer. These are the base objects. Each one of them consists of one basic object or different types of that object. Open force_field directoy to see what is inside.

We will talk about how to develop an object in developer guide in the next versions.

3.3 CAVIAR scripting language

CAVIAR scripting language is simple and easy. It is one command per line. The comments start with a hash #. For CAVIAR interpreter, variables must be suitably declared. For integer numbers, we use int keyword, but for a 2-dimensional and 3-dimensional array of integers, we use int2d and int3d. For floating point kind of numbers we use real. real2d and real3d for 2-dimensional and 3-dimensional real number arrays. The other types are bool for boolean variables and string for strings and characters.

The interpreter supports if conditions and also do while(condition) loops.

```
real x = 5.5
if x < 6.0
  echo x  is less than 6.0
else
  echo x  is higher than 6.0
endif
do x < 11
  echo this is a loop. x is  x
  x = x + 1.0
enddo
```

Two commands print and echo do the same thing and print on output stream.

An example of a do loop is given in bellow:

```
i=1
do i<5
  echo "i:␣" i
  i = i+1
enddo
```

The scope starts with do keyword and ends with enddo.

There is a list of keywords and their functions below.
- Import: import and run scripts from a file.
- Continue: skip current iteration of a loop.
- Break: stop a loop.
- Read: get a number from the user.

If one wants to create a Lennard-Jones force_field, this command would suffice,

```
force_field lj my_force
```

in which my_force is the name of the force_field.

Changing a variable, for examples cutoff value is done like this,

```
my_force cutoff 1.67
```

3.4 Tools

Inside the tools, you see these directories:

- differential capacitance: A code to calculate differential capacitance according to its formula. It imports the induced charges outputs and also the area of boundaries outputs.
- mean induced charges: It calculates the mean of induced charges on the electrodes and also converts the simulation units to the units that the user wants if the conversion coefficient is available. It imports the induced charges outputs and also the area of boundaries outputs. The outputs can be total charge or the charge density using the boundaries area.
- mesh modifier: After creating a mesh, one may need to remove extra internal faces, add some boundary ids or change a tetrahedral mesh to a hexahedral one. It all can be done using this code. It currently only support UNV format. We create mesh using SALOME-PLATFORM which is an available open-source software.
- number of ions in area: If one wants to calculates number of some ions or atoms inside an area, using this code can help. It imports and post-processes the xyz file formats.
- povray scripts: These are some shell files that have been used in the early stages of CAVIAR developments. It helps to produce some simple POVRAY outputs.
- syntax highlighting: Using a new computer language without syntax highlighting is not easy. We have made some for a few text editors. We hope to see more from other users.
- radial distribution: It does post-processing on the output files with xyz formats. It calculates radial distribution function for the atom types that user selects in a specific box area.

These tools have some extra comments and documentations inside. See them and we sure you will find the correct way to use them.

4 CAVIAR in use

4.1 Two atoms with Lennard-Jones Potential

This section explains how to run a simulation with CAVIAR using scripts. CAVIAR uses a specific format of the file for scripts and this format is a text file with casl extension. We tried to make a scripting language with a simple syntax so that the user has to remember as less as possible. At first, we need to learn some of the commands of this language. The syntax is one command per line. Comments start with a # until end of the line. One of the first things we have to do is adding some atoms to our simulation. For adding one atom we use atom that is one of unique class of the CAVIAR. For adding atom a1 we write:

```
unique atom a1
```

Other unique subclasses are Atom_group, Atom_list, Distribution, Element, Grid_1d, Molecule, Molecule_group, Molecule_list, and Random_1d.

Every atom in CAVIAR have some features that need to be specified e.g. type and position of the atom that you can specify them by commands in below:

```
unique atom a1
a1 type 0
a1 position -1 0 0
```

Here a1 is the name of the atom. Every simulation also needs some features such as the domain, boundary condition, force field, integrator and etc. In the domain class, you can choose the geometry and size of the simulation box. In the code below you can see one example of that:

```
domain box dom
dom xmin -50 xmax 50
dom ymin -50 ymax 50
dom zmin -50 zmax 50
```

You can also choose the boundary condition using the domain class e.g. if you want periodic boundary condition you must give 3 numbers each one is in order for x-, y- and z-direction and each number is 0 or 1 that 0 is for not having periodic boundary condition and 1 is for having periodic boundary condition in that direction. We can see an example in the code below:

```
dom boundary_condition 1 0 0
```

This boundary condition only in the x-direction is periodic. After all, we must generate the box we made via generate command in order to use it in our simulation. Another class we have is atom_data that we must make an object of its type. This object will have all the data of atoms, box, charges, masses and etc. For example, we define an object named adata with the type of atom_data by the following command:atom_data basic adata basic is one of the subclasses of atom_data. For adding adata to our simulation, adata needs a simulation box. This can be done by adding dom (we made earlier) to adata by the following command:

```
adata set_domain dom
```

An example of atom_data object is given in the bellow:

```
atom_data basic adata
adata ghost_cutoff 5
adata cutoff_extra 0.01
adata set_domain dom
adata add_atom a1
adata add_atom a2
adata add_type_mass 0 1.0
adata add_type_charge 0 0.0
```

neighbor list is one of the features that you can add to your simulation and have some subclasses such as verlet-list and cell-list and we can define verlet-list like this:

```
neighbor list verlet_list neigh_verlet
```

and adding adata to neigh_verlet:

```
neigh_verlet set_atom_data adata
```

adding cutoff and time step to our verlet-list:

```
neigh_verlet cutoff 15
neigh_verlet dt 0.001
```

in the code below we have shown an example of cell-list:

```
neighbor list cell_list
neigh_cell set_atom_data adata
neigh_cell cutoff 15
neigh_cell set_domain dom
neigh_cell make_neighlist
neigh_cell cutoff_neighlist 10
```

Every Molecular Dynamics (MD) simulation needs a force field that makes the atoms move and interact with other atoms. For example, if we want to define a Lennard-Jones force field that is the most famous MD force field we define like this:

```
force_field lj f_lj
```

The features we defined former can be added to f_lj like the code below

```
f_lj cutoff 10.0
f_lj epsilon 0 0 1.0
f_lj sigma 0 0 1.0
f_lj set_neighborlist neigh_verlet
f_lj set_atom_data adata
```

After the force field, the integrator is also necessary for simulation. By the class integrator we can define an integrator object. velocity_verlet is a child-class of class integrator and we can use that like this:

```
integrator velocity_verlet integ2
```

Like before we add atom_data to this integrator and specify a time step like below:

```
integ2 set_atom_data adata
integ2 dt 0.001
```

CAVIAR can give you an output file that can be visualized by any visualization application. By the class writer, you can define an object that makes an output file. The output file can be customized by a few features that can be added to this object. An example is given in below:

```
writer atom_data w1
w1 set_atom_data adata
w1 xyz_step 200
```

At the end of every script, it is needed to specify a simulation to be done by calling the script. We had to add every object to the simulation to be done as we want it. In the bellow we define a simulation object using md_simulation class and name it sim and add features we specified to this simulation. You can see the example bellow:

```
md_simulator basic sim
sim set_integrator integ2
sim set_atom_data adata
sim add_force_field f_lj
sim add_neighborlist neigh_verlet
sim add_writer w1
sim initial_step 0
sim final_step 20000
sim timestep 0.001
sim run
```

At the end of the script, we type exit to show the end of the script, however it is an optional command.

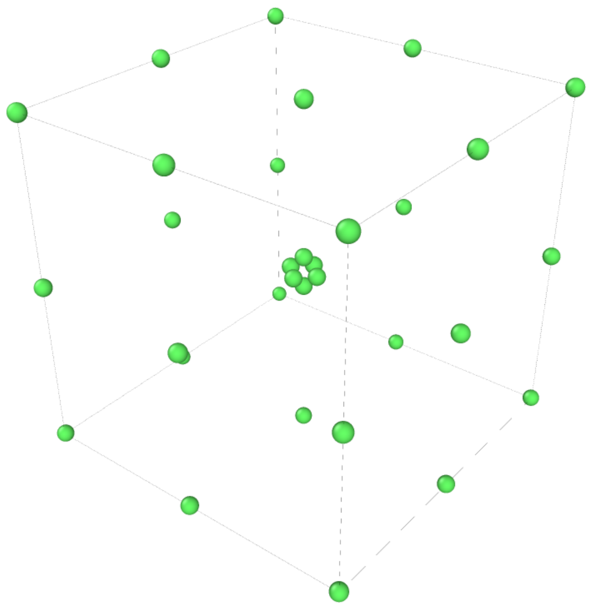


Figure 4.1: Example e2, a periodic test.

4.2 Periodic boundary condition

The second example were meant to be a test of periodic boundary condition in serial and parallel runs, but we thought it is a good way to show how to set the boundary conditions. The system looks like Fig??. If you look at the scripts of e2-periodic.casl, you can find the lines about the domain object creation,

```
domain box dom
dom xmin -11 xmax 11
dom ymin -11 ymax 11
dom zmin -11 zmax 11
dom boundary_condition 1 1 1
dom generate
```

The line boundary_condition 1 1 1 means that the domain is periodic in three dimensions. If you want the system to be periodic in x- and z-direction, just set it to boundary_condition 1 0 1 and for no periodicity it would be boundary_condition 0 0 0. You can play with the parameters and objects to get used to the CAVIAR scripting language.

4.3 Water simulation

CAVIAR contains constraint algorithms such as RATTLE, SHAKE and M-SHAKE. We provided example e3 which you can see how the constraint are inputted. It simulates

SPC/E water model (Fig4.2) with NVE ensemble.

CAVIAR has different types of integrator. The user has to set the correct integrator according to constraint type.

The atoms and the water molecule with bonds are defined like bellow,

```
#===== Atom definition =====
unique atom a_H1
a_H1 type 1
a_H1 position -1 0 0

unique atom a_H2
a_H2 type 1
a_H2 position 1 0 0

unique atom a_O
a_O type 0
a_O position 0 1 0

#===== Molecule definition =====
unique molecule m_water
m_water add_atom a_H1 # 0
m_water add_atom a_H2 # 1
m_water add_atom a_O  # 2
m_water position 0 0 0
m_water atomic_bond 0 1 0 1.633
m_water atomic_bond 1 2 0 1.0
m_water atomic_bond 2 0 0 1.0
```

The only thing that needs to be illustrated is the command atomic_bond . The first two parameters are atom indices according to the addition order to the molecule. The third parameter is the bond type and the forth one is the atomic bond distance.

4.4 A Polymer in Langevin

Polymers can be simulated in CAVIAR. The force_fields Spring_bond and Spring_angle is designed for that purpose. Examples e4 simulate different polymers in a system with Langevin fluctuations. The scripts for force_fields are,

```
#===== force_field
force_field spring_bond f_sb
f_sb set_atom_data adata
f_sb set_domain dom
f_sb elastic_coef 0 100.0
f_sb dissip_coef 0 1.0
#===== force_field
```

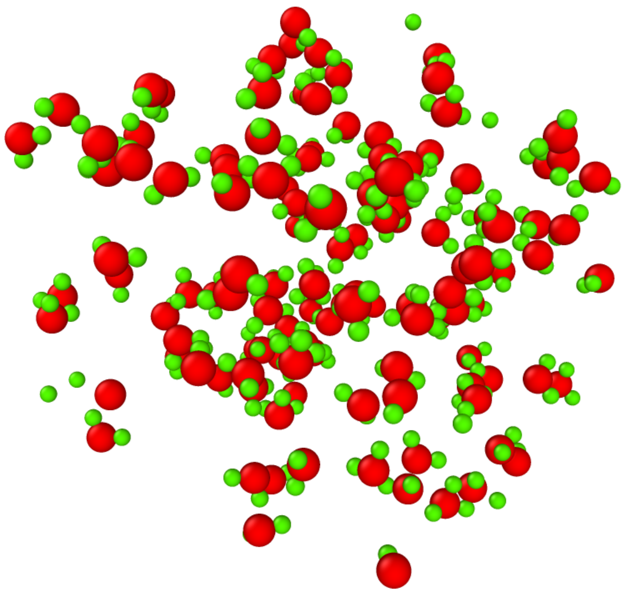


Figure 4.2: Example e3, SPC/E water simulation. The system is periodic in 3 dimensions. If you see some stray atoms because it split across the boundaries.

```
force_field spring_angle f_sa
f_sa set_atom_data adata
f_sa set_domain dom
f_sa elastic_coef 0 50.0
f_sa dissip_coef 0 1.0
```

The zero before elastic and dissipative coefficient values are the type of bond. These force_fields can have unlimited types of elastic and dissipative coefficient. The types are set in the molecules bond definitions.

4.5 Charged particles simulation

In this section, we talk about charged particle simulations near conductive boundaries using Plt_dealii force_field. If you have configured CAVIAR with deal.II linkage, you should be able to run example e5. In order to do that, open a terminal in examples/e5-charged-particle then call CAVIAR executable and direct the script file to it. Visualization of o_xyz.xyz file and addition of vtk files with proper colors should give you the results like Fig4.4. Note that prior to add a VTK file in Ovito, you have to delete some parts of VTK format (vertices and lines). Some older versions of Ovito may need different VTK standards.

One new thing in the script that this simulation have is shape addition.

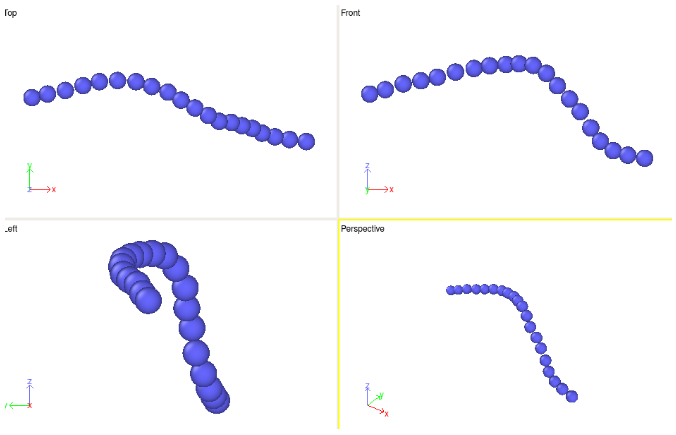


Figure 4.3: Example e4-polymer-big.casl, a polymer simulation using bond spring force field.

```
shape polyhedron s_full
s_full vtk_file_name "vtk-geometry/Fuse_1.vtk"
s_full parameter thickness 10.0
s_full parameter correct_normals
s_full an_inside_point 19.0 7.5 2.5
s_full point_is_inside_method 2
s_full generate
```

What you need to know is that an_inside_point takes a points coordinates which is inside your simulation box, and it is near to the center of a big face of your shape. CAVIAR uses this point to deduce the correct orientation of face normal.

Another one is Electrostatic_ewald1d and Plt_dealii force_field. We describe them line by line.

```
#===== force_field
force_field  electrostatic_ewald1d  f_ee
f_ee set_neighborlist neigh_cell
f_ee set_atom_data adata
f_ee set_domain dom
f_ee sigma 1.0 # a parameter for 1D Ewald
f_ee num_mirrors 3
```

For technical reasons, we only can use Cell_list in this example. The parameter k_electrostatic is the coefficient of electrostatic forces in the simulation units.

```
f_ee k_electrostatic k_elec_all
#===== finite element Mesh import
force_field plt_dealii f_ic
f_ic k_electrostatic k_elec_all
f_ic set_atom_data adata
```

This force_field needs a pointer to the electrostatic force_fields that should act on the particles.

```
f_ic add_force_field f_ee
f_ic add_unv_mesh "mesh-fixed/Mesh_1_fixed.unv"
f_ic read_unv_mesh
f_ic boundary_id_value 2 0.0
f_ic boundary_id_value 5 ic_voltage
f_ic induced_charge_ignore_id 1
f_ic induced_charge_ignore_id 3
f_ic induced_charge_ignore_id 4
f_ic output_induced_charge 500
f_ic set_solve_type faster_adaptive
```

We can refine the imported mesh as much as we need. increasing refinement does not necessarily gives a better answer since we usually use a thermostat to fix system energy or temperature.

f_ic refine_global 1

Also, theres a geometric force_field that should be introduced. It takes a normal LJ force parameter with some shape objects.

```
#===== force_field
force_field geometry_lj f_gelj
f_gelj add_shape s_full_shell
f_gelj epsilon_atom 0 1.0
f_gelj epsilon_atom 1 1.0
f_gelj sigma_atom 0 1.0
f_gelj sigma_atom 1 1.0
f_gelj epsilon_wall 0 1.0
f_gelj sigma_wall   0 0.674  # carbon sigma = 3.37 A
```

This activates automatic cutoff calculations using sigma and epsilon to make a completely repulsive LJ force (Weeks-Chandler-Anderson). One can set the cutoff value explicitly.

```
f_gelj wca
f_gelj set_atom_data adata
```

It is also possible to have non-periodic simulations. It suffices to replace Electorstatic_-ewald1d with Electrostatic force_field.

```
#===== force_field
force_field  electrostatic_ewald1d f_el
f_el set_neighborlist neigh_cell
f_el set_atom_data adata
f_el set_domain dom
```

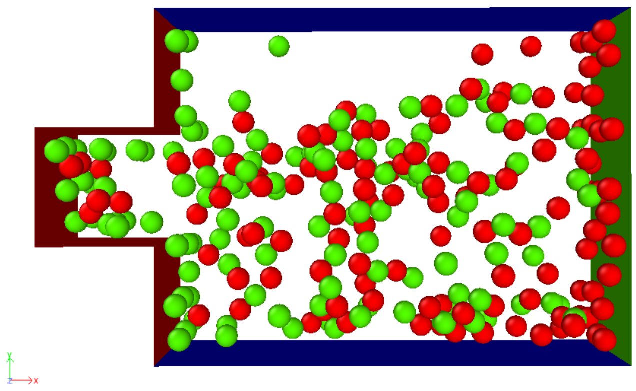


Figure 4.4: Simulation of charged particles between two polarized electrodes (red and green walls), the system is periodic in z direction. Blue walls are only acting with LJ force on the particles.

```
f_el k_electrostatic k_elec_all
```

CAVIAR supports 3D Ewald sums and ELC correction for it. We repeat that all the electrostatic force_fields that acts on the particle should have their pointer given to the Plt_dealii, if not, the answer would be wrong.

There's another example 'e6' that has a more complicated shape for charged particle simulations (Fig4.5). The simulation parameters are discussed in the CAVIAR first paper.

4.6 Granular Materials and Hopper

Example 'e7' contains the input files for a simple granular simulations. The result for the simulation is shown in the Fig4.6. The hopper is made by a geometric VTK file (see Fig4.7). The different objects from other examples is 'Geometry' and 'Granular' force-fields.

```
#===== force_field
force_field granular f_gr
f_gr cutoff 20 #
f_gr elastic_coef 0 10000
f_gr dissip_coef 0 1
f_gr elastic_coef 1 10000
f_gr dissip_coef 1 1
f_gr elastic_coef 2 10000
f_gr dissip_coef 2 1
f_gr gravity 0 0 -10.0
f_gr set_neighborlist neigh_verlet
```

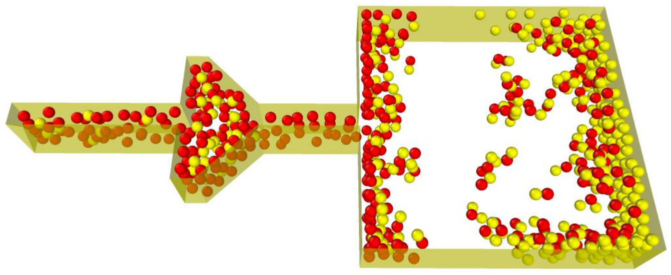


Figure 4.5: Simulation of charged particles between two polarized electrodes, the system is periodic in z direction.

```
f_gr set_atom_data adata
f_gr set_domain dom
```

```
#===== force_field
force_field geometry f_ge
f_ge add_shape s_hopper
f_ge add_shape s_box
f_ge dissip_coef 500.0
f_ge young_modulus 10000
f_ge set_atom_data adata
f_ge set_neighborlist neigh_verlet
```

This force-fields are not complicated. This is just an example and the parameters are not related to any realistic problem.

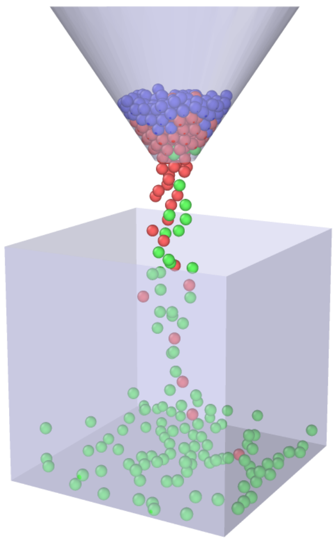


Figure 4.6: Granular materials falling through a hopper.

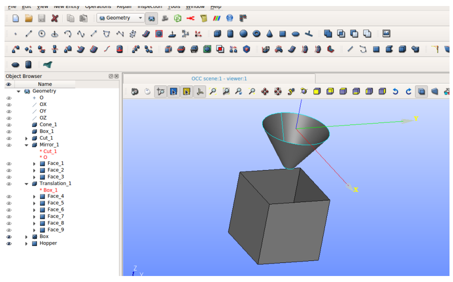


Figure 4.7: The hopper geometry is completely made in SALOME software. A brief tutorial is provided in the next section.

5 Geometry, Mesh and Fixing it

As an example, here we make a geometry for some simple shaped electrodes. Then we make a mesh out of it. Then we fix the mesh, and put boundary ids on it. We are not going to give a complete guide to SALOME, because it is a general software and we only need a small part of its abilities. But we talk as the user has never used SALOME before. For more clarification, use SALOME documentations.

5.1 1. A SALOME tutorial: Geometry module

At the first step, download and extract a version of SALOME-PLATFORM compatible with your operational system. Then open it by running salome file in its main directory.

On the toolbox on the top, click on Geometry icon to start a Geometry module of SALOME. It may ask you if you want to open a new file. Click on New. Now you should see something like the Fig5.1. We have cropped the screen-shots.

In summary, this is the plan: we make some points, then we some make some face of them. Then we make them 3D by using Extrusion.

Now click on the New Entry Rightarrow Basic Rightarrow Point. It opens a window for geometrical point creation. Set this (x,y,z) values and click Apply to the document: (0,0,0), (0,5,0), (-5,5,0), (-5,10,0), (0,10,0), (0,15,0), (20,15,0), (20,10,0), (20,5,0), (20,0,0).

If you change the 3D view using its tools, you could be able to see this view as Fig5.2. Play with the tools to get a hang of it.

Now click on New Entry Rightarrow Blocks Rightarrow Quadrangle Face. This tools default type, ask the users four points to create a quadrangle face. Click on the points in a counter-clockwise direction then Apply and Close. Now use this tool and create something like Fig5.3.

After this step, select all of the quadrangles and open extrusion tool by New Entry > Generation > Extrusion. By changing the view, you should see Fig5.4. Note that here we have chosen the third type of extrusion (choice boxes at the top of extrusion window) and the value for Dz is set to 5, then click on Apply and Close.

Now we have four 3D shapes. In order to make a geometry for geometric force of CAVIAR, we fuse them together. Select all of them and do Operation > Boolean > Fuse > Apply and Close. A Fuse_1 object is created. As you see the separator lines are removed and theres a solid object. Select it and File > Export > VTK select destination and filename, then save it.

Now hide Fuse_1 and show Extrusion_1, Extrusion_2, Extrusion_3 and Extrusion_-4 objects (You can do it using Object Browse). Select all of the extruded objects,

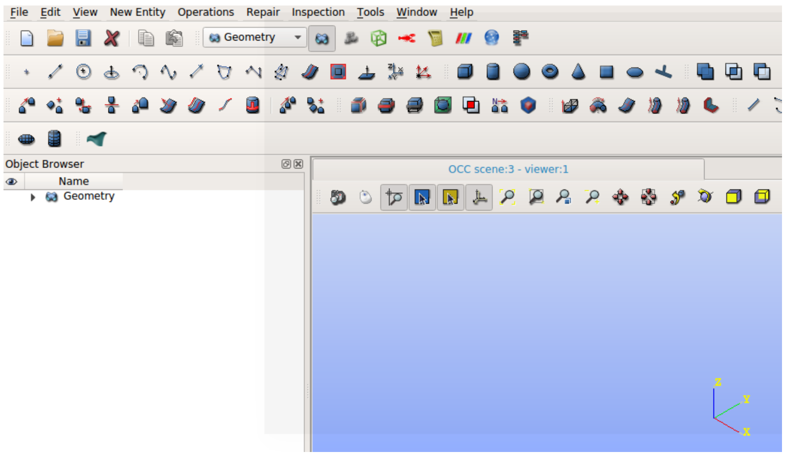

File Edit View New Entity Operations Repair Inspection Tools Window Help
Geometry
Object Browser
Name
Geometry
OCC scene:3 - viewer:1
Z
Y
X

Figure 5.1: SALOME Geometry view

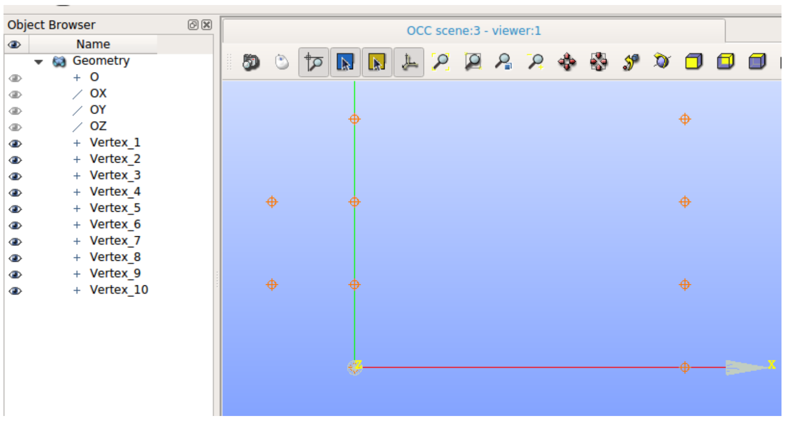


Figure 5.2: SALOME 3D view, after points creation.

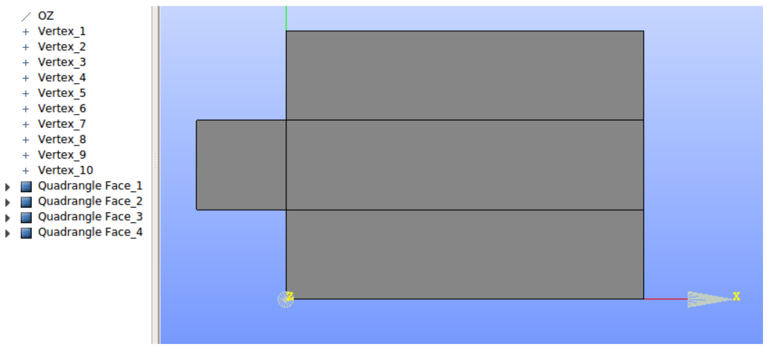
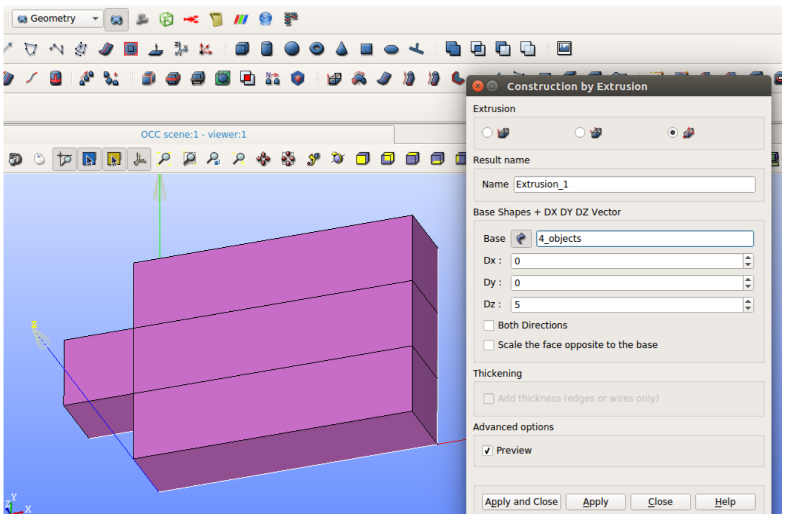


Figure 5.3: SALOME 3D view, quadrangles created of points.



Extrusion

Result name

Name: Extrusion_1

Base Shapes + DX DY DZ Vector

Base: 4_objects

Dx : 0

Dy : 0

Dz : 5

☐ Both Directions

☐ Scale the face opposite to the base

Thickening

☐ Add thickness (edges or wires only)

Advanced options

☑ Preview

Apply and Close | Apply | Close | Help

Figure 5.4: SALOME 3D view, extrusion of quadrangles.

then New Entry > Build > Compound > Apply and Close. This new object has the separators. They are necessary for making a good mesh.

If one needs to simulate the system in periodic condition, the system walls should be different from Fuse_1 object, since it is closed in every direction. For example, we want to have the problem periodic in z direction, we change the object like this: Select Fuse_1 then New Entry > Explode. Choose Face for Sub-shapes Type. Then Apply and Close. Now every face of Fuse_1 is individually selectable. Hide the upper and lower caps and select the rest. Then Fuse them all and create Fuse_2 object, it should be something like Fig5.5. Then export it as a VTK file.

5.2 SALOME tutorial: Mesh module

To make a mesh, click on the Mesh icon. You see everything disappears. Select Compound_1 in the Object Browser, Then Mesh > Create Mesh > Assign a set of hypotheses > 3D Automatic Hexhedralization. Set Number of Segments to 1 and ok > Apply and Close. Now theres an object named Mesh_1 in the Object Browser. Select it and Mesh > Compute. Now you can see the mesh in Fig5.6. Select Mesh_1 then File > Export > UNV file and save it.

This mesh file cannot be directly used in CAVIAR for two reasons. 1: Since the mesh is created from a compound object, there is extra faces between the mesh domains (not visible), 2: Theres no boundary ids, the id of all of the boundaries are set to zero, so the user cannot set potentials to them. Both of them can be fixed and done using SALOME, but it can be tedious for complicated meshes. Because of these reasons, we have developed a C++ library called CAVIAR Mesh Modifier which read UNV files and do the necessary modifications.

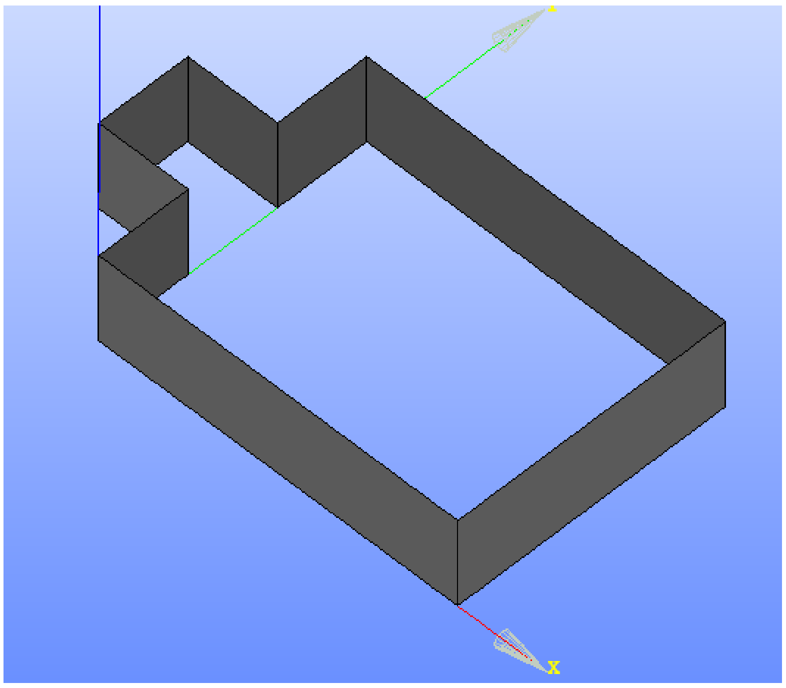


Figure 5.5: SALOME 3D view, Fuse_2 created from fusing some faces of Fuse_1.

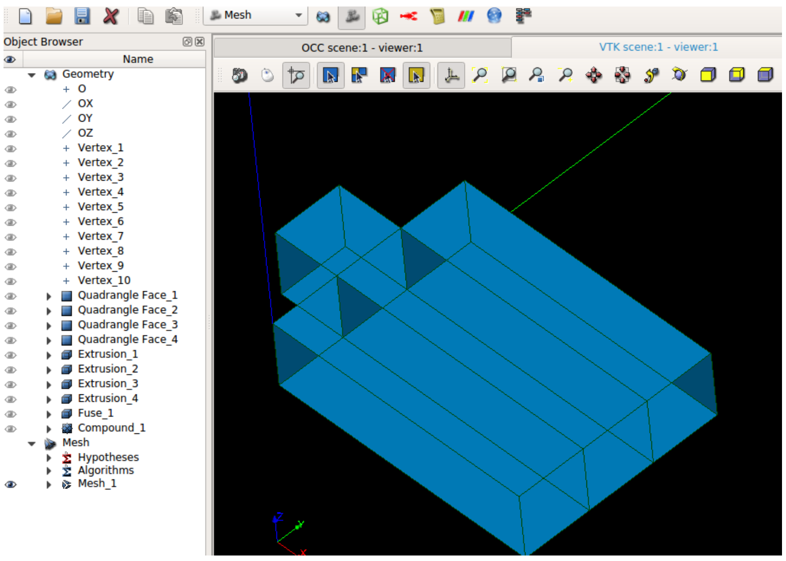


Figure 5.6: SALOME 3D mesh view.

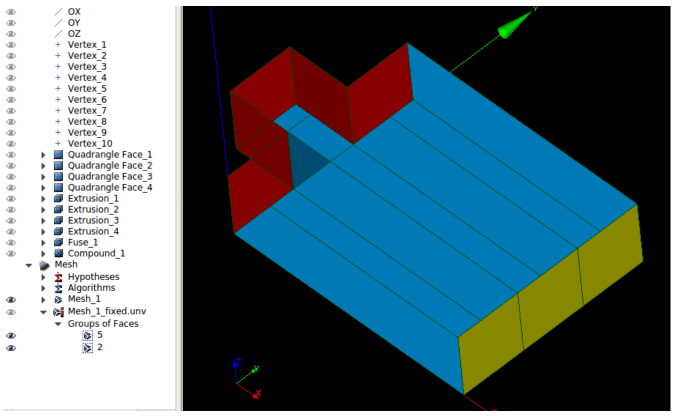


Figure 5.7: SALOME 3D mesh view, Mesh_1 fixed with group addition by using CAVIAR Mesh Modifier.

5.3 Using CAVIAR Mesh Modifier

To fix the UNV file that you have made in the previous section, open a terminal and go to CAVIAR Mesh Modifier (FMM) directory, then build it according to its documentation.

FMM version 1.0.0 does not have any interpreter. Its a library that needs a little C++ programing knowledge. It has the ability to import UNV files that SALOME produces, merge some meshes together, extract and export its boundary as a VTK or STL file, find and remove isolated and unnecessary points, find and remove internal faces that are inside the mesh, scale some part of the mesh, add group ids to the selected mesh boundaries, and change a tetrahedral mesh to a hexahedral one.

After you have built FMM, run the executable file in the terminal. It is fixed on the example e1 in the examples directory of FMM. It reads a mesh from mesh-raw and creates a fixed mesh file inside mesh-fixed . If you import the file,

${FMM_DIRECTORY}/examples/e1/mesh-fixed/Mesh_1_fixed.unv in SALOME Mesh module and play a little with the group colors in Mesh ¿ Edit Group, you can see the results in Fig5.7.

If you want to do any different thing, you need to modify main.cpp in FMM directory. Currently it contains,

```
#include ”mesh_modifier.h”
int main () {
  mesh_modifier::Mesh_modifier mesh_modifier;
  mesh_modifier.import(”../examples/e1/mesh-raw/Mesh_1.unv”,
     true);
  mesh_modifier.remove_hexa_internals (1e-8);
  std::vector<Point_condition> pc_e1, pc_e2;
  pc_e1.push_back( Point_condition(”x”,”<”,0.01) );
  pc_e1.push_back( Point_condition(”z”,”>”,0.01) );
  pc_e1.push_back( Point_condition(”z”,”<”,4.99) );
  pc_e2.push_back( Point_condition("x",">",19.99) );
  mesh_modifier.add_face_to_group_with_condition
    ("2", \&pc_e1);
  mesh_modifier.add_face_to_group_with_condition
    ("5", \&pc_e2);
  mesh_modifier.export_
    ("../examples/e1/mesh-fixed/Mesh_1_fixed.unv",
   false);
}
```

It also contains some extra commented lines which helps you if you want to do anything different from this example. Here we describe what does it all means.

First, we include the main library file mesh_modifier.h. Then in the main function, an object of type mesh_modifier::Mesh_modifier is created and called mesh_modifier. Then we import the UNV file. After that it removes extra internal faces with a tolerance that is not much related here. Then we create a std::vector of Point_condition. This class is used to select vertices that follow our conditions. It is obvious that ("x","<" , 0.01) means x values are less than 0.01. Then we put a numeric label group (here 2 and 5) for the boundaries that their central points follow our conditions. After all of it, we export the file.

At this stage the mesh file is ready to use.

6 Bibliography and references

- **CAVIAR**: Morad Biagooi, Mohammad Samanipour, S. Alireza Ghasemi, and Ehsan Nedaee Oskoee, CAVIAR: A simulation package for charged particles in environments surrounded by conductive boundaries, Under submission.

- **Deal.II library**: D. Arndt, W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, D. Wells, The Library, Version 8.5, Journal of Numerical Mathematics, vol. 25, pp. 137-146, 2017. https://www.dealii.org/

- **Salome-Platform**: A. Ribes and C. Caremoli, "Salom platform component model for numerical simulation," COMPSAC 07: Proceeding of the 31st Annual International Computer Software and Applications Conference, pages 553-564, Washington, DC, USA, 2007, IEEE Computer Society. https://www.salome-platform.org/

- **Paraview**: Ahrens, James, Geveci, Berk, Law, Charles, ParaView: An End-User Tool for Large Data Visualization, Visualization Handbook, Elsevier, 2005, ISBN-13: 978-0123875822 https://www.paraview.org/

- **Ovito**: A. Stukowski, Visualization and analysis of atomistic simulation data with OVITO - the Open Visualization Tool, Modelling Simul. Mater. Sci. Eng. 18 (2010), 015012 http://www.ovito.org/