EPJ Web of Conferences **108**, 02037 (2016)
DOI: 10.1051/epjconf/201610802037


# Visualization for Molecular Dynamics Simulation of Gas and Metal Surface Interaction

D. Puzyrkov[1,a], S. Polyakov[1,b], and V. Podryga[1,c]

[1]*Keldysh Institute of Applied Mathematics (Russian Academy of Sciences), Miusskaya sq., 4, Moscow, 125047, Russia*

**Abstract.** The development of methods, algorithms and applications for visualization of molecular dynamics simulation outputs is discussed. The visual analysis of the results of such calculations is a complex and actual problem especially in case of the large scale simulations. To solve this challenging task it is necessary to decide on: 1) what data parameters to render, 2) what type of visualization to choose, 3) what development tools to use. In the present work an attempt to answer these questions was made. For visualization it was offered to draw particles in the corresponding 3D coordinates and also their velocity vectors, trajectories and volume density in the form of isosurfaces or fog. We tested the way of post-processing and visualization based on the Python language with use of additional libraries. Also parallel software was developed that allows processing large volumes of data in the 3D regions of the examined system. This software gives the opportunity to achieve desired results that are obtained in parallel with the calculations, and at the end to collect discrete received frames into a video file. The software package "Enthought Mayavi2" was used as the tool for visualization. This visualization application gave us the opportunity to study the interaction of a gas with a metal surface and to closely observe the adsorption effect.

## 1 Introduction

The current stage of the computer technologies development and their computational potential makes it possible the calculation of the properties of complex systems at the molecular and even atomic levels. The mathematical models which describe such processes with the methods of molecular dynamics (MD) [1] can contain a huge amount of particles, up to several billion, and each particle can have dozens of properties. Such data volumes can amount to terabytes and they make it difficult to study the involved processes properly. It brings us over to the problem of calculation result representation in a convenient form for further analysis. As we know, MD calculations result in a set of investigated system states, which describes the system evolution over time. For large scale systems the drawing of all the particles is not informative (fig. 1a). In this case it is required to draw not all area of modeling but only the data region of interest (fig. 1b).

[a]e-mail: dpuzyrkov@gmail.com
[b]e-mail: sergepol@mail.ru
[c]e-mail: pvictoria@list.ru

In this work we examine the ability to use the interpreted programming language such as Python [3] for the calculation results analysis and their representation for 3D visualization. Besides the positions of the particles, this visualization should reflect different parameters, e.g., the temperature, the distribution of the pressure, the particle trajectories. All of that makes necessary the preprocessing of large data amounts. As a test gas-metal system we consider a microsystem [2] consisting of 8552352 particles, where the nitrogen was used as the gas part of the system and the nickel as the metal one.

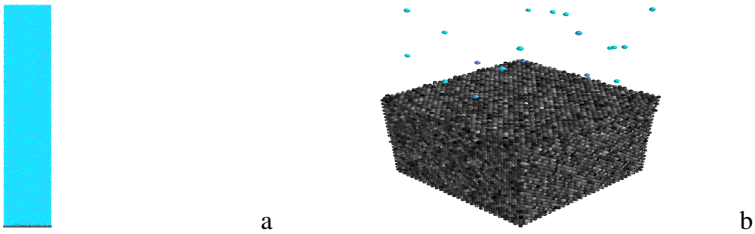


**Figure 1.** Nitrogen-Nickel microsystem in the beginning of simulation: entire system in one image (a) and a close up region $10 \times 10 \times 30$ nm$^3$ (b)

## 2 Technologies overview

There are several software packages for atomistic data visualization. For example: PAVMD, VMD, PyMOL, RasMol. A lot of packages for MD calculations (LAMMPS, GROMACS) have their own visualizers. But for using them in any non-standard case (e.g., to visualize a dataset of non-standard format) it is necessary to script them, or convert the data to a well-known format, which is not always possible. In this study we made an attempt to do fast and easy visualization and analysis and to offer the following tools and technologies.

Python [3] is a widely used general-purpose high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than it would be possible in languages such as C++ or Java. The syntax of the Python kernel is very simple and short, at the same time a standard library gives a large volume of useful functions and convenient data structures. Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management, full introspection, exceptions and multiprocessing. The developers community creates a lot of computer science libraries, that makes Python one of the most commonly used languages for big data analysis and scientific calculations.

NumPy [5], the open-source package for the Python language, is a free but strong alternative to the proprietary packages for calculations with matrices and vectors (such as MATLAB). Plenty of algorithms for data analysis are implemented at low level, a high quality documentation and detailed examples are provided. All of these advantages make NumPy an ideal choice for the acceleration of scientific applications in Python language. NumPy offers rich functionality which includes multidimensional arrays, import and export methods, mathematical functions, Fourier transform, linear algebra, etc.

Numba [6] is open-source project, which makes it possible to optimize Python+NumPy code. Numba is the Just-In-Time optimizing compiler using LLVM (Low Level Virtual Machine) infrastructure for the compilation of Python code into machine codes. For comparison: Intel Core i7, array multiplication, addition and division by constant (see the listing below).

**Table 1.** NumPy and Numba+Numpy performance comparison

| N | NumPy | Numba+NumPy | rel. performance gain |
|---|---|---|---|
| 1000000 | 0.19ms | 0.07ms | 2.77 |
| 10000000 | 1.62ms | 0.74ms | 2.19 |
| 100000000 | 16.06 | 7,4ms | 2.17 |

```
from numba import jit
@jit(nogil=True, nopython=True)
def numpy_numba_func(vx, vy, vz, multiplier=100, divider=3.0):
    return multiplier*((vx*vx) + (vy*vy) + (vz*vz)) / divider

def numpy_func(vx, vy, vz, multiplier=100, divider=3.0):
    return multiplier*((vx*vx) + (vy*vy) + (vz*vz)) / divider
```

The productivity increased about 2 times (see table 1). This result was achieved by applying just one decorator and without any other tricks. Of course, Cython [8] (optimizing static compiler for both Python programming language and the extended Cython programming language) will give more performance in this case, but then you should write not pure-python code, and probably think about its optimization.

ParallelPython (PP) [4] has a mechanism for parallelization of a Python code on the SMP systems and scientific clusters. It is a client-server software, based on the use of "multiprocessing" Python module, and it needs installation on the computational nodes. PP provides a simple application program interface for the execution of the tasks and getting the results when they are finished.

Mayavi2 [7] is an instrument for scientific data visualization. It uses Python for general scripting. Mayavi2 allows to deal with large volumes of data in memory, not screening, and then to render an image and save it to file, or show an interactive window. For the individual use of this package it is necessary to develop your own scripts for loading, processing, animating data and for camera control. For the graphics rendering Mayavi2 uses the powerful technology Visualization Toolkit (VTK), which is a data visualization library with open-source code, and it is widely used in the scientific community. Originally, Mayavi2 was developed as a visualization tool for computational hydrodynamics. After the benefit from its usage in other applications became intelligible, it was converted into a general scientific tool for data visualization.

## 3 Realization details

As an example of dataset, in this work we used the results of the calculations described in [2]. The equilibrium state of the nitrogen-nickel microsystem was calculated using the MD approach. The data obtained in the end of the simulation represents the microsystem states through time. It is a set of binary files that contains atomistic data, such as particles global indexes, coordinates, velocities, forces, and other properties.

For importing and analysing such structures, we have developed the "mmdlab" package. It provides modules for data reading, processing and common visualization routines.

### 3.1 mmdlab.datareader module

Atomistic data import procedure from the above described datasets in the "mmdlab" package is implemented in the "datareader" module. The purpose of this procedure is to parse the binary file and

to create special storage containers, according to the user-defined description of the particles stored. This method can operate in two different ways: fast reading process and operated reading process.

The fast reading process works on the dataset which fits in RAM. It is based on simultaneous loading of all particles and their properties in memory. A vectorized procedure for particle separation into storage containers is performed after the data was loaded and parsed. This process is fast because it makes minimum operations over a single particle in the sequential reading procedure.

The operated reading process takes more time because it sequentially applies a user-defined filter on every particle in the data (for example cutting off the particles which are out of the specified region or only particles with certain index property). Naturally, the speed of such a filter directly depends on its complexity, however, its advantage is that it doesn't load into RAM all the data and eventually it gives the storage container with particles which passed the filter.

Every data reading procedure returns the objects of "ParticleContainer" class, that contains properties of the particles assigned to this container by the filter procedure. After construction, this container automatically calculates temperature and the absolute velocities of every particle assigned to it. This object also contains a user-defined information, for example the atom diameter and its mass.

The Numba+NumPy code for calculation temperatures and velocities looks as simple and clear as a MATLAB code:

```
from numba import jit
import numpy
@jit
def absolute_velocity(vx, vy, vz, multiplier=1, kb=1.38065):
        return  numpy.sqrt(((vx*vx) + (vy*vy) + (vz*vz)))*multiplier;
@jit
def absolute_temp(vx, vy, vz, m, multiplier=100, kb=1.38065):
        return multiplier*m*((vx*vx) + (vy*vy) + (vz*vz)) / (3.0*kb)
```

### 3.2 mmdlab.dataprocessor module

In the "mmdlab" module we implemented some special methods for atomistic data processing. For example, it has a procedure for post-processing the data filtration by a user-defined filter, a filter for cutting off the particles not in the specified spatial region and a routine for merging two different containers in a new "ParticleContainer" object, with additional information about the position of every particle from the first container in the second. The last routine is used in the method asking for particle movement interpolation. Let's consider this feature closely in an application dealing with the particle movement animation task in a user-defined spatial region.

Since the particles can either move into the domain and move out of it, we do not know which of the particles can be in the domain at the next time step. So, first of all it is necessary to determine the particles we want to observe. There are two ways to do that:

- pre-process the whole dataset and obtain the particle indexes appearing at least once in a given domain;
- obtain indexes of the particles which were in the given domain at two consecutive time steps and discard all the particles that appear in the first dataset and do not appear in the second (and vice versa).

It is not possible to use the velocity vector of the particle for the determination of its location in the future steps because the time difference between system states can be quite large and collisions appear (affecting the velocity).

Let's consider the first option. The following algorithm is defined: 1) import a first dataset; 2) apply a cut-off filter to the containers (or use operated reading method for cut-off particles); 3) save an

array of global indices; 4) import the next dataset, repeat steps 2 & 3; 5) unite two received arrays; 6) remove duplicates; 7) repeat steps 4-6 for every dataset representing system states at different times; 8) import first and second datasets; 9) merge containers; 10) draw particles movement; 11) repeat steps 8–10 for all dataset pairs (1,2), (2,3), (3,4), ..., $(n-1, n)$.

Usage of the "mmdlab" package and NumPy + PP for this task is quite simple:

```
import pp
import numpy as np
from mmdlab import datareader, dataprocessor, utils
serv = pp.Server(ncpus = 4)
indexes = np.array([])
def get_indexes(f, region):
    c = mmdlab.datareader.fast_read(f)
    return mmdlab.dataprocessor.region_filter(c, region).indexes
region = [0, 10, 0, 10, 0, 30]
pp_imports = ("mmdlab","mmdlab.datareader","mmdlab.dataprocessor","mmdlab.utils")
jobs = [serv.submit(get_indexes,(f,region), (), pp_imports) for f in datafiles];
for job in jobs:
    indexes = np.unique(np.append(indexes, job()))
```

As a result we receive an array, containing all the particle indexes which appear in the specified region at least once. Then it is necessary to import every system state consecutively, filter them for particles the indexes of which are in the obtained array and merge two sequentially read containers in one containing the particle positions in the first and second states.

```
def get_container(f1,f2,indexes):
    c1 = mmdlab.dataprocessor.index_filter(mmdlab.datareader.fast_read(f1), indexes)
    c2 = mmdlab.dataprocessor.index_filter(mmdlab.datareader.fast_read(f2), indexes)
    return mmdlab.dataprocessor.merge(c1,c2)
jobs=[]
for f in utils.pairs(datafiles):
    jobs.append(serv.submit(get_container,(f[0],f[1],indexes), (), pp_imports))
for job in jobs:
    c = job()
    print c.x[0] #will print x coordinates of particles in first dataset
    print c.x[1] #will print x coordinates of particles in second dataset
```

As for the second option, we define the following algorithm: 1) load the first dataset; 2) apply a cut-off filter to the containers (or use operated reading method for cut-off particles); 3) repeat steps 1 and 2 for the second dataset; 4) draw particles movement; 5) repeat steps 1–4 for each dataset pair (1,2), (2,3), (3,4), ..., $(n-1, n)$.

The steps 1 and 2 of the algorithm can be performed with this simple function:

```
def get_containerv2(f1,f2):
    c1 = mmdlab.dataprocessor.region_filter(mmdlab.datareader.fast_read(f1), region)
    c2 = mmdlab.dataprocessor.region_filter(mmdlab.datareader.fast_read(f2), region)
    return mmdlab.dataprocessor.merge(c1,c2, np.intersect1d(c1.indexes, c2.indexes))
```

Both algorithms can be parallelized with respect to the data due to the independence of the data at the points $[i, i+1]$ and $[i+1, i+2]$. ParallelPython does this task perfectly.

As it was already said earlier, all visualization in this work is carried out by means of the Mayavi2 package providing extensive tools for drawing various data. For a specific objective the most suitable methods for visual data representation are mlab.points3d, mlab.pipeline.volume and mlab.IsoSurface. Using the embedded pipeline browser, any parameter of the visualization can be modified interactively, such as colormaps, isoline values, glyph shapes, etc. The description of these methods is available in the official documentation [7].

## 4 Results

Using the developed "mmdlab" module the visualization and the analysis of the adsorption process which is taking place on the metal-gas border were done (Fig. 2).

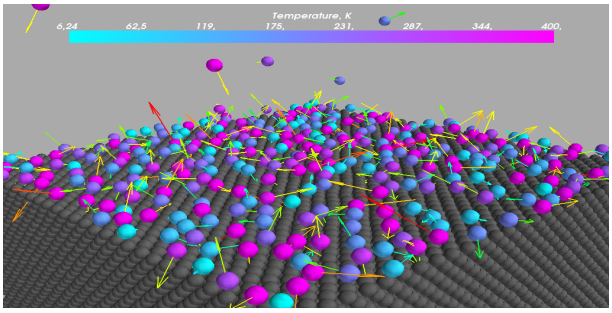


**Figure 2.** Adsorption of nitrogen on the nickel surface. Vectors represent particle velocities direction (in scale), the color represents nitrogen particles temperature.

## 5 Conclusion

In this work the analysis of the possibility of using the Python language for large scale data processing and visualisation was provided. The accelerators for Python such as NumPy and Numba were studied and put into practice. These packages allow to achieve computation performance comparable with the compiled languages. Also, the developed "mmdlab" module made it possible to visualize and examine in details microsystems with two interacting components. We can see the potential opportunity to use the interpreted programming language Python for processing and visualization problems of molecular dynamics calculations.

This work was supported by the Russian Foundation for Basic Researches (projects No. 13-01-12073-ofim, 15-07-06082-a).

## References

[1] D.C. Rapaport, *The Art of Molecular Dynamics Simulation* (Second Edition, Cambridge University Press, Cambridge, 2004) 565 p.
[2] V.O. Podryga, S.V. Polyakov, D.V. Puzyrkov, Journal Vychislitel'nye Metody i Programmirovanie. **16**, 123–138 (2015) [in Russian]
[3] Official site Python, https://www.python.org/
[4] Official site ParallelPython, http://www.parallelpython.com/
[5] Official site NumPy, http://www.numpy.org/
[6] Official site Numba, http://numba.pydata.org/
[7] Official Mayavi2 documentation, http://docs.enthought.com/mayavi/mayavi/mlab.html
[8] Official Cython documentation, http://cython.org/