Herramientas Computacionales para Matemática Aplicada

Curso 2020

Introducción a la Dinámica Molecular

Dinámica Molecular clásica

Molecular dynamics simulation consists of the numerical, step-by-step, solution of the classical equations of motion, which for a simple atomic system may be written

$$m_i \ddot{\boldsymbol{r}}_i = \boldsymbol{f}_i \qquad \boldsymbol{f}_i = -\frac{\partial}{\partial \boldsymbol{r}_i}\mathcal{U} \tag{1}$$

For this purpose we need to be able to calculate the forces $\boldsymbol{f}_i$ acting on the atoms, and these are usually derived from a potential energy $\mathcal{U}(\boldsymbol{r}^N)$, where $\boldsymbol{r}^N = (\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots \boldsymbol{r}_N)$ represents the complete set of $3N$ atomic coordinates. In this section we focus on this function

Distintos algoritmos son utilizados en la integración numérica de las ecuaciones del movimiento, tales como el de Euler, Gear, Verlet, etc.

Velocity Verlet algorithm

The **Velocity Verlet algorithm**, for use in molecular dynamics, is given by [1]:

$$r(t+\delta t) = r(t) + \delta t v(t) + \frac{1}{2}\delta t^2 a(t)$$

$$v\left(t+\delta t\right) = v(t) + \frac{1}{2}\delta t[a(t) + a(t+\delta t)]$$

where r is the position, v is the velocity, a is the acceleration and t is the time. δt is known as the time step.

Software disponible para Dinámica Molecular

Comparison of software for molecular mechanics modeling

From Wikipedia, the free encyclopedia

This is a list of computer programs that are predominantly used for **molecular mechanics** calculations.

- GPU – GPU accelerated
- I – Has interface
- Imp – Implicit water
- MC – Monte Carlo
- MD – Molecular dynamics
- Min – Optimization
- QM – Quantum mechanics
- REM – Replica exchange method

Name	View 3D	Model builder	Min	MD	MC	REM	QM	Imp	GPU	Comments	License	Website
Abalone	Yes	Yes	Yes	Yes	Yes	Yes	I	Yes	Yes	Biomolecular simulations, protein folding.	Proprietary, gratis, commercial	Agile Molecule
BOSS	No	No	Yes	No	Yes	No	Yes	No	No	OPLS	Proprietary	Yale University
CHARMM	No	Yes	Yes	Yes	Yes	I	I	Yes	Yes	Commercial version with multiple graphical front ends is sold by Accelrys (as CHARMm)	Proprietary, commercial	charmm.org
CHEMKIN	No	No	No	No	No	No	No	No	No	Chemical reaction kinetics.	Proprietary	CHEMKIN
GROMACS	No	No	Yes	Yes	No[1]	Yes	I	Yes[2]	Yes	High performance MD	Free open source GNU GPL	gromacs.org
GROMOS	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Intended for biomolecules	Proprietary, commercial	GROMOS website
LAMMPS	Yes	Yes	Yes	Yes	Yes	Yes	I	Yes	Yes	Has potentials for soft and solid-state materials and coarse-grain systems	Free open source, GNU GPLv2	Sandia

lammps

Large-scale Atomic/Molecular Massively Parallel Simulator (**LAMMPS**) is a molecular dynamics program from Sandia National Laboratories.[1] LAMMPS makes use of Message Passing Interface (MPI) for parallel communication and is free and open-source software, distributed under the terms of the GNU General Public License.[1]

LAMMPS was originally developed under a Cooperative Research and Development Agreement (CRADA) between two laboratories from United States Department of Energy and three other laboratories from private sector firms.[1] As of 2016, it is maintained and distributed by researchers at the Sandia National Laboratories and Temple University.[1]

Contents [hide]

1 Features
2 See also
3 References
4 External links

Features [edit]

For computing efficiency, LAMMPS uses neighbor lists (Verlet lists) to keep track of nearby particles. The lists are optimized for systems with particles that repel at short distances, so that the local density of particles never grows too large.

On parallel computers, LAMMPS uses spatial-decomposition techniques to partition the simulation domain into small 3d sub-domains, one of which is assigned to each processor. Processors communicate and store *ghost* atom information for atoms that border their subdomain. LAMMPS is most efficient (in a parallel computing sense) for systems whose particles fill a 3D rectangular box with approximately uniform density.

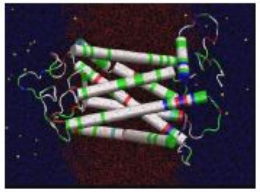
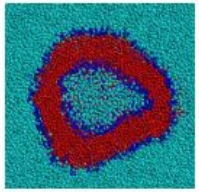
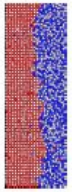
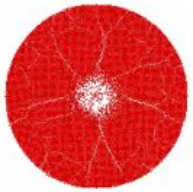
Original author(s)	Steve Plimpton, Aidan Thompson, Stan Moore, Axel Kohlmeyer
Developer(s)	Sandia National Laboratories Temple University
Initial release	1995; 25 years ago
Stable release	07-August-2019 / August 7, 2019; 7 months ago
Repository	github.com/lammps
Written in	C++
Operating system	Cross-platform: Linux, macOS, Windows
Platform	x86, x86-64
Size	304 MB
Available in	English
Type	Molecular dynamics
License	GNU General Public License
Website	lammps.sandia.gov

Repaso de las principales características de lammps:

Large-scale Atomic/Molecular Massively Parallel Simulator

http://lammps.sandia.gov

- Classical MD code
- Open source, portable C++
- 3-legged stool: soft matter, solids, mesoscale

- Particle simulator at varying length and time scales

  electrons ⇒ atomistic ⇒ coarse-grained ⇒ continuum
- Spatial-decomposition of simulation domain for parallelism
- MD, non-equilibrium MD, energy minimization
- GPU and OpenMP enhanced
- Can be coupled to other scales: QM, kMC, FE, CFD, ...

Algunos de los sistemas que han sido implementados empleando lammps

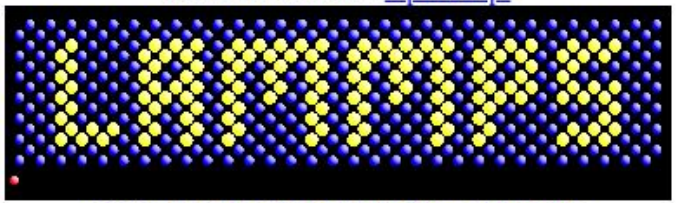
	carbon nanotube fiber design		shock loading of inhomogeneous PBX
	single-point diamond turning		nanoparticle coating structure in the presence of solvent
	two-temperature model for electronic heat conduction		atom-to-continuum coupling with the ATC package
	stress field around dislocations		water interacting with self-assembled monolayers
	coarse-grained self-assembly of lipids and PEG surfactants		spherical polyelectrolyte brushes
	coarse-grained block copolymer generation		polyelectrolyte adsorption and brushes
	stress in metal nanowires with twin boundaries		nanotip indentation of a coated surface
	surface wetting by polymer nanodroplet		shear of Cu bicrystal
	solvated dendritic polymer phase behavior		metal solidification
	lipid membrane self-assembly and fusion		tensile pull on adhesive polymer chains
	crazing of entangled polymer chains		stress in metal nanowires

Manual online de lammps

LAMMPS Molecular Dynamics Simulator

***lamp**: a device that generates light, heat, or therapeutic radiation; something that illumines the mind or soul* -- www.dictionary.com

hover to animate -- input script



physical analog (start at 3:25) & explanation

Big Picture	Code	Documentation	Results	Related Tools	Context	User Support
Features	Download	Manual	Publications	Pre/Post processing	Authors	Mail list
Non-features	GitHub	Developer guide	Pictures	Pizza.py Toolkit	History	IRC channel
Packages	SourceForge	Tutorials	Movies	Offsite LAMMPS packages & tools	Funding	Workshops
FAQ	Latest features & bug fixes	MD to LAMMPS glossary	Benchmarks	Visualization	Open source	Contribute to LAMMPS
Wish list	Report bugs & request features	Commands	Citing LAMMPS	Related modeling codes		.

Configurando y compilando lammps en GNU/Linux:

Pasos a seguir para la instalación:

- Descargar el paquete: lammps-stable.tar.gz en el directorio donde querramos instalarlo.
- Descomprimirlo con
  `tar xvfz lammps-stable.tar.gz`

- Entrar al directorio donde está el código fuente, por ejemplo:
  `cd lammps-5Jun19/src/`

  Si se desean instalar paquetes auxilares, tipear
  `make yes-<paquete>`

  Para desinstalarlos, tipear
  `make no-<paquete>`

```
Installed  NO: package USER-MESO
Installed  NO: package USER-MGPT
Installed  NO: package USER-MISC
Installed  NO: package USER-MOFFF
Installed YES: package USER-MOLFILE
Installed  NO: package USER-NETCDF
Installed  NO: package USER-OMP
Installed  NO: package USER-PHONON
Installed  NO: package USER-PLUMED
Installed  NO: package USER-PTM
Installed  NO: package USER-QMMM
Installed  NO: package USER-QTB
Installed  NO: package USER-QUIP
Installed  NO: package USER-REAXC
Installed  NO: package USER-SCAFACOS
Installed  NO: package USER-SMD
Installed  NO: package USER-SMTBQ
Installed  NO: package USER-SDPD
Installed  NO: package USER-SPH
Installed  NO: package USER-TALLY
Installed  NO: package USER-UEF
Installed  NO: package USER-VTK
Installed  NO: package USER-YAFF
fede@fedeLaptopLenovo:~/lammps/lammps-5Jun19/src$ make ps
```

Configurando lammps

Al tipear `make` sin parámetros nos imprimirá ayuda en pantalla respecto de la configuración de los paquetes adicionales y de Makefiles genéricos disponibles

```
make package                    list available packages and their dependencies
make package-status (ps)        status of all packages
make package-installed (pi)     list of installed packages
make yes-package                install a single pgk in src dir
make no-package                 remove a single pkg from src dir
make yes-all                    install all pgks in src dir
make no-all                     remove all pkgs from src dir
make yes-standard (yes-std)     install all standard pkgs
make no-standard (no-std)       remove all standard pkgs
make yes-user                   install all user pkgs
make no-user                    remove all user pkgs
make yes-lib          install all pkgs with libs (included or ext)
make no-lib           remove all pkgs with libs (included or ext)
make yes-ext                    install all pkgs with external libs
make no-ext                     remove all pkgs with external libs
```

Configurando lammps

Luego, editamos el Makefile más apropiado para nuestra plataforma (por ejemplo, asumamos que es src/MACHINES/Makefile.ubuntu), lo editamos por si fuera necesario hacer alguna modificación y guardamos los cambios. Tener en cuenta que si queremos correr lammps en paralelo, debemos instalar y configurar una biblioteca mpi (por ejemplo openmpi) y elegir un Makefile que compile lammps usando esta capacidad. De hecho la mayoría de los Makefiles lo preconfigurados la usan (no así, evidentemente, el Makefile llamado Makefile.serial), ya que las simulaciones de Dinámica Molecular pueden tener tiempos de ejecución muy largos, así que es muy buena idea correr en lammps en paralelo siempre que podamos!

```
# libfftw3-dev, libjpeg-dev and libpng12-dev to compile LAMMPS with this
# makefile

SHELL = /bin/sh

# ------------------------------------------------------------------------
# compiler/linker settings
# specify flags and libraries needed for your compiler

CC =            mpic++
CCFLAGS =       -g -O3 # -Wunused
SHFLAGS =       -fPIC
DEPFLAGS =      -M

LINK =          mpic++
LINKFLAGS =     -g -O3
LIB =
SIZE =          size

ARCHIVE =       ar
ARFLAGS =       -rc
SHLIBFLAGS =    -shared

# ------------------------------------------------------------------------
# LAMMPS-specific settings, all OPTIONAL
# specify settings for LAMMPS features you will use
# if you change any -D setting, do full re-compile after "make clean"

# LAMMPS ifdef settings
# see possible settings in Section 2.2 (step 4) of manual

LMP_INC =       -DLAMMPS_GZIP -DLAMMPS_JPEG -DLAMMPS_PNG -DLAMMPS_FFMPEG
```

MPI

Interfaz de Paso de Mensajes

Para MPI (Multidimensional Poverty Index), véase Índice de pobreza multidimensional.

MPI ("Message Passing Interface", Interfaz de Paso de Mensajes) es un estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores.

El paso de mensajes es una técnica empleada en programación concurrente para aportar sincronización entre procesos y permitir la exclusión mutua, de manera similar a como se hace con los semáforos, monitores, etc.

Su principal característica es que no precisa de memoria compartida, por lo que es muy importante en la programación de sistemas distribuidos.

La Interfaz de Paso de Mensajes (conocido ampliamente como MPI, siglas en inglés de *Message Passing Interface*) es un protocolo de comunicación entre computadoras. Es el estándar para la comunicación entre los nodos que ejecutan un programa en un sistema de memoria distribuida. Las implementaciones en MPI consisten en un conjunto de bibliotecas de rutinas que pueden ser utilizadas en programas escritos en los lenguajes de programación *C*, C++, Fortran y Ada. La ventaja de MPI sobre otras bibliotecas de paso de mensajes, es que los programas que utilizan la biblioteca son portables (dado que MPI ha sido implementado para casi toda arquitectura de memoria distribuida), y rápidos, (porque cada implementación de la biblioteca ha sido optimizada para el hardware en la cual se ejecuta).

MPI

Implementaciones [editar]

La implementación del lenguaje para MPI es, en general, los lenguajes que intentan optimizar el tiempo de ejecución.

La mayoría de las implementaciones de MPI se realizan en una combinación de C, C + + y el lenguaje ensamblador, C ++, Fortran. Sin embargo, el idioma y la aplicación de usuario final idioma son, en principio, siempre desasociado.

La etapa inicial de aplicación del estándar MPI 1.x fue MPICH, nacida en Argonne National Laboratory y la Universidad del Estado de Mississippi.

IBM también fue una de las primeras en implementar el estándar MPI, y la mayoría de las empresas de superordenador a principio de los años 90 comercializando MPICH o implementando su propia aplicación del estándar MPI 1.x.

LAM/MPI del Centro de Supercomputación de Ohio es otra de las primeras en implementarlo.

El Argonne National Laboratory continuó desarrollando MPICH durante más de una década, y ahora ofrece MPICH2 que se corresponde con la implementación del estándar MPI-2.1

LAM/MPI, y otra serie de esfuerzos recientes de MPI se han fusionado para formar un nuevo proyecto mundial, el llamado OpenMPI, pero este nombre no implica ninguna relación con el estándar.

Microsoft ha añadido una MPI al esfuerzo a sus Kit Cluster Computing (2005), basada en MPICH2. MPI se ha convertido y sigue siendo un elemento vital para la interfaz de programación concurrente a esta fecha de hoy.

Muchas distribuciones de Linux incluyen MPI (uno o ambos MPICH y LAM, como ejemplos particulares), pero es mejor obtener las últimas versiones de MPI de desarrolladores desde los sitios de desarrollo.

Compilación de lammps

Compilamos lammps usando este Makefile, tipeando **dentro del directorio `src/`** :

```
make ubuntu
```

La compilación puede demorar bastante (una hora aproximadamente, obviamente dependiendo de la configuración y de la capacidad del procesador de nuestra computadora).
Si la PC en la que estamos realizando la compilación tiene varios cpu-cores disponibles, podemos acelerar el proceso de compilación mediante una compilación en paralelo. Para esto simplemente poner el flag “-jnum_cores”, como en el siguiente ejemplo:

```
make -j4 ubuntu
```

Si la compilación fue exitosa (es normal que ocurran errores por alguna biblioteca faltante que hay que instalar para luego proseguir con la compilación), se creará un ejecutable (dentro de src/) llamado (por defecto, obviamente este nombre puede cambiarse) `lmp_ubuntu` . Conviene hacer un enlace simbólico a este ejecutable de la manera que fue explicada antes en el curso, para evitar tipear la ruta a `lmp_ubuntu` cada vez que se quiera ejecutar lammps.

Primeros ejemplos: ejecutando lammps en serie

En el directorio `examples` dentro del directorio donde descargamos lammps, existen múltiples ejemplos que muestran los diversos sistemas que pueden simularse con lammps. Por ejemplo, si nos metemos en el directorio obstacle y corremos

```
/ruta/to/lammps/lammps.ubuntu < in.obstacle
```

debería comenzar la ejecución, mostrándonos la siguiente salida:

```
Nlocal:    769 ave 769 max 769 min
Histogram: 1 0 0 0 0 0 0 0 0 0
Nghost:    45 ave 45 max 45 min
Histogram: 1 0 0 0 0 0 0 0 0 0
Neighs:    1618 ave 1618 max 1618 min
Histogram: 1 0 0 0 0 0 0 0 0 0

Total # of neighbors = 1618
Ave neighs/atom = 2.10403
Neighbor list builds = 1622
Dangerous builds = 0
Total wall time: 0:00:02
fede@fedeLaptopLenovo:~/lammps/lammps-5Jun19/examples/obstacle$ ls
in.obstacle  log.27Nov18.obstacle.g++.1  log.27Nov18.obstacle.g++.4  log.lammps
fede@fedeLaptopLenovo:~/lammps/lammps-5Jun19/examples/obstacle$ lammps19 < in.obstacle
```

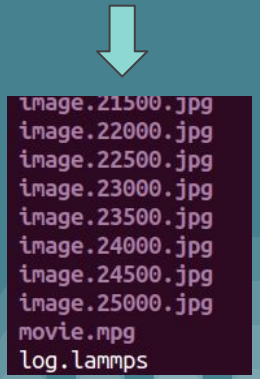
Primeros ejemplos: ejecutando lammps en paralelo

Si tenemos una biblioteca mpi adecuadamente instalada (por ejemplo openmpi), y lammps fue compilado usando tal biblioteca, podemos correr lammps en paralelo , mediante el comando
`mpirun -np 4 /ruta/to/lammps/lammps.ubuntu < in.obstacle`
(en este caso estamos usando 4 procesadores, por supuesto este número lo modificamos de acuerdo a nuestra disponibilidad de cpu-cores libres)

```
Loop time of 1.61871 on 4 procs for 25000 steps with 769 atoms

Performance: 4003175.830 tau/day, 15444.351 timesteps/s
79.7% CPU use with 4 MPI tasks x no OpenMP threads

MPI task timing breakdown:
Section |  min time  |  avg time  |  max time  |%varavg| %total
---------------------------------------------------------------
Pair    | 0.10771    | 0.2098     | 0.34191    |  20.7 | 12.96
Neigh   | 0.050759   | 0.070235   | 0.096265   |   6.4 |  4.34
Comm    | 0.27313    | 0.42409    | 0.50697    |  13.8 | 26.20
Output  | 0.00085711 | 0.0021454  | 0.0059955  |   4.8 |  0.13
Modify  | 0.61465    | 0.66       | 0.75027    |   6.7 | 40.77
Other   |            | 0.2524     |            |       | 15.60

Nlocal:    192.25 ave 243 max 151 min
Histogram: 1 1 0 0 0 0 1 0 0 1
Nghost:    41.75 ave 43 max 39 min
Histogram: 1 0 0 0 0 0 0 1 0 2
Neighs:    408.5 ave 575 max 266 min
Histogram: 1 1 0 0 0 0 0 1 0 1
```

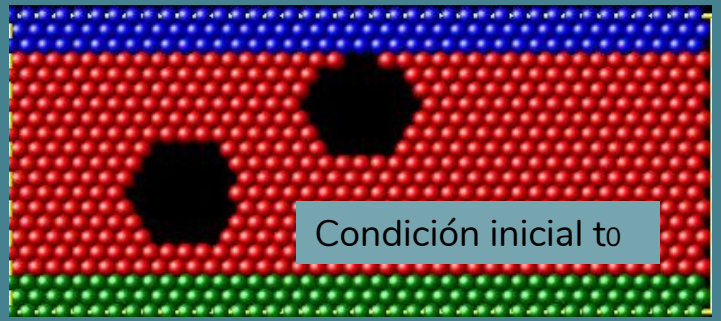
Obteniendo salida gráfica de la simulación:

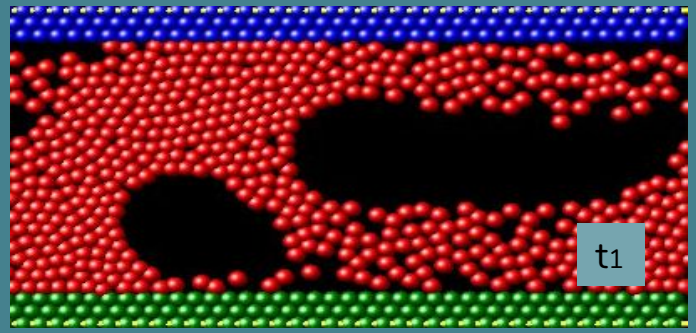
Descomentando las líneas que se refieren al guardado de imágenes (las líneas de comentario en el lenguaje de scripting de lammps comienzan con #) y del archivo de video, la simulación generará varias imágenes en formato jpg y un video en formato mpg con la evolución temporal de la simulación

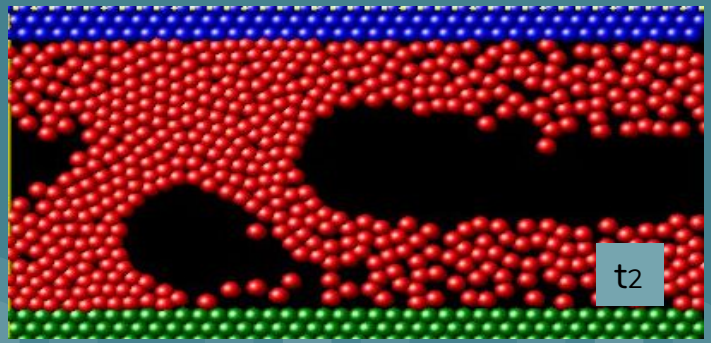


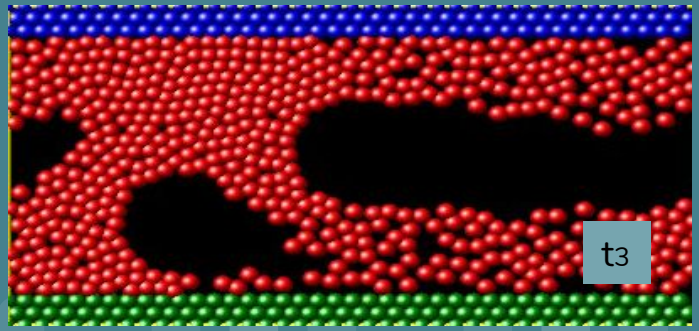

```
region          void1 sphere 10 4 0 3
delete_atoms    region void1
region          void2 sphere 20 7 0 3
delete_atoms    region void2

fix             7 flow indent 100 sphere 10 4 0 4
fix             8 flow indent 100 sphere 20 7 0 4
fix             9 all enforce2d

# Run

timestep        0.003
thermo          1000
thermo_modify   temp mobile

#dump           1 all atom 100 dump.obstacle

dump            2 all image 500 image.*.jpg type type &
                zoom 1.6 adiam 1.5
dump_modify     2 pad 5

dump            3 all movie 500 movie.mpg type type &
                zoom 1.6 adiam 1.5
dump_modify     3 pad 5

run             25000
```

Imágenes de salida de la simulación









Potenciales de interacción de Lennard -Jones

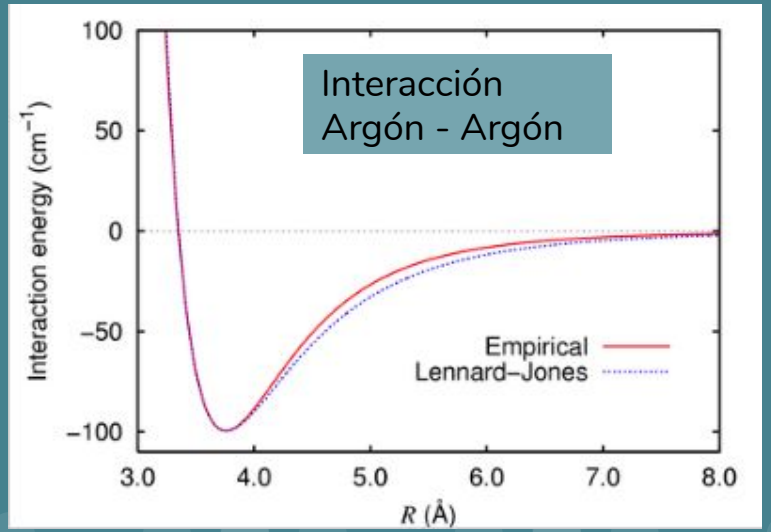
El potencial de Lennard-Jones es de la forma:

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right]$$

donde:

- ϵ es la profundidad del potencial,
- σ es la distancia (finita) en la que el potencial entre partículas es cero y
- r es la distancia entre partículas.

El potencial de LJ es una excelente aproximación para la interacción interatómica en gases nobles. Es ampliamente usado para describir la interacción entre moléculas en otros sistemas, por tener una buena relación entre la bondad de la aproximación al potencial real y el costo computacional.



Potenciales para sistemas condensados

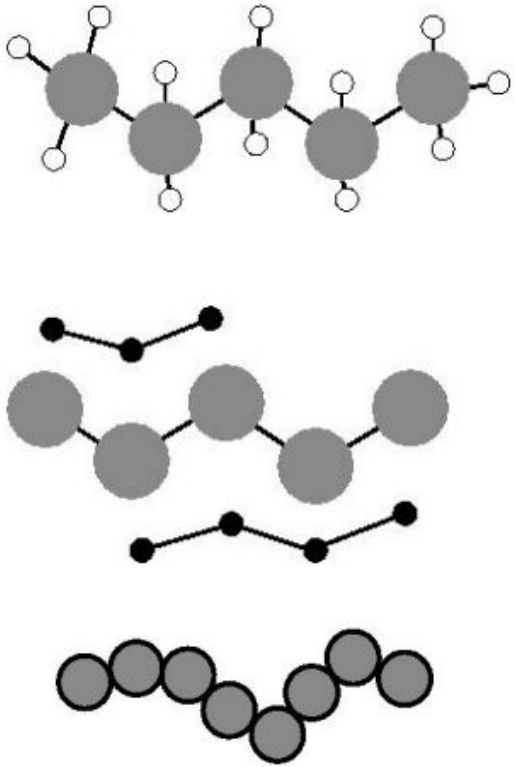
En general, los potenciales de a pares del tipo LJ son una muy mala aproximación por ejemplo para sólidos. Para estos sistemas, se han desarrollado potenciales especializados que capturan de alguna manera los efectos mecanocuánticos que no pueden despreciarse en tales sistemas. En general son potenciales de “muchos cuerpos” (por ejemplo: si A, B, y C son tres átomos muy próximos entre sí, la interacción entre A y B está influenciada por la ubicación de C, contrariamente a lo que ocurre en los potenciales de a pares)

Por ejemplo, para metales son muy utilizados los potenciales del tipo EAM (“Embedded Atom Model”), para cerámicos los del tipo “Tersoff” o de “Stillinger-Weber”, etc.

Enfoques all-atom vs united-atom

El timestep en una simulación de DM es uno de las limitaciones más importantes de la técnica, siendo típicamente del orden de los femtosegundos, es decir E-15 segundos!!

- All-atom:
  - Δt = 0.5-1.0 fmsec for C-H
  - C-C distance = 1.5 Angs
  - cutoff = 10 Angs
- United-atom:
  - # of interactions is 9x less
  - Δt = 1.0-2.0 fmsec for C-C
  - cutoff = 10 Angs
  - 20-30x savings over all-atom
- Bead-Spring:
  - 2-3 C per bead
  - $\Delta t \iff$ fmsec mapping is T-dependent
  - $2^{1/6}\sigma$ cutoff $\Rightarrow$ 8x in interactions
  - can be considerable savings over united-atom

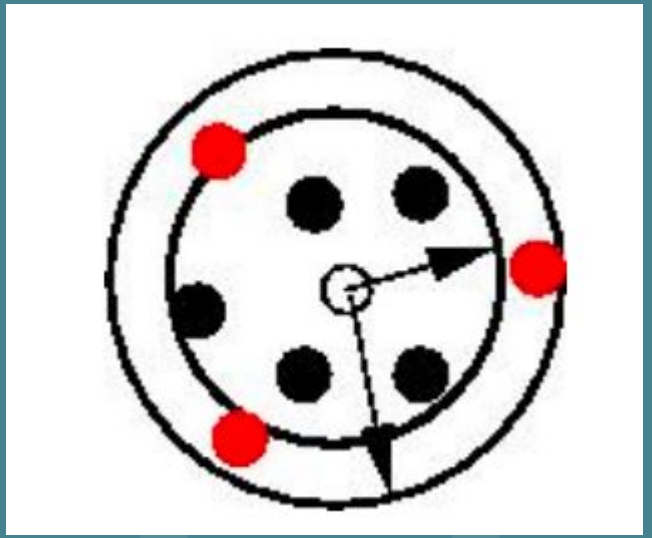


Listas de vecinos

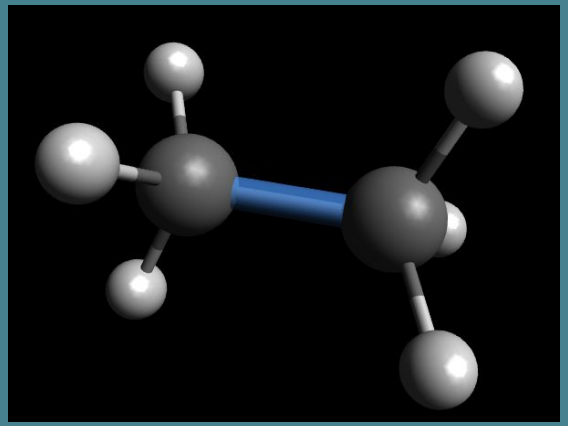
Con el fin de ahorrar tiempo de máquina, es usual en simulaciones de DM emplear listas de vecinos

Verlet lists:

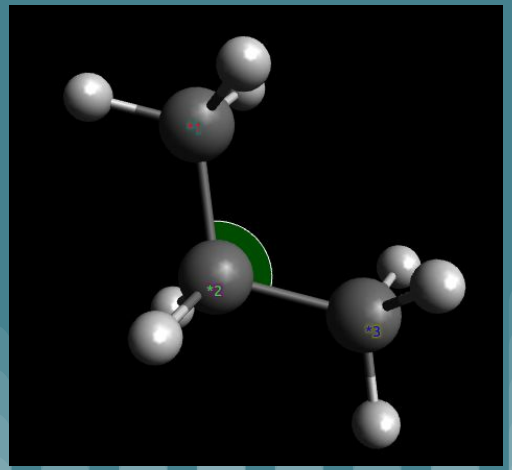
- *Verlet, Phys Rev, 159, p 98 (1967)*
- $R_{neigh} = R_{force} + \Delta_{skin}$
- build list: once every few timesteps
- other timesteps: scan larger list for neighbors within force cutoff
- rebuild: any atom moves 1/2 skin

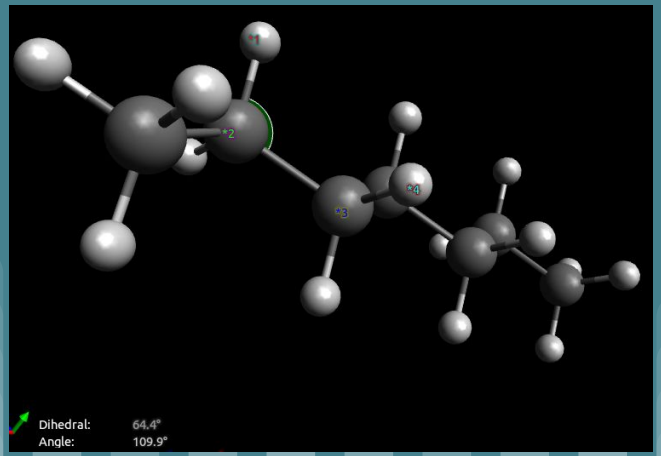


Interacciones intramoleculares



$$\mathcal{U}_{\text{intramolecular}} = \frac{1}{2}\sum_{\text{bonds}} k_{ij}^{r}\left(r_{ij} - r_{\text{eq}}\right)^2 + \frac{1}{2}\sum_{\substack{\text{bend}\\ \text{angles}}} k_{ijk}^{\theta}\left(\theta_{ijk} - \theta_{\text{eq}}\right)^2 + \frac{1}{2}\sum_{\substack{\text{torsion}\\ \text{angles}}}\sum_{m} k_{ijkl}^{\phi,\,m}\left(1 + \cos(m\phi_{ijkl} - \gamma_m)\right)$$





Comandos generales de lammps

An alphabetic list of all general LAMMPS commands.

angle_coeff	angle_style	atom_modify	atom_style	balance
bond_coeff	bond_style	bond_write	boundary	box
change_box	clear	comm_modify	comm_style	compute
compute_modify	create_atoms	create_bonds	create_box	delete_atoms
delete_bonds	dielectric	dihedral_coeff	dihedral_style	dimension
displace_atoms	dump	dump atom/adios	dump custom/adios	dump image
dump movie	dump netcdf	dump netcdf/mpiio	dump vtk	dump_modify
dynamical_matrix	echo	fix	fix_modify	group
group2ndx	hyper	if	improper_coeff	improper_style
include	info	jump	kim_init	kim_interactions
kim_param	kim_query	kspace_modify	kspace_style	label
lattice	log	mass	message	minimize
min_modify	min_style	min_style spin	molecule	ndx2group
neb	neb/spin	neigh_modify	neighbor	newton
next	package	pair_coeff	pair_modify	pair_write
partition	prd	print	processors	python
quit	read_data	read_dump	read_restart	region

Fixes

5.6. Fix commands

An alphabetic list of all LAMMPS fix commands. Some styles have accelerated versions. This is indicated by additional letters in parenthesis: g = GPU, i = USER-INTEL, k = KOKKOS, o = USER-OMP, t = OPT.

adapt	adapt/fep	addforce	addtorque	append/atoms
atc	atom/swap	ave/atom	ave/chunk	ave/correlate
ave/correlate/long	ave/histo	ave/histo/weight	ave/time	aveforce
balance	bocs	bond/break	bond/create	bond/react
bond/swap	box/relax	client/md	cmap	colvars
controller	deform (k)	deposit	dpd/energy (k)	drag
drude	drude/transform/direct	drude/transform/inverse	dt/reset	edpd/source
efield	ehex	electron/stopping	enforce2d (k)	eos/cv
eos/table	eos/table/rx (k)	evaporate	external	ffl
filter/corotate	flow/gauss	freeze (k)	gcmc	gld
gle	gravity (ko)	grem	halt	heat
hyper/global	hyper/local	imd	indent	ipi
langevin (k)	langevin/drude	langevin/eff	langevin/spin	latte
lb/fluid	lb/momentum	lb/pc	lb/rigid/pc/sphere	lb/viscous

Computes

ke/eff	ke/rigid	meso/e/atom	meso/rho/atom	meso/t/atom
momentum	msd	msd/chunk	msd/nongauss	omega/chunk
orientorder/atom	pair	pair/local	pe	pe/atom
pe/mol/tally	pe/tally	plasticity/atom	pressure	pressure/cylinder
pressure/uef	property/atom	property/chunk	property/local	ptm/atom
rdf	reduce	reduce/chunk	reduce/region	rigid/local
saed	slice	smd/contact/radius	smd/damage	smd/hourglass/error
smd/internal/energy	smd/plastic/strain	smd/plastic/strain/rate	smd/rho	smd/tlsph/defgrad
smd/tlsph/dt	smd/tlsph/num/neighs	smd/tlsph/shape	smd/tlsph/strain	smd/tlsph/strain/rate
smd/tlsph/stress	smd/triangle/vertices	smd/ulsph/num/neighs	smd/ulsph/strain	smd/ulsph/strain/rate
smd/ulsph/stress	smd/vol	snap	sna/atom	snad/atom
snav/atom	spin	stress/atom	stress/mop	stress/mop/profile
stress/tally	tdpd/cc/atom	temp (k)	temp/asphere	temp/body
temp/chunk	temp/com	temp/cs	temp/deform	temp/deform/eff
temp/drude	temp/eff	temp/partial	temp/profile	temp/ramp
temp/region	temp/region/eff	temp/rotate	temp/sphere	temp/uef
ti	torque/chunk	vacf	vcm/chunk	voronoi/atom

Estructura típica de un script de lammps :

5.3. Input script structure

This page describes the structure of a typical LAMMPS input script. The examples directory in the LAMMPS distribution contains many sample input scripts; it is discussed on the Examples doc page.

A LAMMPS input script typically has 4 parts:

1. Initialization
2. System definition
3. Simulation settings
4. Run a simulation

The last 2 parts can be repeated as many times as desired. I.e. run a simulation, change some settings, run some more, etc. Each of the 4 parts is now described in more detail. Remember that almost all commands need only be used if a non-default value is desired.

Inicialización y definición del sistema:

Set parameters that need to be defined before atoms are created or read-in from a file.

The relevant commands are units, dimension, newton, processors, boundary, atom_style, atom_modify.

If force-field parameters appear in the files that will be read, these commands tell LAMMPS what kinds of force fields are being used: pair_style, bond_style, angle_style, dihedral_style, improper_style.

There are 3 ways to define the simulation cell and reserve space for force field info and fill it with atoms in LAMMPS. Read them in from (1) a data file or (2) a restart file via the read_data or read_restart commands, respectively. These files can also contain molecular topology information. Or (3) create a simulation cell and fill it with atoms on a lattice (with no molecular topology), using these commands: lattice, region, create_box, create_atoms or read_dump.

The entire set of atoms can be duplicated to make a larger simulation using the replicate command.

Más opciones para configurar:

Once atoms and molecular topology are defined, a variety of settings can be specified: force field coefficients, simulation parameters, output options, and more.

Force field coefficients are set by these commands (they can also be set in the read-in files): pair_coeff, bond_coeff, angle_coeff, dihedral_coeff, improper_coeff, kspace_style, dielectric, special_bonds.

Various simulation parameters are set by these commands: neighbor, neigh_modify, group, timestep, reset_timestep, run_style, min_style, min_modify.

Fixes impose a variety of boundary conditions, time integration, and diagnostic options. The fix command comes in many flavors.

Various computations can be specified for execution during a simulation using the compute, compute_modify, and variable commands.

Output options are set by the thermo, dump, and restart commands.

A molecular dynamics simulation is run using the run command.

Control del termostato de la simulación en lammps

Thermostatting means controlling the temperature of particles in an MD simulation. Barostatting means controlling the pressure. Since the pressure includes a kinetic component due to particle velocities, both these operations require calculation of the temperature. Typically a target temperature (T) and/or pressure (P) is specified by the user, and the thermostat or barostat attempts to equilibrate the system to the requested T and/or P.

Thermostatting in LAMMPS is performed by fixes, or in one case by a pair style. Several thermostatting fixes are available: Nose-Hoover (nvt), Berendsen, CSVR, Langevin, and direct rescaling (temp/rescale). Dissipative particle dynamics (DPD) thermostatting can be invoked via the *dpd/tstat* pair style:

- fix nvt
- fix nvt/sphere
- fix nvt/asphere
- fix nvt/sllod
- fix temp/berendsen
- fix temp/csvr
- fix langevin
- fix temp/rescale
- pair_style dpd/tstat

Seleccionando los tipos de interacción

pair_style coul/slater/cut command
pair_style coul/slater/long command
pair_style born/coul/dsf/cs command
pair_style born/coul/long/cs command
pair_style born/coul/long/cs/gpu command
pair_style born/coul/wolf/cs command
pair_style born/coul/wolf/cs/gpu command
pair_style buck/coul/long/cs command
pair_style coul/long/cs command
pair_style coul/long/cs/gpu command
pair_style coul/wolf/cs command
pair_style lj/cut/coul/long/cs command
pair_style lj/cut/dipole/cut command
pair_style lj/cut/dipole/cut/gpu command
pair_style lj/cut/dipole/cut/omp command
pair_style lj/sf/dipole/sf command

pair_style eam/fs/kk command
pair_style eam/fs/omp command
pair_style eam/fs/opt command
pair_style edip command
pair_style edip/omp command
pair_style edip/multi command
pair_style eff/cut command
pair_style eim command
pair_style eim/omp command
pair_style exp6/rx command
pair_style exp6/rx/kk command
pair_style extep command
pair_style lj/cut/soft command
pair_style lj/cut/soft/omp command
pair_style lj/cut/coul/cut/soft command
pair_style lj/cut/coul/cut/soft/omp command

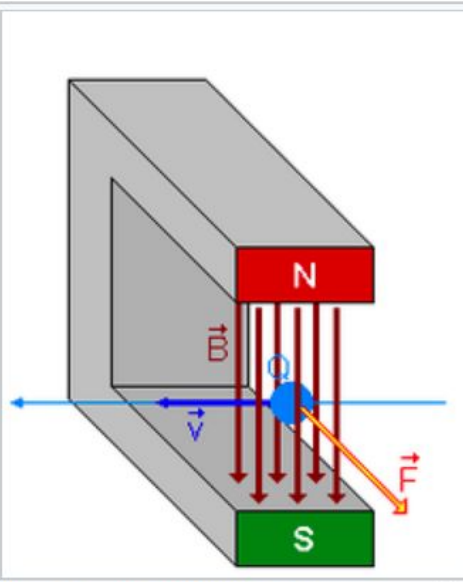
Ejemplo: Partícula en campo electromagnético

Forma clásica [editar]

Para una partícula sometida a un campo eléctrico combinado con un campo magnético, la fuerza electromagnética total o fuerza de Lorentz sobre esa partícula viene dada por:

$$\vec{F} = q(\vec{v} \times \vec{B} + \vec{E}),$$

donde $\mathbf{v}$ es la velocidad de la carga, $\mathbf{E}$ es el vector intensidad de campo eléctrico y $\mathbf{B}$ es el vector inducción magnética



Fuerza sobre una partícula cargada.

Ejemplo: Partícula en campo electromagnético (cont.)

```
#Correr con
#mpirun -np 2 lammps19 -sf opt < ElectronEnCampoEyB.lammps
#si se quiere usar la definicion de una variable  desde linea de
#comandos usar:
#lammps19 -var extern valor -sf opt < ElectronEnCampoEyB.lammps


#probado con lammps-5Jun19
#

#For style real, these are the units:

#mass = grams/mole
#distance = Angstroms
#time = femtoseconds
#energy = Kcal/mole
#velocity = Angstroms/femtosecond
#force = Kcal/mole-Angstrom
#torque = Kcal/mole
#temperature = Kelvin
#pressure = atmospheres
#dynamic viscosity = Poise
#charge = multiple of electron charge (1.0 is a proton)
#dipole = charge*Angstroms
#electric field = volts/Angstrom
#density = gram/cm^dim

units real
boundary        s s s
dimension 3
atom_style full
```

Ejemplo: Partícula en campo electromagnético (cont.)

```
#tipos con que se asocian las particulas
variable TipoA1 equal 1


#Poner la variable "GenerarMovie" en on/off segun se quiera o no generar la movie
variable GenerarMovie string "off"


###XXX#########################################################################################
#######################################SECCCION DE PARAMETROS DE CONTROL DE LA SIMULACION########
#################################################################################################

#caja inicial del sistema
variable xlo equal 0.0
variable xhi equal 300.0
variable ylo equal 0.0
variable yhi equal 300.0
variable zlo equal 0.0
variable zhi equal 300.0




variable NumPasosIntegracion equal 1e6


variable DTime equal 0.00001

#posición inicial
variable x_ini equal 0.0
variable y_ini equal 0.0
variable z_ini equal 100.0
```

Ejemplo: Partícula en campo electromagnético (cont.)

```
variable NumPasosIntegracion equal 1e6


variable DTime equal 0.00001

#posición inicial
variable x_ini equal 0.0
variable y_ini equal 0.0
variable z_ini equal 100.0

#velocidad inicial
variable vx_ini equal 10.0
variable vy_ini equal 0.0
variable vz_ini equal 0.0


#IntensityMagn=0 lo usamos para "apagar" el campo magnético
variable IntensityMagn equal 10.0

#IntensityElectr=0 lo usamos para "apagar" el campo electrico
variable IntensityElectr equal 0.1

#Directorio en el que se van a guardar los archivos de salida
variable DirectorioOut string ./DatosElectronEnCampoElectromagnetico/
#creamos el directorio DirectorioOut (si es que no existe...) Ojo, NO crea subdirectorios...
shell mkdir ${DirectorioOut}

variable Prefix string Z_

variable NombreDeBase string ${Prefix}ElectronEnCampo_PasInt_IntensityMagn${IntensityMagn}_IntensityElectr${IntensityElectr}
NumPasosIntegracion${NumPasosIntegracion}Dt${DTime}
#variable NombreDeBase string ${Prefix}prueba
```

Ejemplo: Partícula en campo electromagnético (cont.)

```
variable NumFigurasParaMovie equal 200
variable NumFigurasPNG equal 1
variable NumFigurasXYZ equal 1
variable NumDatos equal 1000
variable NumEstado equal 1



##No modificar esto que sigue
variable NumTotalPasos equal ${NumPasosIntegracion}
variable PasosGuardarFigurasParaMovie equal ${NumPasosIntegracion}/${NumFigurasParaMovie}
variable PasosGuardarFigurasPNG equal ${NumPasosIntegracion}/${NumFigurasPNG}
variable PasosGuardarFigurasXYZ equal ${NumPasosIntegracion}/${NumFigurasXYZ}
variable PasosDatos equal ${NumPasosIntegracion}/${NumDatos}
variable PasosEstado equal ${NumPasosIntegracion}/${NumEstado}
##

#esto lo pongo para poder definir variables que usen info de
#atomos individuales, como en
#variable MiVariable equal x[34]
atom_modify map array

timestep ${DTime}

region CAJA_SISTEMA block ${xlo} ${xhi} ${ylo} ${yhi} ${zlo} ${zhi}

create_box 1 CAJA_SISTEMA
###XXX#############################################################################################
#####################################SECCCION DE INSERCION DE ATOMOS EN EL SISTEMA#############
#############################################################################################

create_atoms 1 single ${x_ini} ${y_ini} ${z_ini}
```

Ejemplo: Partícula en campo electromagnético (cont.)

```
create_atoms 1 single ${x_ini} ${y_ini} ${z_ini}


#masa del electrón en gramos/mol
variable MasaElectron equal 0.00054854
mass 1 ${MasaElectron}

#seteamos la carga del electrón, consistentemente con el sistema de
#unidades de lammps "real"
variable CargaElectron equal -1
set type 1 charge ${CargaElectron}

pair_style none

#seteamos la velocidad del electrón
velocity all set ${vx_ini} ${vy_ini} ${vz_ini}

#seteamos el campo eléctrico externo
fix fix_campo_electrico all efield 0.0 ${IntensityElectr} 0.0

#########comienzo seccion de configuración el campo magnetico externo#################
#calculamos las componentes de la fuerza de Lorentz q(v x B), asumimos
#B=(0,1,0)
variable FLx equal -${CargaElectron}*${IntensityMagn}*vz[1]
variable FLy equal 0
variable FLz equal ${CargaElectron}*${IntensityMagn}*vx[1]



fix fix_campo_magnetico all addforce v_FLx v_FLy v_FLz

#-------------fin seccion de configuración el campo magnetico externo----------------#
```

Ejemplo: Partícula en campo electromagnético (cont.)

```
#termostato para la integración a energía constante
fix termostato all nve


if "${GenerarMovie}==on" then &
"dump           dump_movie all movie ${PasosGuardarFigurasParaMovie} ${DirectorioOut}${NombreDeBase}.mpg type type size 1200
1200 &
zoom 2.5 axes no 0 0 box no 0"




variable TrazaX equal x[1]
variable TrazaY equal y[1]
variable TrazaZ equal z[1]

fix printTraza all print ${PasosDatos} &
"${tiempo} ${TrazaX} ${TrazaY} ${TrazaZ}" &
file ${DirectorioOut}${NombreDeBase}__traza screen no &
title "#tiempo [fms]   X [Ang]    Y [Ang]       Z [Ang]"



#Definimos algunas variables utiles para el output de la info termodinamica
variable tiempo equal time
variable EnergiaCinetica equal 0.5*${MasaElectron}*(vx[1]*vx[1]+vy[1]*vy[1]+vz[1]*vz[1])


fix printEnergia all print ${PasosDatos} &
"${tiempo} ${EnergiaCinetica}" &
file ${DirectorioOut}${NombreDeBase}__energia_cinetica screen no &
title "#tiempo [fms] EnergiaCineticaTotal [grams.angs^2/(mole.fms^2)]"

run ${NumPasosIntegracion}
```

Postprocesando los resultados: visualizadores moleculares

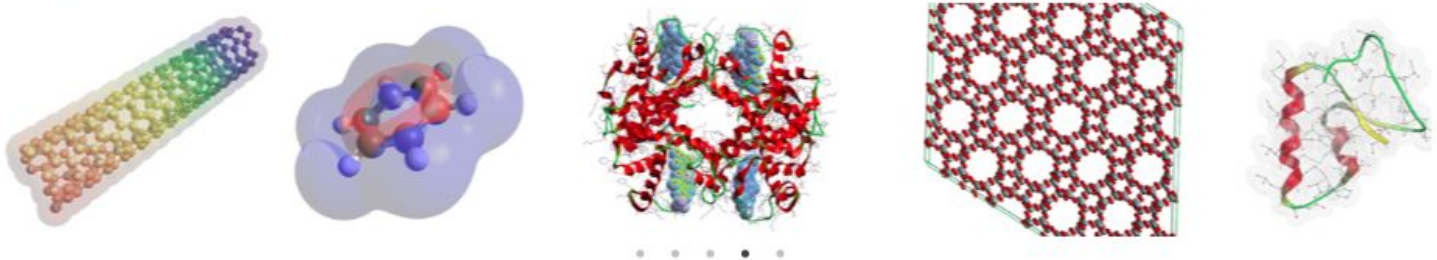
Avogadro	MM XRC MD	Free open-source, GPL	C++, Qt, extensible via Python modules
BALLView	MM NA XRC SMI	Free open-source, GPL	C++, Python; Windows, Linux, Solaris, Mac OS X
CAVER Analyst	MD EM XRC	Proprietary, free use noncommercial	Java; Windows, Linux, Solaris, Mac OS X
CCP4mg	XRC MM NA SMI	Free open-source	C++, Python, OpenGL, Windows, OS X, Linux
ChemDoodle 3D	MM	Proprietary	Java, OpenGL, Windows, macOS, Linux
Cn3D		Free open-source	Standalone program
CheVi	SMI	Free open-source	Standalone program
chemkit	MM MD	Free open-source	C++
chemlab	MM MD	Free open-source, GPL	Python

...

Qmol	MM	Free open-source	C
RasMol		Free open-source	C standalone program
SAMSON	MM MD SMI	Free	Windows, Linux, Mac. C++ (Qt)
Sirius		Free open-source	Java 3D applet or standalone program
Scigress	MM QM	Proprietary[30]	Standalone program
Spartan	MM QM	Proprietary[32]	Standalone program
Tapering lines		Proprietary, free for academic use	Standalone program
UCSF Chimera	XRC SMI EM MD	Proprietary, free use noncommercial[34]	Python
VMD	EM MD MM	Proprietary, free use noncommercial[38]	C++

Visualizador molecular (Open Source): Avogadro

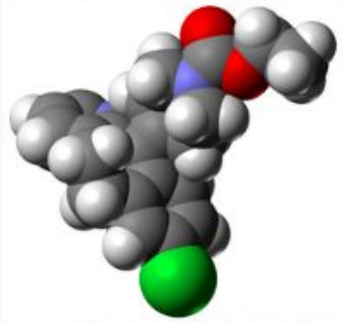
Avogadro is an advanced molecule editor and visualizer designed for cross-platform use in computational chemistry, molecular modeling, bioinformatics, materials science, and related areas. It offers flexible high quality rendering and a powerful plugin architecture.



- **Cross-Platform:** Molecular builder/editor for Windows, Linux, and Mac OS X.
- **Free, Open Source**: Easy to install and all source code and documentation is available to modify or extend.
- **International**: Translations into Chinese, French, German, Italian, Russian, Spanish, and others, with more languages to come.
- **Intuitive**: Built to work easily for students and advanced researchers both.
- **Fast**: Supports multi-threaded rendering and computation.
- **Extensible**: Plugin architecture for developers, including rendering, interactive tools, commands, and Python scripts.
- **Flexible**: Features include Open Babel import of chemical files, input generation for multiple computational chemistry packages, crystallography, and biomolecules.

Avogadro

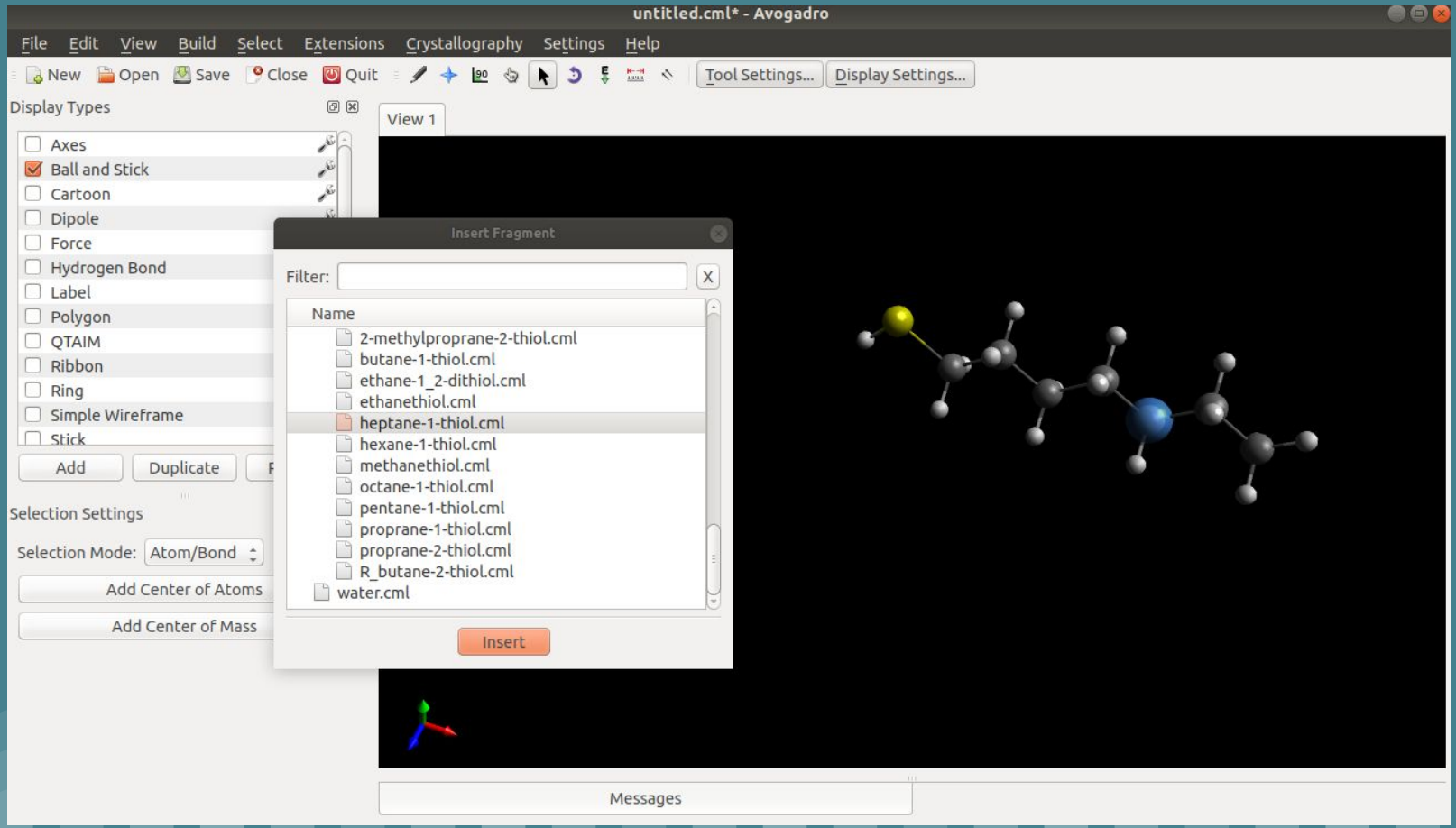
avogadro.cc



Tipo de programa	aplicación informática software libre
Modelo de desarrollo	Software Libre
Desarrollador	Open Molecules
Género	Modelización Molecular
Programado en	C++ (Qt)
Sistema operativo	Linux, OS X, Windows
Licencia	GNU General Public License
En español	✓ Sí

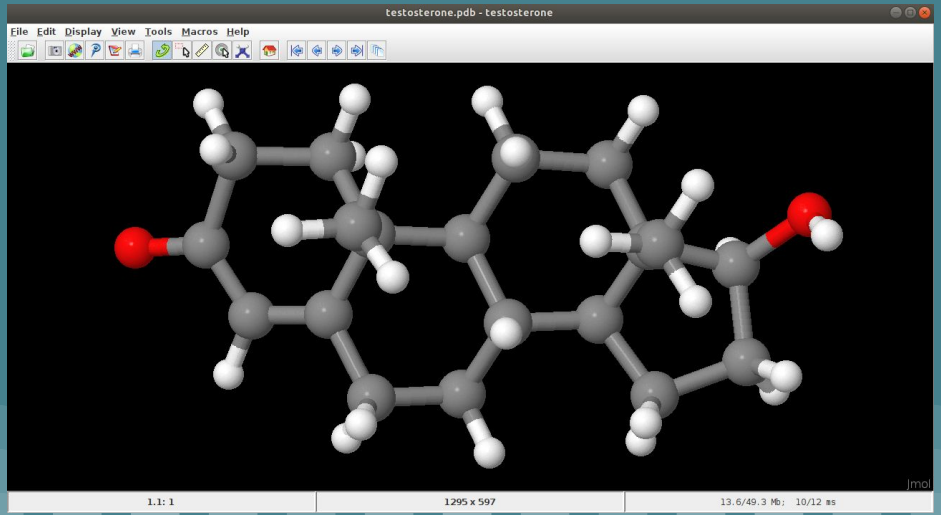
[editar datos en Wikidata]

Visualizador molecular (Open Source): Avogadro



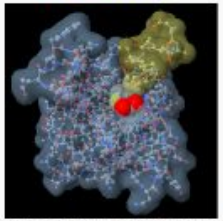
Visualizador molecular (Open Source): jmol

Jmol es un visor de código abierto de estructuras químicas en 3D.[2] Jmol devuelve una representación tridimensional de una molécula que puede usarse como herramienta de enseñanza[3] o para la investigación, por ejemplo en química y bioquímica. Es software libre y de código abierto, escrito en Java y por ello se puede ejecutar en Windows, Mac OS X, Linux y sistemas Unix. Existe una **aplicación independiente** y un kit de herramientas de desarrollo que puede integrarse en otras aplicaciones Java.



Jmol

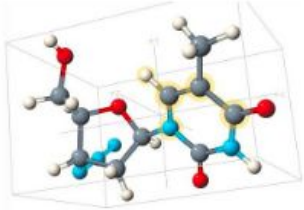
www.jmol.org y wiki.jmol.org



Jmol es un visor de Java moleculares para estructuras químicas en 3D

Tipo de programa	software libre
Desarrollador	equipo de desarrollo de Jmol
Última versión estable	13.0.4 10 de septiembre de 2012 (7 años, 7 meses y 1 día)
Última versión en pruebas	13.1.4 10 de septiembre de 2012 (7 años, 7 meses y 1 día)
Género	Modelización molecular
Programado en	Java
Sistema operativo	multiplataforma
Plataforma	máquina virtual Java
Licencia	GPL
Idiomas	catalán, chino, checo, danés, holandés, inglés, francés, alemán, húngaro, indonesio,

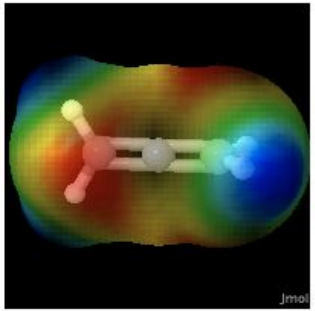
Visualizador molecular (Open Source): jmol



Display of bounding box and axes of coordinates space. Note the transparency of the yellow halo around selected atoms.

Representación de la caja limitante y los ejes de las coordenadas espaciales. Observa la transparencia del halo amarillo alrededor de los átomos seleccionados.

Affichage de la boîte englobante et des axes du système de coordonnées. Il faut noter la transparence du halo jaune autour des atomes sélectionnés.



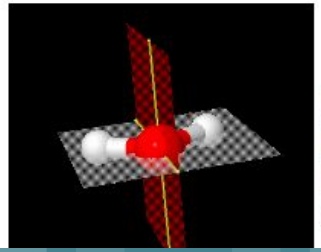
The electrostatic potential of allene mapped onto a translucent surface.

Potencial electrostático del aleno, mapeado sobre una superficie translúcida.

Le potentiel électrostatique de l'*allene* mappé sur une surface translucide.

(Nick Greeves)

Ver galería de imágenes en la página web de jmol
http://jmol.sourceforge.net/screenshots/



Two translucent planes (pMesh) illustrating symmetry planes for water molecule.

Dos planos translúcidos (pMesh) ilustrando los planos de simetría para la molécula de agua.

Deux plans translucides (pMesh) illustrant les plans de symétrie pour une molécule d'eau.

(Bob Hanson)

Visualizador molecular (Open Source): VMD

VMD

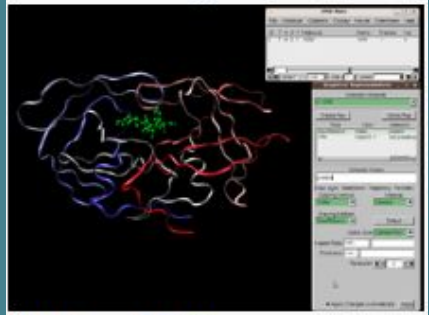
VMD (**V**isual **M**olecular **D**ynamics) es un programa de modelamiento molecular y visualización de estructuras.[1] VMD fue principalmente desarrollado como una herramienta para ver y analizar los resultados de las simulaciones de dinámica molecular, pero también incluye herramientas para trabajar con datos de volumen, secuencias y objetos gráficos arbitrarios como conos, cilindros o esferas. Las escenas mostradas pueden ser exportadas a herramientas de renderizado como POV-Ray, Renderman, Tachyon, VMRL y muchas otras. Los usuarios pueden ejecutar sus propios scripts de TCl y Python, dado que VMD también incluye intérpretes para estos lenguajes. VMD es gratuito y de código libre..[2]

Historia [editar]

VMD fue desarrollado por "Theoretical and Computational Biophysics Group" de la Universidad de Illinois y el Instituto Beckman. La versión original, llamada VRChem fue desarrollada en 1992 por Mike Krough, Bill Humphrey y Rick Kufrin. La versión inicial de VMD fue escrita por William Humphrey, Andrew Dalke, Ken Hamer, Jon Leech y James Phillips en 1995.

VMD

[1]



Tipo de programa	software
Desarrollador	Universidad de Illinois en Urbana-Champaign
Última versión estable	1.9.2 (info) 29 de diciembre de 2014 (5 años, 3 meses y 10 dias)
Género	Modelamiento molecular
Programado en	C
Sistema operativo	Multiplataforma
Plataforma	«x86», «x86-64»
Licencia	University of Illinois license
Estado actual	Con soporte
Idiomas	1
En español	✗ No

[editar datos en Wikidata]

Visualizador molecular (Open Source): VMD

NIH CENTER FOR MACROMOLECULAR MODELING & BIOINFORMATICS | UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Type Keywords SEARCH

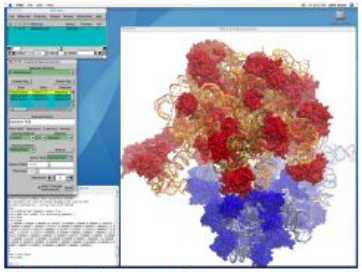
THEORETICAL *and* COMPUTATIONAL BIOPHYSICS GROUP

Home | Research | Publications | Software | Instruction | News | Galleries | Facilities | About Us

- Home
- Overview
- Publications
- Research
- Software
  - VMD Molecular Graphics Viewer
  - NAMD Molecular Dynamics Simulator
  - BioCoRE Collaboratory Environment
  - MD Service Suite
  - Structural Biology Software Database
  - Computational Facility
- Outreach

What is VMD?

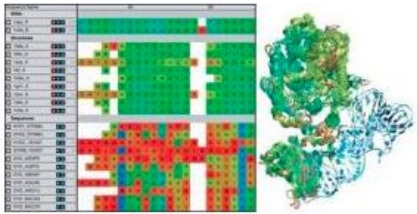
VMD is designed for modeling, visualization, and analysis of biological systems such as proteins, nucleic acids, lipid bilayer assemblies, etc. It may be used to view more general molecules, as VMD can read standard Protein Data Bank (PDB) files and display the contained structure. VMD provides a wide variety of methods for rendering and coloring a molecule: simple points and lines, CPK spheres and cylinders, licorice bonds, backbone tubes and ribbons, cartoon drawings, and others. VMD can be used to animate and analyze the trajectory of a molecular dynamics (MD) simulation. In particular, VMD can act as a graphical front end for an external MD program by displaying and animating a molecule undergoing simulation on a remote computer.



VMD 1.8.5 Screen Shot

Key features of VMD include:

- Support for all major computer platforms
- Support for multicore processors
- Support for **GPU accelerated computation**
- Many excellent **VMD tutorials** developed locally, and by the research community at large
- No limits on the number of molecules, atoms, residues or number of trajectory frames, except available memory
- Many molecular rendering and coloring methods
- Stereo display capability



Formatos de archivos típicamente soportados por los visualizadores moleculares

Contents [hide]

1 Distinguishing formats
2 Chemical Markup Language
3 Protein Data Bank Format
4 GROMACS format
5 CHARMM format
6 GSD format
7 Ghemical file format
8 SYBYL Line Notation
9 SMILES
10 XYZ
11 MDL number
12 Other common formats
13 Converting between formats
14 The Chemical MIME Project
14.1 Support
15 Sources of chemical data
16 See also
17 References
18 External links

Chemical Markup Language [edit]

Chemical Markup Language (CML) is an open standard for representing molecular and other chemical data. The open source project includes XML Schema, source code for parsing and working with CML data, and an active community. The articles Tools for Working with Chemical Markup Language and XML for Chemistry and Biosciences discusses CML in more detail. CML data files are accepted by many tools, including JChemPaint, Jmol, XDrawChem and MarvinView.

Protein Data Bank Format [edit]

The Protein Data Bank Format is commonly used for proteins but it can be used for other types of molecules as well. It was originally designed as, and continues to be, a fixed-column-width format and thus officially has a built-in maximum number of atoms, of residues, and of chains; this resulted in splitting very large structures such as ribosomes into multiple files. However, many tools can read files that exceed those limits. For example, the E. coli 70S ribosome was represented as 4 PDB files in 2009: 3I1M, 3I1N, 3I1O and 3I1P. In 2014 they were consolidated into a single file, 4V6C.

XYZ [edit]

The XYZ file format is a simple format that usually gives the number of atoms in the first line, a comment on the second, followed by a number of lines with atomic symbols (or atomic numbers) and cartesian coordinates.

Converting between formats [edit]

OpenBabel and JOELib are freely available open source tools specifically designed for converting between file formats. Their chemical expert systems support a large atom type conversion tables.

```
babel -i input_format input_file -o output_format output_file
```

For example, to convert the file epinephrine.sdf in SDF to CML use the command

```
babel -i sdf epinephrine.sdf -o cml epinephrine.cml
```

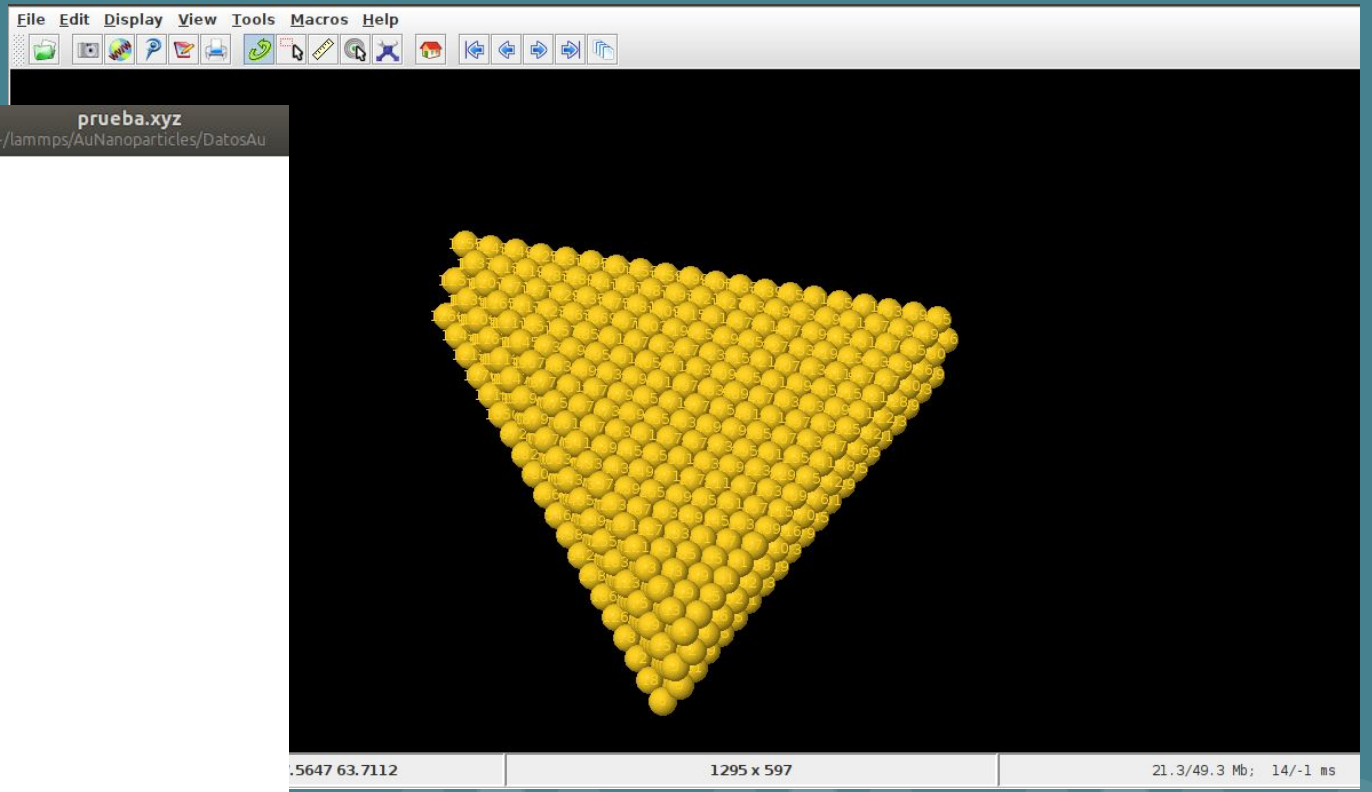
The resulting file is epinephrine.cml.

A number of tools intended for viewing and editing molecular structures are able to read in files in a number of formats and write them out in other formats. The tools JChemPaint (based on the Chemistry Development Kit), XDrawChem (based on OpenBabel), Chime, Jmol, Mol2mol[5][*citation needed*] and Discovery Studio fit into this category.

Ejemplo formato XYZ

Open ▾ prueba.xyz
~/lammps/AuNanoparticles/DatosAu

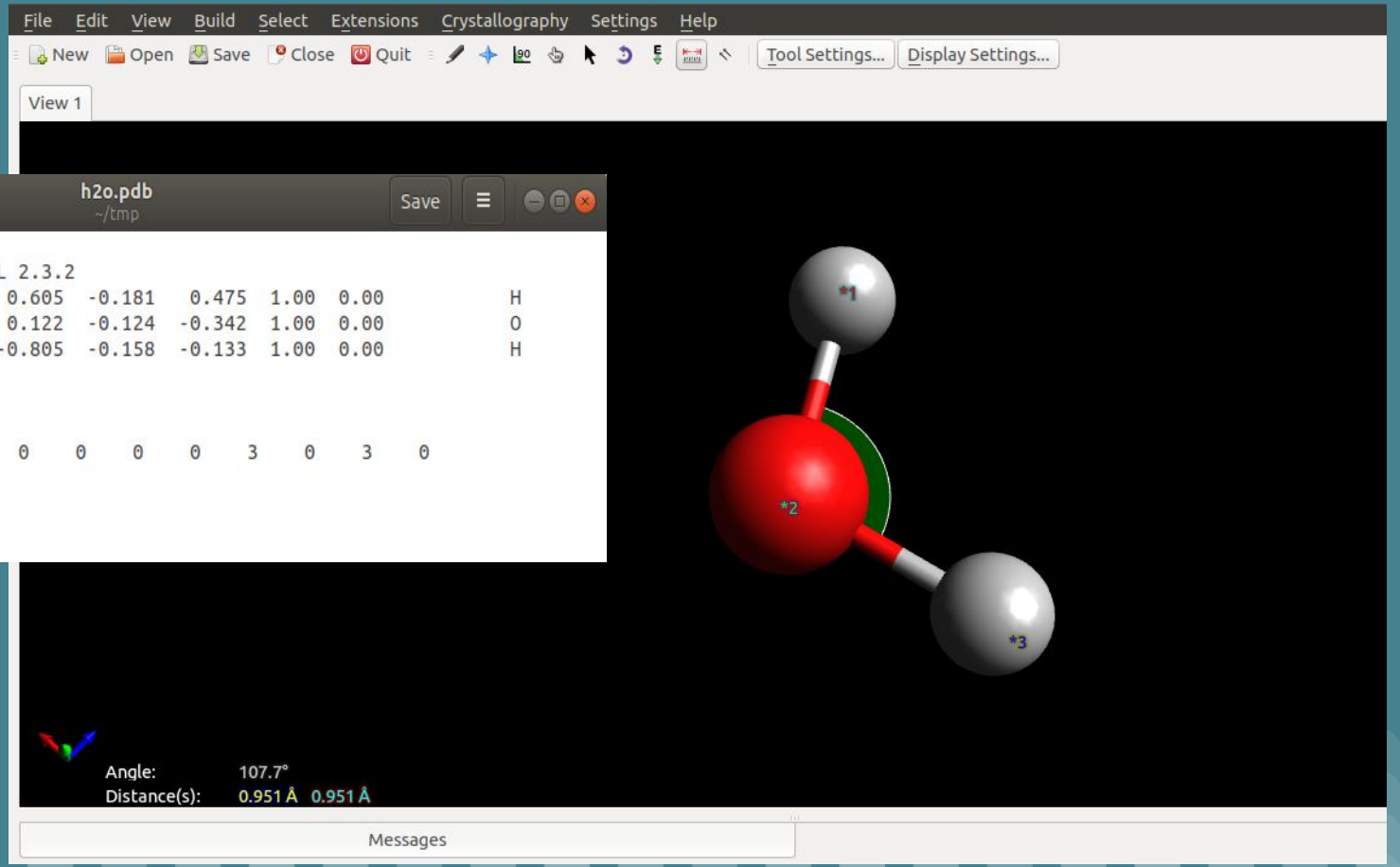
```
1260
Atoms. Timestep: 0
Au 44.795 35.8737 51.9128
Au 43.35 36.7079 54.2725
Au 43.35 35.0394 56.6322
Au 44.795 35.8737 58.9919
Au 43.35 36.7079 61.3516
Au 43.35 35.0394 63.7112
Au 46.24 38.3765 51.9128
Au 44.795 39.2107 54.2725
Au 44.795 37.5422 56.6322
Au 46.24 38.3765 58.9919
Au 44.795 39.2107 61.3516
Au 44.795 37.5422 63.7112
Au 47.685 35.8737 51.9128
Au 46.24 36.7079 54.2725
Au 46.24 35.0394 56.6322
Au 47.685 35.8737 58.9919
Au 46.24 36.7079 61.3516
Au 46.24 35.0394 63.7112
Au 49.13 38.3765 51.9128
Au 47.685 39.2107 54.2725
Au 47.685 37.5422 56.6322
Au 49.13 38.3765 58.9919
Au 47.685 39.2107 61.3516
Au 47.685 37.5422 63.7112
Au 47.685 40.8793 51.9128
Au 46.24 41.7136 54.2725
Au 46.24 40.045 56.6322
Au 47.685 40.8793 58.9919
Au 46.24 41.7136 61.3516
Au 46.24 40.045 63.7112
```



Ejemplo formato PDB

Open ▾ h2o.pdb ~/tmp Save

```
COMPND    UNNAMED
AUTHOR    GENERATED BY OPEN BABEL 2.3.2
HETATM    1  H   HOH     0       0.605  -0.181   0.475  1.00  0.00           H
HETATM    2  O   HOH     1       0.122  -0.124  -0.342  1.00  0.00           O
HETATM    3  H   HOH     0      -0.805  -0.158  -0.133  1.00  0.00           H
CONECT    1    2
CONECT    2    1    3
CONECT    3    2
MASTER        0    0    0    0    0    0    0    0    3    0    3    0
END
```



Referencias:

- *Introduction to Molecular Dynamics Simulation*, Michael P. Allen Computational Soft Matter: From Synthetic Polymers to Proteins, Lecture Notes.
- https://lammps.sandia.gov/
- https://avogadro.cc/
- http://jmol.sourceforge.net/
- https://www.ks.uiuc.edu/Research/vmd/