aLENS

Release 0.1

Adam Lamson <alamson@flatironinstitute.org>

May 27, 2023

CONTENTS

Note: This project is under active development.

CHAPTER

ONE

CONTENTS

1.1 aLENS quickstart

This a *gentle* introduction to running aLENS on a local computer using docker.

1.1.1 Pre-software installation

- Docker desktop
  - Sign up at docker hub https://hub.docker.com/
  - Download docker desktop https://www.docker.com/products/docker-desktop/
  - (Optional) Docker engine
    - Linux https://www.linux.com/topic/desktop/how-install-and-use-docker-linux/
    - Mac https://medium.com/crowdbotics/a-complete-one-by-one-guide-to-install-docker-on-your-mac-os-using-homebre
- paraview https://www.paraview.org/download/
- hdf5 view
  - Sign up for service https://www.hdfgroup.org/register/
  - Download at https://www.hdfgroup.org/downloads/hdfview/#download

1.1.2 Creating a container

1. Pull latest docker image from dockerhub

```
docker pull lamsoa729/alens:latest
```

2. Make a folder to access your simulation data stored generated in docker container

```
mkdir my_alens_data; cd my_alens_data
```

3. Create and run a docker container from the image pulled from dockerhub

```
docker run --volume=<path/to/my_alens_data>:/root/Run --name alens -dit lamsoa729/
↪alens:latest
```

replacing <path/to/my_alens_data> with the path to the directory made in the previous step. For completeness, here are the descriptions of the used options:

- `--volume`: option mounts the containers `/root/Run` directory to your local machines `<path/to/my_alens_data>` folder so you can interact with data from your local machine. This is useful for playing with visualizations and post-processing of data without increasing the size or memory usage of the container.
- `--name`: Gives the container a name to easily identify what the container is for and access its command line later interface (CLI) later.
- `-dit`: Combined shorthand options of `--detach`, `--interactive`, and `--tty`. This essentially runs the container perpetually in the background until you manually stop it. While it is running, you can interact with it through a terminal – originally called a TeleTYpe (TTY) for historical reasons.

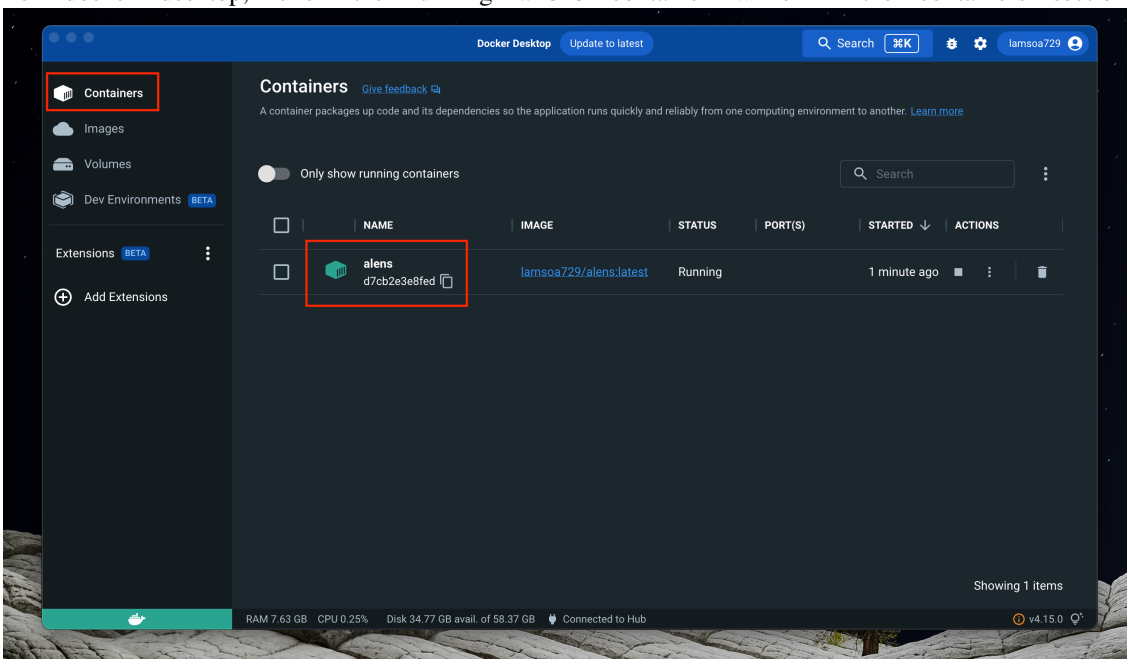
You now have access to an environment that can run aLENS but will create data files on your local computer.

1.1.3 Running aLENS inside docker containter

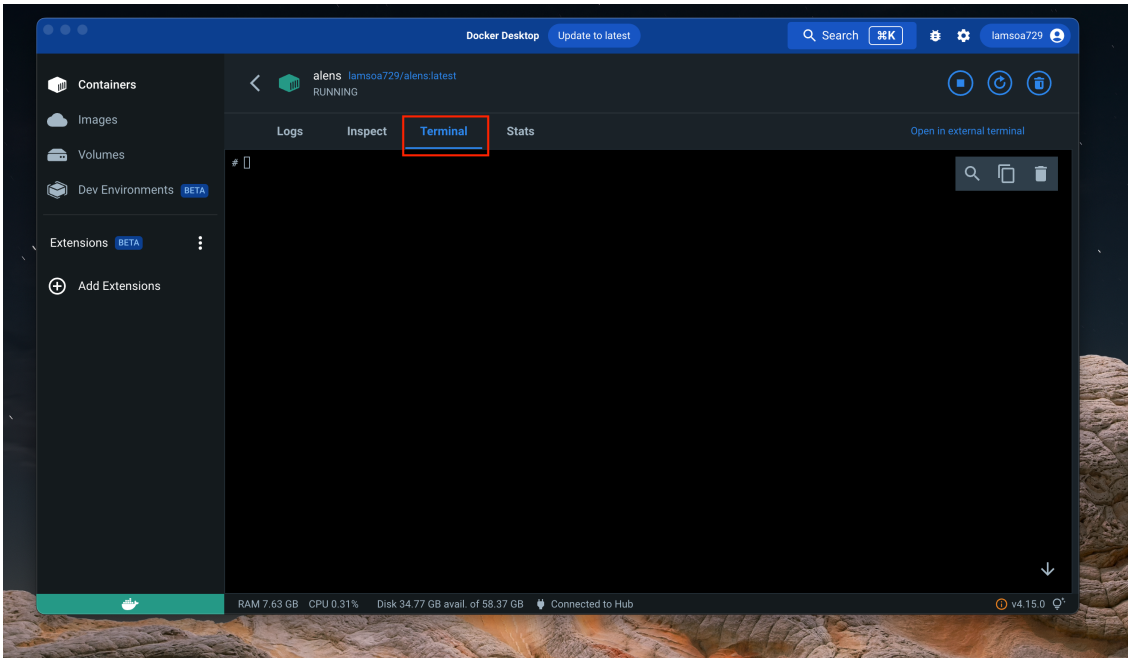
1. To open a CLI with your docker container, you can either use your native terminal or docker desktop's GUI.
   - For native terminal

```
docker exec -it alens /bin/bash
```

   - For docker desktop, click the running `alens` container while in the containers section



then click the `terminal` tab in the upper middle of the window



You may treat this CLI just like any terminal that is connected to a remote server.

2. From this CLI, navigate to the `Run` directory

```
cd /root/Run
```

3. While still in the CLI, copy the example configuration to the data folder

```
cp -r ~/aLENS/Examples/MixMotorSliding .
cd MixMotorSliding
```

4. Copy the contents of the `Run` template directory from aLENS to the data folder as well

```
cp -r ~/aLENS/Run/* .
```

You should now see an `aLENS.X` executable in this directory along with `result` and `script` directories containing useful scripts for processing, storing, and cleaning up generated files.

5. Run *aLENS*

```
./aLENS.X
# or to control the number of cores used
OMP_NUM_THREADS=<number_of_cores> ./aLENS.
```

6. Stop the run by pressing `[ctrl+c]`

7. Execute run again as we did in step 4. Notice that the aLENS simulation continues from the last snapshot. This is a very useful restart feature for longer runs.

1.1.4 Parameter and initial configuration files

The executable `aLENS.X` reads 2 input files (4 if specifying starting object configurations):

- `RunConfig.yaml` specifies simulation parameters for the system and rod-like objects (sylinders).
- `ProteinConfig.yaml` specifies types and parameter of crosslinking and motor objects (proteins).
- `TubuleInitial.dat` specifies initial configuration of sylinders (optional).
- `ProteinInitial.dat` specifies initial configuration of proteins (optional).

Example parameter config files

`RunConfig.yaml`:

```yaml
#Example of system and sylinder configuration file
conMaxIte: 10000
conResTol: 1e-5
conSolverChoice: 0
logLevel: 3
monolayer: false
rngSeed: 1234
simBoxHigh:
  - 20.0
  - 1.0
  - 1.0
simBoxLow:
  - 0.0
  - 0.0
  - 0.0
simBoxPBC:
  - false
  - false
  - false
sylinderColBuf: 1.0
sylinderDiameter: 0.025
sylinderDiameterColRatio: 1.0
sylinderFixed: false
sylinderLength: 0.5
sylinderLengthColRatio: 1.0
sylinderLengthSigma: 0
sylinderNumber: 4000
dt: 1.0e-05
timeSnap: 0.01
timeTotal: 10.0
timerLevel: 3
viscosity: 1.0
```

`ProteinConfig.yaml`:

```yaml
#Example of crosslinker and motor configuration file
KBT: 0.00411 # pN.um, at 300K
proteins:
    - tag: 0 # Type 0, active Kinesin-1
```

(continues on next page)

(continued from previous page)

```
    #properties:
    walkOff: true
    PtoAPratio: 1.0
    fixedEnd0: true
    freeLength: 0.05 # um
    rc: 0.038 # um ( freeLength/2 + D/2 )
    kappa: 100 # pN/um
    fstall: 7.0 # pN
    lambda: 0.5 # dimensionless, energy dependent unbinding
    vmax: [0, 1.0] # um/s
    diffUnbound: 1.0 # 0.436 um^2/s when viscosity=0.02
    diffBoundS: [0.0, 1.0e-2] # um^2/s
    diffBoundD: [0.0, 1.0e-2] # um^2/s
    # KMC parameters
    useBindVol: false
    lookupType: 1
    lookupGrid: 2048
    eps: 400 # um^{-1}
    Ka: [0, 10.0] # (uM)^{-1}
    ko_s: [0, 1.0] # 1/s or [0.3, 10.0]
    Ke: [0, 10.0] # (uM)^{-1}
    ko_d: [0, 1.0] # 1/s or [0.3, 10.0]
    #numbers:
    freeNumber: 0
    fixedLocationPerMT: [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,␣
↪2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,␣
↪2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,␣
↪2] # 80
    - tag: 1 # Type 0, passive Kinesin-1
    #properties:
    walkOff: true
    PtoAPratio: 1.0
    fixedEnd0: true
    freeLength: 0.05 # um
    rc: 0.038 # um ( freeLength/2 + D/2 )
    kappa: 100 # pN/um
    fstall: 7.0 # pN
    lambda: 0.5 # dimensionless, energy dependent unbinding
    vmax: [0, 0.0] # um/s
    diffUnbound: 1.0 # 0.436 um^2/s when viscosity=0.02
    diffBoundS: [0.0, 0.0] # um^2/s
    diffBoundD: [0.0, 0.0] # um^2/s
    # KMC parameters
    useBindVol: false
    lookupType: 1
    lookupGrid: 2048
    eps: 400 # um^{-1}
    Ka: [0, 10.0] # (uM)^{-1}
    ko_s: [0, 0.1] # 1/s or [0.3, 10.0]
    Ke: [0, 10.0] # (uM)^{-1}
    ko_d: [0, 0.1] # 1/s or [0.3, 10.0]
    #numbers:
```

(continues on next page)

(continued from previous page)

```
    freeNumber: 0
    fixedLocationPerMT: [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2] # 20
```

NOTE: All default values and further explanation of parameters can be found in `SimToolbox/Sylinder/SylinderConfig.hpp` and `Protein/ProteinConfig.hpp` files in the `aLENS` gitrepo.

Initial configuration file lines (optional)

`TubuleInitial.dat`

First two lines: Number of sylinders, time step. (These do not get read for initial conditions.)

	Sylinder type	Global ID	Radius	Minus end x-pos	Minus end y-pos	Minus end z-pos	Plus end x-pos	Plus end y-pos	Plus end z-pos	Group
Option/parameter type	'C' for regular cylinder or 'S' stationary	int	float	float	float	float	float	float	float	int
Example line	C	0	.0125	0	0.5	0.57	20	0.5	0.57	-1

`ProteinInitial.dat`

First two lines: Number of proteins, time step. (These do not get read for initial condition.)

Name	Protein character	Global ID	Protein tag	End 0 x-pos	End 0 y-pos	End 0 z-pos	End 1 x-pos	End 1 y-pos	End 1 z-pos	End 0 bind ID	End 1 bind ID
Option/parameter type	'P'	int	int	float	float	float	float	float	float	int (-1 if not bound)	int (-1 if not bound)
Example line	P	41	0	8.85976	0.5	0.5	8.85976	0.5	0.5	-1	2

NOTE: For example initial configuration files, navigate to `Examples/MixMotorSliding/TubuleInitial.dat` and `Examples/MixMotorSliding/ProteinInitial.dat`. If no configuration files are present in the directory `aLENS.X` is run, objects will be generated according to the parameters in `RunConfig.yaml` and `ProteinConfig.yaml`.

1.2 Installing aLENS from scratch

1.2.1 General tasks to do beforehand if developing aLENS

- Get github account and collaborator status on aLENS and SimToolbox
- Install git and setup up token access in global git config
- (Optional) Get Intel license to install MKL on system

1.2.2 Ubuntu (version 22)

WARNING: Make sure you have more than 2Gb of RAM and 30Gb of hard drive storage. There are work arounds for reduced memory and storage but these are not covered in this tutorial.

1. (Optional) If on virtual machine set up port forwarding to ssh into desktop SSH into virtual machine in virtual box
2. Upgrade to make sure you have latest `apt` version

```
sudo apt update
sudo apt upgrade
```

3. Install MKL on system (from this blog)

```
wget -P /tmp [https://apt.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-
↪PRODUCTS-2019.PUB](https://apt.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-
↪SW-PRODUCTS-2019.PUB)
sudo apt-key add /tmp/GPG-PUB-KEY-INTEL-SW-PRODUCTS-2019.PUB
sudo sh -c 'echo deb https://apt.repos.intel.com/mkl all main > /etc/apt/sources.
↪list.d/intel-mkl.list'
sudo apt update
sudo apt install intel-mkl-64bit-2020.0-088
```

   `apt-key` command may throw a warning. This is fine, but for further reading, please see: What commands (exactly) should replace the deprecated apt-key?

4. Tell computer where to find mkl by setting

```
export MKLROOT=/opt/intel/mkl
```

   Linux usually installs the mkl package in `/opt` directory but check to make sure `intel/mkl` is there. Otherwise, go searching for it. Be sure to add `MKLROOT` variable to `.bashrc` file so that you don't always have to type above command when you open a new terminal. Quick way to add this command to your path is with the command

```
echo 'export MKLROOT=/opt/intel/mkl' >> ~/.bashrc
```

5. Install other necessary dependencies

```
sudo apt install gcc g++ cmake openmpi-bin openmpi-doc libopenmpi-dev lbzip2
```

   Make sure your `make` version <4.3 otherwise Trilinos12 will not compile. This can be installed by first removing `make` and then downloading and installing the debian package

```
sudo apt-get remove make
wget http://ftp.de.debian.org/debian/pool/main/m/make-dfsg/make_4.2.1-1.2_amd64.deb
sudo dpkg -i make_4.2.1-1.2_amd64.deb
```

6. Make directories to store program files

```
mkdir -p ~/projects ~/local/aLENS
```

7. Create project structure and clone

```
cd ~/projects
git clone --recursive git@github.com:flatironinstitute/aLENS.git
cd aLENS/Dep
```

8. Download depency softwares using the provided script

```
python3 download_all.py
```

   This can take longer than you think. Just be patient.

9. Make necessary changes to `compile_all.py` file which includes changing installation directory to

```
# your installation destination, use ABSOLUTE PATH
##From
#os.environ["SFTPATH"] = os.environ['HOME'] + \
#    '/envs/alens_env'
##To
os.environ["SFTPATH"] = os.environ['HOME'] + \
    '/local/'
```

10. Compile all the dependencies

```
python3 compile_all.py
```

    This will take a ***LONG*** while. Go take a break and come back.

11. (Now following the installation guide on github landing page.) Go back to aLENS root directory and create a build directory

```
cd ..
mkdir build
```

12. Modify `cmake-example.sh` by replacing the variables `SFTPATH` and `CMAKE_INSTALL_PREFIX` definitions with

```
-D CMAKE_INSTALL_PREFIX="${HOME}/local/aLENS" \
-D SFTPATH="${HOME}/local" \
```

13. Go to build directory, run cmake script, and make executable

```
cd build
sh ../cmake-example.sh
make -j <number_of_compiling_cores>
```

    WARNING: Leaving <number_of_compiling_cores> blank creates threads equal to the number of cores on your local machine. If you do this with a virtual machine, it will most likely crash during compilation.

Compiling creates the executable file `aLENS.X` which you can run in any of the example simulations included in the `Examples` directory of aLENS.

1.2.3 Flatiron cluster (2022-05-18)

This is much easier but only because all dependencies are already installed on our module system.

1. Make useful directories to store program files

```
mkdir -p ~/projects ~/local/aLENS
```

2. Create project structure and clone

```
cd ~/projects
git clone --recursive git@github.com:flatironinstitute/aLENS.git
cd aLENS
```

3. Load essential modules

```
module purge
module load modules gcc cmake gsl boost lib/fftw3 intel-mkl openmpi4 trilinos/12.18.
↪1-mpi eigen vtk
```

4. Export environment variables

```
export MKL_INTERFACE_LAYER=GNU,LP64
export MKL_THREADING_LAYER=GNU
export BOOST_ROOT=$BOOST_BASE
export FFTWDIR=$FFTW3_BASE
unset OMPI_CC
unset OMPI_CXX
export OMP_DISPLAY_ENV=true
export OMP_MAX_ACTIVE_LEVELS=1
```

5. Modify `cmake-example.sh` by replacing SFTPATH and CMAKE_INSTALL_PREFIX definitions with

```
-D CMAKE_INSTALL_PREFIX="${HOME}/local/aLENS" \
-D SFTPATH="/cm/shared/sw/nix/store" \
```

6. Make and go to build directory, run cmake script, and make the executable

```
mkdir build
cd build
sh ../cmake-example.sh
make -j
```

This will produce an executable file `aLENS.X` that you can run in any of the simulation directory included in the `Examples` directory of aLENS.

1.3 Visualization

1.3.1 Overview of aLENS data files

aLENS produces 3 types of data files when run:

- <Object>Ascii_<Snapshot#>.dat: Contains positional, geometric, and state information of all <Object>s at <Snapshot#>. These are in the same format as the initial data files like TubuleInitial.dat or ProteinInitial.dat. (See *initial file configurations*.)
- <Object>_r<Rank#>_<Snapshot#>.vtp: XML vtk format in base64 binary encoding for all <Object> information. aLENS.X is written such that each MPI rank writes its own set of data to a unique vtp file marked by <Rank#>. These are not human readable but can be conveniently loaded into Paraview for visualization or read by VTK (either python or cpp) for data processing.
- <Object>_<Snapshot#>.pvtp: (pvtp = parallel vtp) An index to a set of vtp files (serial vtp), which holds the actual data. Therefore the number of vtp files in each pvtp file index is equal to the number of MPI ranks. The restriction is that the index pvtp file must appear in the same location as those vtp data files. For a comprehensive explanation of these pvtp files, read the official guide of vtk file format: https://kitware.github.io/vtk-examples/site/VTKFileFormats/#parallel-file-formats

The data files are saved in different folders, but for postprocessing and visualization, files of a given sequence must appear in the same folder otherwise postprocessing or visualization programs may fail to load the entire sequence. The python script Result2PVD.py solves this restriction. More on this later in the tutorial.

1.3.2 First visualization (using pre-made visualization file)

Visualizations are created and interacted with using paraview. Be sure to have this software installed https://www.paraview.org/download/ (This tutorial was made using ParaView-5.9.1). Tutorials for using paraview for general data visualization can be found on their website.

1. After running your *first simulation*, your results directory will have multiple result subdirectories inside of it.

```
$ cd ~/Run/MixMotorSliding/result
$ ls
Clean.sh  PNG  Result2PVD.py  compress.sh  result0-399  result400-799  result800-
↪1199  simBox.vtk  uncompress.sh
```

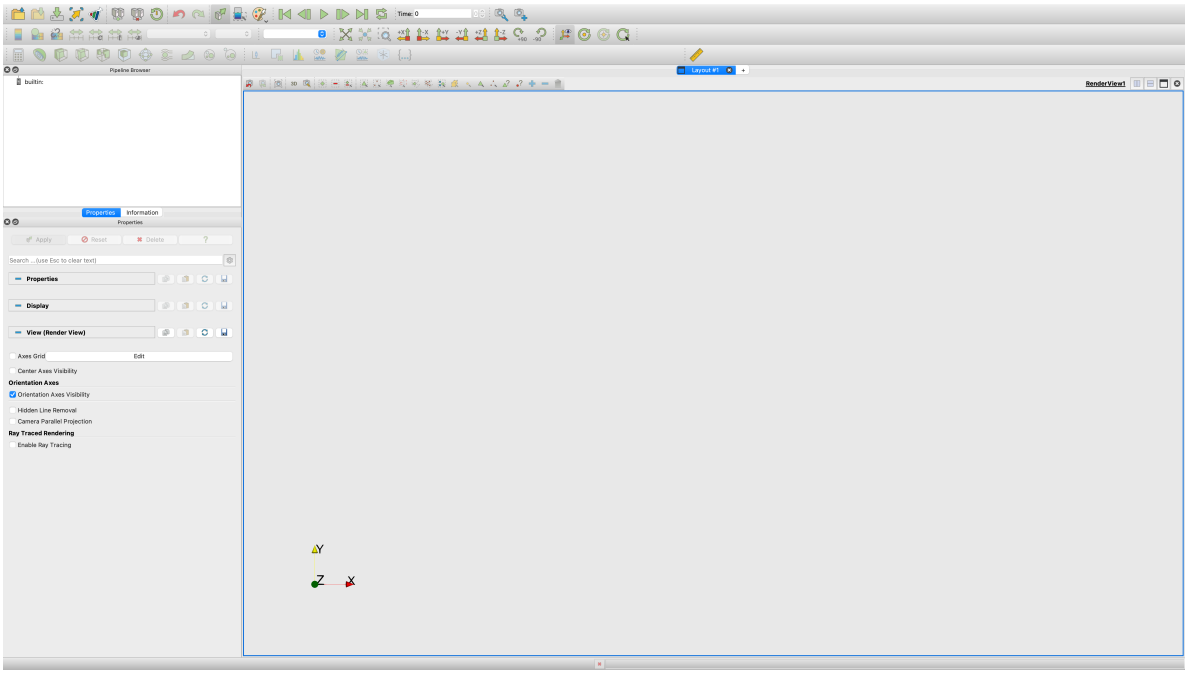
   NOTE: All generated data is held in the result<FirstSnapshot#>-<LastSnapshot#> directories. Data is divided among multiple directories to prevent too many files or too much data from being held in a single location, which can put unwanted stress on distributed files systems common among computing clusters. <FirstSnapshot#> specifies the earliest snapshot data held in that directory and <LastSnapshot#> the latest. The difference of these two numbers changes based on the number of objects in the simulation and number of MPI ranks used to run the simulation. Because of this, you can not change the number of ranks if you wish to restart a simulation using the builtin restart feature.

2. To visualize the data, we must first run the Result2PVD.py script.

```
$ python3 ./Result2PVD.py
$ ls ./*.pvd
./ConBlockpvtp.pvd  ./Proteinpvtp.pvd  ./Sylinderpvtp.pvd
```

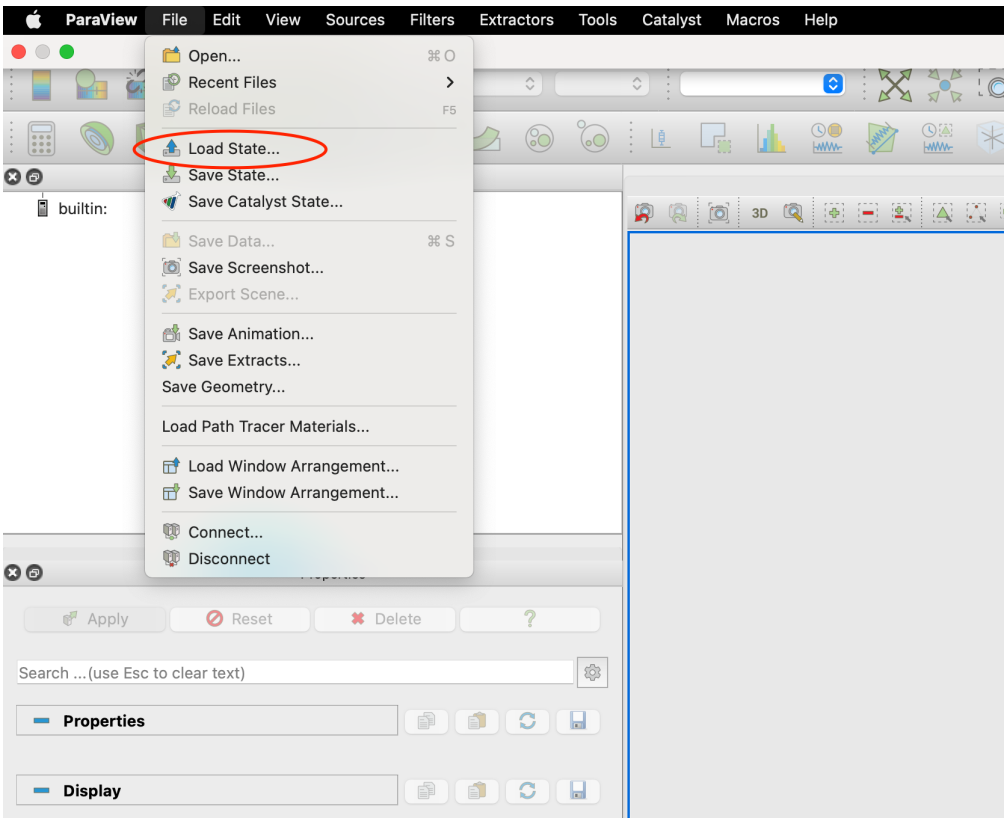
   This creates ParaView data (.pvd) files that point to the .pvtp files in the various result subfolders, allowing Paraview to load all the files belonging to a sequence in the correct order. It is safe to execute this script while aLENS.X is still running and writing data so analysis can occur before a simulation finishes.

3. Open up the paraview GUI.



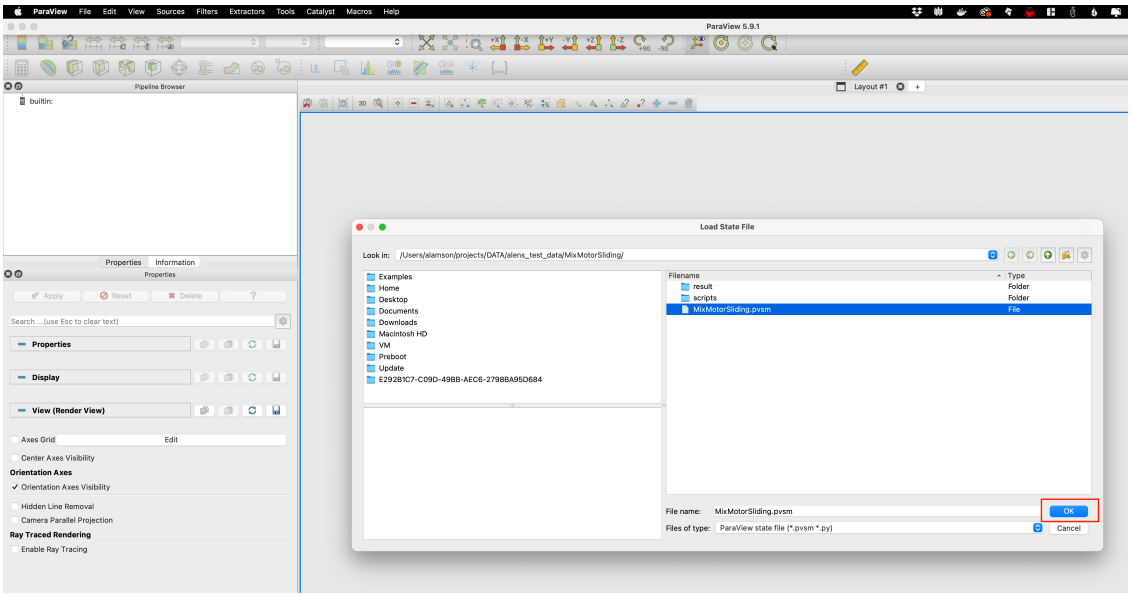
4. The quickest way to get interpretable visualizations is to load a pre-made ParaView state (.pvsm) file that has filters already applied to the data to be loaded. One has been provided in the MixMotorSliding example.

   - In the upper left hand corner, click `Files -> Load State`.

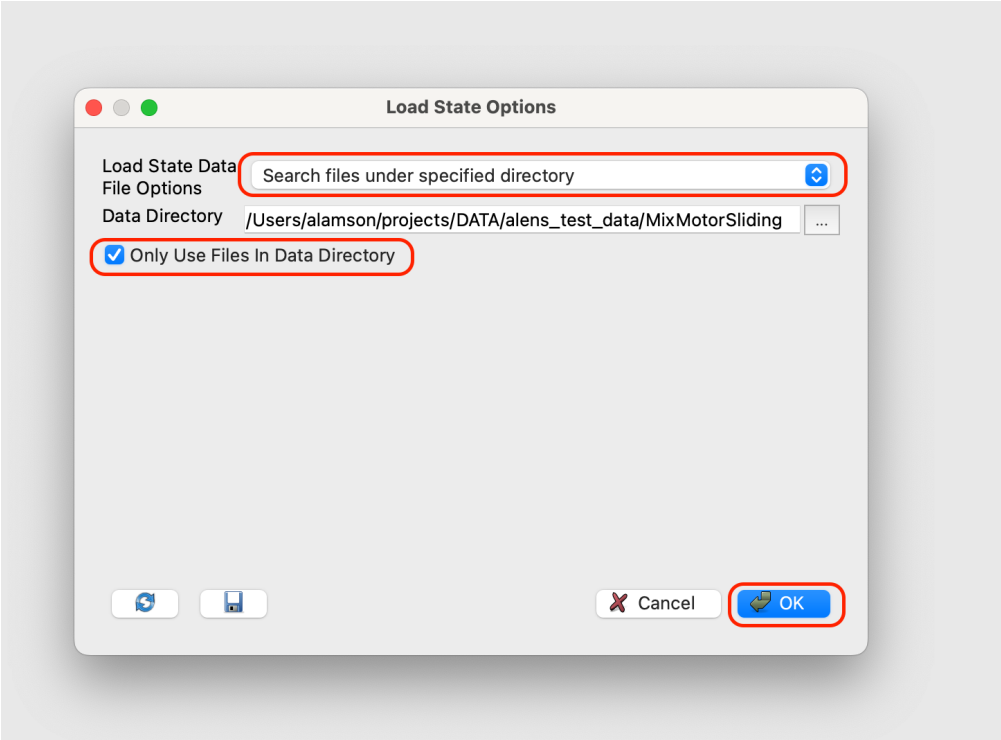


   - From the pop up window, navigate to the mounted Run directory on your local machine and click

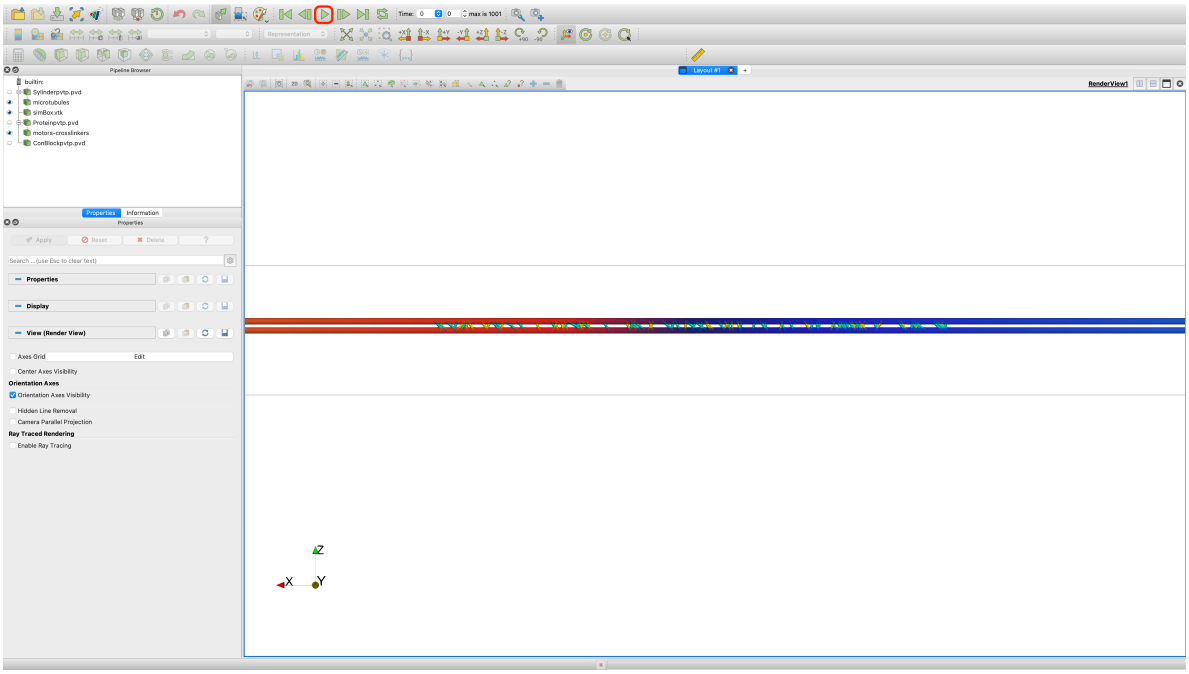
`MixMotorSliding.pvsm` and click OK.



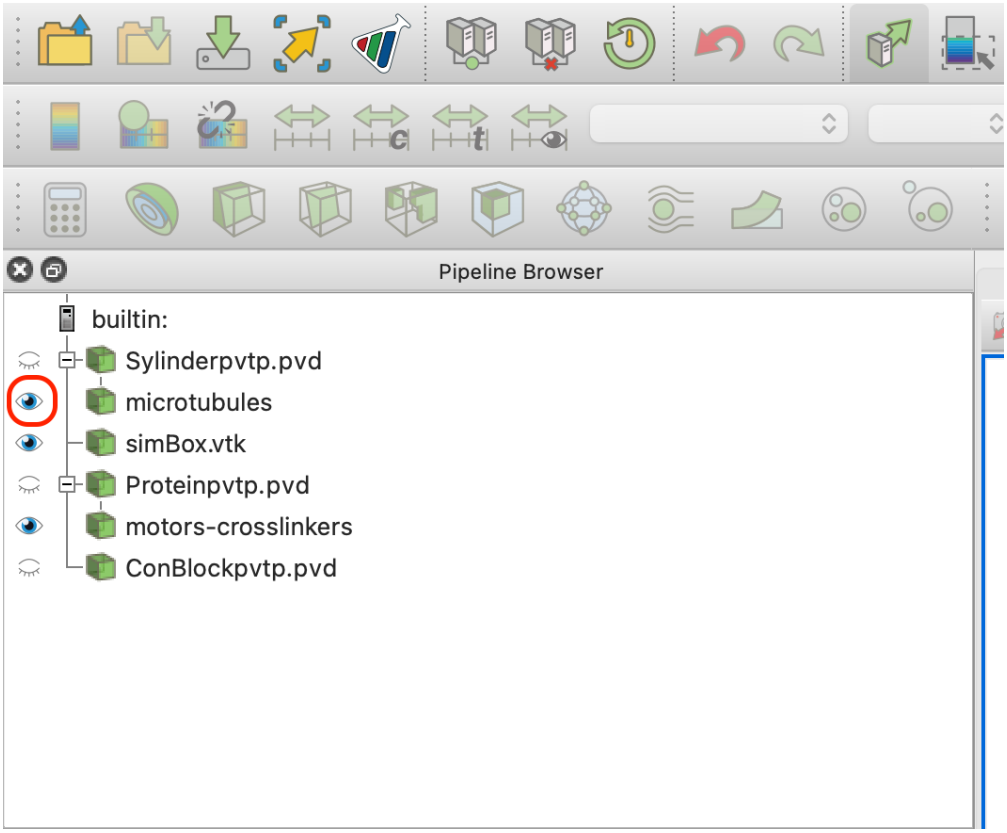
5. In Load State Options choose ‘Search files under specified directory. Data Directory should populate with the correct run directory on your local machine. Select ‘Only Use Files In Data Directory’ and press OK.

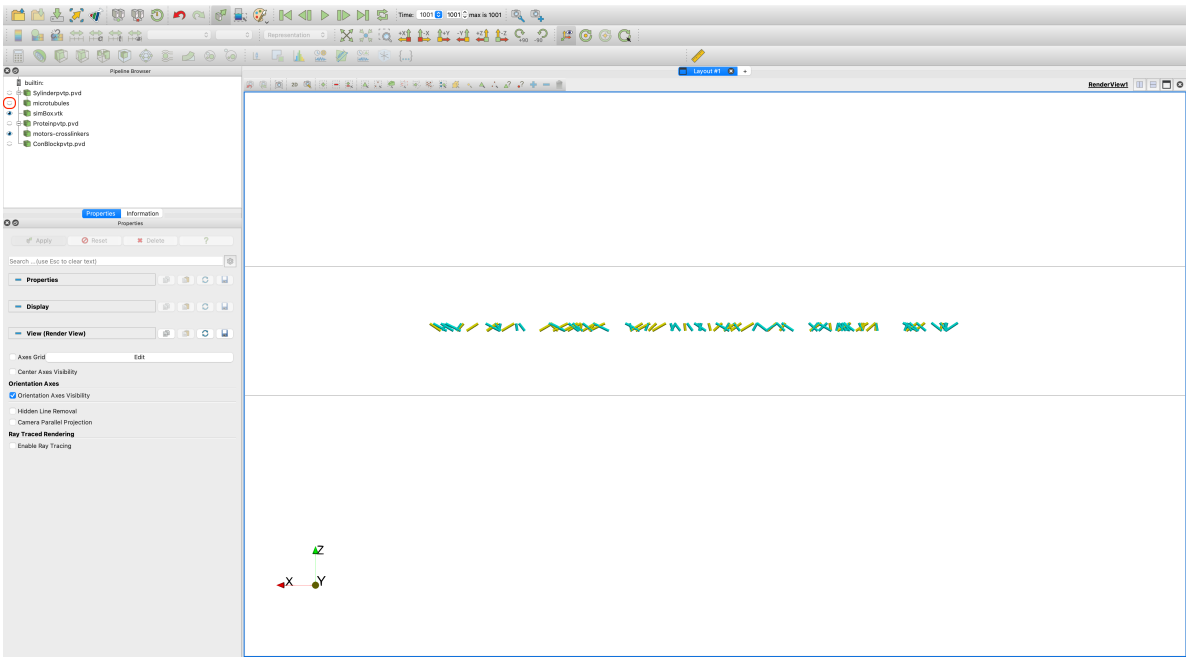


6. Run visualization using run button at the top of the screen.



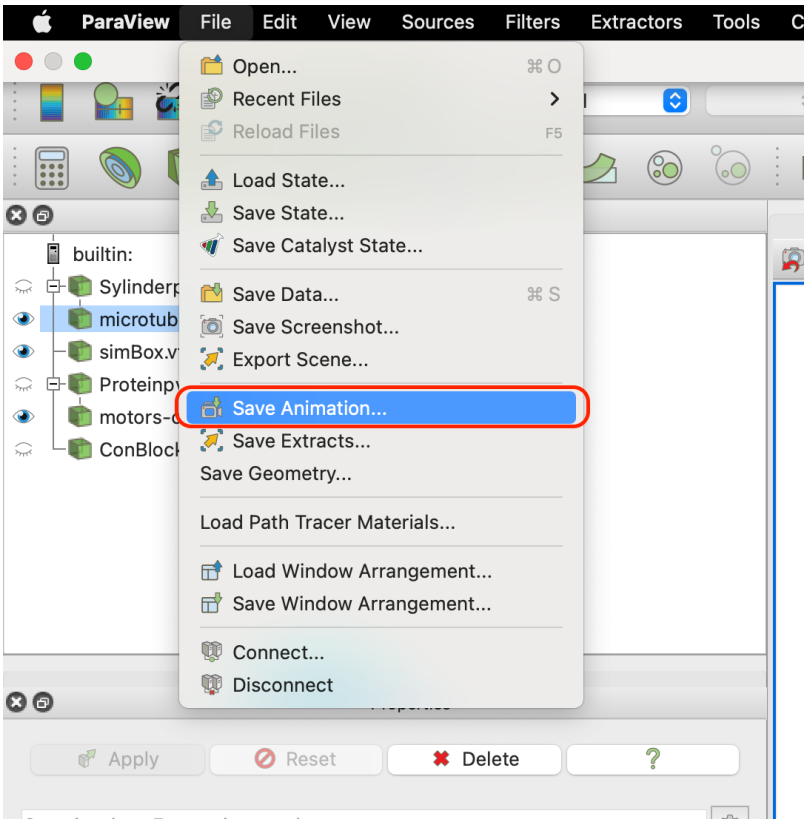
7. Objects in the simulation are shown on the left-hand bar and can be turned on and off by clicking on the eye icon next the the names. (The more objects shown, the slower the visualization will run)



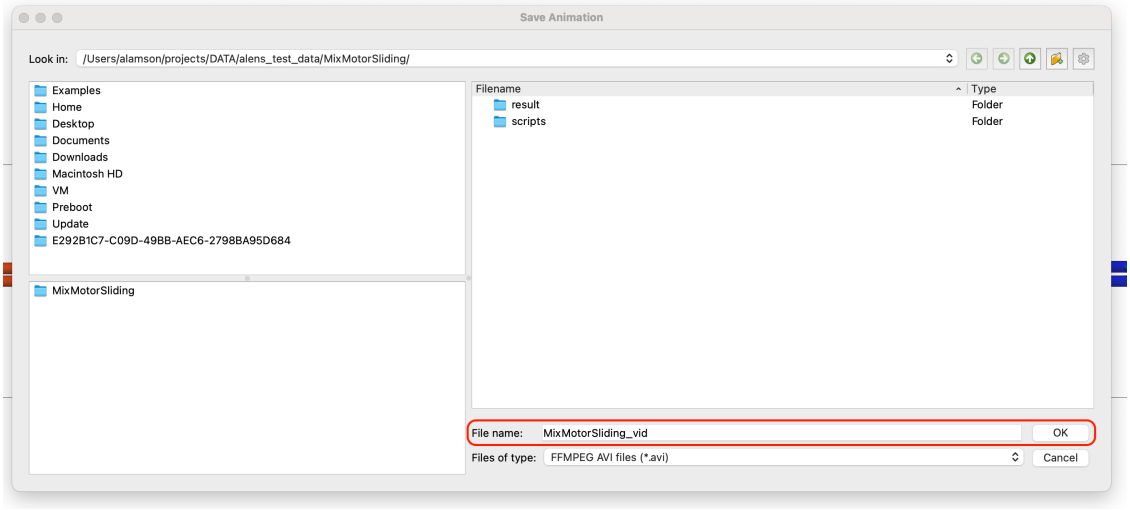


8. To save a movie, one can either save each frame to a PNG file and then stitch them together using FFMPEG (script provided in `result/PNG/MovieGen.sh`) or generate an .avi file from ParaViews internal functionality. We will use the latter for now.

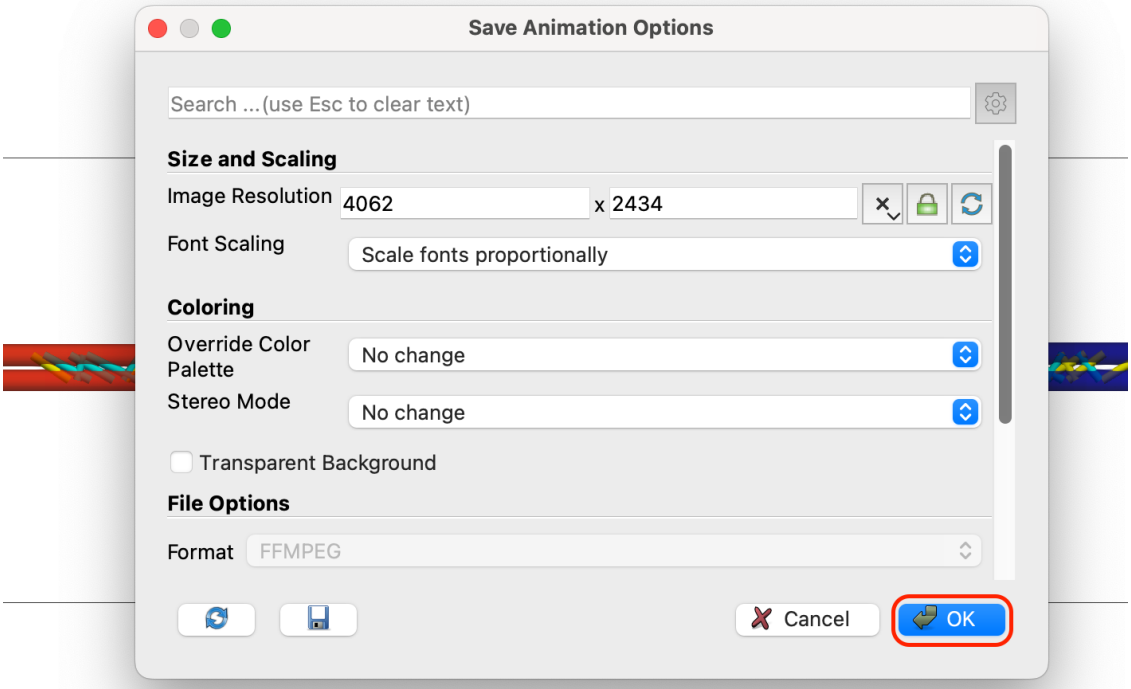
   - Click `File->Save Animation...`



   - Navigate to `MixMotorSliding` on your local if Save Animation window does not already bring your there and type in a name for your video file. Hit OK.



- Options are available to control size, frame rate, resolution, etc. of file. Hit OK when done.



You should now see a .avi file in your simulation directory.

1.3.3 Creating a your own ParaView state file (coming soon)