Aksimentiev Group
Department of Physics and
Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign

Multi-resolution simulations of self-assembled DNA nanostructures

Chris Maffeo and Aleksei Aksimentiev

Contents

1 Introduction **1**

2 Getting started **2**
2.1 Required software 2
2.2 Layout of the tutorial 3
2.3 Overview of the model 4

3 Multi-resolution simulations with the `mrdna` script **5**
3.1 Visualization and analysis 7
3.2 Solvent-free atomic simulations 11
3.3 Multi-resolution modeling for fast structural relaxation 11
3.4 Working with other design file formats 13

4 Detailed description of the model **15**

5 Description of the Python modeling framework **16**

6 Writing scripts using the `mrdna` package **19**

1 Introduction

The DNA origami method has brought nanometer-precision fabrication to molecular biology labs, offering myriads of potential applications in the fields of synthetic biology, biomolecular medicine, molecular computation, etc. Advancing the method further requires controlling self-assembly down to the atomic scale. Hence, fast, accurate and detailed structure prediction can advance the development of complex, functional DNA origami constructs. Towards this goal, we developed a protocol that unites the rapid and robust structural relaxation of low-resolution coarse-grained modeling with the detail and accuracy of atomistic molecular dynamics (MD) simulation. In this tutorial we will show you how to use the multi-resolution DNA (`mrdna`) Python package to perform such simulations.

2 Getting started

This tutorial introduces a Python3 package called `mrdna` that makes it easy to run simulations of self-assembled DNA nanostructures. The multi-resolution approach provides the benefits of coarse-grained modeling, while resulting in a high-resolution structure suitable for atomistic simulations.

Please note that this tutorial is written for the Linux platform and x86 architecture. Proceed at your own risk using other operating systems and architectures. Currently, a CUDA-enabled GPU is required for running simulations using the ARBD engine.

You should ideally be familiar with the Unix command line, the molecular visualization package VMD, and basic Python syntax. If any is unfamiliar, please refer to the resources below.

- Command line tutorial: http://rik.smith-unna.com/command_line_bootcamp
- VMD tutorial: https://www.ks.uiuc.edu/Training/Tutorials
- Python tutorial: http://www.learnpython.org (basic tutorials + numpy are recommended)

2.1 Required software

This tutorial requires Python3 with the `mrdna` and `jupyter` packages and all dependencies installed. Installing `mrdna` can be achieved by downloading or cloning the package from https://gitlab.engr.illinois.edu/tbgl/tools/mrdna, changing to the `mrdna` directory, and running `python3 setup.py install`. In addition to these Python packages, you must install the biomolecular visualization program VMD and the CUDA-accelerated simulation engine ARBD. Installation of the atomic simulation engine NAMD is recommended.

2.2 Layout of the tutorial

Tutorial is divided into five steps. This document will walk you through the first two steps, showing you how to run multi-resolution simulations using a command-line utility. Each of the remaining three steps contain an interactive Jupyter notebook, which teach you how to write scripts that manipulate multi-resolution models. The tutorial aims to be a comprehensive guide, and you will likely want to work through across at least two sessions. A good time to take a break is between `step2` and `step3`. The layout of directories and files in this tutorial is depicted below.

```
multi-resolution-dna-nanotechnology/
|-- multi-resolution-dna-nanotechnology.pdf
|-- README.md
|-- step1/
|   |-- curved-hextube.json
|   |-- example-output/
|   |-- hextube.json
|   |-- load-mrdna.tcl
|   |-- pris-6conn-86b.json
|   ‘-- slider-dist-exp.dat
|-- step2/
|   |-- box.mmp
|   |-- example-output/
|   |-- hextube-cando.pdb
|   |-- hextube-nab.pdb
|   |-- hextube-oxdna.pdb
|   |-- load-mrdna.tcl -> ../step1/load-mrdna.tcl
|   ‘-- vHelix/
|-- step3/
|   |-- basic-mrdna.ipynb
|   |-- example-output/
|   |-- hextube.json
|   |-- load-mrdna.tcl -> ../step1/load-mrdna.tcl
|   ‘-- notebook-src/
|-- step3.ipynb
|-- step4/
|   |-- advanced-mrdna.ipynb
|   |-- example-output/
|   |-- load-mrdna.tcl -> ../step1/load-mrdna.tcl
|   ‘-- pris-6conn-86b.json -> ../step1/pris-6conn-86b.json
‘-- step5/
    |-- 2-layer-plate.json
    |-- comsol-to-dx/
    |-- example-output/
    |-- load-mrdna.tcl -> ../step1/load-mrdna.tcl
    |-- potential-600.dx
    |-- sequence.txt
    ‘-- voltage-sensor.ipynb
```

2.3 Overview of the model

The `mrdna` Python package ships with a command-line script that provides users with convenient access to the modeling framework. To run a multi-resolution simulation, all one has to do is provide an appropriate DNA nanostructure design file.

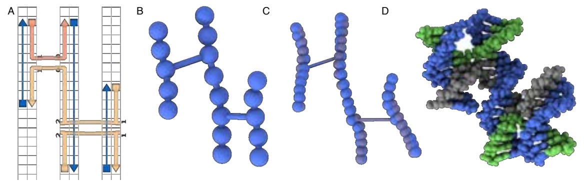

Figure 1: Multi-resolution modeling of a simple object made in cadnano.

Currently the script supports cadnano 'json' files, vHelix 'ma' files, and atomistic PDB files.

Regardless of the type of input, the script will run a low-resolution coarse-grained simulation with approximately 5 basepairs represented by each bead to provide fast relaxation of the DNA structure. With such a coarse representation we found that it was not practical to represent the twist locally. Instead we represent the twist via a torsion between consecutive crossovers within a helix.

The low-resolution simulation is followed by a pair of high-resolution (roughly 2 bead/basepair) coarse-grained simulations in which the orientation of each basepair is represented locally using an "orientation" bead. A harmonic potential is applied to the dihedral angle formed by adjacent basepair beads and their orientation beads to represent the twist elasticity of DNA in a local manner. When mapping from low- to high-resolution, the orientation of a basepair is not known, and hence the linking number or number of turns for a given stretch of dsDNA may be incorrect. The first of the high-resolution simulations uses dihedral angle potentials for the local twist that are smoothly truncated at about 3 k_BT, allowing the linking number of a DNA helix to relax. In the second of the high-resolution simulations, the potential increases harmonically, effectively preventing the linking number from changing.

The secondary structure of the DNA is fixed in both low- and high-resolution models; the models are intended for structural relaxation given the double-helical structure of the DNA. If you suspect hybridization to be important for modeling your object, we suggest application of the oxDNA model [9, 11], which was parametrized specifically to capture melting thermodynamics of DNA. Even if you choose to use the oxDNA model, the models presented here may help you improve oxDNA simulation's initial conditions.

3 Multi-resolution simulations with the `mrdna` script

To get started, open a terminal and navigate to the `step1` directory, and then call the following command:

```
mrdna -d sim1 hextube.json
```

Congratulations! You are now running your first set of multi-resolution simulations of a DNA origami object! **Please be aware that the entire simulation would take a long time to complete. A little later in the tutorial, you will be instructed to stop the simulation before it ends.** For now, continue reading and working through the tutorial while the simulation runs. The `mrdna` script should print some limited information as it builds the simulation system. It will then create the `sim1` directory (supplied using the '`-d`' option), populate the directory with the configuration files needed to run a simulation using the ARBD engine. Then the script will call ARBD, running the first of three coarse-grained simulations.

At any time, you can open another terminal and inspect the contents of the `sim1` directory, which should look something like what is shown below:

```
sim1/
|-- hextube-1.bd
|-- hextube-1.particles.txt
|-- hextube-1.pdb
|-- hextube-1.psf
|-- output/
|   |-- hextube-1.dcd
|   ‘-- hextube-1.restart
‘-- potentials/
    |-- angle-14.123-180.000.dat
    ‘-- ...
```

Briefly, `hextube-1.bd` is the configuration file for ARBD, which is passed to the simulation engine. This file references a number of other automatically generated text files to describe the system, such as `hextube-1.particles.txt`, which describes the coordinates of each bead. The many files in the `potentials` directory describe all the interactions in the system in simple two-column format. The files `hextube-1.psf` and `hextube-1.pdb` can be loaded into VMD to view the initial configuration of the system, but are not actually used by the ARBD engine. Finally, the `output` directory contains the DCD trajectory and simulation restart files output by ARBD. Load the part of the trajectory that has already been written using VMD with `vmd hextube-1.psf output/hextube-1.dcd` (or from the File menu within VMD). You should see a set of connected beads like in the left panel of Fig. 2.

By default, each step of the multi-resolution will run for a ten million steps, writ-

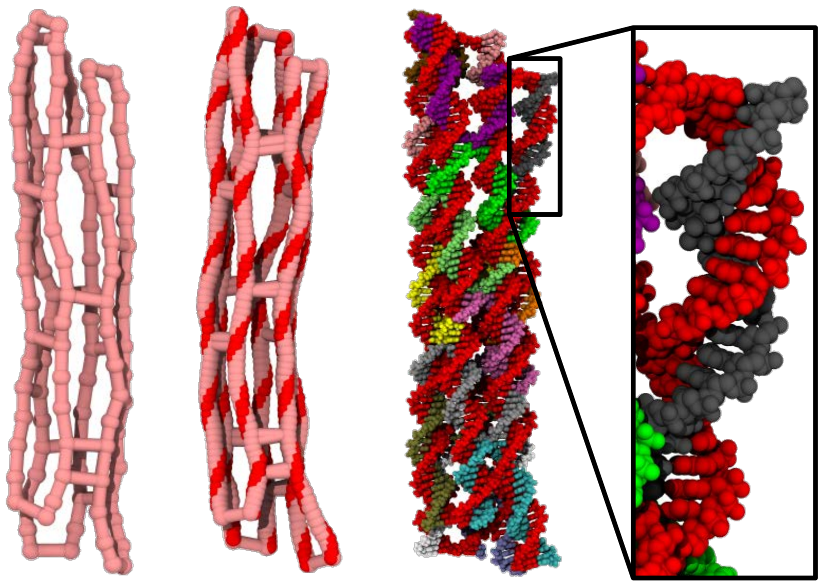
Figure 2: Multi-resolution simulation of a six-helix bundle of DNA.

ing frames to the DCD every 10,000 steps. However, this might be an insufficient or excessive number of steps for a given system; the required duration of the simulation for the structure to relax will, in general, increase with the size of the object. For typical-sized DNA origami objects, this number of steps is usually more than enough, requiring roughly thirty minutes of real time using a recent consumer grade GPU (such as a GTX-1080). With older hardware, the simulation could require substantially more time.

Close VMD and return to the first terminal with the simulation running and type `Control-c` to stop the simulation. Now run `mrdna` with the following options to quickly perform a complete (though short) set of multi-resolution simulations.

```
mrdna -d sim2 --coarse-steps 1e4 --fine-steps 1e4 --output-period 1e2 \
  hextube.json
```

3.1 Visualization and analysis

When the simulations are finished, use the following command to load the simulations into VMD:

```
vmd -e load-mrdna.tcl -args sim2
```

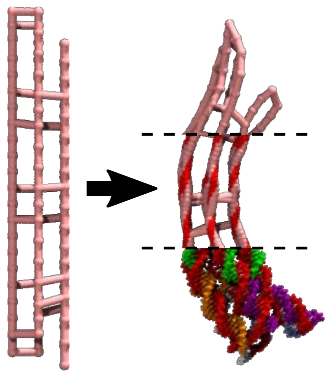

Figure 3: Multi-resolution simulation of a curved six-helix bundle of DNA. The representation of the DNA is “CPK” for the coarse-grained models, and “vdW” for the atomic model. The atomic DNA is colored by “segname”.

The `load-mrdna.tcl` script is provided to simplify and streamline the process of loading all of the multi-resolution simulation trajectories into VMD. Notice the differences between the coarse- and fine-resolution simulations. Also notice that the `mrdna` script has generated an atomistic model of the DNA object.

During the simulation, the object drifts and rotates. You can remove these global motions by typing the following commands into the tkConsole:

```
set ref [atomselect 0 all frame 0]
set sel [atomselect 0 all]
for {set f 0} {$f < [molinfo 0 get numframes]} {incr f} {
  $sel frame $f ;# work on frame $f
  set R [measure fit $sel $ref] ;# get the 4x4 transformation
  $sel move $R
}
puts "transformation at last frame:"
puts [lindex $R 0]
puts [lindex $R 1]
puts [lindex $R 2]
puts [lindex $R 3]
```

Now, you can use root mean square deviation (RMSD) as a metric to see how far the structure has deviated from its initial position.

```
set ch [open "hextube-rmsd.dat" w]
for {set f 0} {$f < [molinfo 0 get numframes]} {incr f} {
  $sel frame $f
  set RMSD [measure rmsd $sel $ref]
  puts $ch "$f $RMSD"
}
close $ch
```

Then you can plot the RMSD using your favorite plotting software, for example `xmgrace` or `matplotlib`. Do you think the simulation was long enough? Using RMSD with the idealized coordinates as a reference is best for determining whether a simulation needs to run longer for the structure to converge. Having the RMSD become flat does not necessarily mean the structure is done changing. However, for these small structures, one can usually tell if the simulation has run for long enough by visual inspection.

Another interesting quantity to examine is the root mean square fluctuation (RMSF) of each bead in the model. The first step in computing the RMSF is normally to perform a global alignment, but you've already done this. Next, we simply compute the RMSF of each bead. Instead of outputting the RMSF to a file for plotting, you can set the RMSF to one of the atoms' fields in VMD allowing you to visualize the fluctuations.

```
set oldBeta [$sel get beta] ;# save the old values so they can be restored
set RMSF [measure rmsf $sel]
$sel set beta $RMSF
```

Now open the representation window of VMD (Main → Graphics → Representations), select molecule "0", and change the "Coloring Method" to "beta". The color of the structure will range from red to white to blue with increasing RMSD. The values corresponding to the ends of the color range can be set in the "Trajectory" tab of the Graphical Representations window. When you are done examining the structure, restore the original Beta values.

```
$sel set beta $oldBeta
```

DNA nanostructures can be tagged by complementary fluorescent dyes to allow fluorescent resonance energy transfer (FRET) measurements. In these measurements, the donor dye absorbs light from a (typically) green laser. A portion of the energy absorbed by the dye is re-emitted as green fluorescence. The remaining energy is transfered through a distance-dependent quantum process to the nearby acceptor dye, which radiates (typically) red light. The energy transfer is mediated by dipole–dipole interactions and occurs with a likelihood roughly proportional to $1/r^6$, where r is the distance between the dyes. The ratio of red and green intensity captured then reports semi-quantitatively on the distance between the dyes. These measurements are typically most sensitive to changes in the distance between the dyes when they are located at about 6 nm. Hence, simulations can help guide the choice of dye attachement location before expensive and time-consuming FRET experiments are performed.

Suppose that you are interested in deformations of the ends of simple DNA origami nano structures, and you are considering labeling several different sites on a six-helix bundle. You plan to label the $3'$ ends of select staple strands. In particular you are considering adding donor/acceptor pairs at the following sites: (3[74] & 1[78]), (3[74] & 5[81]) and (1[78] & 5[81]), where X[Y] refers to the basepair in cadnano in helix with index X at the base index Y. Based on the simulations, determine which of the sites should provide the greatest sensitivity of the FRET efficiency to a given change in distance. First, create graphical representations of the candidate beads using the occupancy and beta field, which have been populated by the `mrdna` package with values corresponding to the candano helix and base indices, respectively. For each pair of dyes, create selections using the text from the graphical representations

```
## Replace X1/Y1/X2/Y2 with appropriate values
set donor [atomselect 0 "occupancy X1 and beta Y1"]
set acceptor [atomselect 0 "occupancy X2 and beta Y2"]
```

You can now write the distances between the dyes to a file with:

```
set ch [open hextube-dye-distance1.dat w]
set f 0
set i [$donor get index]
set j [$acceptor get index]
foreach d [measure bond "$i $j" molid 0 frame all] {
  puts $ch "$f $d"
  incr f
}
close $ch
```

Do this for each pair of dyes, and use your favorite plotting software to extract distance distributions to decide which attachment location will be best.

3.2 Solvent-free atomic simulations

All-atom molecular dynamics (MD) simulation with explicit representation of solvent remains the gold-standard for structural simulation of DNA nanostructures. This method is slow, even with access to supercomputing resources, and is therefore poorly suited for routine structure prediction. We previously demonstrated [6] that, using an elastic network of restraints within each DNA helix, a structure for the "pointer" DNA origami object could be obtained *without solvent* that compared favorably with the available cryo-electron microscopy reconstruction. The Elastic Network of Restraints Guided MD (ENRG-MD) approach allowed structural relaxation of the "pointer" in an overnight simulation on a workstation.

The `mrdna` script generates a NAMD configuration file that will perform and ENRG-MD simulation, that you may optionally run with:

```
cd sim2
namd2-cpu hextube-4.namd >& output/hextube-4.log &
```

The symbols ">&" redirect all output into the file `output/hextube-4.log`, and the ampersand (`&`) at the end of the line makes the simulation run in the background so you don't have to wait for the simulation to end to use your terminal. You can monitor the simulation with `less output/hextube-4.log` and you can stop the simulation by typing `fg` to bring the job into the foreground, and then typing "Control-c" to interrupt it.

3.3 Multi-resolution modeling for fast structural relaxation

The six-helix bundle is a very simple object whose relaxed configuration differs only a little from the initial structure, and as a result, one can easily perform an ENRG-MD simulation of this object without first performing coarse-grained structural relaxation, simply by uploading the design file to our ENRG-MD server.

Cadnano designs often include a pattern of skips and insertions that program a twisted or curved shape. In contrast to simple structures like the six-helix bundle, such designs can be problematic for the ENRG-MD protocol because the local DNA structure quickly distorts to accommodate a bad global structure, becoming kinetically trapped in local energy minima. In fact, this problem originally motivated the development of the multi-resolution modeling scheme described in this document.

We have provided an example of such an object in `curved-hextube.json`. Open the file using cadnano. Notice that one side of the structure contains insertions and one side contains deletions. When you are satisfied, close cadnano and run the design using `mrdna` as follows.

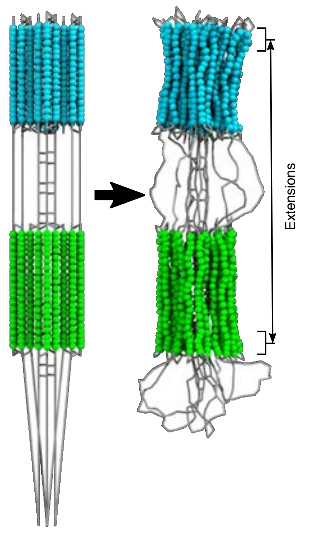

Figure 4: Coarse-grained simulation of the a sliding object designed by the Castro group [7]. The cyan region depics the fixed base of the object, and the green region depicts the sliding bearing.

```
mrdna --coarse-steps 2e5 --fine-steps 1e5 --output-period 1e3 -d sim3 \
  curved-hextube.json
```

When the simulation is finished, enter the `sim3` directory and visually inspect atomic structure using VMD. Is the object curved as expected? Is it twisted in ways that are unexpected? In this way, simulations can provide valuable feedback during an iterative design process.

The low-resolution simulations can also be useful for sampling the configurational space explored by “dynamic” DNA objects. As an example, you can simulate the DNA “slider” object developed by the Castro group [7] with:

```
mrdna --coarse-steps 1e7 --fine-steps 1e3 --output-period 1e2 -d sim4 \
  pris-6conn-86b.json
```

This simulation may take some time to complete. If you are pressed for time, it is recommended that you verify that the simulation works before stopping it and refering to the trajectory in `example-output/sim4`. Note that a significantly longer simulation is needed to get fully converged results.

Load the simulation results into VMD with the `load-mrdna.tcl` script. The Castro group took many transmission electron microscope images of these objects, and from the images they were able to measure distributions of the distance between the bearing base and the slider [7], see the file `slider-dist-exp.dat`. You can extract a similar distribution from your simulation. To achieve this you need to create two atom selections: one for the base of the slider, as far from the bearing as possible, and one for the opposite side of the bearing, which slides over the central six helices. Start by selecting the entire base and the entire bearing, as shown in Fig. 4. **Go to frame 0** and open the Graphical Representations window in VMD. Your first selection should be "`z > -300 and name "D.*"`". Your second selection should be quite similar, but should use a different geometric selection. In addition it should somehow exclude the central six helices. This can be achieved with a geometric selection or by excluding beads based on the cadnano virtual helix ID. Conveniently, the `mrdna` package writes the virtual helix ID from cadnano to the "occupancy" field of the pdb so, for example, you can select helix 5 and 7 from cadnano in VMD using the syntax "occupancy 5 7". Now refine the geometric selection so only a few layers of beads are selected from the base and from the bearing. Now, copy the selection text from the Graphical Representations window into the tkConsole, modifying the first two lines below:

```
set sel1 [atomselect 0 {z > -100 and name "D.*"}]
set sel2 [atomselect 0 {z < -780 and name "D.*" and not occupancy 7 8 21 22 35 36}]
set ch [open slider-dist.dat w]
for {set f 0} {$f < [molinfo 0 get numframes]} {incr f} {
  $sel1 frame $f
  $sel2 frame $f
  set r1 [measure center $sel1]
  set r2 [measure center $sel2]
  set distance [veclength [vecsub $r2 $r1]]
  set distance [expr 0.1*$distance] ;# convert to nm
  puts $ch "$f $distance"
}
close $ch
```

Close VMD and plot contents of the file `slider-dist.dat` with your favorite plotting tool. Does the data look converged? It may be interesting to examine the longer simulation provided in the `example-output` directory. Finally, you construct a histogram of the distances and compare to the experimentally obtained distribution available in `slider-dist-exp.dat` [7]. Note that you will need to normalize the distributions by the area under each curve to obtain a fair comparison.

3.4 Working with other design file formats

The field of structural DNA nanotechnology is currently dominated by DNA origami, but there are a number of exciting alternative approaches to making self-assembled nanoscale objects out of DNA. Accordingly there are a number computational tools for designing DNA nanostructures besides cadnano. These include, but are not limited to, Nanoengineer-1 [8], Tiamat [13], vHelix [2], DAEDALUS [12], and nucleic acid builder [3]. Being a relatively new project, the `mrdna` package so far only implements readers for cadnano json and vHelix Maya design files.

Change to the `step2` directory and you will find the `vHelix` directory containing a file called `bunny.ma` that was created by Björn Högberg's group [2]. You can run a simulation of the bunny object (or any of the other objects in `vHelix`) using the same `mrdna` command as before, but supplying the vHelix file as input.

```
mrdna --coarse-steps 1e6 --fine-steps 1e5 --output-period 1e4 -d sim1 \
  vHelix/bunny.ma
```

Note that full relaxation of such a large object would require a longer simulation. When the simulation completes, load the results into VMD for visual inspection. When you are done, close VMD.

In addition to the cadnano and vHelix readers, we recently created a reader for atomistic PDB files. This makes it possible to use the output from many existing design and simulation tools as the input for a multi-resolution simulation. For example, we have supplied the original DNA box design in `box.mmp`, which was produced using Nanoengineer-1. Although we do not yet have a Nanoengineer reader in `mrdna`, we can use an already-developed web-service to convert the nanoengineer file into an atomistic PDB. In a web browser, navigate to bionano.physics.illinois.edu/nanoengineer2pdb, upload `box.mmp`, name your job "box", check the "PDB only" option, click "Convert Structure", wait for the server to generate your pdb, and finally click on the link to download your pdb file. Extract `box.pdb` from the archive to the current directory before proceeding (*e.g.* `tar -xzf box.tar.gz`). Then, you can run a short simulation of the box using the following:

```
mrdna --coarse-steps 1e6 --fine-steps 1e6 --output-period 1e4 -d sim2 box.pdb
```

Examine the simulation results in VMD. Notice that the box fluctuates significantly during the coarse-grained simulation. Such simulations can be useful for estimating the underlying cause of FRET fluctuations.

PDB files written by different software packages often follow slightly different conventions. We've tested our PDB reader using files output by the popular DNA nanotechnology design and simulation packages CanDo, oxDNA (via the `convert_to_atomic.py` script), and NucleicAcidBuilder. You can run each a six-helix bundle made from each of these tools by using the following bash command:

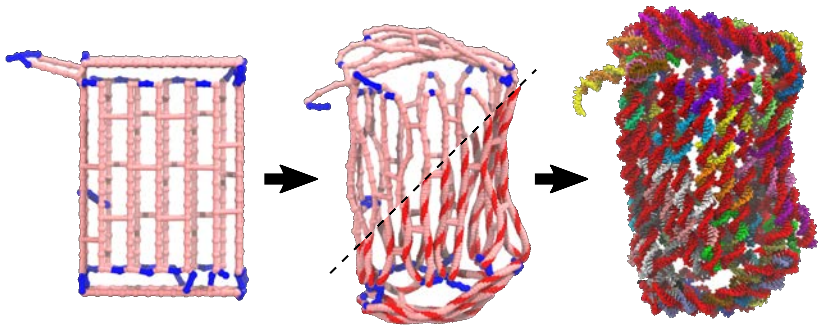

Figure 5: Multi-resolution modeling of a DNA box designed with Nanoengineer-1.

```
for f in hextube*pdb; do
  mrdna --coarse-steps 1e4 --fine-steps 1e3 --output-period 1e2 -d sim3 $f
done
```

We encourage you to test PDB files produced by other software that you use. If you encounter any issues, please file a bug report at:
https://gitlab.engr.illinois.edu/tbgl/tools/mrdna/issues/new or contact Chris Maffeo at cmaffeo2@illinois.edu.

4 Detailed description of the model

This section provides a more detailed description of how the model is constructed. Reading this material is optional; if you are pressed for time it is recommended that you skip this section, returning later when you have time.

The `mrdna` package converts model of DNA with position and orientation of each helix described parametrically into a bead-based realization of the model. First, beads are placed at the connections between segments every crossover and at every intrahelical connection. When two of the connection beads are placed within one nucleotide of one another, they are merged together.

Next, in each segment, "intrahelical" beads are placed between each pair of "consecutive" connection beads with a minimum user-specified density. A bond connects consecutive intrahelical beads with a separation-dependent rest-length (3.4 Å/bp; 5 Å/nt) and spring constant derived from the elastic moduli of dsDNA and ssDNA [4]. A harmonic angle potential is placed on consecutive triplets of intrahelically-connected beads, with a 180° rest angle and spring constant derived from the persistence length of ssDNA. Optionally, an orientation bead is placed alongside each dsDNA backbone bead, attached to its parent bead by a harmonic potential (1.5 Å rest-length; k_{spring} =30 kcal/mol. An harmonic angle (90° rest angle; $k_{\text{spring}} = 0.5 k_{\text{L}_\text{p}}$ kcal/mol) potential is placed between consecutive intrahelical beads and each of the orientation beads. A dihedral angle potential is placed between each orientation bead, its parent intrahelical bead, the adjacent intrahelical bead and the corresponding orientation bead with a separation-dependent rest angle derived from the $\sim$ 90-nm twist-persistence length of dsDNA [5].

To reduce the number of bead types, a hierarchical clustering algorithm is applied, classifying the bead types according to number of nucleotides represented by the bead.

Finally, a harmonic bond is placed between intrahelical beads on either side of each crossover connection. The spring constant (4 kcal/mol-Å^2) and rest length (18.5 Å) of the bond were derived from the distribution of crossover distances observed in atomistic simulations [10]. If the model is constructed without a local representation of the twist a dihedral angle potential is applied between the intrahelical beads at consecutive crossovers within each helix. If the model is constructed with a local representation of the twist, an angle potential is placed on each side of the crossover between the orientation bead, its parent bead and the intrahelical bead on the other side of the crossover at the adjacent helix with a rest length of 120°.

5 Description of the Python modeling framework

This section attempts to provide an overview of the Python framework. Reading this material is optional; you can proceed to the next section if you want to immediately get started using the framework.

Requisite knowledge. The remainder of the tutorial covers usage of the `mrdna` Python *package* (not the script). Hence, the tutorial assumes basic familiarity with Python. For example, you should be familiar with concepts such as importing modules, using `defs` (functions) and objects, and preferably working with numpy arrays. If any of these is unfamiliar, we strongly recommend that you take a few moments to work through some basic Python tutorials before completing this section. A concise, interactive web-based set of tutorials is freely available through learnpython.org.

The contents of the `mrdna` package are listed below.

```
.
|-- coords.py
|-- __init__.py
|-- model/
|   |-- arbdmodel.py
|   |-- CanonicalNucleotideAtoms.py
|   |-- dna_sequence.py
|   |-- __init__.py
|   |-- nbPot.py
|   `-- nonbonded.py
|-- readers/
|   |-- cadnano_segments.py
|   |-- __init__.py
|   |-- polygon_mesh.py
|   |-- segmentmodel_from_lists.py
|   `-- segmentmodel_from_pdb.py
|-- resources/
|   `-- ...
|-- segmentmodel.py
`-- simulate.py
```

The __init__.py files contain code that is executed when the `mrdna` package or a subpackage (such as `model` or `readers`) is first imported. Classes defined in the module `mrdna.model.arbdmodel` can be used to build models from point-particles interacting through bonded and non-bonded potentials described in nbPot.py and nonbonded.py. The `Particle` class in arbdmodel is meant to represent a single point-like particle with a position in 3D space. It should normally have a `type_` attribute (following convention, the underscore distinguishes the attribute name from a built-in Python keyword) that should normally refer to an object of the `ParticleType` class. The `ParticleType` class is meant to hold information like the charge or mass of the particle.

The `arbdmodel` module also defines the `Group` class, which has a "children" attribute and can be used with `Particle` objects to compose hierarchies for bead-based models. Every object contained in the "children" list should have a "parent" attribute that points back to the `Group` object. `Group` objects can have an orientation matrix and position vector, which is applied to each of it's child objects when their coordinates are requested.

Also in arbdmodel.py, the class `ArbdModel` extends the `Group` class to set up and perform coarse-grained simulations using the GPU-accelerated package ARBD [1]. This class can additionally write NAMD configuration files for atomistic models using the CHARMM36 force-field, but this interface is currently specific to DNA simulations and does not attempt to wrap the many options available during NAMD configuration.

Please note that the types defined in arbdmodel.py are flexible and convenient for development, but do not scale very well beyond ten-thousand particles. Hence, they can be expected to change substantially in the future.

The class `SegmentModel` in the segmentmodel module, along with several associated classes, provides a general data structure for modeling objects built using DNA nanotechnology with a given secondary structure (*i.e.* the modeling framework does not currently support melting and hybridization). A `SegmentModel` is composed from a list of one or more `DoubleStrandedSegment` and `SingleStrandedSegment` objects that inherit from the `Segment` class, which in turn inherits from the `Group` class. A `Segment` object represents a contiguous polymeric stretch of DNA in 3-dimensional space using splines. Each segment can contain `Connection` objects that joins two `Location` objects, one on each `Segment`. The `Location` objects specify a position in terms of contour-length and strand (in forward or reverse direction with respect to $5'$-to-$3'$ direction). Both of these objects can have a "type_" attribute that is used to indicate whether a connection is a "crossover" or "intrahelical" connection. The `SegmentModel` can generate beads with a minimum resolution specified by the user to realize a coarse-grained particle-based model of the DNA system that is compatible with ARBD. This functionality is likely to be refactored into another class in the future to make the code more modular. After a simulation, the coordinates taken from the final frame or from a time-averaged window of the trajectory can be used to update the splines representing the position and orientation of each single- or double-stranded region of DNA. In this way, the `SegmentModel` facilitates mapping between different-resolution models of a DNA strand. Finally, the `SegmentModel` object can create an atomistic model of a DNA nanotechnology object with ENRG-MD restraints.

We have already implemented several functions that construct a SegmentModel from various input sources including cadnano data structures and file-based output from vHelix. In addition to importing models form these sources, it is possible, though challenging, to directly use the classes in segmentmodel.py to construct simple DNA nanotechnology objects. We intend to simplify and streamline the model-building API to make such manipulation easier for end-users with limited programming experience.

Once a model has been created, one usually would like to run a simulation with the model. The `ArbdModel` class (from which `SegmentModel` inherits) has a `simulate` method for directly running a simulation. However, the `mrdna.simulate` module provides the function `multiresolution_simulation`, which takes a model as an argument and automates the multi-resolution simulation protocol.

These modules can be combined in various ways to run simulations of DNA nanostructures. For example, the code in these modules was used to almost trivially write the `mrdna` script that you have been using.

6 Writing scripts using the `mrdna` package

The remaining portion of the tutorial uses an interactive notebook to demonstrate how models can be constructed and manipulated from code. The notebook will walk you through the creation of simple models, culminating with the reproduction of a portion of a recently published study.

Navigate to the root directory of the tutorial. Here you can find a Jupyter notebook that will interactively walk you through some aspects of the modeling framework. Open the notebook by typing `jupyter notebook step3.ipynb`. This should start the jupyter server and open a browser to the page for your notebook.

References

[1] Atomic resolution brownian dynamics. http://bionano.cpanel.engr.illinois.edu/arbd, 2016.

[2] E. Benson, A. Mohammed, J. Gardell, S. Masich, E. Czeizler, P. Orponen, and B. Högberg. DNA rendering of polyhedral meshes at the nanoscale. *Nature*, 523(7561):441–444, 2015.

[3] R. A. Brown and D. A. Case. Second derivatives in generalized Born theory. *J. Comput. Chem.*, 27(14):1662–1675, 2006.

[4] S. Cocco, J. F. Marko, and R. Monasson. Theoretical models for single-molecule DNA and RNA experiments: from elasticity to unzipping. *Biophysique*, 2002.

[5] D. J. Kauert, T. Kurth, T. Liedl, and R. Seidel. Direct mechanical measurements reveal the material properties of three-dimensional DNA origami. *Nano Lett.*, 11(12):5558–5563, 2011.

[6] C. Maffeo, J. Yoo, and A. Aksimentiev. De novo prediction of DNA origami structures through atomistic molecular dynamics simulation. *Nucleic Acids Res.*, 44(7):3013–3019, 2016.

[7] A. E. Marras, L. Zhou, H.-J. Su, and C. E. Castro. Programmable motion of DNA origami mechanisms. *Proc. Natl. Acad. Sci. U. S. A.*, 112(3):713–718, 2015.

[8] Nanorex, Inc. Nanoengineer-1, v.a8, 2006.

[9] T. E. Ouldridge, A. A. Louis, and J. P. K. Doye. Structural, mechanical, and thermodynamic properties of a coarse-grained DNA model. *J. Chem. Phys.*, 134(8):085101, Feb. 2011.

[10] S. Slone, C.-Y. Li, J. Yoo, and A. Aksimentiev. Molecular mechanics of DNA bricks: *in situ* structure, mechanical properties and ionic conductivity. *New J. Phys.*, 18(5):055012, 2016.

[11] B. E. Snodin, F. Randisi, M. Mosayebi, P. Šulc, J. S. Schreck, F. Romano, T. E. Ouldridge, R. Tsukanov, E. Nir, A. A. Louis, and J. P. Doye. Introducing improved structural properties and salt dependence into a coarse-grained model of DNA. *J. Chem. Phys.*, 142(23):06B613_1, 2015.

[12] R. Veneziano, S. Ratanalert, K. Zhang, F. Zhang, H. Yan, W. Chiu, and M. Bathe. Designer nanoscale DNA assemblies programmed from the top down. *Science*, 352(6293):1534–1534, 2016.

[13] S. Williams, K. Lund, C. Lin, P. Wonka, S. Lindsay, and H. Yan. Tiamat: A three-dimensional editing tool for complex DNA structures. In A. Goel, F. C. Simmel, and P. Sosík, editors, *DNA Computing*, volume 5347 of *Lecture Notes in Computer Science*, pages 90–101. Springer Berlin Heidelberg, 2009.