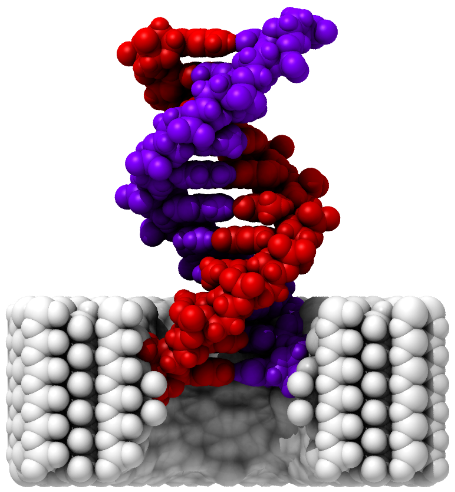
University of Illinois at Urbana-Champaign
Beckman Institute for Advanced Science and Technology
Theoretical and Computational Biophysics Group
Computational Biophysics Workshop

Nanotechnology Tutorial



Alek Aksimentiev

Jeffrey Comer

May 2007

Contents

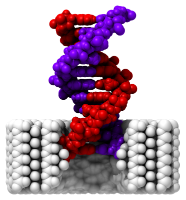
Introduction

This tutorial is designed to guide users of VMD and NAMD in all the steps required to set up a molecular dynamics (MD) simulation of a bionanotechnology device. The tutorial assumes that you already have a working knowledge of VMD and NAMD. For the accompanying VMD and NAMD tutorials go to:
`http://www.ks.uiuc.edu/Training/Tutorials/`
This tutorial has been designed specifically for VMD 1.8.5, and should take about 4 hours to complete in its entirety.

Structure building for biomolecules is likely familiar to most VMD and NAMD users and the interested reader is referred to the in-depth treatment given in the other VMD and NAMD tutorials. Constructing models of solid-state inorganic systems, however, requires a slightly different approach. Therefore, we begin in the first unit by learning how to build models of synthetic devices, starting with only a crystal unit cell. We'll then add solution and end by simulating ionic current through a nanoscale pore in a crystalline membrane. The second unit will guide you through combining a biomolecule (DNA) with a crystalline membrane and simulating the resulting system. Many of the steps in this tutorial depend on the results of previous steps. If some steps are not completed and you would like to move on, exemplary output is available in `bionano-tutorial-files/example-output/`.

Throughout the text, some material will be presented in separate "boxes". These boxes include information complementary to the tutorial, such as details of the systems used in bionanotechnology research, tips or technical details, and suggestions for more in-depth simulations.

If you have any questions or comments on this tutorial, please email the TCB Tutorial mailing list at tutorial-l@ks.uiuc.edu. The mailing list is archived at http://www.ks.uiuc.edu/Training/Tutorials/mailing_list/tutorial-l/.



High-throughput DNA sequencing. This tutorial will focus on the interaction of DNA and a Si_3N_4 nanopore about 2 nm in diameter, which is the key element in proposed technology for high-throughput DNA sequencing. Currently, two months and approximately ten million dollars are required to determine a human genome to the desired 99.99% accuracy—obviously too slow and too costly for use in personal medicine. A nanopore device, along with an integrated semiconductor detector, has promise to reduce the time and expense of genome sequencing by orders of magnitude (For example, see Heng et al., *Bell Labs Technical Journal* **10**, 5–22 (2005)).

Required programs

The following programs are required for this tutorial:

- **VMD:** Available at http://www.ks.uiuc.edu/Research/vmd/ (for all platforms)
- **NAMD:** Available at http://www.ks.uiuc.edu/Research/namd/ (for all platforms)

Getting Started

You can find the files for this tutorial in the `bionano-tutorial-files` directory. Below you can see in Fig. 1 the directory structure for this tutorial.

To start VMD type `vmd` in a Unix terminal window, double-click on the VMD application icon likely located in the `Applications` folder in Mac OS X, or click on the Start → Programs → VMD menu item in Windows.

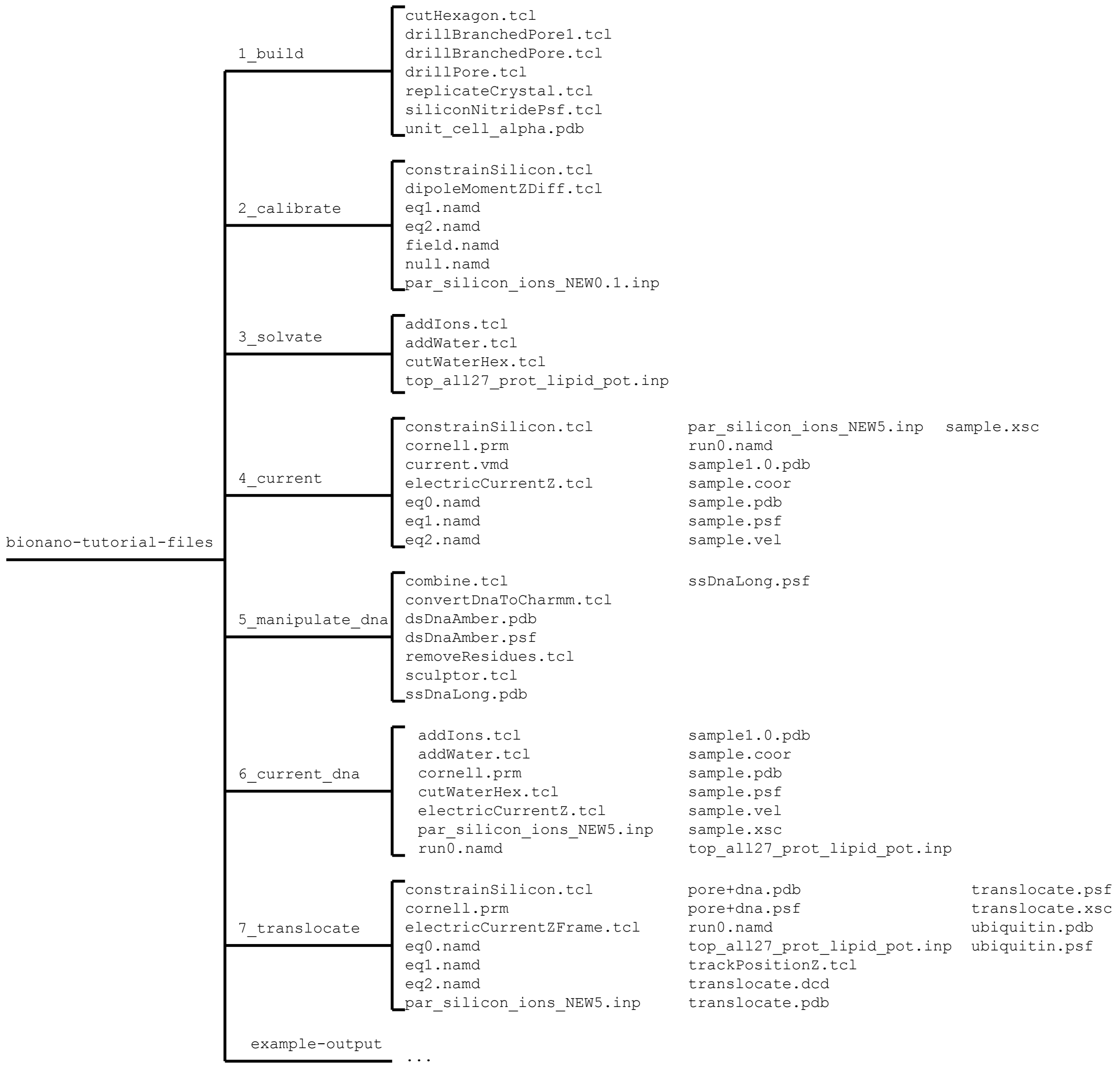


Figure 1: Directory structure of `bionano-tutorial-files`.

1 Simulation setup and protocols

In this unit you will learn to construct synthetic systems and simulate the passage of ions through a nanopore device.

1.1 Building a crystal

In this section, we'll learn how to build a crystalline membrane from its unit cell.

1 Let's take a look at the unit cell in VMD. If you have not already opened VMD, do so now. Open the Tk Console by selecting Extensions → Tk Console. Open the directory with the files for this section and load the unit cell by entering the following:

```
cd ‘‘your working directory’’
cd bionano-tutorial-files
cd 1_build
mol new unit_cell_alpha.pdb
```

Next, select Graphics → Representations.... In the Graphical Representations window, set the Drawing Method to CPK. Now we can clearly see the configuration of atoms in the unit cell. This configuration of eight nitrogen atoms and six silicon atoms will form the basis for our extended Si_3N_4 crystal.

To generate a crystal membrane from the unit cell, we will execute a script included with this tutorial. The main steps in the script are as follows. First, we open an output PDB and write `REMARK` lines specifying the geometry of the crystal. Next, we read the unit cell PDB, extracting the records for each atom. We then generate the crystal by repeatedly writing the atom records from the unit cell PDB to the output PDB, albeit with new positions that are displaced by periodic lattice vectors. In the following part of the tutorial, the script, `replicateCrystal.tcl`, we will use to generate the crystal is presented. Each portion of the script is preceded by text describing how it works. If you'd like to move on with the tutorial without examining the details of the script, simply skip ahead to page 13.

As will be common to most of the scripts presented in this tutorial, the first section of the script defines variables that act as the arguments of the

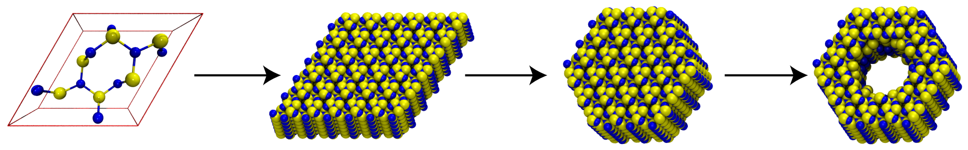


Figure 2: Process of modeling a silicon nitride device. First the unit cell is replicated to form a crystal membrane. This membrane is then cut to a more convenient geometry. Finally, a pore is produced in the membrane by the removal of atoms.

script. The names of input and output files will often appear here, as in the case below, where the name of the PDB file containing the crystal's unit cell is stored in `unitCellPdb` and that of the resultant PDB is stored in `outPdb`. The variables `n1`, `n2`, and `n3` determine the number of times that the unit cell will be replicated along the respective crystal axis. The remaining variables describe the geometry of the unit cell. The unit cell is a parallelepiped with sides of lengths `l1`, `l2`, and `l3` along directions given by the unit vectors `basisVector1`, `basisVector2`, and `basisVector3`. Thus, the set of vectors $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$, where $\mathbf{a}_i = l_i\,\mathbf{basisVector}_i$, generates the translational symmetry of the lattice.

replicateCrystal.tcl

```
# Read the unit cell of a pdb and replicate n1 by n2 by n3 times.
# Input:
set unitCellPdb unit_cell_alpha.pdb
# Output:
set outPdb membrane.pdb
# Parameters:
# Choose n1 and n2 even if you wish to use cutHexagon.tcl.
set n1 6
set n2 6
set n3 6
set l1 7.595
set l2 7.595
set l3 2.902
set basisVector1 [list 1.0 0.0 0.0]
set basisVector2 [list 0.5 [expr sqrt(3.)/2.] 0.0]
set basisVector3 [list 0.0 0.0 1.0]
```

The two following Tcl procedures extract data from the PDB. The first returns a list of 3D vectors corresponding to the $\{x\ y\ z\}$ coordinates of each atom in the unit cell. The second simply extracts each line atom record from the PDB and returns it as a list.

```
# Return a list with atom positions.
proc extractPdbCoords {pdbFile} {
     set r {}

     # Get the coordinates from the pdb file.
     set in [open $pdbFile r]
     foreach line [split [read $in] \n] {
          if {[string equal [string range $line 0 3] "ATOM"]} {
               set x [string trim [string range $line 30 37]]
               set y [string trim [string range $line 38 45]]
               set z [string trim [string range $line 46 53]]

               lappend r [list $x $y $z]
          }
     }
     close $in
     return $r
}

# Extract all atom records from a pdb file.
proc extractPdbRecords {pdbFile} {
     set in [open $pdbFile r]

     set pdbLine {}
     foreach line [split [read $in] \n] {
          if {[string equal [string range $line 0 3] "ATOM"]} {
               lappend pdbLine $line
          }
     }
     close $in

     return $pdbLine
}
```

Given the coordinates of all atoms in the unit cell, the `displaceCell` procedure shifts them by a lattice vector. In other words, this procedure is where the crystal is actually replicated—the basis on which the rest of the script rests.

```
# Shift a list of vectors by a lattice vector.
proc displaceCell {rUnitName i1 i2 i3 a1 a2 a3} {
     upvar $rUnitName rUnit
     # Compute the new lattice vector.
     set rShift [vecadd [vecscale $i1 $a1] [vecscale $i2 $a2]]
     set rShift [vecadd $rShift [vecscale $i3 $a3]]

     set rRep {}
     foreach r $rUnit {
          lappend rRep [vecadd $r $rShift]
     }
     return $rRep
}
```

The procedure `makePdbLine` is essential to the correct formation of the output PDB. The lines of the unit cell PDB obtained by `extractPdbRecords` are altered to reflect the new coordinates of the translated unit cells.

```
# Construct a pdb line from a template line, index, resId, and coordinates.
proc makePdbLine {template index resId r} {
     foreach {x y z} $r {break}
     set record "ATOM  "
     set si [string range [format "     %5i " $index] end-5 end]
     set temp0 [string range $template 12 21]
     set resId [string range "    $resId"  end-3 end]
     set temp1 [string range $template  26 29]
     set sx [string range [format "       %8.3f" $x] end-7 end]
     set sy [string range [format "       %8.3f" $y] end-7 end]
     set sz [string range [format "       %8.3f" $z] end-7 end]
     set tempEnd [string range $template 54 end]

     # Construct the pdb line.
     return "${record}${si}${temp0}${resId}${temp1}${sx}${sy}${sz}${tempEnd}"
}
```

The final procedure drives the script. The series of `puts` commands near the top of the procedure store the geometry of the crystal in `REMARK` lines in the output PDB. These lines will be needed later when we modify the shape of the crystal. The lattice vectors are defined by

$$\mathbf{R}(i,j,k) = i\mathbf{a}_1 + j\mathbf{a}_2 + k\mathbf{a}_3,$$

where i, j, and k are integers. The main loop iterates through all unique (i,j,k) for $0 \le i < n_1$, $0 \le j < n_2$, and $0 \le k < n_3$, producing a crystal.

```
# Build the crystal.
proc main {} {
     global unitCellPdb outPdb
     global n1 n2 n3 l1 l2 l3 basisVector1 basisVector2 basisVector3

     set out [open $outPdb w]
     puts $out "REMARK Unit cell dimensions:"
     puts $out "REMARK a1 $a1"
     puts $out "REMARK a2 $a2"
     puts $out "REMARK a3 $a3"
     puts $out "REMARK Basis vectors:"
     puts $out "REMARK basisVector1 $basisVector1"
     puts $out "REMARK basisVector2 $basisVector2"
     puts $out "REMARK basisVector3 $basisVector3"
     puts $out "REMARK replicationCount $n1 $n2 $n3"

     set a1 [vecscale $l1 $basisVector1]
     set a2 [vecscale $l2 $basisVector2]
     set a3 [vecscale $l3 $basisVector3]

     set rUnit [extractPdbCoords $unitCellPdb]
     set pdbLine [extractPdbRecords $unitCellPdb]
     puts "\nReplicating unit $unitCellPdb cell $n1 by $n2 by $n3..."

     # Replicate the unit cell.
     set atom 1
     set resId 1
     for {set k 0} {$k < $n3} {incr k} {
          for {set j 0} {$j < $n2} {incr j} {
               for {set i 0} {$i < $n1} {incr i} {
                    set rRep [displaceCell rUnit $i $j $k $a1 $a2 $a3]

                    # Write each atom.
                    foreach r $rRep l $pdbLine {
                         puts $out [makePdbLine $l $atom $resId $r]
                         incr atom
                    }
                    incr resId

                    if {$resId > 9999} {
                         puts "Warning! Residue overflow."
                         set resId 1
                    }
               }
          }
     }
     puts $out "END"
     close $out

     puts "The file $outPdb was written successfully."
}

main
```

2 To execute the script, enter `source replicateCrystal.tcl` in the VMD Tk Console.

3 We will now edit the script `replicateCrystal.tcl` in order to make a thicker Si_3N_4 block. Open the file `replicateCrystal.tcl` in your text editor of choice, e.g., by typing `nedit replicateCrystal.tcl &` in the terminal window. First, change the line 8 to `set outPdb block.pdb`. Next change the value of `n3` by altering line 13 to read `set n3 16`. Save the file and exit the text editor.

4 To generate this thicker block of Si_3N_4, execute the modified script by entering `source replicateCrystal.tcl` as before.

5 We've now created two Si_3N_4 crystals. To view the first, type the following in the Tk Console window:

```
mol delete all
mol new membrane.pdb
```

This is the membrane that we will use for ionic current measurement and DNA translocation. Notice that the cross section of the system in xy-plane is parallelogram.

6 Similarly, open the thicker block, which we'll use in **Task 1**. Enter the following:

```
mol delete all
mol new block.pdb
```

1.2 Constructing synthetic nanopores

Now we'll construct a nanopore in our Si_3N_4 membranes.

Our subsequent MD simulations will use periodic boundary conditions, so the shape of our system must be such that the Si_3N_4 lattice matches at the system's boundaries. A hexagonal prism shape can match this lattice and is more convenient than the parallelpiped we just created for housing a nanopore with a roughly circular cross section. In the script for this purpose, we first obtain the crystal geometry from the `REMARK` lines in the PDB and write a file describing the hexagonal periodic boundary conditions. Next, for convenience, we shift the crystal so that its centroid coincides with the origin of the coordinate system. We finally copy the atom records from the input PDB to the output PDB, skipping those that do not lie within the hexagonal prism. If you'd like to skip the details of this script, move on to page 18.

The first section again contains what serves as arguments to the script. To save time when the script is altered to act on different files, a file name prefix is defined which gives the output files systematic names based on the name of the input file. In addition to cutting the system to a hexagonal prism, the script `cutHexagon.tcl` also produces a boundary file (with a `.bound` extension) that contains the periodic simulation cell vectors needed to form bonds between atoms at the boundaries and run simulations in NAMD.

`cutHexagon.tcl`

```
# Remove atoms from a pdb outside of a hexagonal prism
# along the z-axis with a vertex along the x-axis.
# Also write a file with NAMD cellBasisVectors.

set fileNamePrefix membrane
# Input:
set pdbIn ${fileNamePrefix}.pdb
# Output:
set pdbOut ${fileNamePrefix}_hex.pdb
set boundaryFile ${fileNamePrefix}_hex.bound
set pdbTemp tmp.pdb
```

This procedure executes VMD's `measure center` method to center the system at the origin, which is done for convenience.

```
# Write a pdb with the system centered.
proc centerPdb {pdbIn pdbOut} {
     mol new $pdbIn
     set all [atomselect top all]
     set cen [measure center $all]
     $all moveby [vecinvert $cen]
     $all writepdb $pdbOut
     $all delete
     mol delete top
}
```

The procedure `readGeometry` extracts the crystal geometry from the `REMARK` lines we added to the PDB in last script and writes the boundary file mentioned above.

```
# Read the geometry of the system and write the boundary file.
# Return the radius of the hexagon.
proc readGeometry {pdbFile boundaryFile} {
     # Extract the remark lines from the pdb.
     mol new $pdbFile
     set remarkLines [lindex [molinfo top get remarks] 0]
     foreach line [split $remarkLines \n] {
          if {![string equal [string range $line 0 5] "REMARK"]} {continue}
          set tok [concat [string range $line 7 end]]

          set attr [lindex $tok 0]
          set val [lrange $tok 1 end]
          set remark($attr) $val
          puts "$attr = $val"
     }
     mol delete top

     # Deterimine the lattice vectors.
     set vector1 [vecscale $remark(basisVector1) $remark(a1)]
     set vector2 [vecscale $remark(basisVector2) $remark(a2)]
     set vector3 [vecscale $remark(basisVector3) $remark(a3)]

     foreach {n1 n2 n3} $remark(replicationCount) {break}
     set pbcVector1 [vecadd [vecscale $vector1 [expr $n1/2]] \
     [vecscale $vector2 [expr $n2/2]]]
     set pbcVector2 [vecadd [vecscale $vector1 [expr -$n1/2] ] \
     [vecscale $vector2 [expr $n2]]]
     set pbcVector3 [vecscale $vector3 $n3]

     puts ""
     puts "PERIODIC VECTORS FOR NAMD:"
     puts "cellBasisVector1         $pbcVector1"
     puts "cellBasisVector2         $pbcVector2"
     puts "cellBasisVector3         $pbcVector3"
     puts ""

     set radius [expr 2.*[lindex $pbcVector1 0]/3.]
     puts "The radius of the hexagon: $radius"

     # Write the boundary condition file.
     set out [open $boundaryFile w]
     puts $out "radius            $radius"
     puts $out "cellBasisVector1 $pbcVector1"
     puts $out "cellBasisVector2 $pbcVector2"
     puts $out "cellBasisVector3 $pbcVector3"
     close $out

     return $radius
}
```

Here, in the procedure `cutHexagon`, we read each atom record from the input PDB and extract the serial number and coordinates. The record is then written to the output PDB if and only if the position of the atom is within a hexagon of radius R in the xy-plane, centered at the origin, which has a vertex along the x-axis. All three of the following geometric criteria must hold:

$$\begin{array}{rcccl} -\frac{\sqrt{3}}{2}R & < & y & < & \frac{\sqrt{3}}{2}R, \\ \sqrt{3}(x-R) & < & y & < & \sqrt{3}(x+R), \\ \sqrt{3}(-x-R) & < & y & < & \sqrt{3}(-x+R). \end{array}$$

```
proc cutHexagon {r pdbIn pdbOut} {
     set sqrt3 [expr sqrt(3.0)]

     # Open the pdb to extract the atom records.
     set out [open $pdbOut w]
     set in [open $pdbIn r]
     set atom 1
     foreach line [split [read $in] \n] {
          set string0 [string range $line 0 3]

          # Just write any line that isn't an atom record.
          if {![string match $string0 "ATOM"]} {
               puts $out $line
               continue
          }

          # Extract the relevant pdb fields.
          set serial [string range $line 6 10]
          set x [string range $line 30 37]
          set y [string range $line 38 45]
          set z [string range $line 46 53]

          # Check the hexagon bounds.
          set inHor [expr abs($y) < 0.5*$sqrt3*$r]
          set inPos [expr $y < $sqrt3*($x+$r) && $y > $sqrt3*($x-$r)]
          set inNeg [expr $y < $sqrt3*($r-$x) && $y > $sqrt3*(-$x-$r)]

          # If atom is within the hexagon, write it to the output pdb
          if {$inHor && $inPos && $inNeg} {
               # Make the atom serial number accurate if necessary.
               if {[string is integer [string trim $serial]]} {
                    puts -nonewline $out "ATOM  "
                    puts -nonewline $out \
                    [string range [format "     %5i " $atom] end-5 end]
                    puts $out [string range $line 12 end]

               } else {
                    puts $out $line
               }

               incr atom
          }
     }
     close $in
     close $out
}
```

In the main part of the script, we extract the radius of the hexagon and write the boundary file, center the crystal, and finally cut the crystal into a hexagonal prism.

```
set radius [readGeometry $pdbIn $boundaryFile]
centerPdb $pdbIn $pdbTemp
cutHexagon $radius $pdbTemp $pdbOut
```

1 Enter `source cutHexagon.tcl` in the Tk Console. The script acts on `membrane.pdb`, producing the file `membrane_hex.pdb`. We also need to cut `block.pdb` to a hexagonal prism.

2 Open `cutHexagon.tcl` in your text editor. Change line 7 to read `set fileNamePrefix block` and save the file. Execute the script by reentering
`source cutHexagon.tcl` in the Tk Console.

3 Let's look at our system in VMD to make sure it has been cut into a hexagonal prism correctly. Type:

```
mol delete all
mol load pdb membrane_hex.pdb
```

4 Also, look at the second system. Enter `mol delete all` and `mol load pdb block_hex.pdb` in the Tk Console.

Now we'll shape our crystals into nanopore devices. The script `drillPore.tcl` has been designed for this purpose. We'll produce a pore with the shape of two intersecting cones, which has hourglass-like cross sections in the xz- or yz- planes. First, we read the length of the pore along the z-axis from the boundary file. Subsequently, we remove atoms from the PDB file that are within the pore. You can skip the details of this script by turning to page 22.

The parameters `radiusMin` and `radiusMax` define the minimum and maximum radius of the double-cone pore.

<u>`drillPore.tcl`</u>

```
# Cut a double-cone pore in a membrane.

# Parameters:
set radiusMin 8
set radiusMax 15
# Input:
```

```
set pdbIn membrane_hex.pdb
set boundaryFile membrane_hex.bound
# Output:
set pdbOut pore.pdb
set boundaryOut pore.bound
```

This procedure extracts the length of the pore along the z-axis, which is necessary for defining the geometry of the double cone pore.

```
# Get cellBasisVector3_z from the boundary file.
proc readLz {boundaryFile} {
     set in [open $boundaryFile r]
     foreach line [split [read $in] \n] {
          if {[string match "cellBasisVector3 *" $line]} {
               set lz [lindex $line 3]
               break
          }
     }
     close $in
     return $lz
}
```

In a membrane of thickness l_z, the cylindrical coordinate s that corresponds to the radius of the pore at height z for a double cone with a center radius of s_{min} and a maximum radius of s_{max} is given by

$$s(z) = s_{\text{min}} + 2\frac{s_{\text{max}} - s_{\text{min}}}{l_z} |z| .$$

Whether the point (x, y, z) is within the pore is determined by

$$x^2 + y^2 < s(z)^2.$$

Later, in **Task 1**, you will modify a similar procedure to produce a topologically more complicated pore.

```
# Determine whether the position {x y z} is inside the pore and
# should be deleted.
proc insidePore {x y z sMin sMax} {
     # Get the radius for the double cone at this z-value.
     set s [expr $sMin + 2.0*($sMax-$sMin)/$lz*abs($z)]

     return [expr $x*$x + $y*$y < $s*$s]
}
```

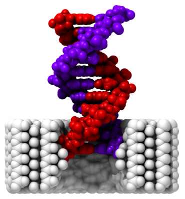
The final procedure is nearly identical to the `cutHexagon` procedure in `cutHexagon.tcl`. It writes only lines satisfying geometrical constraints, this time given by the result of the procedure `insidePore`.

```
proc drillPore {sMin sMax lz pdbIn pdbOut} {
     set sqrt3 [expr sqrt(3.0)]

     # Open the pdb to extract the atom records.
     set out [open $pdbOut w]
     set in [open $pdbIn r]
     set atom 1
     foreach line [split [read $in] \n] {
          set string0 [string range $line 0 3]

          # Just write any line that isn't an atom record.
          if {![string match $string0 "ATOM"]} {
               puts $out $line
               continue
          }

          # Extract the relevant pdb fields.
          set serial [string range $line 6 10]
          set x [string range $line 30 37]
          set y [string range $line 38 45]
          set z [string range $line 46 53]

          # If atom is outside the pore, write it to the output pdb.
          # Otherwise, exclude it from the resultant pdb.
          if {![insidePore $x $y $z $sMin $sMax]} {
               # Make the atom serial number accurate if necessary.
               if {[string is integer [string trim $serial]]} {
                    puts -nonewline $out "ATOM  "
                    puts -nonewline $out \
                    [string range [format "    %5i " $atom] end-5 end]
                    puts $out [string range $line 12 end]

               } else {
                    puts $out $line
               }

               incr atom
          }
     }
     close $in
     close $out
}

set lz [readLz $boundaryFile]
drillPore $radiusMin $radiusMax $lz $pdbIn $pdbOut
```

5 In the Tk Console, enter `source drillPore.tcl`.

6 Let's examine the pore we just created in VMD. Enter `mol delete all` and `mol load pdb pore.pdb` in the Tk Console. Setting the Drawing Method to VDW and the Selected Atoms edit box to `abs(y) < 5` should make the double pore cross section apparent.

7 The file produced, `pore.pdb`, needs an accompanying boundary file. In the Tk Console, enter `cp membrane_hex.bound pore.bound`.



Double-cone pore. The membrane is drilled using geometrical criteria which result in a pore shaped like two intersecting cones. This pore resembles those sculptured in silicon nitride by high-energy electron beam (See Heng et al., *Biophysical Journal* **87**, 2905–2911 (2004)).

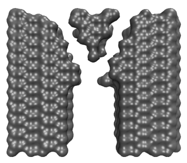
Task 1: Branching pore. Now let's produce a device with a more complex topology. The following criteria define the interior of a Y-shaped branched pore, which one could possibly encounter in a nanofluidic application:

$$\begin{array}{ll} \text{if } z < 0, & x^2 + y^2 < r_0^2 \\ \text{if } z > 0, & y^2 + \frac{1}{5}(z - 2x)^2 < r_1^2 \quad \text{OR} \quad y^2 + \frac{1}{5}(z + 2x)^2 < r_1^2 \end{array} \tag{1}$$

For $z < 0$, the pore is defined by a single cylindrical region, which runs parallel to the z-axis. At the xy-plane, the pore branches; moreover, for $z > 0$, the pore is defined by two cylindrical regions oblique to the z-axis.

Open `drillBranchedPore.tcl` in your text editor. To drill the pore described above, we need to complete the Tcl procedure `insidePore` that begins on line 12. The procedure accepts the atomic coordinates $\{x \ y \ z\}$ and returns 1 if the atom is inside the pore and needs to be removed and returns 0 otherwise. The first portion of the conditional is done for you and defines the shape of the pore for $z < 0$. Using this as your guide, your assignment is to alter the `expr` commands in lines 21 and 22 to correspond to the two criteria (1) that define the two branches for $z > 0$. Note that line 23 returns the logical OR of the two criteria; therefore, you do not need to include this in your modification.

Execute your modified script by entering `source drillBranchedPore.tcl`. In VMD, delete any molecules you have open and open the branched pore (`branch.pdb`). In the Graphical Representations window, set the Drawing Method to MSMS. Setting the Selected Atoms edit box to `abs(y) < 5` reveals the pore's cross section. Does it look how you expected? Compare your pore to the figure below. Did you apply the geometric criteria correctly?



1.3 Generating the structure file

We've constructed two crystalline membranes and, from them, two nanopores; however, we have only generated atom coordinates. We have not defined bonds of any sort between the atoms. In this section, we'll construct a PSF file that describes the bonds (connections between two atoms) and angles (connections between three atoms) in our systems as well as other items needed for subsequent MD simulations. To do this we'll use the script `siliconNitridePsf.tcl`. This script is somewhat longer than those we have seen thus far, so its description has been left to the appendix.

A quick synopsis of the script's operation is as follows. The first step is to find bonds simply by searching for atoms that are within some threshold distance of one another. However, this step misses bonds that exist across the periodic boundaries. To find these, we displace the system by the periodic cell vectors and find bonds between the original system and its periodic image (Fig. 3). Next we determine the angles and then finally write all of the information to a PSF file.

You may be used to calling upon `psfgen` to produce the structure files for proteins and other biomolecules. This would be possible for Si_3N_4 as well. However, due to the nature of the material, it is somewhat more straightforward to generate the PSF directly as we do with this script.

1 We'll now build the PSF structure file for our membrane. Type `source siliconNitridePsf.tcl` in the Tk Console. The structure information for our pore is now contained in `pore.psf`.

2 Let's also build the structure for the pristine membrane. Change line 5 of `siliconNitridePsf.tcl` to `set fileNamePrefix membrane_hex`. Since we will use the pristine membrane to calibrate the dielectric constant of the silicon nitride, we do not want any surfaces. Change line 16 to read `set zPeriodic 1`. Save the script and then execute it.

3 Take a look at the system in VMD by entering `mol delete all` and `mol load psf pore.psf pdb pore.pdb` in the Tk Console. Select Graphics → Representations.... Notice that there appear to be bonds crisscrossing the pore. This occurs because VMD can't correctly display bonds across the periodic boundaries. Set the drawing method Drawing Method to VDW, which does not illustrate bonds. The pore should now be clearly visible.

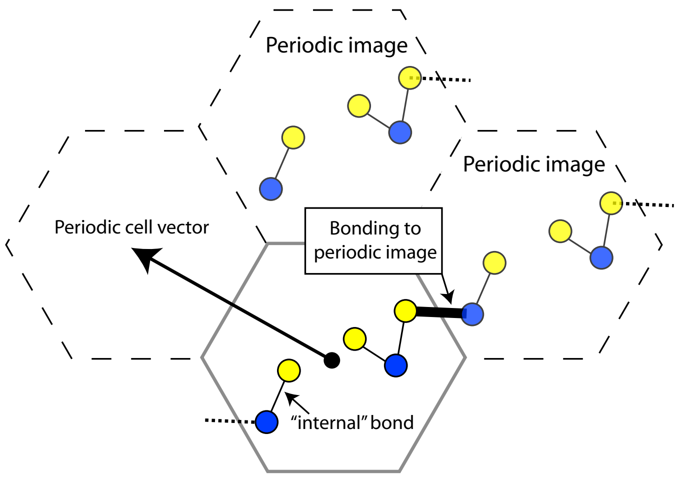


Figure 3: Bonding to periodic images. The periodic image is produced by translating the system by a periodic cell vector. To find bonds across the periodic boundary, a distance search is performed between the original coordinates of the atoms and those in each periodic image.

Charge neutrality. The script `drillPore.tcl` operates by removing atoms defined by geometric constraints. In doing this, it is likely that the ratio of the number of Si atoms to that of N atoms is no longer exactly 3:4. To perform MD simulations with PME electrostatics, the total charge of the simulated system needs to be adjusted to zero. To accomplish this, the charges on all of the nitrogen atoms are tuned by the equation $q_{\mathrm{N}} = -\frac{N_{\mathrm{Si}}\, q_{\mathrm{Si}}}{N_{\mathrm{N}}}$ where N_i and q_i are the number and charge of each species, respectively. For this pore, the adjustment to q_{N} is less than 2% times its absolute value, which is negligible for most purposes.

1.4 Calibrating the force field

Bionanotechnology enters uncharted territory by placing together biomolecules and synthetic materials that have rarely been studied in contact. In addition, simulations of inorganic solids such as Si_3N_4 usually employ vastly different methods than those used in computational molecular biology. Thus, simulating systems with both synthetic and biomolecular constituents is challenging and, in general, an unsolved problem. Because much research in bionanotechnology involves electrostatic interactions between biomolecules and silicon-based materials, we'll focus on getting our Si_3N_4 model to reproduce experimental data for just one property: the dielectric constant. With this model we can expect to have a realistic electric field within the pore.

To determine the dielectric constant, we will apply an electric field to a block of Si_3N_4 with no free surfaces and measure the electric dipole moment. Hence, we will use the structure `membrane_hex.psf` that we generated in the last section, for which we generated bonds along all three lattice directions.

1 Type `cd ../2_calibrate/` in the Tk Console.

2 Open the parameter file `par_silicon_ions_NEW0.1.inp` in your text editor. Notice that the file has three sections. The first two give energy function parameters for harmonic bonds and harmonic angle bending between two bonds, respectively. The last gives the parameters for the non-bonded interactions. You may close the file now.

We take the non-bonded parameters, as well as the values for the partial charges on the Si and N atoms in `siliconNitridePsf.tcl`, from quantum mechanical calculations using the biased Hessian method (John A. Wendell and William A. Goddard III, *Journal of Chemical Physics* **97**, 5048–5062 (1992)). However, the bonded interactions from the same source lead to a dielectric constant that is practically the same as a vacuum (1.0). To overcome this, we set the bonded interaction constants to be much lower than those given in the reference. In this section, we'll set them to 0.1 kcal/(mol Å). To match the experimental dielectric constant we include in our force field harmonic restraints, which can easily be applied in NAMD, that pull each atom of Si_3N_4 towards its equilibrium position in the Si_3N_4 crystal. It is the spring constant associated with these constraint forces that we will calibrate to reproduce the experimentally-determined dielectric constant of Si_3N_4.

Silicon nitride parameters. To change the bonded and van der Waals interaction parameters, you need only to modify the parameter file `par_silicon_ions_NEW0.1.inp`. However, note that the atomic charges of the Si_3N_4 are defined in the PSF file. If you'd like to alter these, you must change the variable `chargeSi` in the PSF generating script (See Appendix).

3 The Tcl script `constrainSilicon.tcl` produces PDB files where the spring constant is placed in the B (known in VMD as beta) column of the PDB. Open the script in your text editor. A constraint PDB will be produced for each spring constant (kcal/(mol Å^2)) in the list defined in line 7. We'll determine the dielectric constant for values 1.0 and 10.0. Hence, change line 7 of the script to `set betaList {1.0 10.0}`. Execute `constrainSilicon.tcl`, whose contents follow.

<u>constrainSilicon.tcl</u>

```
# Add harmonic constraints to silicon nitride.

# Parameters:
# Spring constant in kcal/(mol A^2)
set betaList {1.0}
set selText "resname SIN"
set surfText "(name \"SI.*\" and numbonds<=3) \
or (name \"N.*\" and numbonds<=2)"
# Input:
set psf ../1_build/membrane_hex.psf
set pdb ../1_build/membrane_hex.pdb
# Output:
set restFilePrefix siliconRest

mol load psf $psf pdb $pdb
set selAll [atomselect top all]

# Set the spring constants to zero for all atoms.
$selAll set occupancy 0.0
$selAll set beta 0.0

# Select the silicon nitride.
```

```
set selSiN [atomselect top $selText]

# Select the surface.
set selSurf [atomselect top "(${selText}) and (${surfText})"]

foreach beta $betaList {
     # Set the spring constant for SiN to this beta value.
     $selSiN set beta $beta
     # Constrain the surface 10 times more than the bulk.
     $selSurf set beta [expr 10.0*$beta]
     # Write the constraint file.
     $selAll writepdb ${restFilePrefix}_${beta}.pdb
}
$selSiN delete
$selSurf delete
$selAll delete
mol delete top
```

4 Since the silicon atoms are already in their equilibrium positions, we'll forgo the energy minimization step in the usual simulation sequence. Instead, we'll start by raising the temperature gradually to 295 K. During this time, we'll use constraints of 1.0 kcal/(mol Å^2).

Before we start, however, we need to put the system dimensions in the NAMD configuration file `eq1.namd`. Open it and `1_build/membrane_hex.bound` (if you did not use InorganicBuilder), which we generated in Section 1.2, in your text editor. If you used InorganicBuilder, refer instead to the vectors you recorded. Copy the values of `cellBasisVector1`, `cellBasisVector2`, and `cellBasisVector3` into lines 8, 9, and 10, respectively, of the configuration file. Also, examine the constraint parameters at the bottom of the file. Save the configuration file and exit the text editor.

5 Enter `namd2 eq1.namd > eq1.log` to raise the system's temperature. This may take a couple of minutes.

6 To equilibrate the system at constant temperature, enter `namd2 eq2.namd > eq2.log`.

7 Next we compute the dielectric constant for each constraint value. To do this, we calculate the difference between the dipole moments of identical systems with and without an applied electric field. Open the files `field.namd` and `null.namd` in your text editor. Modify line 2 to read `set constraint 1.0`. First simulate the system without the applied electric field by entering `namd2 null.namd >! null1.0.log` and then with a field of 16 kcal/(mol Å e) by entering `namd2 field.namd >! field1.0.log`. Do the same for the other constraint value, i.e., alter the variable `constraint` in `field.namd` and `null.namd` and run NAMD.

8 We'll now compute the electric dipole moment for each run and from these calculate the dielectric constant for the material. Open the script `dipoleMomentZDiff.tcl` in your text editor. The script operates by loading DCD trajectory files for the system with and without an applied field. We then compute the dipole moment for each frame and write the time (ns) in the first column and the difference in the dipoles (e Å) in the second column of a text file.

The values of `dcdFreq` and `timestep`, taken from the NAMD configuration file, allow us to determine the time between the frames of the DCD trajectory file. We'll set the variable `startFrame` to 4 to give the system 500 fs to equilbrate before computing the dipole moment. The electric dipole moment is computed by VMD's `measure dipole` command which employs the following formula. For a set of N atoms with partial charges q_i and positions $\mathbf{r}_i$ the electric dipole moment is

$$\mathbf{p} = \sum_{i=1}^{N} (q_i - q_0)\mathbf{r}_i,$$

where $q_0 = \frac{1}{N}\sum_{i=1}^{N} q_i$. Subtraction of q_0, the monopole component, makes the result independent of the choice of the origin. Finally, the script computes the average of the difference in the dipole moments and the associated standard error.

`dipoleMomentZDiff.tcl`

```
# Calculate dipole moment of the selection
# for a trajectory.
```

```
set constraint 10.0
set dcdFreq 100
set selText "all"
set startFrame 0
set timestep 1.0

# Input:
set psf ../1_build/membrane_hex.psf
set dcd field${constraint}.dcd
set dcd0 null${constraint}.dcd
# Output:
set outFile dipole${constraint}.dat

# Get the time change between frames in femtoseconds.
set dt [expr $timestep*$dcdFreq]

# Load the system.
set traj [mol load psf $psf dcd $dcd]
set sel [atomselect $traj $selText]
set traj0 [mol load psf $psf dcd $dcd0]
set sel0 [atomselect $traj0 $selText]

# Choose nFrames to be the smaller of the two.
set nFrames [molinfo $traj get numframes]
set nFrames0 [molinfo $traj0 get numframes]
if {$nFrames0 < $nFrames} {set nFrames $nFrames}
puts [format "Reading %i frames." $nFrames]

# Open the output file.
set out [open $outFile w]

# Start at "startFrame" and move forward, computing
# the dipole moment at each step.
set sum 0.
set sumSq 0.
set n 1
puts "t (ns)\tp_z (e A)\tp0_z (e A)\tp_z-p0_z (e A)"
for {set f $startFrame} {$f < $nFrames && $n > 0} {incr f} {
     $sel frame $f
     $sel0 frame $f

     # Obtain the dipole moment along z.
     set p [measure dipole $sel]
     set p0 [measure dipole $sel0]
     set z [expr [lindex $p 2] - [lindex $p0 2]]

     # Get the time in nanoseconds for this frame.
     set t [expr ($f+0.5)*$dt*1.e-6]

     puts $out "$t $z"
     puts -nonewline [format "FRAME %i: " $f]
     puts "$t\t[lindex $p 2]\t[lindex $p0 2]\t$z"

     set sum [expr $sum + $z]
     set sumSq [expr $sumSq + $z*$z]
}
close $out

# Compute the mean and standard error.
set mean [expr $sum/$nFrames]
set meanSq [expr $sumSq/$nFrames]
set se [expr sqrt(($meanSq - $mean*$mean)/$nFrames)]

puts ""
puts "********Results: "
puts "mean dipole: $mean"
puts "standard error: $se"
mol delete top
mol delete top
```

9 Execute the script `dipoleMomentZDiff.tcl` twice, setting `constraint` (in line 6 of the script) to each of the values in our simulations. Be sure to write down the mean dipole and standard error for each.

10 Plot the time versus dipole moment data stored the resulting files `dipole10.0.dat` and `dipole1.0.dat`. You should see that the dipole moments are changing little with time by the end of the simulation and that the values are significantly different for the two different constraint parameters.

11 To calculate the dielectric constant we apply the formula

$$\kappa = 1 + \frac{\Delta p}{\epsilon_0 E V},$$

where Δp is the magnitude of the difference in the dipole moment between identical systems with and without an electric field, E is the magnitude of the applied electric field, and V is the volume of the system dielectric material (See Dong Xu, et al., *The Journal of Physical Chemistry* **100**, 12108–12121 (1996) for further discussion). The permittivity of free space is given in NAMD units by $\epsilon_0 = 2.398 \times 10^{-4}$ (mol e^2)/(kcal Å). We can calculate the volume of our hexagonal prism by

$$V = \frac{3\sqrt{3}}{2} R^2 l_z,$$

where R is the radius of the hexagon and l_z is the height of the prism. Obtaining R and l_z from `membrane_hex.bound`, we find $V = 23485$ Å^3. Given that $E = 16.0$ kcal/(mol Å e) calculate the dielectric constants for the two constraint values using the mean dipole values. Note that you can use the form Tk Console as a calculator by typing `expr` commands. Is the difference in the dielectric constant between the two significant?

This section is only meant to be a demonstration of how the calibration is performed. Sampling the entire parameter space takes a good deal of time, but you should now have a good understanding of how to calibrate the constraints to reproduce the experimental dielectric constant. In subsequent sections, we will use a parameter file with the bond constants set to 5.0 kcal/(mol Å^2) and a constraint file with constants of 1.0 kcal/(mol Å^2), which have been found to be optimal by the procedure above.

1.5 Solvating the nanopore

Now that we've demonstrated how to calibrate the force field of our Si_3N_4 model, we're ready to prepare our nanopore for simulations.

1 In the Tk Console, type `cd ../3_solvate/`.

All biological systems rely on water to function. If our synthetic device is to interact with them, it must be immersed in water.

2 Open the system we wish to solvate by entering `mol load psf ../1_build/pore.psf pdb ../1_build/pore.pdb` in the Tk Console.

3 To open the Solvate plugin, select Extensions → Modeling → Add Solvation Box from the VMD menu.

4 You should already see `../1_build/pore.psf` and `../1_build/pore.pdb` in the edit boxes labeled PSF and PDB, respectively. Set Output to `pore_solv`. Since we wish to have water above and below the membrane, set the minimum and maximum Box Padding in the direction z to 20.

5 Press Solvate.

6 Notice that the Solvate plugin adds the water in a right rectangular prism, which does not conform to our hexagonal prism periodic boundary conditions. Type `mol delete all` in the Tk Console.

We'll now remove water from outside of the hexagonal boundaries with the script `cutWaterHex.tcl`. It uses VMD's atom selection interface to obtain the set {`segname`, `resid`, `name`}, which uniquely specifies each atom, for all atoms violating the geometric constraints that we used in Section 1.2 to cut a hexagon from our crystal. Then by applying the `psfgen` command `delatom`, violating atoms are deleted. We estimate the radius of the hexagon with the `measure minmax` command provided by VMD.

<u>`cutWaterHex.tcl`</u>

```
# This script will remove water from psf and pdf outside of a
# hexagonal prism along the z-axis.
package require psfgen 1.3

# Input:
set psf pore_solv.psf
set pdb pore_solv.pdb
```

```
# Output:
set psfFinal pore_hex.psf
set pdbFinal pore_hex.pdb

# Parameters:
# The radius of the water hexagon is reduced by "radiusMargin"
# from the pore hexagon. The distance is in angstroms.
set radiusMargin 0.5
# This is the stuff that is removed.
set waterText "water or ions"
# This selection forms the basis for the hexagon.
set selText "resname SIN"

# Load the molecule.
mol load psf $psf pdb $pdb

# Find the system dimensions.
set sel [atomselect top $selText]
set minmax [measure minmax $sel]
$sel delete
set size [vecsub [lindex $minmax 1] [lindex $minmax 0]]
foreach {size_x size_y size_z} $size {break}
# This is the hexagon's radius.
if {[expr $size_x > $size_y]} {
     set rad [expr 0.5*$size_x]
} else {
     set rad [expr 0.5*$size_y]
}
set r [expr $rad - $radiusMargin]

# Find water outside of the hexagon.
set sqrt3 [expr sqrt(3.0)]
# Check the middle rectangle.
set check "($waterText) and ((abs(x) < 0.5*$r and abs(y) > 0.5*$sqrt3*$r) or"
# Check the lines forming the nonhorizontal sides.
set check [concat $check "(y > $sqrt3*(x+$r) or y < $sqrt3*(x-$r) or"]
set check [concat $check "y > $sqrt3*($r-x) or y < $sqrt3*(-x-$r)))"]
set w [atomselect top $check]
set violators [lsort -unique [$w get {segname resid}]]
$w delete

# Remove the offending water molecules.
puts "Deleting the offending water molecules..."
resetpsf
readpsf $psf
coordpdb $pdb
foreach waterMol $violators {
     delatom [lindex $waterMol 0] [lindex $waterMol 1]
}

writepsf $psfFinal
writepdb $pdbFinal
mol delete top
```

7 Enter `source cutWaterHex.tcl` to remove water outside of the hexagonal boundaries.

8 Open the new structure by entering `mol load psf pore_hex.psf pdb pore_hex.pdb`. Does the system now conform to a hexagonal prism?

Many biomolecules are sensitive to the ionic strength of the surrounding solvent; therefore, salt is added to the solutions used in experiments to mimic physiological conditions. In addition, ions facilitate measurements of small currents in nanopore systems by substantially increasing the conductivity of the solution.

9 To open the Autoionize plugin, select Extensions → Modeling → Add Ions from the VMD menu.

10 You should already see `pore_hex.psf` and `pore_hex.pdb` in the edit boxes labeled PSF and PDB, respectively. Set Output to `pore_all`. Since we wish to have a 2 mol/kg KCl concentration, set Concentration to 4. Set both Min. distance from molecule and Min. distance between ions to 2. Also, because we are using a KCl solution instead of NaCl, select the checkbox labeled Switch to KCl instead of NaCl.

11 Execute the Autoionize plugin by pressing Autoionize.

12 For convenience, copy the solvated structure into the directory for the next section by typing `cp pore_all.psf ../4_current/` and `cp pore_all.pdb ../4_current/`.

1.6 Measuring ionic current

In experiment, ionic current is a macroscopic quantity that gives insight into nanoscale processes. Ionic current measurements are used to characterize single nanopores and their interactions with biological molecules. In this subsection, we'll learn to simulate our nanopore system with an applied voltage and calculate the ionic current from the trajectory.

1 Enter `cd ../4_current/` in the Tk Console to change to the directory for this subsection. Be sure that you have copied the files `pore_all.psf` and `pore_all.pdb` into this directory as instructed at the end of the last subsection.

2 First we need to generate the constraint file using the parameters that reproduce the experimental dielectric constant. In the Tk Console, enter
`source constrainSilicon.tcl`.

3 Now we need to equilibrate our system. We'll start by performing energy minimization. Take a look at the NAMD configuration file `eq0.namd` in your text editor. The values given for `cellBasisVector1` and `cellBasisVector2` match those given in `../1_build/membrane_hex.bound`. If you used InorganicBuilder to generate the pore, you should replace values with your own. The third basis vector is dependent on the size of the water box we added. To determine it, in the Tk Console window (Extensions → Tk Console) type the following commands:

```
mol delete all
mol load psf pore_all.psf pdb pore_all.pdb
set all [atomselect top all]
set minmax [measure minmax $all]
set lz [expr [lindex $minmax 1 2]-[lindex $minmax 0 2]]
$all delete
```

The value of `lz` gives us the size (Å) of the system along the z-axis. We don't want to put this value directly into the NAMD configuration file, however.

It is better for a few water molecules to be crowded at the ends at this point than risk introducing a vacuum region at the ends. While the minimization step can easily rearrange water molecules that have been placed too close together due to wrapping at the periodic boundaries, small regions of vacuum can cause inaccuracies in simulations, especially those performed at constant pressure, that can be difficult to catch.

For this reason, we set `cellBasisVector3` to `lz` minus about 5 Å. Since we get about 55.9 Å for `lz`, line 13 of `eq0.namd` should read `cellBasisVector3 0.0 0.0 51.0`.

4 While we have our system open in VMD, let's take a look at it. Select Graphics → Representations.... In the Graphical Representations window, set Selected Atoms to `resname SIN` to see only the Si_3N_4. Set the drawing method Drawing Method to VDW. Now create a new representation (by pressing Create Rep) with Selected Atoms set to `ions`. The K^+ and Cl^- ions within the pore should be visible. When you are finished examining the system, enter `mol delete all` in the Tk Console.

5 To perform energy minimization, enter `namd2 eq0.namd > eq0.log` in the terminal window. This may take a few minutes to execute. During this time you may want to take a look at the next step in the equilibration process `eq1.namd`. When the minimization completes, check the end of log file `eq0.log` to be certain that the simulation completed successfully.

NAMD script	*steps*	*description*
eq0.namd	201	energy minimization
eq1.namd	500	raise temperature from 0 to 295 K, constant V
eq2.namd	1000	equilibrate, constant p and Langevin thermostat
run0.namd	1000	apply 20 V, constant V

The table above summarizes the NAMD runs we will perform in this section. It consists of three equilibration stages and one run with an applied field. Stages such as these are used in most production simulaions.

6 Enter `namd2 eq1.namd > eq1.log` to gradually raise the system's temperature from 0 K to 295 K at constant volume.

7 Examine the NAMD configuration file `eq2.namd` in your text editor. Notice the block of commands below the comment `# pressure control`. These set the parameters for the Langevin piston Nosé-Hoover method implemented in NAMD to maintain atmospheric pressure. Close the text editor and equilibrate the system by entering `namd2 eq2.namd > eq2.log`.

8 Constant pressure simulations allow the volume of the system to change. As a necessary condition for equilibrium, the volume should fluctuate about a mean value. Select Extensions → Analysis → NAMD Plot from VMD's menu. In the NAMD Plot window, select File → Select NAMD Log File, highlight `eq2.log`, and press Open. Select for VOLUME for the y-axis data. Now, plot the system volume versus time step by selecting File → Plot Selected Data. You should notice a significant downward trend in the volume. At equilibrium, the volume fluctuates about a mean value for an NpT system such as this. Hence, we have not equilibrated long enough. Since our time in this tutorial is limited, a system that has been equilibrated for 0.5 ns is included in this directory.

9 We are now ready to apply an electric field and simulate the flow of ionic current. Because the total current is more simply related to voltage than the electric field magnitude, we are going to apply a potential difference of 20 V along the $-z$-axis of our system. The corresponding uniform electric field is calculated by $E_z = -U/l_z$, where U is the potential difference and l_z is the size of the system along the z-axis. The NAMD unit for electric field is kcal/(mol Å e); thus, the appropriate conversion factor for U in V and l_z in Å is 23.0605492. That is,

$$\texttt{eField}_z/\left(\frac{\text{kcal}}{\text{mol Å e}}\right) = -23.060549\frac{U/\text{V}}{l_z/\text{Å}}.$$

To obtain the value of l_z, open the NAMD extended system configuration file `sample.xsc` in your text editor. Write down `c_z`, the tenth number in the row of system parameters. Using $V = 20$ V and $l_z = -$`c_z` $\AA$, calculate eField$_z$. Note that since the potential difference is applied the along $-z$-axis, eField$_z$ is positive.

10 Now open `run0.namd`. At the bottom of the file you will see the following lines:

```
eFieldOn on
eField 0.0 0.0 0.0
```

Change the third component of `eField` to the value of eField$_z$ that you calculated. Before you close the `run0.namd`, note that the pressure control lines are absent. Applying an electric field to a pressure-controlled system will distort it, leading to erroneous results. In addition, note that the Langevin temperature control is only applied to the silicon nitride. Applying Langevin forces to the ions, whose motion due to the electric field we are trying to measure, could lead to a subtle bias in the current.

11 Begin the simulation by entering `namd2 run0.namd > run0.log` in the terminal window. The simulation may require a couple minutes. Feel free to read ahead while it runs.

12 We are using a very high applied electric field due to the time constraints of this tutorial. If you analyze the temperature of the simulation versus time step using the VMD plugin NAMD Plot (whose use was described during the equilibration phase of this section), you'll see that the temperature rises above 450 K, because of the large ionic current. Such temperatures would render a production simulation invalid. In real simulations, we would be using a much smaller electric field.

13 Load the VMD save state by selecting File → Load State... and then the file `current.vmd`. Step through your trajectory and you should notice that the K^+ ions (in red) move upward, while the Cl^- (in blue) ions move downward. Enter `mol delete all` in the Tk Console.

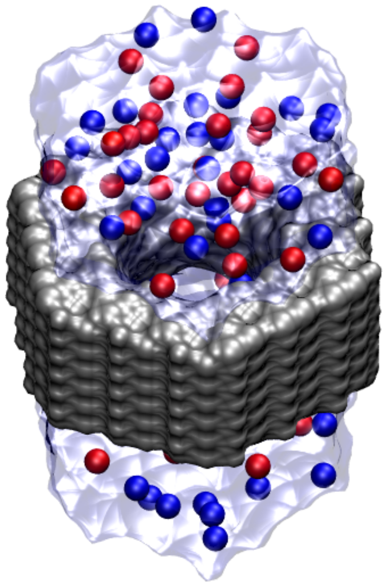


Figure 4: Complete silicon nitride nanopore (grey) including water and potassium (red) and chloride (blue) ions.

14 The parameter `dcdFreq` is set to 100 in the NAMD configuration file. As you may already know, this means that NAMD writes the coordinates of every atom to a DCD file every 100 simulation steps. To calculate the ionic current, we will execute the Tcl script `electricCurrentZ.tcl`. It computes the ionic current by

$$I(t + \Delta t/2) = \frac{1}{\Delta t\, l_z} \sum_{i=1}^{N} q_i(z_i(t + \Delta t) - z_i(t)),$$

where z_i and q_i are respectively the z-coordinate and charge of ion i and Δt is the simulation time represented by `dcdFreq`. Execute the script by entering `source electricCurrentZ.tcl`.

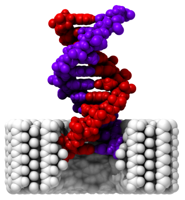
electricCurrentZ.tcl

```
# Calculate the current for a trajectory.
# Results are in "time(ns) current(nA)"

set dcdFreq 100
set selText "ions"
set startFrame 0
set timestep 1.0

# Input:
set pdb  sample.pdb
set psf  sample.psf
set dcd  run0.dcd
set xsc run0.restart.xsc
# Output:
set outFile curr_20V.dat

# Get the time change between frames in femtoseconds.
set dt [expr $timestep*$dcdFreq]

# Read the system size from the xsc file.
# Note: This only works for lattice vectors along the axes!
set in [open $xsc r]
foreach line [split [read $in] "\n"] {
     if {![string match "#*" $line]} {
          set param [split $line]
          puts $param
          set lx [lindex $param 1]
          set ly [lindex $param 5]
          set lz [lindex $param 9]
          break
     }
}
puts "NOTE: The system size is $lx $ly $lz.\n"
close $in

# Load the system.
mol load psf $psf pdb $pdb
```

```
set sel [atomselect top $selText]

# Load the trajectory.
animate delete all
mol addfile $dcd waitfor all
set nFrames [molinfo top get numframes]
puts [format "Reading %i frames." $nFrames]

# Open the output file.
set out [open $outFile w]
#puts $out "sum of q*v for $psf with trajectory $dcd"
#puts $out "t(ns) I(A)"

for {set i 0} {$i < 1} {incr i} {
     # Get the charge of each atom.
     set q [$sel get charge]

     # Get the position data for the first frame.
     molinfo top set frame $startFrame
     set z0 [$sel get z]
}

# Start at "startFrame" and move forward, computing
# current at each step.
set n 1
for {set f [expr $startFrame+1]} {$f < $nFrames && $n > 0} {incr f} {
     molinfo top set frame $f

     # Get the position data for the current frame.
     set z1 [$sel get z]

     # Find the displacements in the z-direction.
     set dz {}
     foreach r0 $z0 r1 $z1 {
          # Compensate for jumps across the periodic cell.
          set z [expr $r1-$r0]
          if {[expr $z > 0.5*$lz]} {set z [expr $z-$lz]}
          if {[expr $z <-0.5*$lz]} {set z [expr $z+$lz]}
          lappend dz $z
     }

     # Compute the average charge*velocity between the two frames.
     set qvsum [expr [vecdot $dz $q] / $dt]

     # We first scale by the system size to obtain the z-current in e/fs.
     set currentZ [expr $qvsum/$lz]
     # Now we convert to nanoamperes.
     set currentZ [expr $currentZ*1.60217733e5]
     # Get the time in nanoseconds for this frame.
     set t [expr ($f+0.5)*$dt*1.e-6]

     # Write the current.
     puts $out "$t $currentZ"
     puts -nonewline [format "FRAME %i: " $f]
     puts "$t $currentZ"

     # Store the postion data for the next computation.
     set z0 $z1
}
close $out
mol delete top
}
```

15 The script `electricCurrentZ.tcl` produces an output file `curr_20V.dat`, which has two columns that record the time (ns) and the current (nA). Open the file `curr_20V.dat` in a text editor. Is the current steady? What is its mean value?

Challenge: Ionic current in branched pore. Measure the ionic current of the branched pore (or another pore of your design). First, use `siliconNitridePsf.tcl` to generate the structure `branch.psf` for `branch.pdb`. Next, follow the steps in Section 1.3 to solvate the branched pore. Then equilibrate the system and run it with an applied electric field as described in this section. How does the current compare to double-cone pore?



Detecting single molecules by the measurement of ionic current. When DNA is driven into a pore, be it a natural protein channel or synthetic nanopore, large changes in the ionic current can be measured experimentally. While within the pore, the molecule often causes a transient reduction in the ionic current. The duration of this current reduction has been found to be proportional to the length of the DNA molecule and sensitive to single nucleotide substitution in DNA hairpins (See Kasianowicz et al., *Proceedings of the National Academy of Sciences* **93**, 13770–13773 (1996) and Bezrukov et al., *Nature* **370**, 279–281 (1994)). Thus, measurement of ionic current through a nanopore can be used to detect and study single molecules.

2 Simulations of DNA permeation through nanopores

In the second unit, you will learn how to manipulate DNA molecules and simulate their permeation through a synthetic nanopore.

2.1 Manipulating DNA

1 Enter `cd ../5_manipulate_dna/` to start this section.

2 In the Tk Console type

`mol load psf dsDnaAmber.psf pdb dsDnaAmber.pdb` In the Graphical Representations window, set the Drawing Method to Licorice and the Coloring Method to ResName.

3 You should now see an 8-basepair molecule of double-stranded DNA (dsDNA), colored by the residue names ADE, CYT, GUA, and THY; which correspond respectively to the bases adenine, cytosine, guanine, and thymine. (See Fig. 5. Try setting Selected Atoms in the Graphical Representations window to `resname ADE`, `resname CYT`, `resname GUA`, and `resname THY` in turn. Which colors correspond to which bases?

4 To determine the base sequence for the first strand, type the following in the Tk Console:

```
set a [atomselect top "segname ADNA and name C1'"]
puts [$a get {resid resname}]
$a delete
```

What is the sequence of the first strand (segment name ADNA)? What is the sequence of its complementary strand (segment name BDNA)?

5 There are several sets of parameters available for molecular modeling of DNA. We'll use the AMBER topology given in `cornell.rtf` and the interaction parameters given in `cornell.prm`. Another popular model of DNA uses the Charmm topology and parameter set. The script `convertDnaToCharmm.tcl` can produce a Charmm model from our AMBER model. The script applies patches using `psfgen` to change

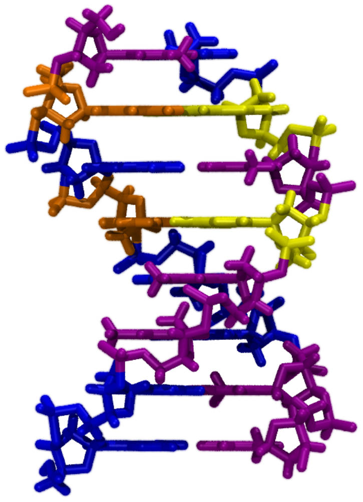


Figure 5: Double-stranded DNA colored by base type.

the topology from that of the AMBER model to that of the Charmm model using the Charmm topology file `top_all27_prot_na.inp`. Execute this script by typing `source convertDnaToCharmm.tcl` in the Tk Console.

6 In the Tk Console, enter
`mol load psf dsDnaCharmm.psf pdb dsDnaCharmm.pdb`. Set the Drawing Method to Licorice, the Coloring Method to Molecule, and Selected Atoms to `all` for both the AMBER DNA that we loaded earlier and the Charmm DNA. No difference in structure between the AMBER model and the Charmm model should be apparent. In the Tk Console, enter `mol delete all`.

7 Now we would like to produce single-stranded DNA (ssDNA) from `dsDnaAmber.psf` and `dsDnaAmber.pdb`. The script `removeResidues.tcl` deletes the residues of all atoms in a given selection. Open the script in your text editor. The first and second DNA strands have the segment names ADNA and BDNA, respectively. Set the value of `selText` in line 6 to `segname BDNA` so that the script will delete the second DNA strand. Save your changes and execute the script.

8 Let's check that we produced the ssDNA correctly. Enter `mol load psf ssDna.psf pdb ssDna.pdb` in the Tk Console. After examining your 8-mer ssDNA, type `mol delete all`.

ssDNA is much more flexible than dsDNA and easily bends into various conformations. The details of these conformations can be important for applications of bionanotechnology. For example, if ssDNA is to pass through a nanopore device, such as is proposed for a means of fast sequencing, it must be aligned somewhat along the axis of the pore. Molecules lying in the plane of the membrane or contorted in certain ways can make translocation more difficult or impossible. For this reason, we want the ability to easily generate any desired DNA conformation *in silico*.

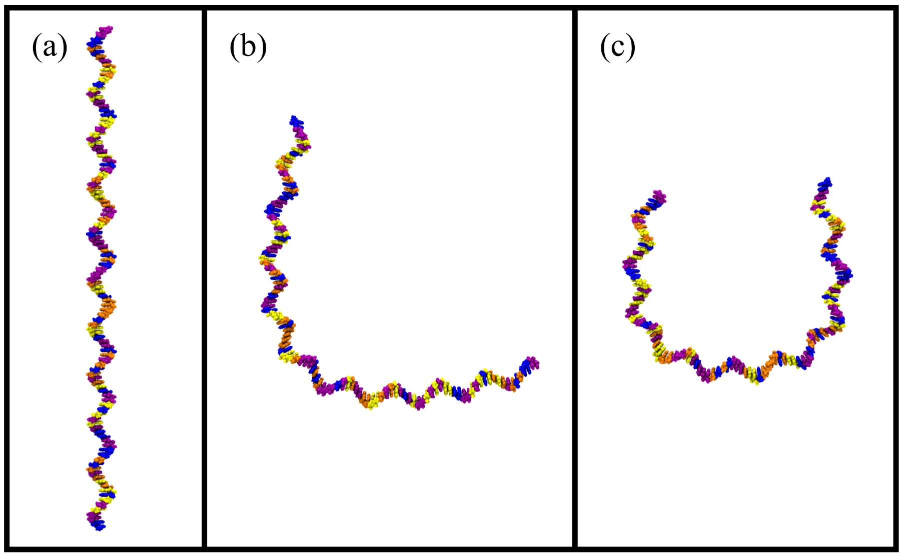


Figure 6: Shaping single-stranded DNA. (a) The DNA begins in a straight conformation. (b), (c) Bending the DNA with Sculptor using two different paths as described in the text.

9 Here we will use the VMD script `sculptor.tcl` to shape ssDNA to our will. In the Tk Console, enter the following lines to load a 110-mer ssDNA molecule and open Sculptor:

```
mol load psf ssDnaLong.psf pdb ssDnaLong.pdb
source sculptor.tcl
sculptorGui
```

The Sculptor window should open. The script will map any long molecule aligned along the z-axis to a cubic spline whose form is given by the points in Path. If we are careful, the cubic spline allows us to bend the ssDNA smoothly, leading to conformations, that with some equilibration, could occur in nature. However, using Sculptor on structures that are not relatively straight along the z-axis, applying a tortuous path, or pressing the Sculpt button more than once without undoing the last operation will result in highly distorted and unphysical conformations. If this happens, simply reload the molecule.

10 Let's start by bending ssDNA into an L-shape. Delete the contents of Path, and type `{0 0 1} {0 0 0} {1 0 0}`. Press Sculpt. Rotate the molecule a bit and then press Undo. Your result should look like Fig. 6(b).

11 Now we'll bend the ssDNA in a U-shape. Delete the contents of Path and type `{0 1 2} {0 1 0} {0 -1 0} {0 -1 2}`. Imagine the positions of these coordinates in space. You should see that they form three sides of a rectangle. Production of a cubic spline from these control points will yield a U-shape as shown in Fig. 6(c). Press Sculpt. Undo this and then produce a few conformations of your own. Close Sculptor when you are finished. Then enter `mol delete all`.

2.2 Combining DNA and the synthetic nanopore

1 We now will combine our 8-mer ssDNA molecule with the Si_3N_4 nanopore. Execute the script `combine.tcl`, which will create `pore+dna.psf` and `pore+dna.pdb`. As shown below, the script combines the pore we created in Section 1.2 with the ssDNA using `psfgen`. The script is rather general, but can run into problems if segment names are duplicated between the scripts.

`combine.tcl`

```
# Input:
set psf0 ../1_build/pore.psf
set pdb0 ../1_build/pore.pdb
set psf1 ssDna.psf
set pdb1 ssDna.pdb
# Output:
set finalPsf pore+dna.psf
set finalPdb pore+dna.pdb

# Load the topology and coordinates.
package require psfgen
resetpsf
readpsf $psf0
coordpdb $pdb0
readpsf $psf1
coordpdb $pdb1

# Write the combination.
writepdb $finalPdb
writepsf $finalPsf
```

2 We've added the ssDNA without regard for the position of the pore. We now need to adjust the position of the molecule so that it is in a reasonable position for our translocation simulation. What is the charge of DNA? Which way will it move in an electric field pointing along the z-axis? Enter `mol load psf pore+dna.psf pdb pore+dna.pdb` in the Tk Console. Examine the system in the VDW representation. Using selection text like `segname ADNA and within 4.0 of resname SIN` allows us to see where the DNA has been placed too close to the Si_3N_4.

Type the following commands into the Tk Console:

```
set sel [atomselect top "segname ADNA"]
$sel moveby {4 1 7}
set all [atomselect top all]
$all writepdb pore+dna.pdb
$sel delete
$all delete
```

VMD will not automatically update a selection defined by `within` commands after the ssDNA has been moved. To see the changes, simply change one letter in the Selected Atoms box, change it back, and press `Enter`. When you are convinced that the ssDNA is not too close to the Si_3N_4, enter `mol delete all`.

Task 2: A different conformation. Load your combined system by entering `mol load psf pore+dna.psf pdb pore+dna.pdb`. By applying Sculptor to just the DNA (by setting Selection Text in the Sculptor window) and moving the DNA with `moveby` commands, create situation where DNA is blocking the pore, but with a substantially different conformation than before. Save the result as `pore+dna_other.pdb`.

2.3 Measuring ionic current with DNA

1 We've been running a lot scripts in our VMD session, some of which may have large global variables. This might be a good time to exit VMD and start a new VMD session to free any memory in these variables.

2 Enter `cd ../6_current_dna/` in the Tk Console. Execute the solvation scripts `addWater.tcl`, `cutWaterHex.tcl`, and `addIons.tcl` in sequence.

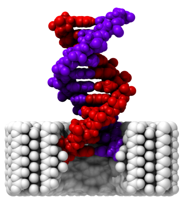
3 To save the time it takes to equilibrate the system, we've included an equilibrium system (`sample*`) with which you can continue.

4 Calculate the value of `eField` necessary to apply 20 V along the $-z$-axis of the system with data from `sample.xsc` as you did in Section 1.6. Place this value in the configuration file `run0.namd` and execute NAMD with this file.

5 Execute the script `electricCurrentZ.tcl` to determine the ionic current. How does it compare with what you measured with no DNA in the system?

Task 3: Comparing ionic currents. Plot the ionic current as function of time for the DNA-free system of Section 1.6 and the system from this section. How does the presence of DNA affect the current?

Challenge: Dependence of ionic current on the conformation. By changing the input files to the solvate scripts, solvate the system you created in **Task 2**, defined in the files `pore+dna_other.pdb` and `pore+dna.psf`. Equilibrate the system and calculate the ionic current as in the previous section. Does the difference in conformation change the results?



Enhanced ionic current. In some situations, the presence of DNA in the pore leads to an increase rather than a decrease in ionic current. There appear to be two competing mechanisms whose dominance depends on the bulk ion concentration. First, the DNA mechanically blocks the pore, excluding ions in the volume it occupies. However, because DNA is charged, the concentration of ions in its vicinity is greater than in the bulk. These ideas are discussed further in Chang et al., *Nano Letters* **4**, 1551–1556 (2004) and Smeets et al., *Nano Letters* **6**, 89–95 (2006).

2.4 Simulating DNA translocation

1 Enter `cd ../7_translocate/`. In production simulations, translocation would be performed in solution. However, due to time constraints, we'll perform the translocation simulation in vacuum and then analyze the provided trajectory for a similar simulation in solution. We will also be using only short-range electrostatics (with a 12 Å cutoff) instead of PME electrostatics because the vacuum system has a nonzero charge. Electrostatic cutoffs are not recommended for most production simulations.

2 Execute `constrainSilicon.tcl`.

3 Run the NAMD configuration scripts shown in the table below sequentially. If you generated the pore with InorganicBuilder, you need to change `cellBasisVector1` and `cellBasisVector2` in `eq0.namd` to those you recorded. The final simulation may take several minutes to run, so if you are short on time you may want to skip this step and the one that follows.

NAMD script	*steps*	*description*
eq0.namd	201	minimization
eq1.namd	500	raise temperature from 0 to 295 K at constant V
eq2.namd	1000	constant p and Langevin thermostat
run0.namd	8000	electric field 150 kcal/(mol Å e) at constant V

4 View the resulting trajectory in VMD by entering `mol delete all` and `mol load psf pore+dna.psf dcd run0.dcd`. Change the representation to VDW. Does the ssDNA translocate from one side of the pore to another?

5 Since your simulation was performed in a vacuum, we cannot analyze the ionic current. For this reason, the trajectory `translocate.dcd` along with the structure `translocate.psf` and extended system `translocate.xsc` has been provided. The data is from a 6 V translocation simulation of dsDNA. Execute `electricCurrentZFrame.tcl` to calculate the ionic current for this trajectory. Unlike the script of a similar name we used previously, `electricCurrentZFrame.tcl` records the time in DCD frames, instead of nanoseconds, to facilitate comparision with the trajectory. The results are placed in the file `curr_6V.dat`.

6 The script `trackPositionZ.tcl` operates in much the same way as `electricCurrentZFrame.tcl` except that it determines the center of mass of the DNA relative to the center of the pore instead of the current. Enter `source trackPositionZ.tcl`. The z-position of the center of mass is stored as a function of frame number in `pos_6V.dat`.

7 Open the trajectory in VMD with the following commands:

```
mol delete all
mol load psf translocate.psf dcd translocate.dcd
```

In the Graphical Representations window, change Selected Atoms to `resname SIN and y > 0`. Change the Drawing Method to Beads. Create a new representation with the selection `segname ADNA BDNA` and the drawing method VDW.

8 Now plot current versus frame (`curr_6V.dat`) and center-of-mass position versus frame (`pos_6V.dat`) and compare it with the events that take place in the trajectory. How does the passage of the DNA change the current?

Challenge: Protein translocation. Perform the translocation simulation with the protein ubiquitin instead of DNA using the files `ubiquitin.psf` and `ubiquitin.pdb`.

3 Appendix

Here we describe in detail the operation of the script `siliconNitridePsf.tcl`*, which is used to generate the structure file for our synthetic subsystems in section 1.3.*

Looking below, you'll see that the script `siliconNitridePsf.tcl` has a number of parameters. Besides having the usual input and output files, we have three flags which determine whether the script should search for angles (`findAngles`), whether bonds should be formed across hexagonal periodic boundaries in the *xy*-plane (`hexPeriodic`), and whether bonds should be formed between the hexagonal faces at the top and bottom (`zPeriodic`). We want to determine the angles and bond across the periodic boundaries; however, we wish to have water above and below the membrane, so we do not bond the top of the membrane to its bottom.

The next important parameter to note is `bondDistance`. It is the threshold distance between atoms below which bonds are created. The remaining parameters define the properties of the silicon and nitrogen atoms—such as their masses and charges. NAMD matches the atom type to values in the parameter files that give the bond and non-bonded force constants between atoms. We take a look at one of these parameter files in section 1.4.

siliconNitridePsf.tcl

```
# Make a psf file for Si3N4.

set fileNamePrefix membrane
# Input:
set pdbFile ${fileNamePrefix}.pdb
set boundaryFile ${fileNamePrefix}.bound
# Output:
set psfFile ${fileNamePrefix}.psf
set surfPdb surf.pdb
# Parameters:
# Should angles be calculated in addition to bonds?
set findAngles 1
set hexPeriodic 1
set zPeriodic 0
# "bondDistance" is used to determine whether a bond exists between atoms.
```

```
set bondDistance 2.0
# Si parameters
set nameSi "SI.*"
set massSi 28.085500
set chargeSi 0.7710
set typePrefixSi SI_
set numBondsSi 4
# N parameters
set nameN "N.*"
set massN 14.00700
# chargeN is determined by neutrality.
set chargeN 0.
set typePrefixN N_
set numBondsN 3
```

The `main` procedure is the driver for the script. Note that it determines the nitrogen charge to enforce charge neutrality in the system. See the box "Charge neutrality" in section 1.3 for more information.

```
proc main {} {
     global pdbFile boundaryFile psfFile surfPdb
     global findAngles hexPeriodic zPeriodic
     global bondDistance
     global nameSi massSi chargeSi typePrefixSi numBondsSi
     global nameN massN chargeN typePrefixN numBondsN

     set selTextSi "name \"${nameSi}\""
     set selTextN "name \"${nameN}\""

     # Load the pdb.
     mol load pdb $pdbFile
     set nAtoms [molinfo top get numatoms]

     # Get the number of nitrogen and silicon atoms.
     set silicon [atomselect top $selTextSi]
     set numSilicon [$silicon num]
     $silicon delete
     set nitrogen [atomselect top $selTextN]
     set numNitrogen [$nitrogen num]
     $nitrogen delete

     # Determine the nitrogen charge.
     set chargeN [expr -$chargeSi*$numSilicon/$numNitrogen]
     puts "Charge on nitrogen atoms: $chargeN"
```

The procedure first calls `bondAtoms` to find bonds between internal atoms and then finds bonds across the periodic boundaries by bonding to periodic images with `bondPeriodic` (Fig. 3). The location of the periodic images are obtained by extracting information from the boundary file with `readRadius` and `readLz`. To save time, we do not attempt to bond all atoms to the periodic images, only those that did not receive a complete set of bonds (defined by `numBondSi` and `numBondsN`) during the first bonding step. To accomplish this, a temporary PDB file, `surf.pdb`, is created in which all the atoms that are incompletely bonded are marked 0.0 in the B column of the PDB. The procedure `bondPeriodic` is used to search for the bonds. If both `hexPeriodic` and `zPeriodic` are not true, then some atoms will never have a complete set of bonds. These are the true surface atoms—those that will be in contact with water molecules in the simulations. Next we put the bond lists in a more convenient form. We then call `findAngles` and finally write the PSF file with `manifestPsf`.

```
# Find the internal bonds.
puts "Bonding internal atoms..."
set bond [bondAtoms all $bondDistance]
puts "Internal bonds: [expr [llength $bond]/4]"

# Bond to periodic images.
if {$hexPeriodic || $zPeriodic} {
     # Create the surface atom pdb.
     set all [atomselect top all]
     $all set beta 1.0
     puts "Searching for surface atoms..."
     set nSurfSi [markSurface $bond $selTextSi $numBondsSi]
     set nSurfN [markSurface $bond $selTextN $numBondsN]
     puts "Number of surface silicons: $nSurfSi"
     puts "Number of surface nitrogens: $nSurfN"
```

```
$all writepdb $surfPdb
$all delete

# Load it up.
mol delete top
mol load pdb $surfPdb

if {$hexPeriodic} {
     puts "The system has hexagonal periodic boundary conditions."
     set radius [readRadius $boundaryFile]
     puts "Hexagon radius: $radius"

     # Determine the centers of the image hexagons.
     set pi [expr 4.0*atan(1.0)]
     set hexCen {}
     set d [expr sqrt(3.)*$radius]
     for {set i 0} {$i < 6} {incr i} {
          set theta [expr $pi/6.*(2*$i-1)]
          lappend hexCen [list [expr $d*cos($theta)] \
          [expr $d*sin($theta)] 0.]
     }
     puts "Periodic image displacements: $hexCen"

     # Find the bonds on the periodic boundaries.
     puts "Bonding to the periodic image..."
     foreach r $hexCen {
          set bond [concat $bond \
          [bondPeriodic all $bondDistance $r]]
     }
}

if {$zPeriodic} {
     puts "The system is periodic along the z-axis."
     set lz [readLz $boundaryFile]
     puts "Period in z: $lz"
     set zCen [list [list 0 0 -${lz}] [list 0 0 $lz]]

     # Find the bonds on the periodic boundaries.
```

```
                puts "Bonding to the periodic image..."
                foreach r $zCen {
                     set bond [concat $bond \
                     [bondPeriodic all $bondDistance $r]]
                }
           }

     }
     mol delete top

     puts "Counting bonds on each atom..."
     countBonds count $bond $nAtoms
     puts "Reorganizing bond lists..."
     set bond [reorganizeBonds $bond]
     puts "Removing redundancy..."
     set bond [removeRedundantBonds $bond]
     set totalBonds [llength $bond]
     puts "Number of bonds: $totalBonds"

     set angle {}
     if {$findAngles} {
          puts "Determining the angles..."
          set angle [findAngles $bond]
          set totalAngles [llength $angle]
          puts "Number of angles: $totalAngles"
     }

     puts "Writing psf file..."
     manifestPsf $psfFile $pdbFile $nAtoms bond angle count
     puts "The file $psfFile was written successfully."
}
```

The procedure `bondAtoms` uses VMD's atom selection interface to find other atoms within `bondDistance` of each atom. Because the procedure searches for neighbors of each atom, the resulting list contains each bond twice, since a bond between atom 1 and atom 2 is the same as a bond between atom 2 and atom 1. Note that the algorithm has a quadratic growth rate in the number of atoms. For the small systems in this tutorial, the method used here should be fast enough. However, by effecting a spatial decomposition of the system, we could reduce the growth rate to linear in the number of atoms.

```
# Find bonds between internal atoms and return them.
proc bondAtoms {selText bondDistance} {
     set sel [atomselect top $selText]
     set pos [$sel get {x y z}]
     set index [$sel get index]
     $sel delete

     set bondDistance2 [expr $bondDistance*$bondDistance]
     set bond {}
     foreach r $pos ind $index {
          # Select neighboring atoms.
          foreach {x y z} $r { break }
          set nearText "($x-x)^2+($y-y)^2+($z-z)^2 < $bondDistance2"
          set near [atomselect top \
          "$selText and $nearText and not index $ind"]
          set nearNum [$near num]
          set nearIndex [$near get index]
          $near delete

          # Add them to the bond list.
          foreach i $nearIndex {lappend bond $ind $i}
     }
     return $bond
}
```

The following two procedures extract information about the system's geometry from the boundary file for use in bonding across periodic boundaries.

```
# Get the radius from the boundary file.
proc readRadius {boundaryFile} {
     set in [open $boundaryFile r]
     foreach line [split [read $in] \n] {
          if {[string match "radius *" $line]} {
               set radius [lindex $line 1]
```

```
                break
            }
        }
        close $in
        return $radius
}

# Get the cellBasisVector3_z from the boundary file.
proc readLz {boundaryFile} {
        set in [open $boundaryFile r]
        foreach line [split [read $in] \n] {
            if {[string match "cellBasisVector3 *" $line]} {
                set lz [lindex $line 3]
                break
            }
        }
        close $in
        return $lz
}
```

The procedure `bondPeriodic` acts much like `bondAtoms` except that the entire system is shifted to its periodic image using VMD's `moveby` command (Fig. 3).

```
# Try to bond surface atoms to the periodic image.
proc bondPeriodic {selText bondDistance periodicDisp} {
        set selText "$selText and beta == 0.0"
        set sel [atomselect top $selText]
        set pos [$sel get {x y z}]
        set index [$sel get index]

        # Shift all of the atoms into this periodic image.
        $sel moveby $periodicDisp

        set bondDistance2 [expr $bondDistance*$bondDistance]
        set bond {}
        foreach r $pos ind $index {
            # Select neighboring atoms.
```

```
          foreach {x y z} $r { break }
          set nearText "($x-x)^2+($y-y)^2+($z-z)^2 < $bondDistance2"
          set near [atomselect top \
          "$selText and $nearText and not index $ind"]
          set nearNum [$near num]
          set nearIndex [$near get index]
          $near delete

          # Add them to the bond list.
          foreach i $nearIndex {lappend bond $ind $i}
     }

     # Return all atoms to their original position.
     $sel set {x y z} $pos
     $sel delete

     return $bond
}
```

The following procedure sets the beta value to 0.0 for all atoms that do not have a full set of bonds. This includes both atoms that will later be bonded to the periodic image and those that are truly on the surface. Marking these atoms allows most of the atoms to be skipped when bonding to the periodic images.

```
# Find the atoms that have fewer than "numBonds" bonds.
# Mark surface atoms by beta = 0.0.
# Warning! The bond list is assumed to be flat and redundant.
proc markSurface {bond selText numBonds} {
     set sel [atomselect top $selText]
     set index [$sel get index]
     set nSurfAtoms 0

     foreach i $index {
          # Find the number of bonds for each atom.
          set n [llength [lsearch -all $bond $i]]
          # Assume each bond is in the list twice.
          set n [expr $n/2]
          # Set the beta value to 0.0 if the atom is on the surface.
          if {$n < $numBonds} {
               set s [atomselect top "index $i"]
               $s set beta 0.0
               incr nSurfAtoms
               $s delete
          }
     }
     $sel delete

     return $nSurfAtoms
}
```

The procedure `countBonds` creates a Tcl array linking each atom to the total number of bonds that it has. We need this number because the string in the PSF type column is determined by the number of bonds. For example, the type N_2 refers to a nitrogen atom with two bonds. Consequently, we can define different bond constants in the parameter files depending on the coordination of the atom. None of the parameter files we will use here discriminate in this way, however.

```
# Count the number of bonds on each atom and return an array (zero-based).
# The result is placed in a variable name countVar.
# Warning! The bond list is assumed to be flat and redundant.
proc countBonds {countVar bond nAtoms} {
     upvar $countVar count

     set num {}
     for {set i 0} {$i < $nAtoms} {incr i} {
          set n [llength [lsearch -all $bond $i]]
          set n [expr $n/2]
          lappend num $i $n
     }

     array set count $num
}
```

The following two procedures reformat the bond lists. The first converts the flat list of bonds into nested lists containing pairs of atom indices. It also adds 1 to all of the indices since the first PSF index is 1, but VMD atom indices are 0-based. The second of the two procedures removes bonds that are permutations of one another, as mentioned earlier.

```
# Put the bonds into sublists.
# Reindex to a 1-based index.
proc reorganizeBonds {bond} {
     set ret {}
     foreach {b0 b1} $bond {
          incr b0
          incr b1
          lappend ret [list $b0 $b1]
     }
     return $ret
}

# We should now have all of the bonds twice.
# Find the unique bonds.
proc removeRedundantBonds {bond} {
     set ret {}
     foreach b $bond {
          set bPerm [list [lindex $b 1] [lindex $b 0]]
          set match [lsearch $ret $bPerm]

          # Add the bond to "ret" only if it is unique.
          if {$match == -1} {lappend ret $b}
     }
     return $ret
}
```

The `findAngles` procedure searches through all unique pairs of bonds and finds triplets of atoms such that atom A is bonded to atom B and atom B is bonded to atom C. Since each atom not on the surface has a fixed number of bonds, the number of bonds is proportional to number of atoms. Thus, the algorithm is in $\Theta(N^2)$ where N is the number of atoms. We could reduce the asymptotic complexity by writing the algorithm more cleverly;

however, for our purposes here this method is fast enough. Because of the quadratic complexity and the fact that it is written entirely in Tcl—it uses no fast built-in VMD commands—this procedure can take a long to time run for large systems.

```
# Find the angles.
proc findAngles {bond} {
     set totalBonds [llength $bond]
     set totalBonds1 [expr $totalBonds - 1]

     # Find bonds that share atoms.
     set angle {}
     for {set i 0} {$i < $totalBonds1} {incr i} {
          for {set j [expr $i+1]} {$j < $totalBonds} {incr j} {
               foreach {a0 a1} [lindex $bond $i] {break}
               foreach {b0 b1} [lindex $bond $j] {break}

               if {$a0 == $b0} {
               lappend angle [list $a1 $a0 $b1]
               } elseif {$a0 == $b1} {
                    lappend angle [list $a1 $a0 $b0]
               } elseif {$a1 == $b0} {
                    lappend angle [list $a0 $a1 $b1]
               } elseif {$a1 == $b1} {
                    lappend angle [list $a0 $a1 $b0]
               }
          }
     }
     return $angle
}
```

The final procedure writes all of the information we have determined thus far to a PSF file. There are three sections of the PSF format important for our Si_3N_4 systems. The first is the atom record section which replicates much of the identifying information contained in the PDB as well the atom's type, mass, and charge, which are essential for simulations. The next section of the PSF contains the bonds. The bonds are stored in eight columns of indices, with each pair of columns in a row representing a single bond between two atoms. Hence, each line of the bond section of the PSF describes four bonds (except the last, which may not be full). The final section of import to us is the angles section which contains nine columns of indices, which as groups of three define three bonds in each row.

```
# Write the psf file.
proc manifestPsf {psfFile pdbFile nAtoms bondVar angleVar countVar} {
     global nameSi massSi chargeSi typePrefixSi numBondsSi
     global nameN massN chargeN typePrefixN numBondsN

     # Import the big pass-by-reference stuff.
     upvar $bondVar bond
     upvar $angleVar angle
     upvar $countVar count

     set dummy "          0"
     set totalBonds [llength $bond]
     set totalAngles [llength $angle]
     set out [open $psfFile w]

     ##### HEADER
     puts $out "PSF"
     puts $out ""
     puts $out "       1 !NTITLE"
     puts $out " REMARKS original generated structure x-plor psf file"

     ##### ATOMS
     puts "Writing atom records..."
     puts $out ""
     puts $out "[format %8i $nAtoms] !NATOM"

     # Open the pdb to extract the atom records.
     set inStream [open $pdbFile r]
     set atom 1
     foreach line [split [read $inStream] \n] {
          set string0 [string range $line 0 3]
          if {![string match $string0 "ATOM"]} {continue}
```

```
# Extract each pdb field.
set record [string range $line 0 5]
set serial [string range $line 6 10]
set name [string range $line 12 15]
set altLoc [string range $line 16 16]
set resName [string range $line 17 19]
set chainId [string range $line 21 21]
set resId [string range $line 22 25]
set iCode [string range $line 26 26]
set x [string range $line 30 37]
set y [string range $line 38 45]
set z [string range $line 46 53]
set occupancy [string range $line 54 59]
set beta [string range $line 60 65]
set segName [string range $line 72 75]
set element [string range $line 76 77]
set charge [string range $line 78 79]

# Determine the type names.
set numBonds $count([expr $atom-1])
set typeSi ${typePrefixSi}${numBonds}
set typeN ${typePrefixN}${numBonds}

# Write the atom record.
puts -nonewline $out [format "%8i " $atom]
puts -nonewline $out [format "%-4s " $segName]
puts -nonewline $out [format "%-4i " $resId]
puts -nonewline $out [format "%-3s " $resName]
puts -nonewline $out [format "%-4s  " $name]
if {[regexp $nameSi $name]} {
     puts -nonewline $out [format "%-4s  " $typeSi]
     puts -nonewline $out [format "% 5.6f       " $chargeSi]
     puts -nonewline $out [format "%6.4f " $massSi]
} else {
     puts -nonewline $out [format "%-4s  " $typeN]
     puts -nonewline $out [format "% 5.6f       " $chargeN]
     puts -nonewline $out [format "%6.4f " $massN]
}
```

```
     puts $out $dummy

     incr atom
}
close $inStream
puts $out ""

##### BONDS
# Write the bonds.
set total [format %8i $totalBonds]
puts $out "$total !NBOND: bonds"
set num 0
foreach b $bond {
     puts -nonewline $out [format "%8i%8i" [lindex $b 0] [lindex $b 1]]

     incr num
          if {$num == 4} {
          puts $out ""
          set num 0
     }
}
puts $out ""

##### ANGLES
# Write the angles.
puts $out "[format %8i $totalAngles] !NTHETA: angles"
set num 0
foreach a $angle {
     puts -nonewline $out \
     [format "%8i%8i%8i" [lindex $a 0] [lindex $a 1] [lindex $a 2]]

     incr num
     if {$num == 3} {
          puts $out ""
          set num 0
     }
}
puts $out ""
```

```
# Write everything else.
##### DIHEDRALS
set nDihedrals 0
puts $out ""
puts $out "[format %8i $nDihedrals] !NPHI: dihedrals"
puts $out ""

##### IMPROPERS
set nImpropers 0
puts $out ""
puts $out "[format %8i $nImpropers] !NIMPHI: impropers"
puts $out ""

##### DONORS
set nDonors 0
puts $out ""
puts $out "[format %8i $nDonors] !NDON: donors"
puts $out ""

##### ACCEPTORS
set nAcceptors 0
puts $out ""
puts $out "[format %8i $nAcceptors] !NACC: acceptors"
puts $out ""

##### NON-BONDED
set nNB 0
puts $out ""
puts $out "[format %8i $nNB] !NNB"
puts $out ""

set tmp [expr int($nAtoms/8)]
set tmp2 [expr $nAtoms -$tmp*8]
for {set i 0} {$i <$tmp} {incr i} {
     puts $out "       0       0       0       0       0       0       0
}
set lastString ""
```

```
     for {set i 0} {$i <$tmp2} {incr i} {
         set lastString "${lastString}       0"
     }
     puts $out $lastString

     ####### GROUPS
     puts $out ""
     puts $out "       1       0 !NGRP"
     puts $out "       0       0       0"
     puts $out ""
     puts $out ""
     close $out
}

main
```