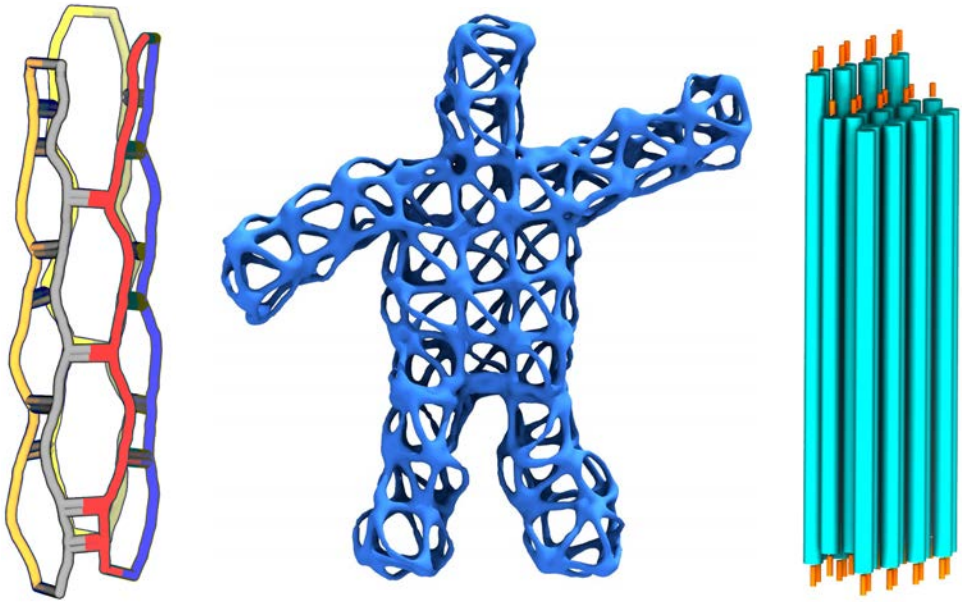
Aksimentiev Group
Department of Physics,
Center for the Physics of Living Cells and
Beckman Institute for Advanced Science and Technology,
University of Illinois at Urbana-Champaign

Images & animations of DNA nanostructures with VMD



Authors:
David Winogradoff
Manish Shankla
Aleksei Aksimentiev

Contents

1 Introduction **2**

2 Representations specific to DNA nanostructures **5**
2.1 Chickenwire 5
2.2 Nice quicksurf representation 6
2.3 Cylinders *a la* cadnano 8

3 Creating a fancy image **9**

4 Animation of a static structure **13**

5 Animation of a trajectory **16**
5.1 Straightforward animation of a trajectory 16
5.2 Advanced topic: movie with representation transitions 18

1 Introduction

In this tutorial, you will be shown how to make an image using Tachyon through VMD's graphical interface. Tachyon is a ray-tracing program written to run very quickly, taking advantage of multiple computer cores. In general, ray-tracing renderers use a realistic lighting model that can properly model shadows, yielding a sense depth to the final image.

This tutorial focuses on features recently added to VMD [1] that affect Tachyon renderings, and does not attempt to be a comprehensive guide to image making with VMD. We assume you have a working knowledge of VMD and refer inexperienced readers to the VMD tutorial. Furthermore, we assume the reader has VMD, Tachyon, and MSMS installed.

If you are unfamiliar with VMD, please take a look at Fig. 1. For the purpose of this tutorial, we recommend that you have all four of the windows shown throughout all your VMD sessions. Fig. 1 provides an explanation of how to initialize the "Representations window" and "Tk Console" (often abbreviated 'TKcon').

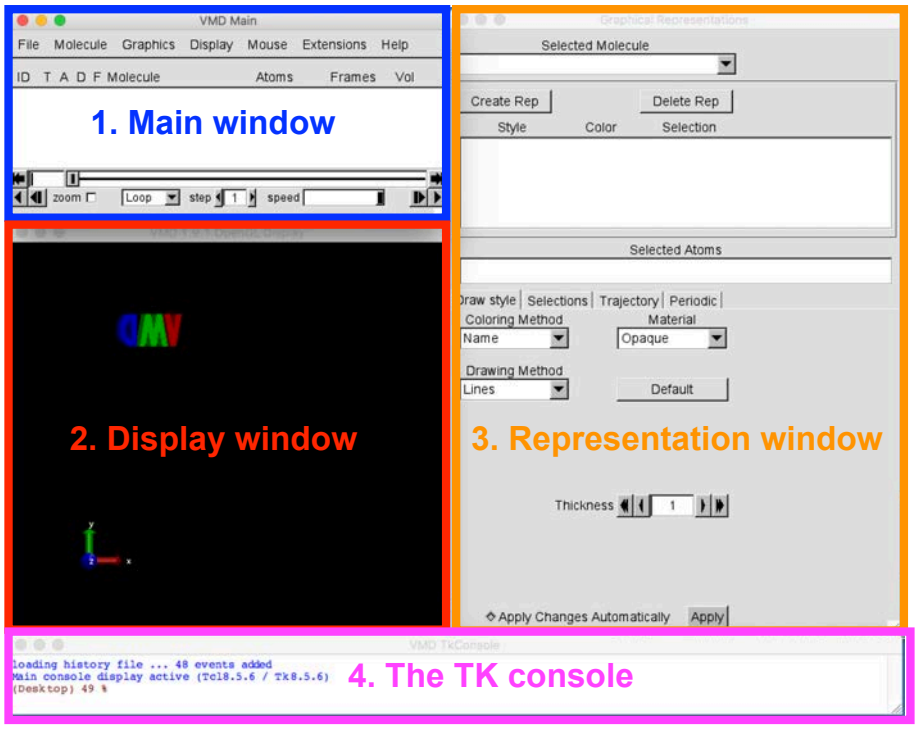


Figure 1: Here we highlight four different windows we suggest you to open in any session of VMD. By default, only windows 1 & 2 will appear. Open window 3 by clicking on the "Graphics" tab of the main window, then on "Representations...". To open window 4, click on the main window's "Extensions" tab, and then on "Tk Console".

Below is an outline of the entire tutorial, with the subfolders and files provided.

```
tutorial-files-provided:
    |
    |-- 1.0-chickenwire/
    |       |-- hextube.psf
    |       |-- equil_k0.pdb
    |       |-- chickenwire.psf, chickenwire.for.make_ndx
    |       |-- pdb2chickenwire.pl
    |-- 1.1-nice-quicksurf/
    |       |-- stickman-1.psf
    |       |-- stickman-1.0.dcd
    |       |-- renderFrame.tcl
    |-- 1.2-cylinders/
    |       |-- FS-v4_02_topBeam_cadnano25.json
    |-- 2.0-pointer-fancy/
    |       |-- pointer_v1_12_from_dietz-3.{psf,pdb}
    |       |-- draw_stage_y.tcl
    |-- 3.0-animation-from-static-structure/
    |       |-- bunny-3.{psf,pdb}
    |       |-- view_change_render.tcl
    |-- 4.0-fret-plate-movie/
    |       |-- plate-capture-1.{psf,pdb,dcd}
    |       |-- pot-600-cut-ds2.dx
    |       |-- fret-plate.mpg
    |-- 4.1-pointer-trajectory-movie/
    |       |-- 1_make_movie.py
    |       |-- Final_clip.mp4
    |       |-- frames/
    |       |   |-- frame_{0..458}.png
    |       |-- images/
    |       |   |-- png frames for transitions
    |       |-- pointer_v1_12_from_dietz-1.{psf,pdb,dcd}
    |       |-- pointer_v1_12_from_dietz.json
    |       |-- step1-cylinders.vmd
    |       |-- step1_make_transparent.tcl
    |       |-- step2_showPlane.tcl
    |       |-- step3_highlight_junction.tcl
    |       |-- step4_convertTGAtoPNG.sh
    |
    |-- follow along with: dna-nanostructure-visualization.pdf
```

2 Representations specific to DNA nanostructures

In Section 2, you will make three static images, each with a different representation. In the first "chickenwire" subsection, you will make a new PDB file based on atomistic topology and coordinate files using a perl script, then render an image of that PDB with VMD. The second "quicksurf" subsection will provide an example of using a single script to automate the entire process of loading and rendering a coarse-grained structure in VMD. In the third "cylinders" subsection, you will start from a `.json` cadnano file, and generate topology and coordinate files, along with a TCL script that will draw a set of cylinders as separate graphics.

2.1 Chickenwire

To start, you will generate an image of a relaxed atomistic structure in what we refer to as a chickenwire representation. To access the required files, you need to cd into the `1.0-chickenwire` subfolder of (`/home/ubuntu/Desktop/TCBGTutorials/dna-nanostructure-visualization`), and run `pdb2chickenwire.pl`. This script requires `chickenwire.for.make_ndx`, which we have already provided. It can also be found in an archive generated from the ENRG MD webserver (*http://bionano.physics.illinois.edu/origami-structure*). From the commandline (**Note: \ indicates line continuation; this should be run as a single line**):

```
>> ./pdb2chickenwire.pl --ndx=chickenwire.for.make_ndx \
equil_k0.pdb > chickenwire.pdb
```

Using the provided file `chickenwire.psf` (which also can be generated from the ENRG MD webserver), you can load `chickenwire.pdb` into VMD, from the command line:

```
>> vmd chickenwire.psf chickenwire.pdb
```

To generate the image in Fig. 2A, we used the following settings (which can be set in the GUI): a white background; Coloring Method of "SegName"; Drawing Method "Licorice" with Bond Radius 2.5, Sphere Resolution 50, Bond Resolution 50; Material Goodsell; Shadows On; Amb. Occl. On; AO Ambient 1.0; AO Direct 0.2.

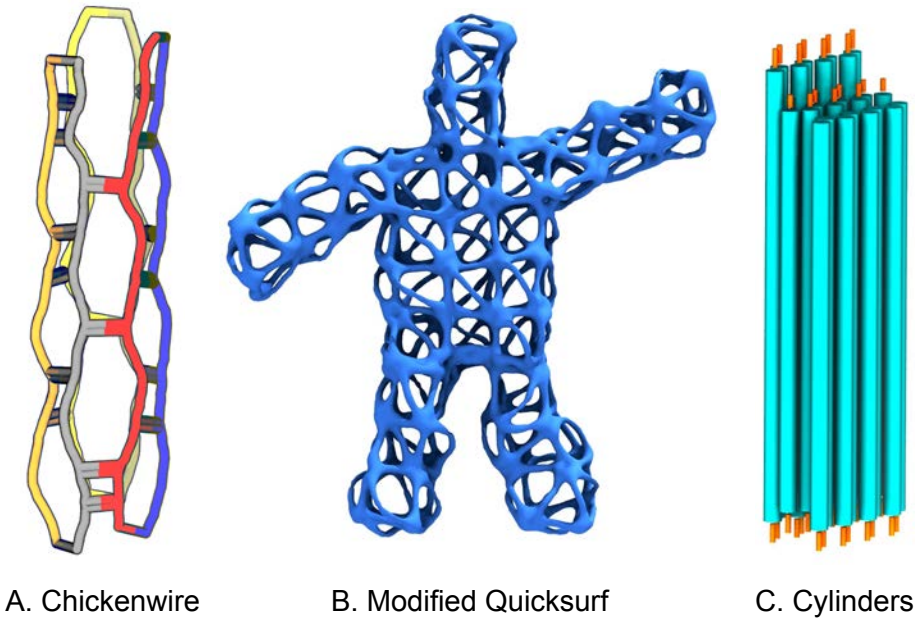


Figure 2: Origami-specific representations from VMD.

2.2 Nice quicksurf representation

In this section, we will provide an example of how to load, modify and render a coarse-grained structure within a single script, `renderFrame.tcl`. After cd'ing into the `1.1-nice-quicksurf` subfolder of (`/home/ubuntu/Desktop/TCBGTutorials/dna-nanostructure-visualization`), open a session of VMD (click on the icon, or type `>> vmd` from the commandline). Once VMD is initialized, open a TK console window (VMD Main → Extensions → TK Console); click on that window and type the following:

```
TK>> source renderFrame.tcl
```

"Sourcing" runs every line in the script. Below, we annotate what different blocks of code in the TCL script do.
Load in the molecule:

```
set PSF stickman-1.psf
set DCD stickman-1.0.dcd
set ID [mol new $PSF]
mol addfile $DCD waitfor all
```

Define display settings

```
color Display Background white
display backgroundgradient off
display depthcue 0
display projection Perspective
display distance -1
display height 1
```

Redefine “red” and “blue”:

```
color change rgb blue 0.27 0.52 0.9
color change rgb red 0.9 0.45 0.33
```

Set selection and material:

```
mol material AOChalky
mol representation QuickSurf 5.700000 1.000000 4.000000 1.000000
mol selection "all"
mol color colorID 0
mol addrep $ID
```

Set ambient occlusion and lighting options:

```
display aodirect 0.600000
light 0 off
display shadows on
display ambientocclusion on
display aoambient 0.900000
display dof off
display dof_fnumber 660.000000
display dof_focaldist 3.700000
render aasamples TachyonInternal 16
render aosamples TachyonInternal 64
```

Resize the display window, rotate and scale the structure:

```
display resize 800 800
display update ui
rotate x by -90
display update ui
scale by 0.4
```

Perform the rendering:

```
render TachyonInternal stickman-tach-inter.ppm
```

This last line (`render ...`) can be modified to choose another renderer (e.g. TachyonLOptiXInternal), or this line can be commented out, and the user could modify the viewing angle or representation further using the GUI.

The result from this script is visually shown in Fig. 2B.

2.3 Cylinders *a la* cadnano

At the time of writing this tutorial, you cannot load a cadnano JSON file directly into VMD. For this third subsection, you will be provided with just a `.json` file, and you will run `mrdna` with an optional tag (`--draw-cylinders`) to generate atomistic topology and coordinate files, along with a TCL script that will separately draw cylinders in VMD (as graphics separate from the molecule). The cylinder representation was inspired by the standard graphical output style of cadnano. You need to cd into the `1.2-cylinders` subfolder of (`/home/ubuntu/Desktop/TCBGTutorials/dna-nanostructure-visualization`). In this example, you will use `FS-v4_02_topBeam_cadnano25.json`.
After cd'ing into the `1.2-cylinder` subfolder, run the following instance of `mrdna` from the commandline:

```
>> mrdna -d . -g 0 --draw-cylinders \
FS-v4_02_topBeam_cadnano25.json
```

The model will be built (i.e. `FS-v4_02_topBeam_cadnano25.psf,pdb,namd,exb`), along with the additional file `FS-v4_02_topBeam_cadnano25.cylinders.tcl`. Inside an open VMD session, run the following in the TKcon:

```
TK>> mol new FS-v4_02_topBeam_cadnano25.psf
TK>> mol addfile FS-v4_02_topBeam_cadnano25.pdb
TK>> source FS-v4_02_topBeam_cadnano25.cylinders.tcl
```

Double-stranded DNA helices will be represented by cyan cylinders, and stretches of single-stranded DNA by narrower orange cylinders. After sourcing the TCL file, the following options were set: Background white; AO Ambient 1.2 (Note: to have such a value above 1.0, from the TKcon you must run (`TK>> display aoambient 1.2`)); AO Direct 0.2; Shadows On; Amb. Occl. On.

The resulting visualization is Fig. 2C.

3 Creating a fancy image

In Section 3, we will only provide you with coarse-grained topology and coordinate files, and we will walk you through using VMD to make a fancy image, featuring multiple selections, representations, and transparencies. First, cd into the `2.0-pointer-fancy` subfolder of (`/home/ubuntu/Desktop/TCBGTutorials/dna-nanostructure-visualization`), and load your molecule of interest. From the commandline:

```
>> cd 2.0-pointer-fancy
>> vmd pointer_v1_12_from_dietz-3.{psf,pdb}
```

This structure is output from `mrdna` run with `pointer_v1_12_from_dietz.json`. One key feature of a PDB file is that it has two fields (i.e. columns) known as "occupancy" and "beta" for every atom (or bead) in the file. `mrdna` has been designed to output PDB files every major step of the multi-resolution process that contain the helix ID as "occupancy" and the cadnano nucleotide ID as "beta", so the beta values will be determined by your `.json` file.

To create a fancy image, we will make four different representations (abbreviated as rep. henceforth). Open a representations window (VMD Main → Graphics → Representations...), and create four reps. writing the following into the "Selected Atoms" field:

1. `z<-275 and not (same occupancy as (abs(x)<5))`
2. `beta 79 to 81`
3. `same occupancy as (abs(x)<5)`
4. `same occupancy as (abs(x)<5)`

Rep. 1 combines a geometric condition with logic about helices. Rep. 2 selects a range of cadnano nucleotide IDs at the center of the pointer structure. Reps. 3 & 4 select a range of helices that span the y- and z-axes.

Drawing Method. The Drawing Method of reps. 1, 2, 4 are Quicksurf, with some variation in the specific presets for (Radius Scale, Density Isovalue, Grid Spacing): 1 (3.5, 0.5, 3.0), 2 (6.0, 0.5, 3.0), 4 (4.0, 0.5, 3.0). And rep. 3 will be drawn as "CPK" with Sphere Scale 10.0, Sphere Resolution 50, Bond Radius 10.0, Bond Resolution 50.

Color. After setting the background to white (VMD Main → Graphics → Colors... → Display → Background → white), we will distinguish each rep. with a different "ColorID"; click on the down arrow next to "Coloring Method" (Graphical Representations → Coloring Method) and select "ColorID"; then, for

each selection, choose the following colors: 1 (1 red), 2 (0 blue), 3 (4 yellow), 4 (8 white). To modify VMD's definitions of "red" and "blue" type the following into the TKConsole:

```
TK>> color change rgb red 0.9 0.45 0.33
TK>> color change rgb blue 0.27 0.52 0.9
```

Material. Reps. 1–3 will be drawn with Material "AOChalky". For rep. 4, we will create a new material that is semi-transparent. Navigate to Materials (VMD Main → Graphics → Materials...). Then, highlight "AOChalky" and click on the button "Create New". This will copy the presets (i.e. Ambient, Diffuse, Specular, ...) of "AOChalky" with the new name "Material23" (if desired, you can change this name of your new material). For Material23, drag the "Opacity" slider to 0.20 (Note: this can also be done from the TkConsole `TK>> material change opacity Material23 0.2`). Also for Material23, turn on "Angle-Modulated Transparency" by clicking of the box marked such.

Display Settings. You have a choice between "Perspective" and "Orthographic" views under the "Display"tab of VMD Main. To make this image, We will select "Perspective" Under the display settings (VMD Main → Display → Display Settings), set the "Screen Hgt" to 1.0 and "Screen Dist" to -1.0. Be aware: this setting will make the image shown in your display window very sensitive to rotations made by moving your mouse. Under Ray Tracing Options, turn shadows and ambient occlusion on. Turn AOambient to 1.2, and AODirect to 0.2. Furthermore, turn on DoF (i.e. depth-of-field), keeping the f/stop and focal distances as default, since they will be changed interactively later when rendering the image. Note: setting AOAmbient to above 1.0 needs to be done from the TKcon:

```
TK>> display aoambient 1.2
```

Adding a stage. Optionally, you can add a stage beneath your image. We have chosen to orient our structure so that, in VMD's display window, the positive y-axis is vertical, positive z is to the right, and positive x is pointing into the screen. This orientation can be achieved by hand in the rotation mode in VMD's display window. If the axes are not showing in your window, turn them on (VMD Main → Display → Axes) and select "Origin" from the pull-down menu. The positive x-, y-, and z-directions will be shown as red, green, and blue arows, respectively. In such a case, we want a stage to be a plane with a specific value in y. Once you have the orientation described above, type the following into the TKcon:

```
TK>> source draw_stage_y.tcl
TK>> draw_stage_y -100
```

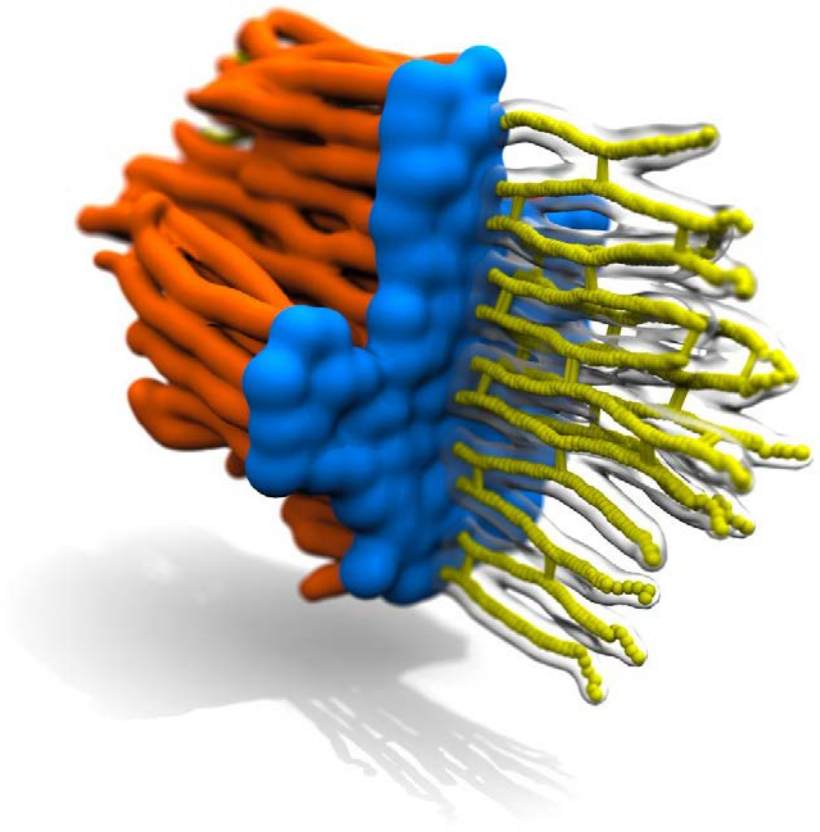


Figure 3: DNA origami pointer structure, drawn in VMD with the interactive Tachyon OptiX renderer. It includes several different selections, drawing methods, materials, and colors. Also note that its shadow is cast upon a user-drawn stage beneath the structure.

You can examine the content of `draw_stage_y.tcl` with a text editor, and copy the file with a new name to make a script that would draw a plane in x or z, if desired. After writting the lines above in the TKcon, it is possible that the plane drawn will slice through the structure itself. If that is the case, you can move the structure vertically in y, also in the TKcon, e.g. :

```
TK>> set all [atomselect top "all"]
TK>> $all moveby "0 10 0"
```

These lines select the entire structure, and then move it 10 Å in the positive y direction. The second line can be repeated several times, to obtain the desired height above the stage drawn. Note: `draw_stage_y.tcl` draws a white stage, in AOChalky material, both of which can be changed easily.

Rendering. Before rendering, we recommend that you save your current visualization state (VMD Main → File → Save Visualization State); you will then be prompted to provide a filename, including the path desired; we recommend the path to be the same location as the `psf,pdb` and the name to be something along the lines of "pointer-visual-state.vmd"; you can include the date in the filename as well, if desired.

In this example we will explain how to use the "TachyonL-OptiX (interactive, GPU-accelerated)" renderer, for which you will need to have a GPU. There are several advantages to this renderer: (1) it is interactive, i.e. you can rotate, zoom, and translate after starting to render and before the image file is generated; and (2) OptiX rendering is *fast*, much faster than the standard Tachyon renderer.

Go to the "File Render Controls" (VMD Main → File → Render...), select the above-stated renderer, provide a unique filename (e.g. `pointer-stage-optic.ppm`), and press the "Start Rendering" button. Because depth-of-field is on, the initial image will appear very blurry. Use zooming and translating (while having VMD's display window selected, type "t" on the keyboard, hold right-click down, and drag your mouse to the right or left) to focus on the image appropriately. Rotations can be made as well (go back to rotation mode by pressing "r" on the keyboard).

After every change you make, the interactive window will display exactly what your rendered image will look like. In this mode, you will see the shadow the pointer casts on the stage drawn beneath it, for example. Note: after a change, it may take a few seconds for the interactive window display to settle on how the image will look. When you are happy with exactly how the image looks, close the interative window, and the image file (`pointer-stage-optic.ppm`) will be generated.

Lastly, you can edit the `.ppm` file in another software, if desired. This allows you to change the saturation and exposure of the image. The final result of all the steps detailed above is Fig. 3.

4 Animation of a static structure

In Section 4, we will provide an example of how to generate an animation based on a single structure, using the "View-Change-Render" GUI of VMD. You need to cd into the `3.0-animation-from-static-structure` subfolder of (`/home/ubuntu/Desktop/TCBGTutorials/dna-nanostructure-visualization`), First, load your desired structure into VMD, e.g. :

```
>> vmd bunny-3.psf bunny-3.pdb
```

Then, choose how to represent your structure. Note, with the method presented here, you cannot alternate between different representations throughout the animation. For this example, we have chosen Drawing Material to be "Quicksurf" (Radius Scale 5.5, Density Isovalue 1.0, Grid Spacing 4.0), Material to be "AOChalky", and Coloring Method "ColorID" (after redefining red: `color change rgb red 0.9 0.45 0.33`).

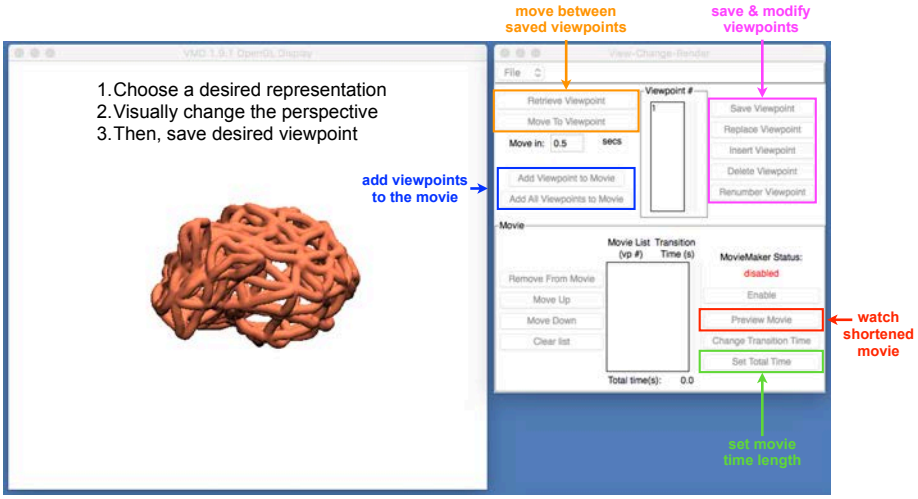


Figure 4: Computer screenshot highlighting VMD's windows for display and "View-Change-Render". The annotated colored boxes explain different options. Shown is the default initial view, achieved by pressing "=" on your keyboard when VMD's display window is highlighted.

Then open a "View-Change-Render" window (from VMD Main: Extensions → Visualization → ViewChangeRender).

In Fig. 4 we highlight the different options for the "View-Change-Render" window. Keeping the same representation, modify the perspective (as seen

in the display window), then click on the "Save Viewpoint" button; they will automatically be numbered in order of assignment (1,2,3,...). After several viewpoints are saved, you can move to one from your current visualized state with "Move To Viewpoint".

Figure 5: Steps required to make the example animation. 1–7 display viewpoints to save. Step 8 is adding those viewpoints to the movie. Step 9 is making the movie itself.

In Fig. 5, we provide screenshots of viewpoints 1–7 saved in our example, which can then be added to the movie (blue box in Fig. 5).

- Viewpoint 1 is the default view (which we can go to by pressing "=" on the keyboard during a VMD session).
- Viewpoints 2 & 3 were generated by doing 90 degree rotations (from the Tk console `TK>> rotate x by -90`, & `TK>> rotate y by 90`).
- Viewpoint 4 was achieved by zooming in.
- Viewpoint 5 by translating into the structure (Ctrl-t (to enter translate mode), right-click with the mouse and drag to the right; alternatively, from the Tk console: `TK>> translate by 0 0 3`).
- Viewpoint 6: select "Viewpoint #" 4, then press on "Move to Viewpoint", then click "Save Viewpoint".
- Viewpoint 7 is achieved in a similar manner to Viewpoint 6, saving after moving to 3.

After saving all the desired viewpoints, in the desired order, click on "Add All Viewpoints to Movie". Note: the "MovieMaker Status" is disabled by default; after adding all your viewpoints, press on the "Enable" button under that status "Preview Movie" will let you watch a short version of the animation you will generate. You can elongate the movie length by pressing "Set Total Time" (for our demonstration, we set the total length to be 30 seconds). Note, the viewpoints can be saved for use in the future using the "File" tab of "View-Change-Render" (File → Save). After previewing the movie, and setting the desired total time, you will use VMD's movie maker (from VMD Main: Extensions → Visualization → Movie Maker) to render frames and encode an animation.

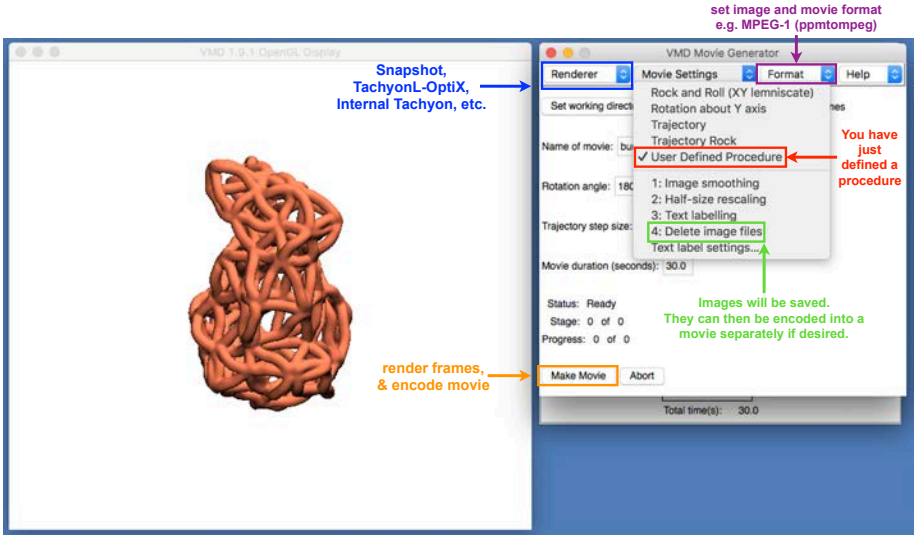


Figure 6: Using VMD's movie maker. Colored boxes annotate different options and functionalities.

First, make a new folder, where you want the individual frames and the entire movie to be made, and enter that location into "Set working directory". Then, choose a renderer. "Snapshot" can be chosen first, to make sure VMD's Movie Generator is generating the frames you expect it to.

Then we recommend switching to one of the tachyon options (e.g. "TachyonL-OptiX", "Internal Tachyon") for a production-quality animation. Under "Movie Settings", click on "User Defined Procedure" to use the procedure you just defined using "View-Change-Render". Optionally, unclick "Delete image files" (which is on by default) if you want to save all the images and have the option of encoding the movie yourself later, as a separate step.

The Movie Maker's "Format" tab allows you to control the file formats for the images rendered and movie encoded (`.ppm` images and an `.mpg` movie by default). Note: one way to check that the movie will follow your procedure is to look at the "Movie duration (seconds): " field. That amount of time (30.0 in Fig. 6) should match what you set in the "View-Change-Render" window.

After setting all options as desired, click on the "Make Movie" button, and the rest is automated. By setting the Name of the movie to "bunny-animation", you will generate MPEG `bunny-animation.mpg` and images `bunny-animation.XXXXX.ppm`, where "XXXXX" is replaced by numbers.

If you save the image files, you could perform your own encoding, e.g. from commandline :

```
>> ffmpeg -i bunny-animation.%05d.tga -vcodec libx264 \
-crf 25  -pix_fmt yuv420p bunny-animation.mp4
```

That's it. This process should work in general, for any set of defined viewpoints.

5 Animation of a trajectory

In the final section, we will cover how to make an animation in VMD based on a trajectory, a file containing many simulation snapshots of a single molecule. The first subsection will provide a straightforward approach in which you will load coarse-grained topology and trajectory files (as well as a DX file that representations a nanopipette), and then use VMD's built-in "Movie Maker" plugin to make an MPEG movie file. In the final subsection of this tutorial, we will walk you through making a movie with smooth transitions between different representations, using VMD and TCL scripts to make the transitions and a python script to encode the final animation.

5.1 Straightforward animation of a trajectory

This section will step you through loading a trajectory into VMD and using the built-in movie maker plugin. You need to cd into the `4.0-fret-plate-movie` subfolder of
(`/home/ubuntu/Desktop/TCBGTutorials/dna-nanostructure-visualization`),
After initializing VMD, load a DX file (a text-based 3D potential file `pot-600-cut-ds2.dx` that represents a nanopipette) by typing the following into the TKCon:

```
TK>> mol new pot-600-cut-ds2.dx
```

Then you can make a nice representation of this potential. In the Graphical Representations window, select the DX molecule, for which the "Selection" field will be disabled and displayed only as `<volume>`, set "Drawing Method" to Isosurface, "Draw" to Solid Surface, "Show" to Isosurface, color the surface silver ("Coloring Method" to ColorID 6), "Material" AOChalky, and set the "Isovalue" field to 3.

At this point, you can also add all the display settings presets: a white background (VMD Main → Graphics → Colors... → Display → Background → white); hide the axes (VMD Main → Display → Axes → Off); under VMD Main, Display → Orthographic; under display settings (VMD Main → Display → Display Settings...) turn "Shadows" and "Amb. Occl." on, "AO Ambient" to 1.3 (from the TKcon `TK>> display aoambient 1.3`), "AO Direct to 0.0".

Once the nanopipette potential has a nice representation, load the fret plate's topology file (`plate-capture-1.psf`) and trajectory file (`plate-capture-1.dcd`); from the TKcon:

```
TK>> mol new plate-capture-1.psf
TK>> mol addfile plate-capture-1.dcd
```

After loading the trajectory, orient the perspective so you can see the downward motion of the fret plate (`TK>> rotate x by 90`; `TKcon>> rotate y by -90`; then enter translation mode by pressing "t" on the keyboard when VMD's display window is active). The inner channel of the nanopipette should be visible from this perspective.

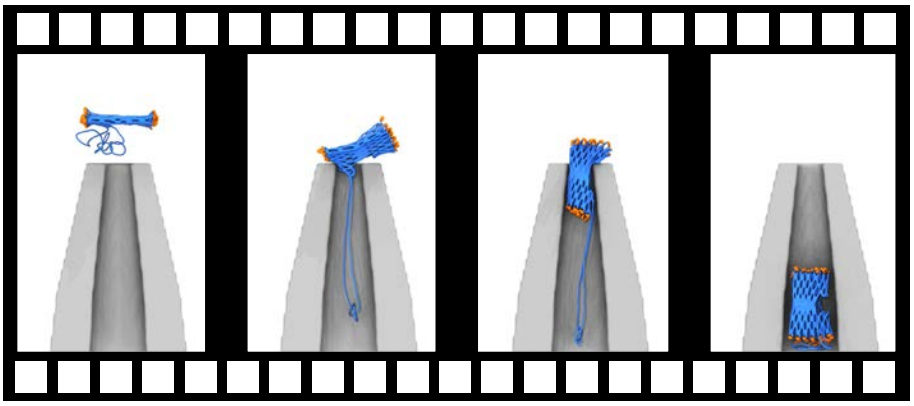


Figure 7: Individual frames of the fret plate movie you will generate using VMD's movie maker plugin.

Then make two different representations of the fret plate (in the Graphical Representations window) with selections (1) "name DNA" and (2) "name NAS". They will select the double-stranded and single-stranded DNA separately. Display both with Material AOChalky and Drawing Method Quicksurf with "Density Isovalue" 0.5 and "Grid Spacing" 1.0. Set the "Radius Scale" of (1) to 5.7, and that of (2) to 7.5, and use the "ColorID" Coloring Method setting (1) to blue and (2) to orange. As done previously throughout this tutorial, redefine VMD's "blue" (from the TKcon: `TK>> color change rgb blue 0.27 0.52 0.9`).

After an individual frame has the desired appearance, you will "smooth" the trajectory. As you may not have noticed, the Graphical Representations window actually has four tabs: "Draw Style", "Selections", "Trajectory", and "Periodic". Only one of these tabs will be shown at any given time, and the default one is "Draw Style". Switch over to the "Trajectory" tab, and click on the small right arrow next to the "0" under "Trajectory Smoothing Window Size", increasing the value to "3". You can rewatch the trajectory now, if desired. Note: smoothing needs to be done for the two representations separately (set both to the same value).

You will now make a movie based on this trajectory using VMD's "Movie Maker" plugin (VMD Main → Extensions → Visualization → Movie Maker). This will open a separate window. In another terminal, in your current working directory, make a new subfolder called "movie" (from the commandline `>> mkdir movie`). Click on the "Set working directory:" button and type in the location of this newly created "movie" subfolder. Change the name of the movie to "fret-plate", or another unique name if desired. For the sake of time, when rendering your first movie, keep the "Renderer" as the default ("Snapshot (Screen Capture)"), which could be changed to make a production-level movie in the future (e.g. to "Tachyon Internal" or "TachyonL-OptiX").

Then click on "Movie Settings", and select "Trajectory". Finally, press the "Make Movie" button, which automates the process of rendering frames and encoding the movie for you. You can track the progress by looking at the three fields at the bottom: Status, Stage, and Progress. "Status" will provide the name of the current "Stage" (there are 8 stages in total), and "Progress" will display the number of frames rendered. Keeping the "Format" tab as default, the entire process will result in making an MPG (`fret-plate.mpg`). We show individual frames from such a movie in Fig. 7.

5.2 Advanced topic: movie with representation transitions

In this section, you will make a sophisticated movie of the DNA origami pointer structure. You need to cd into the `4.1-pointer-trajectory-movie` subfolder of

(`/home/ubuntu/Desktop/TCBGTutorials/dna-nanostructure-visualization`), The final animation you generate will transition between several different representations of the starting, idealized structure. And then, after highlighting a specific structural feature, the trajectory will be played.

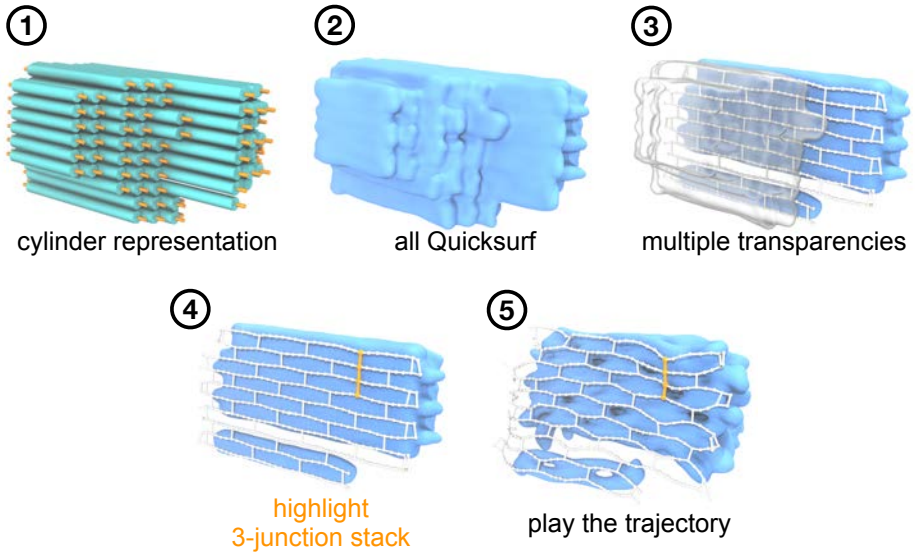


Figure 8: Process of making a more complex movie. Shown are the representations you will transition between to make a more complex movie.

You are provided with the DNA origami's JSON file (`pointer_v1_12_from_dietz.json`), PSF (`pointer_v1_12_from_dietz-1.psf`) that provides the topological information, PDB (`pointer_v1_12_from_dietz-1.pdb`) that provides the initial idealized coarse-grained structural coordinates, and DCD (`pointer_v1_12_from_dietz-1.dcd`), which is a trajectory file containing 1001 frames. Fig. 8 illustrates the stages you will transition between.

Note: we have already provided `.png` files in the "frames" subfolder that were generated from the procedure described below. To encode the file movie without rendering the frames through VMD, skip ahead to the **"Encoding with python"** section below.

To render the frames yourself in VMD, first, from the commandline, initialize a VMD session while sourcing a saved visualization state (`step1-cylinders.vmd`):

```
>> vmd -e step1-cylinders.vmd
```

This will setup your display settings (white background, ambient occlusion, perspective, etc.), load the `psf`, `pdb`, and `dcd` files for the coarse-grained pointer,

and draw cylinders representing the pointer structure (which are graphics separate from the molecule itself).

After this is done, type the following into the TKCon:

```
TK>> source step1_make_transparent.tcl
```

This script transitions from cylinders to a Quicksurf representation (Fig. 8, stages 1 to 2), saving individual frames of that transition to the subfolder "frames". Essentially, you are creating two custom materials, and, in a for loop (text below), you are changing the opacity of the cylinders from fully opaque to completely transparent, and doing the reverse for the Quicksurf representation.

```
for {set i 0 } {$i <= $num_frames_step1} {incr i } {
    set m [expr  (1.0/$num_frames_step1) * $i]
    material change opacity Material29  $m #quicksurf appear
    material change opacity Material30  [expr 1.0 - $m]
    render TachyonLOptiXInternal frames/frame_${i}.tga
}
```

A similar approach will be taken to transition from stages 2–4 (i.e. $2 \rightarrow 3 \rightarrow 4$). Run each of the following lines in the TKCon, one after the other:

```
TK>> source step2_showPlane.tcl
TK>> source step3_highlight_junction.tcl
```

`step2_showPlane.tcl` will do the transition (and save frames) from stages $2 \rightarrow 3$; highlighting a specific plane of the pointer (i.e. selecting the atoms `occupancy 40 to 49`). `step3_highlight_junction.tcl` renders frames for the transition from stages $3 \rightarrow 4$, which higlights a specific structural motif of 3 vertically-stacked junctions (in orange).

Finally, we convert the TARGA frames to PNG files to reduce size of the output movie in the code block below. Note: the PNG frames are already provided in the "frames" directory so this step can be skipped.

```
>> ./step4_convertTGAtoPNG.sh
```

Encoding the movie with python.

With the set of frames generated in the previous section (or with the `.png` files provided in "frames" if you have skipped ahead), you will generate the final

movie that includes various overlaid images. This can be achieved by running the `1_make_movie.py` python script from the commandline (the moviepy library must be installed):

```
>> python 1_make_movie.py
```

Moviepy (*https://github.com/Zulko/moviepy*) is a versatile python library for editing movies. As a short example of the overall work-flow using the moviepy library, we take sets of frames a transition,

```
import numpy as np
from moviepy.editor import *
startFrame, endFrame = 0, 50
#make list of frames
frames = list(map(lambda x: 'frames/frame_{}.png'.\
        format(int(x)),\
        np.arange(startFrame,endFrame)))
        #create first clip
        clip1 = ImageSequenceClip(frames,fps=25)\
        .resize(0.55).speedx(0.5)
```

and overlay images and textual elements found in the images folder,

```
#get an image to overlay
num1 = ImageClip(f'images/one.png',duration=4)\
        .resize(0.1)\
        .set_pos((15,15))\
        .set_start(0)
```

to encode a video clip,

```
#put together image and clip
#set start times and sizes
Final_clip = CompositeVideoClip([
        clip1.set_start(0),
        num1.set_start(4)],
        size=(cX,cY))
#encode clip using FFMPEG
Final_clip.write_videofile(f'Final_clip.mp4',\
        codec='mpeg4',threads=14,\
        preset='veryslow',\
        ffmpeg_params=['-tune','film','-crf', '0','-b:v','6000k'])
```

For the full work-flow, refer to the `1_make_movie.py` script provided, and a complete guide to the syntax and available methods can be found in the moviepy documentation (*http://zulko.github.io/moviepy/*).

References

[1] W. Humphrey, A. Dalke, and K. Schulten. VMD: Visual molecular dynamics. *J. Mol. Graphics*, 14(1):33–38, 1996.