Visualization of electronic density

Bastien Grosso\textsuperscript{a,b}, Valentino R. Cooper\textsuperscript{c}, Polina Pine\textsuperscript{d}, Adham Hashibon\textsuperscript{e}, Yuval Yaish\textsuperscript{a}, Joan Adler\textsuperscript{a,}\textsuperscript{*}

\textsuperscript{a} Technion-IIT, Haifa, Israel
\textsuperscript{b} EPFL, Lausanne, Switzerland
\textsuperscript{c} Oakridge National Laboratory, TN, USA
\textsuperscript{d} Loyola University, IL, USA
\textsuperscript{e} Fraunhofer IWM, Freiburg, Germany

\textbf{A B S T R A C T}

The spatial volume occupied by an atom depends on its electronic density. Although this density can only be evaluated exactly for hydrogen-like atoms, there are many excellent algorithms and packages to calculate it numerically for other materials. Three-dimensional visualization of charge density is challenging, especially when several molecular/atomic levels are intertwined in space. In this paper, we explore several approaches to this, including the extension of an anaglyphic stereo visualization application based on the AViz package for hydrogen atoms and simple molecules to larger structures such as nanotubes. We will describe motivations and potential applications of these tools for answering interesting physical questions about nanotube properties.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction and educational applications

The Computational Physics group at the Technion developed a desktop visualization code for their needs in Atomistic Visualization, called AViz, \textsuperscript{[1–3]}\textsuperscript{*}. It is based on Mesa/OpenGl and Qt. Initially we modeled atoms as balls, spins as cones or vectors and quadrupolar molecules or liquid crystals or pores as cylinders. In a project motivated by educational use we invoked an “off-label” AViz implementation to illustrate the electronic probability density as calculated from the H atom analytic solution in a smoke rendering form \textsuperscript{[4]}, using dots to enable semi-transparency. The dot representation of AViz, originally created to enable quick selection of viewing angle etc. for atomistic samples, creates a translucent effect whereby the sample’s interiors are visible. Combined with color and rotation it gives excellent insight into the nature of the different electronic states \textsuperscript{[5]}. In order to draw the electronic density we (obviously) first need to calculate it. In brief, for the H atom one calculates the electronic density on a grid, and defines a box around each grid point. Dots are then drawn at randomly chosen points within each box at an average density equal to the local electronic density at the center of the box. Each of these points is given x, y and z coordinates and is drawn using the dot feature of AViz. The \texttt{.xyz} format is common to many molecular visualization packages, but its normally used to indicate atoms, not density points. For the hydrogen 2s case the datafiles contain some 50,000 points, rather larger than those typically used in atomic visualization, although since they are not solid spheres, the rendering time is reasonable. In the left frames of Fig. 1 we show the AViz visualization of the electronic density of the 2s state of the H atom in both grayscale and color. Three dimensional visualizations of hydrogen atom wavefunctions are very helpful for teaching Modern Physics or Quantum Mechanics classes. The concept of electronic density is hard to grasp. Animated gifs of these samples in rotation are found at \textsuperscript{[5]} using binned color and have been found to be helpful to students \textsuperscript{[6]}.

Visualization of electronic density for atoms or small molecules is a bit different from volume visualization of, for example, fluid flow around an airplane wing because in the latter case there are one (or more) solid wings with fluid around, whereas in the atomic case we have one or many pinpoint nuclei and different electron orbitals for each. It is also different than water density in a cloud as electronic densities are more structured. Even the electronic density around a metal has a surface comprised of the inner conduction bands as a platform and a few additional free
explored specific orbitals [10, 12, 13]. Sample. We have also been able to show methane molecules both with and without orbital and stripped of the lower densities, and have also explored specific orbitals [10, 12, 13].

Our next attempt at electronic density visualization was to periodically bounded samples, employing plane wave DFT calculations as implemented in the Vienna Ab Initio Simulation package VASP [14, 15]. In preparation for the larger samples, we returned to some of the simple molecules with the VASP code. At that time we used slice visualization with VESTA [16], since dot visualization for many atom samples was limited by issues related to the large datafile size. Despite the VESTA solid visualization, rather than the 3D dot-smoke type, we confirmed that the main features agree. Note that VESTA images also include green in the color range, the early AViz ones only used a red–blue scale for better depth perception.

3. Stereo, binned color smoke rendering

The next stage in our visualization development was to move to 3D stereo. We selected an approach that has a long history, even predating GL. This old concept of anaglyphic stereo relies on two images, slightly displaced, and viewed on a regular screen/projector or poster [17] through colored glasses, or two squares of cellophane. Stereo Vision (SV) works by showing a different image to each eye, thus creating the illusion of a 3D image. AViz 6.1 [2, 18–20] has incorporated the possibility of SV, and although more than two colors are possible there remains some color washout, depending on color selection. The SV images generated by AViz, such as those in this paper, are best viewed using red–cyan anaglyphic glasses. The images in Fig. 2 show nanotube atoms at two different viewing angles in stereo.

In Fig. 2 we show (on the left) the electron density of an hydrogen atom in stereo. Improved colors for stereo for the H atom, as well as clearer instructions were given by Meital Kreif in [21, 22]. Two examples are given in Fig. 3 (center) and (right), the former for the \( n = 3, l = 1, m = 1 \) orbital and the latter for the \( n = 4, l = 3, m = 2 \) orbital of the H atom. On the website [21] all images can be rotated to further aid in depth perception.

4. Motivation and preparation for studying the electronic density of nanotubes

A nanotube vibrates at a frequency that is a function of its width, length, tension, boundary conditions and for certain boundary conditions also of its type. A molecule placed on such a vibrating tube will change this frequency, enabling elucidation of the mass of the adsorbed molecule. The description of these systems with analytic models is limited in cases when both ends are not completely clamped, as occurs in the laboratory. The essential parameter for model analysis is the width of the nanotube wall, and it is the electronic cloud around the atomic nuclei that determines this.
In a series of papers and a thesis, Pine and coworkers [23–29] reviewed the literature and carried out extensive classical molecular dynamics simulations at the atomistic scale to carefully determine the limits of applicability of the analytic theory. While values for the wall width were deduced indirectly by us and many others, direct estimation is of course more desirable. It would be even more desirable to automate this. We are interested in the effect of nanotube local distortions (bending, stretching) on the width, and also in the effect of total strain.

The issue of why the nanotube width is important was discussed in [24], but since it is our major physical motivation, we will recapitulate here. For a nanotube of radius $r$, length $L$, and volume $V$ the frequency of the $n$th mode, $f_n$, of its vibrations depend on the nanotube’s width, $w$ and radius $r$ via two quantities used in its calculation: the moment of inertia, $I = \pi r w (4r^2 + w^2)$ and mass linear density $\rho_L = 2\pi r w \rho_v$ ($\rho_v$ is the density per unit length and $\rho_v$ the density per unit volume) which are related to the tubes’ Young’s modulus, $E$. The exact relation depends on the boundary conditions and model; and the main point of [24–28] was to deduce these frequencies without explicit use of any of the competing analytic models. Literature estimates of $E$ range between 1–6 Tpa, and this is known as Yakobson’s paradox [30]. Huang et al. [31] claimed this range is a result of the scattering of width estimates, and proceeds to study this with sheets of graphene and different potentials as well as reporting on some ab initio results. They also mention the importance of boundary conditions and we have discussed factors such as chirality and strain. The width range they quote is between 0.69 and 0.0617 nm. In two of our papers [24,28] we found via two distinct selfconsistent procedures with simulations of nanotubes of multiple lengths and radii that 0.067 nm is consistent with all our simulations and with earlier studies [30,31]. However we wish to be able to estimate changes in this value directly when we place adsorbed molecules to be weighed, hence the present investigation.

Of course any study to estimate the width directly has to be quantum mechanical. Given the limitations of computer resources, and the need for relatively long tubes, simulated for long times, with a range of parameters it is natural that such width estimates should be deduced with a multiscale approach. Simulations in tandem at an atomistic scale and at an electronic scale are desirable.

The question of multiscale simulations in an efficient manner that minimizes the difficulty of using diverse codes for different scales, inputs and outputs is currently an important research issue. For example, in the European Union FP7 program several collaborations under the banner of “NMP (Nanosciences, Nanotechnologies, Materials and new Production technologies) multiscale modelling” are researching this issue. In particular the project “Simulation framework for multiscale phenomena in nano and microscaled systems, or SimPhoNy” [32] aims towards a uniform environment for scales from electronic to macroscopic, with visualization at all scales. The present study, in addition to its intrinsic interest provides a prototype test bed for SimPhoNy. Together with the atomistic scale and analytic continuum models developed previously, the present study explores the electronic scale and its visualization development and transfer from atomistic scale. There is the caveat that the wrapper codes described below are C++-based and will have to be moved to python for the SimPhoNy environment.

4.1. Molecular dynamics as the starting point

The nanotube simulations at the atomistic scale that form the basis for the present study are described in [24–28] with codes given in [29]. In brief the tube is equilibrated with periodic boundary conditions, then “cut open” and clamped with the appropriate boundary conditions. When strain is needed the tube is carefully stretched before clamping. It is then allowed to vibrate for a long time and the vibration spectrum analyzed with MATLAB codes given in [29].

We used the Brenner [33] potential, and for the present study we selected four nanotubes with different strains from [28] and used rings near the center of the tubes so as to minimize boundary effects. We show one of the nanotubes with 10% strain from [28] in
vectors. One has to translate the coordinates using the C++ code to all atoms to be within its simulation box, defined by the cell input format needed by Quantum Espresso. The latter requires processing routine pp.x to extract the chargedensity. We have with the pw.x subroutine and then executing the post-full DFT structural relaxation (plane-waves self-consistent field) for a onering example, see Fig. 4. The Quantum Espressoparthastwostages: firstperforming the DFT code in four quadrants which have to be recombined prior to visualization.

We also recommend that one dilutes the points prior to visualization and this is currently done in this implementation as part of nano4parts.cpp. (This step was not needed in the H atom and simple molecule cases because the approach used to generate points led naturally to a far more dilute concentration.)

We first describe the calculation for a single ring without strain. We then describe the additional stages needed for systems of several rings and for systems under strain. The details of the simulations are given in Appendix A, and details of the transformations on the output to create AViz input files in Appendix B and recombination of the 4 parts in Appendix C.

6. Visualization of the charge density in 3D

Once all the steps described above are carried out, we reach the most interesting aspect of this procedure—the visualization of the charge density in 3D. We make an initial coarse graining of the density in order to use the binned color approach using dots. The actual color selection is done interactively within AViz, and can be adapted to the user’s preference, optimal selection if there is to be grayscale printing or the user’s requirements in the case of a colorblind person. It would also be different if anaglyphic stereo is to be implemented, because due to the anaglyphic “washout” some color palettes are better than others. At this point we have not varied the number of dots in accordance with their local density as was done for the analytic solutions. The color binning is done at the same time as the extraction from the QE output format and all details are in Appendix B. It helps to determine which region of space has a higher probability of electron localization. In Fig. 8 we see a straight-on view, with colors respectively indicating successively lower densities. The lowest density is not colored for viewing ease, and with colors black, red, green, yellow and blue we show successively higher densities. These very bright colors were selected for their distinctiveness; better palettes for 3D viewing are displayed below. This image is not diluted, the diluted version (details in Appendices) is given in Fig. 9.

We observe that there are bad moire effects here. An improved palette for non-stereo viewing and better angular selection is given in two examples of the extended systems, a 3 ring system in Fig. 10 and for a 6 ring one in Fig. 11.

4. Earlier nanotube visualizations

For the nanotube simulations we carried out still and animated visualization with AViz, creating .xyz files in the simulations and drawing them post-processing. In addition to valuable help in the debugging phase, we could show, for example, regions where bonds become stretched (or compressed), with an example in Fig. 5. This figure as well as the previous one are shown to assist the reader in grasping the local tube geometry. We note that code files that enable direct processing of LAMMPS output into AViz input have also been prepared at the Technion [34]. However for the present project potential problems of explosion of Brenner potential samples in LAMMPS meant that we continued to work with our own older Molecular Dynamics code. The group has also prepared wrappers for outputting Monte Carlo simulations into AViz with both simulations and wrappers in python [35].

4.3. Earlier visualization from electronic density simulations

In addition to our VASP–VESTA studies mentioned above in which we did not succeed to create smoke rendering input, we also carried out exploratory simulations [36] with the Quantum Espresso (QE) [37,38] DFT code and FORTRAN wrappers for transfer to AViz, which led to the development of the present more substantial study.

5. General aspects of our calculations and visualizations

For this calculation we selected the Quantum Espresso plane-wave DFT code [37,38]. The selection was based partly on a preference for public domain codes with clear documentation of their format for the charge density and partly on the possibility of more local support.

We illustrate the complete protocol in Fig. 6. From the initial molecular dynamics simulation we select one or several rings and place their coordinates in an .xyz file. We confirm their validity with an atomistic AViz visualization, for a one ring example, see Fig. 7. The Quantum Espresso part has two stages: first performing a full DFT structural relaxation (plane wave self-consistent field) with the pw.x [39] subroutine and then executing the post-processing routine pp.x [40] to extract the charge density. We have written C++ wrappers to transform the .xyz coordinates into the input format needed by Quantum Espresso. The latter requires all atoms to be within its simulation box, defined by the cell vectors. One has to translate the coordinates using the C++ code many_rings.cpp in order to have the first atom at the origin and paste the new coordinates in the appropriate section of the input file for pw.x. Another C++ wrapper is then used to convert the output of pp.x into the .xyz format for AViz electronic density visualization. AViz requires an input file in the .xyz format with two initial rows, the first being the integer number of atoms or dots and the second a comment line. All other rows have a letter to indicate atomic type or dot color and at least three real number spatial coordinates.

All files that we use can be found in a single tar file on the website [7] in ddl_charge_density.tar. In addition the input and additional wrapper routines, currently written in C++ perform the tasks listed below. As well as being part of the total download file they are also provided as separate tar files in [7] with the name of the file in brackets in this list:

- charge_density_xyz.cpp [7] extracts the charge from pp.x in the correct input format for AViz (in ddl_example_input_QE.tar)
- many_rings.cpp provides the input coordinates for pw.x (in ddl_many_rings.tar)
- nano4parts.cpp recombines the 4 quarters into a single ring (in ddl_charge_density.tar) (In the current setup QE initially calculates the density in four quadrants which have to be recombined prior to visualization).

We also recommend that one dilutes the points prior to visualization and this is currently done in this implementation as part of nano4parts.cpp. (This step was not needed in the H atom and simple molecule cases because the approach used to generate points led naturally to a far more dilute concentration.)

We first describe the calculation for a single ring without strain. We then describe the additional stages needed for systems of several rings and for systems under strain. The details of the simulations are given in Appendix A, and details of the transformations on the output to create AViz input files in Appendix B and recombination of the 4 parts in Appendix C.
7. Stereo visualization of electronic density around a nanotube

Further insight into the structure can be found by invoking several advanced AViz features. The fovy (field of view in the y direction) can be tweaked [20] in the panel from the viewpoint button on the AViz main panel (Fig. 12) via the explicit option (Fig. 13) so that the moire effects of a straight-on cartesian view are minimized.

The anaglyphic stereo, as presented in [19] and adjusted by the buttons on the left with the glasses images (Fig. 14) gives an image (Fig. 15) which when viewed with cyan–red glasses appears to come out of the page. In this figure one can see the higher density (red color) semitransparently behind the green gray lower density even without the glasses. A rotating version of this image is shown on [41].

8. Further analysis and conclusions

Having carefully described the computation and visualization of the charge density, we now turn to physical aspects; thus returning to our initial motivations. The first was research into the deformation of nanotube walls under distortions. The present project has demonstrated that we can calculate and visualize the electronic density surrounding the tube. We can observe local variations in its thickness at different axial locations, see for example Figs. 10 and 11. Further investigations with larger samples...
Fig. 9. Final result with dilution. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 10. Visualization of charge density for a 3-rings system without any strain.

Fig. 11. Visualization of charge density for a 6-rings system without any strain.

Fig. 12. The viewpoint panel of AViz, the “view” button was pressed and the explicit option selected to bring up the explicit panel.

and a careful numerical data analysis that falls beyond the scope of the present paper have begun and will be extended in the near future to larger and more distorted tubes.

The second motivation concerned the integration of simulations on electronic and atomistic scales as part of SimPhoNy [32]. The present computations provide one stage of efforts towards this
goal. We have successfully with initial molecular dynamics input transferred to electronic density functional theory calculations in a smooth, automatic and reproducible manner.

This new visualization technique for viewing electronic density has many potential applications. For example, to the study of any system that involves proximity and contact between different materials such as adsorption of molecules on surfaces. One such case (but certainly not limited to) is the next stage of the nanotube vibration project [27]. Our motivation for the study of nanotube vibrations was to study the way that the adsorption of DNA onto a nanotube would change the vibrational frequencies, hence enabling evaluation of the DNA mass. Part of this project also relates [26] to the nature of the nanotube boundary conditions, and in order to mimic the experimental case, where it sits on two side supports we need to know how the nanotube settles on these. To determine these we need to include the electron clouds in our calculations; and then observe their structure.

Acknowledgments

This study is part of the EU "SimPhoNy" collaboration (GA Nr. 604005). V.R.C. was supported by the U.S. Department of Energy, Office of Science, Basic Energy Sciences, Division of Materials Science and Engineering. This research used resources of the National Energy Research Scientific Computing Center, supported by the Office of Science, U.S. Department of Energy under Contract No. DEAC02-05CH11231. The main calculations were made on TAMNUN, supported by the RBNL Nanotechnology Institute at the Technion. We thank Arik Landau for advice on running Quantum Espresso on TAMNUN, Yulia Halupovich for her endless support on TAMNUN and Ariella Richardson for help with C++.

Appendix A. Details and code links for the simulation part

All tar files are opened with tar -xvf name_of_file. The instructions assume use of a LINUX system for the compilation and execution (tested on Red Hat), and terminal software that enables ssh -X access if the desktop is a different system to where the simulations and visualizations are carried out. For completeness we also include comments related to batch (PBS) submission on our local SGI RedHat Linux parallel cluster called TAMNUN; with the exception of commands related to MPI we have also checked their validity on a shared memory computer. TAMNUN has Quantum Espresso installed with Intel compilers and Aviz 6.1 in the /usr/local partition. TAMNUN is a very typical small parallel cluster, and its use in the following can be considered generic. The Aviz used here is the Qt3 6.1 version, but the visualization has been checked on the newer Aviz 7.0 Qt4 version [9] on UBUNTU. Initial versions of the single ring system were run at NERSC on a Cray XT system (hopper).

The following sections contain explanations for sample input files and parameter choices; more details are in the full set of downloadable files at [7], and non-system specific aspects are in the QE documentation [37,38]. Three cases—single ring, multiple rings and strained nanotubes are considered. For the single ring case one downloads the example_input.tar file.

A.1. Input file for pw.x

Fig. A.16 shows a typical input file for pw.x. We now present a brief discussion of the different lines of the input file; a full description of the documentation is linked above. The path to an user directory is /u/username/, where username has to be replaced by the login name that is used to connect on TAMNUN or the appropriate value for another system. In detail:

Fig. A.16. Input file for pw.x.

A.1.1. &control

- calculation : defines the type of calculation that one wants to do, ‘scf’ is for self-consistent field.
- pseudo_dir : the path where the potential file(s) are located. This is a user specific choice.
- prefix : ‘NANO’ is an example, one can change it and write whatever he/she wants but it is important that all pw.x and pp.x input files use the same prefix.
- outdir : this path is to the place where all the files will be saved during and after the computation.

A.1.2. &system

- ibrav : defines the type of lattice that is used. In this example, 8 is an orthorhombic unit cell with basis vectors: \( \mathbf{v}_1 = (a, 0, 0) \), \( \mathbf{v}_2 = (0, b, 0) \), \( \mathbf{v}_3 = (0, 0, c) \).
- celldm : this part defines the crystallographic lattice constants. The default units are Bohr and not Angstroms (1 Bohr = 0.529177249 Å). In the orthorhombic case, celldm(1) = a, celldm(2) = b/a and celldm(3) = c/a. Here an explanation about the values shown in the example (Fig. A.16) is necessary. In this example the vector along the length of the nanotube is given by celldm(2). The two others are set to a big value in order to "insulate" the system from boundary conditions effects (celldm(1) = 40, means that a = 40 bohr, and celldm(3) = 1, means that c = 40 bohr). To define the value that one has to put for celldm(2), the way is to compute the distance between
two consecutive rings (perhaps using a visualization program), convert this value to Bohr (1 nm = 18.8971 Bohr) and divide it by celldm(1), because the measured value will give ‘b’, but

\[
\text{celldm}(2) = \frac{b}{a}.
\]

- nat : is the number of atoms in the simulation. It has to be equal to the number of lines under the “ATOMIC_POSITIONS” section.
- ntyp : defines the number of different types of atoms present in the simulation.
- ecutroho : is the cutoff for the kinetic energy of the wavefunctions.
- ecutwfc : is the cutoff for the kinetic energy of the charge density and potential, the default value (if not specified) is 4 * ecutwfc.
- input_dft : defines the exchange and correlation functionals employed in the calculation. Here we chose the vdw-DF non-local correlation functional \([42,43]\) with the C09 \([44]\) exchange functional in order to account for London dispersion interactions (or van der Waals forces) within our calculations.

### A.1.3. ATOMIC_SPECIES

Under this header one has to specify the type of atom, the mass of the atom and the exact name of the pseudopotential that will be used for this atom (located in ‘pseudo_dir’).

### A.1.4. ATOMIC_POSITIONS

These are the coordinates of the atoms present in the system. There are many ways to do this; we just specified the type of atom (Carbon in the example Fig. A.16) and the coordinates using an .xyz format without the two first lines. After ATOMIC_POSITIONS, we specified the unit (see documentation for more details). For the structures here we employed Cartesian coordinates in Angstroms.

### A.2. Run pw.x

Now that the input file is created, one can run pw.x. Using MPI (on TAMNUN or a similar SYSTEM) the line to execute the code would be:

```
mpirun -np n /usr/local/espresso/bin/pw.x < name.in > name.out
```

where \(n\) has to be replaced by the number of processors. The path before pw.x is the right one for TAMNUN but could differ on another system. The result of this first part should be a folder with the name ‘prefix’.save, where ‘prefix’ is the name chosen for the prefix, (NANO in our example) and a file name.out.

### A.2.1. vdw-DW calculations

We have included dispersion (van der Waals) interactions within our DFT calculations using the vander Waals density functional \([42,44]\). To include these interactions in QE it is necessary to first generate the vDW-DW kernel table. This can be done using the following procedure:

- copy the file from /usr/local/espresso/PW/src/generate_vdw_kernel_table.x into the user’s directory.
- go into the destination folder and use the command (note the . at the end, it is important):
  ```
cp/usr/local/espresso/PW/src/generate_vdw_kernel_table.x
```
- run it with : ./generate_vdw_kernel_table.x
- move the resulting table file (vdw_kernel_table) into the directory where the pseudopotentials are located.

**Remark.** It can take a moment to execute the generate_vdw_kernel_table.x file but it only needs to be generated once. If this is not done properly you may get the following error while running pw.x:

```
Error in routine read_kernel_table (1):
No “vdw_kernel_table” file could be found
```

If you see this error try to repeat the above directions carefully.

### A.3. Systems of multiple rings

Adding more rings is not totally trivial and should be omitted in a first calculation. As explained in Appendix A.1.2, it is clear that if the number of rings changes, the number of atoms will change (nat) and the unit cell length along the \(b\)-axis, \(\text{celldm}(2)\) will also change. For example, if the system is now a 6-rings system the value of \(\text{celldm}(2)\) has to be multiplied by 6.

The position of the atoms is another aspect that requires care. Indeed, the \(\text{celldm}\) vectors define the simulation box. We recall the values are given in Bohr. On the other hand, the values of the positions are given in Angstrom. The fact that the positions are in Angstroms is a special case, it could be defined in Bohr if we wanted to. So an important thing to check is that all the atoms are contained in the volume defined by the \(\text{celldm}\) vectors. For simplicity our box starts from \((0,0,0)\) and has the size of the \(\text{celldm}\) vectors’ values. (Remark : 1 Bohr = 0.5291 Å). So if the atoms are not contained in these boundaries they have to be translated. To check, the \(y\)-coordinate of the atom (in Angstrom) has to be in the interval \([0, \text{celldm}(2)]\). N.B. The atoms just need to be within one unit cell of each other and could be defined starting at say \((-1/4, -1/4, -1/4)\).

The translation is done by a script that needs as input file a .xyz file with the coordinates. After downloading and opening the many_rings.tar file,

- compile the code with : g++ many_rings.cpp -o exec_many_rings
- put the .xyz file in the same folder and execute the code with : ./exec_many_rings
- enter the exact name of the file (.xyz included)
- select to do a manual translation or automatic one.

A file with the same name plus “.translated.xyz” will be created in the current directory. These are the new coordinates that will be used.

In the case one chooses to translate automatically, the program will find the minimum value in the first and last ring and translate every atom by this value along the \(y\) direction in order to have one atom at the origin and all the others contained inside of the simulation box.

### A.4. System under strain

The third case that has to be considered with care is the case where there is a strain in the nanotube. This can also be omitted in a first trial. The consequence of the strain will be to change the distance between the atoms. When one adds a strain in the nanotube the distance along \(y\) is increased and the distance along \(x\) and \(z\) (direction of the radius) is decreased. As the constraint is constant the distance between two neighbor-cells along one given direction should be the same. Nevertheless, as one can probably notice in Fig. 4 the slice of the nanotube does not fall along a perfectly straight line, which means that the distance is not constant everywhere between the consecutive cells. This problem is due to the way that the nanotube was built numerically and more specifically due to the boundary conditions. A way to minimize this problem is to take the atoms in the middle of the nanotube in order to be as far as possible from the edges. Every ring has 28 atoms. For the nanotube presented in Fig. 4, if one wants to select a system with 3 rings, the way to select them in the middle is to remove the 18 first rings removing the 504 first coordinates, then jump 84 coordinates and remove the rest. One can check that the distance is now constant between all the atoms. This has been tested for a system of 3 rings under 2.5%, 5% and 10% strain. For a larger number of rings this could be problematic if the distance is not constant.
A.5. Input file for pp.x

After the first step, one should have the two files mentioned above. The next step is to extract the charge density from the output file. To do so, one will create an input file for pp.x (see Fig. A.17) and obtain an output file that will contain the charge density for each point of the grid, as shown in our later figures. In detail:

A.5.1. &inputpp
- prefix: as mentioned before it is really important that the name of the prefix is exactly the same as it was for pw.x.
- filplot: this is the name of the output file and can be changed.
- plot_num: defines the quantity that one is interested in during the post-processing, 0 is for the charge density (see the QE documentation for other values).
- outdir: same as before.

A.5.2. &plot
This section may not be needed for other systems but from our experience on TAMNUN it is better to specify it.

- filepp[1]: is the name of the output file that will contain the quantity to be plotted and saved in fileout, here it is the charge density for example.
- output_format: the integer defines the format of the output file (see documentation).
- fileout: name of the file that contains the data to do the plot. This is not used for AViz but it can be useful if one wants to use other software such as xcrysden [45].

A.5.3. Remark
On TAMNUN it is important to put the outdir at the end of the section &inputpp. Another observation is that it is possible to specify the amount of data saved during the computation using ‘disk_io’. If this is not specified the default value is ‘low’ but the less data saved the more RAM is used on some systems and the code can crash/terminate if the available RAM is not sufficient. The default value is advised.

A.6. Run pp.x

The execution line is similar to the previous one:

```
mpirun -np n /usr/local/espresso/bin/pp.x < name.in > name.out
```

A.7. Example of a 1 ring system

For the one ring system (illustrated in Fig. 7), the tar file contains an input file for pw.x and another one for pp.x and also two bash scripts that create these files and submit them through the queuing system on TAMNUN.

A.8. Creation of the .xyz file for the charge density

The output file from pp.x should now be name.charge. This file has the structure shown in Fig. A.18. The first line has 8 numbers, the three first numbers are the size of the grid in the three spatial directions, the three following are the same repeated. The seventh
number corresponds to the number of atoms and the last one to the number of different type of atoms. On the second line, the first number is the type of Bravais lattice, and the three following numbers are the cell dm defined in the input file of pw.x. The first lines relate to the lattice constants. The charge density, is defined for each point of the grid starting at line 34.

Our aim is to create a .xyz format file with the charge density. More precisely, each value of charge density given in the output file corresponds to a point in the grid. To convert this file into an appropriate format one can download a script written in C++ as part of the example_input file and carry out the following:

1. compile and execute the code : g++ charge_density.xyz.cpp -o exec_charge_density
2. copy the file you wish to process into the directory where the script is
3. run ./exec_charge_density to open the program
4. enter the name of the file to convert and press enter.

A new file with the right format is now created in the same folder and is ready to be visualized. An older fortran conversion file for QE output is at [2].

Appendix B. Further data processing and visualization

B.1. Initial visualization with AViz

As it was mentioned previously, the raw charge density output from QE for this system is displayed in four different parts (see Fig. B.19). This is a special case and could have been done differently.

B.1.1. Input file for AViz

The output file after the execution of the C++ routine is a standard .xyz file for AViz (see Fig. B.20 and Section 5) having the first line with an integer total number of dots, then a comment line and then the third and subsequent lines containing the information. All numbers from line 3 onward have to be real ones. The different columns in our example are:

1. A letter (a,b,c,d,e,f) that indicates the colors (transparent, black, red, green, yellow, blue)
2. The x coordinate
3. The y coordinate
4. The z coordinate
5. The charge density at that point.

In this file the coordinates go from 1 to 200 in each direction. The colors are in order of increasing density ranging from less than 0.0005 to 0.5 or above in units of aₐ⁻³₀, where a₀ is the Bohr radius, see Fig. B.21.

In Fig. B.21 we show the color key and a realization of one quarter.

Appendix C. C++ code to recombine the circle

In this part the different important functions implemented will be presented. The general procedure is to read the file mentioned above, find which part corresponds to which quarter of circle, recombine them and create a new input file for AViz.

C.1. Function to read the initial .xyz file

The first step is to read the file in order to work on it after. To do so, it is possible to read line after line and store each line in a vector of strings (Fig. C.22).
the same line but translated in the y-direction. That means the second layer of points forming the plane.

This means that in order to separate the four parts, one has to take the lines labeled from 1 to 100 for the first half of the file (part1), the lines from 101 to 200 for the first half of the file (part2) and the same for the second part of the file (part3 and part4). This work is done by the loop shown in Fig. C.23. Each part is stored in a vector of strings.
C.3. Recombine the circle

This is the most difficult part. The elements of the different vectors are strings but we have to extract the numbers for the coordinates and also the charge density, process the coordinates and recreate the file. To read the numbers from a string, one can use the function `sscanf()`, but the input line to read has to be a `char`. So the idea is to read each line of the vector, convert the string into a `const char` using `.c_str()`, extract the different variables in the appropriate format, modify the coordinates and store the line in the same format as it was initially. One has just to be careful to erase the string and char variables after each iteration. (See Fig. C.24.)

```cpp
char color[2];
double x(0), y(0), z(0);
float charge(0.0);
std::string str0("\n");

const char *cstr("\n");
stringstream ss0;

for (unsigned int i(0); i<vector_part.size(); i++)
{
    str = vector_part[i];
cstr = str.c_str();

    sscanf(cstr, "%3s %lf %lf %lf %E", color, &x, &y, &z, &charge);
    if (number_part==1)
    {
        ss0 << " " << color << " " << x << " " << y << " " << z << " " << charge;
    }
    else if (number_part==2)
    {
        ss0 << " " << color << " " << x-200. << " " << y << " " << z << " " << charge;
    }
    else if (number_part==3)
    {
        ss0 << " " << color << " " << x << " " << y << " " << z-200. << " " << charge;
    }
    else if (number_part==4)
    {
        ss0 << " " << color << " " << x-200. << " " << y << " " << z-200. << " " << charge;
    }

    ss0.str("\n");
    ss0.clear();
}
```

Fig. C.24. Code to recombine the four parts of the sample and create a vector containing the information.

C.4. Dilute points

The sample is quite thick and the points are dense but in order to see the physics “inside” the sample one needs to dilute the points randomly. The idea is to delete a given number of points randomly in the vector that contains the information about the coordinates and charge density. (See Fig. C.25.)

C.5. Create the final output

The last step is to write an output file with the `.xyz` format described above but with the changed coordinates to have the right circle. This can be done using `ofstream`. The two first lines contain

```cpp
vector_dilute = vector_ordered;
int element_to_delete(0);

for (unsigned int i(0); i<dilution_number; i++)
{
    element_to_delete = rand()%vector_dilute.size()-2+1;
    if (element_to_delete > vector_ordered.size())
    {
        cout << "size = " << vector_dilute.size() << " random number = " << element_to_delete << endl;
        vector_dilute.erase(vector_dilute.begin() + element_to_delete);
    }
}
```
the number of points followed by a comment line. The rest can be done with a loop reading each element of the vector of strings recombined and writing it in the output file. (See Fig. C.26).

C.6. Run the code and results

To run the code, one has to compile it first using the command:

g++ -o name_of_executable nano4parts.cpp

This will create an executable file named nano4parts which should be placed in the same folder as the initial input file executed. The name of the input file will be requested, enter it (include the extension in the name; example.xyz) and press enter. The code generates five .xyz files: four to visualize each of the four parts of the sample and the fifth is the full circle. They will be generated in the same folder. One can then visualize the full_circle.xyz file with AViz and obtain the result presented in Fig. 8.

An easy way to view the inside of the sample is to make a random dilution. After this random dilution, one obtains the result shown in Fig. 9.

C.7. Download: recombine circle

The C++ code is downloadable on [7].

In order to compile and execute the program, one has to:

1. type g++ -o name_of_executable nano4parts.cpp
2. be sure that the file with the data is in the same folder
3. run the program using : name_of_executable

References


Fig. C.26. Code to create the output file with the changed coordinates.