SELF-ASSEMBLED ORGANIC MONOLAYERS AS MOLECULAR SIEVES: *AB INITIO* AND MOLECULAR DYNAMICS STUDIES

A Dissertation

presented to

the Faculty of the Graduate School

at the University of Missouri

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

BY

ALEXANDER VICTOR ST. JOHN

DR. CARLOS WEXLER, DISSERTATION SUPERVISOR

DECEMBER 2015

The undersigned, appointed by the Dean of the Graduate School, have examined the dissertation entitled:

SELF-ASSEMBLED ORGANIC MONOLAYERS AS MOLECULAR SIEVES:
AB INITIO AND MOLECULAR DYNAMICS STUDIES

presented by Alexander Victor St. John,
a candidate for the degree of Doctor of Philosophy and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Carlos Wexler

Dr. Ioan Kosztin

Dr. Paul Miceli

Digitally signed by rothm5@nku.edu
DN: cn=rothm5@nku.edu
Date: 2015.12.07 06:51:45 -05'00'

Dr. Michael Roth

Dr. Thomas Sewell

Acknowledgements

I would like to thank my academic advisor, Dr. Carlos Wexler, for his guidance and support through my graduate studies, as well as his encouragement for me to follow and trust my passion and opportunity in my professional development. I thank Dr. Matthew Connolly for helping me with a strong start on writing the code and using the software used in this project. I thank Dr. Michael Roth for his encouragement and instruction in data reduction, and Dr. Lucyna Firlej and Dr. Bogdan Kuchta for their efforts to begin this project, and for our following fruitful discussions. I also thank Dr. Ioan Kosztin for the several rigorous courses that I've taken from him in theoretical and computational physics. Lastly, I want to express my gratitude to the University of Missouri and the entire Physics Department for their acceptance of, and investment in my potential.

I would like to acknowledge financial support by the American Chemical Society Petroleum Research Fund Grant No. 52696-ND5, and the University of Missouri Research Board and Research Council.

TABLE OF CONTENTS

List of Tables

List of Figures

Abstract

We present the results of extensive fully atomistic Molecular Dynamics (MD) simulations of a self-assembled organic monolayer consisting of interdigitated molecules of 1,3,5-tristyrlbenzene substituted by alkoxy peripheral chains containing n = 6, 8, 10, 12, or 14 carbon atoms (TSB3,5-C*n*, or TSB35, for generality or brevity), deposited onto highly ordered pyrolytic graphite (HOPG). We focus on the TSB3,5-C6 and the TSB3,5-C10 systems. Our studies begin with the *ab initio* determination of the optimal electronic structure of the TSB35 molecule, from which we find the optimal lattice geometry and reproduce experimental results for the TSB35 monolayer. The structure and functionality of the TSB35 monolayer as a "molecular sieve" is then explored through fully atomistic MD simulations where we find the monolayer to be stable and well ordered around room temperature. We calculate probability distributions for TSB35 dihedral angles and pore diameters of the molecular sieve to characterize order in the monolayer and investigate how it transitions from order to disorder. We then introduce guest organic molecules (e.g., benzene, pyrene, coronene) into the nanoporous host structure and observe how the stability of the TSB35 monolayer is affected by such additions.

Chapter 1

Introduction

The emergence of nanoscience and the great successes in building nanotechnology have pushed the boundaries on what is possible to control at the molecular and atomic scales. The ability to manipulate molecular degrees of freedom, for example, has led to the assembly of nanometer-scale materials such as carbon nanotubes [1], zeolite-like [2] and monolayer [3] porous materials. One intriguing mechanism is the self-ordering growth or *self-assembly* of individual molecules into molecular structures that may be exploited for novel properties and reduced dimensionality [4,5,6]. With knowledge of the mechanisms that occur on the nanoscale, and what may drive them, such as concentration [7,8] or dipole interactions [9,10], we may gain insight into the underlying interactions between subsystems that cause large-scale order and inspire application of such structures, paving the way for bottom-up techniques of nanoscale fabrication [11]. **Figure 1.1** visualizes the scaling down of various methods of fabrication.

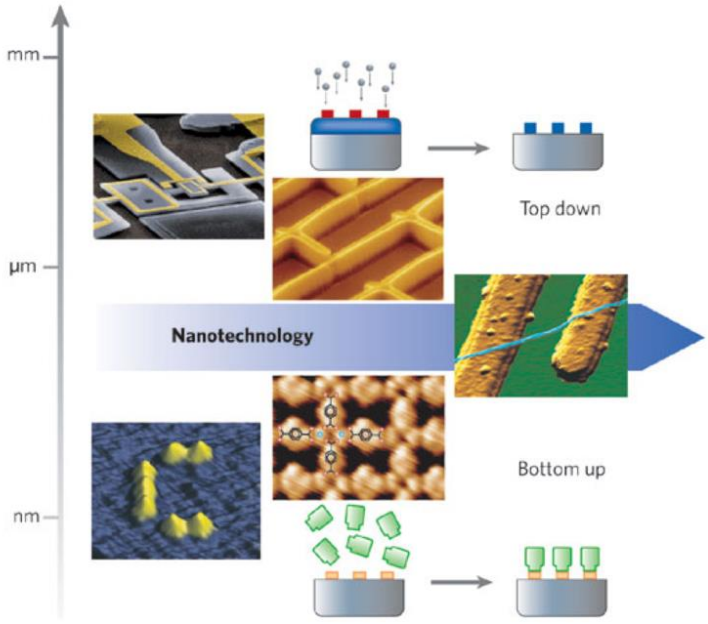


Figure 1.1. *Scaled-down manufacturing techniques are proving to be more controllable and efficient, after Ref. [3].*

Spontaneous molecular self-assembly is a promising route for bottom-up manufacturing of two-dimensional (2D) nanostructures with specific topologies [11,12,13,14, 15,16] on atomically flat surfaces. Of particular interest is the possibility of selective lock-and-key interaction of guest molecules inside cavities, or pores, formed by self-assembled host structures with complex geometries. This type of surface nanostructure exhibits promising properties, such as selective adsorption and catalysis in solution. The self-assembly process is also an appealing route for bottom-up fabrication, being more efficient and cheaper than conventional top-down fabrication techniques, as the processes and mechanisms exploited are inherent to the molecular system. The use of organic materials is also becoming an increasingly attractive route for fabrication, for example, in organic photovoltaics [17] and printable devices [18].

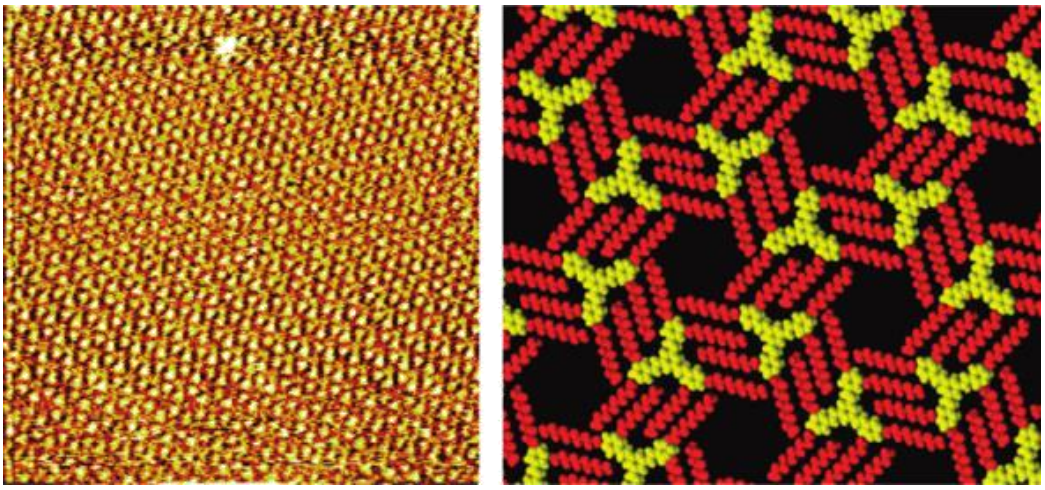


Figure 1.2. *(Left) An example of a self-assembled TSB35 monolayer as observed by scanning tunneling microscopy (STM) at the solid-liquid interface (70 nm by 70 nm). (Right) A computer generated scheme of alkyl chain intercalation, forming a 2D nanostructure of TSB35 molecules, after Ref. [14].*

Various experiments have been carried out on the self-assembled monolayer (SAM) studying the structural topology and the thermodynamic stability of such 2D nanostructures [12,13,14], see, e.g., **Fig. 1.2**. It was soon observed that these structures could act as 2D molecular "sieves", and be used as a means of adsorption with particular size and shape selectivity ("lock-and-key interactions"), based on the sieve's topology

[11,19,20]. These systems have also shown rotational [21,22] and vibrational [13] dynamics of the adsorbed molecules occurring within the pores, see **Fig. 1.3**.

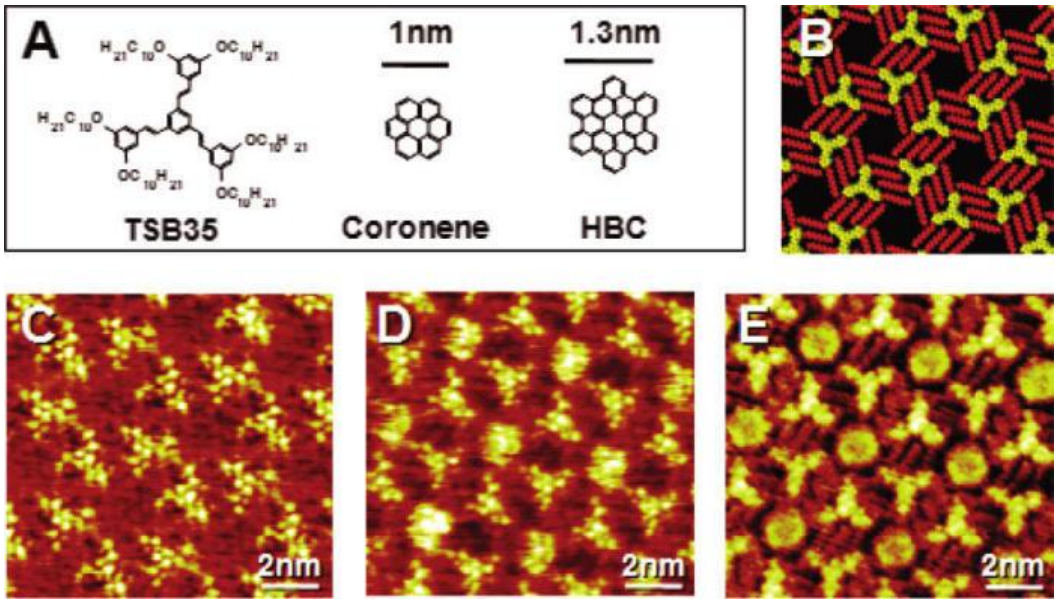


Figure 1.3. (A) Molecular structures of host molecule TSB35 (see text), and two guest molecules, coronene, and hexabenzocoronene (HBC). (B) Scheme of a monolayer assembly of TSB35. (C) STM image of the self-assembled monolayer of TSB35 on HOPG. (D,E) Introduction of guest molecules, coronene, and HBC. After Ref. [13].

Our host structure is a monolayer consisting of interdigitated 1,3,5-tristyrylbenzene substituted by alkoxy peripheral chains containing n = 6, 8, 10, 12, or 14 carbon atoms (TSB3,5-Cn, or TSB35 generically) deposited on a highly ordered pyrolytic graphite (HOPG) surface. This type of molecule, and others in the class of stilbenoids, have been of interest in several experimental settings [23,24,25]. The geometry of the TSB35 molecule, with its peripheral alkoxy chains, drives the self-assembly process through a lock-and-key mechanism, as it relaxes onto the HOPG substrate. Experimental and computational studies have been carried out on a variety of molecules of similar characteristics [26,27,28], which tend to form honeycomb, or hexagonal, supramolecular networks through this self-assembly process. The resulting network geometry is influenced by the TSB35 concentration [29] and by the introduction of guest molecules in the

application of selective adsorption [29,30]. Linear networks have been observed, but do not exhibit nanoporous features, and can be transformed to the porous honeycomb structure by control of the adsorbate peripheral chain length [24], the solvent concentration of adsorbate molecules (TSB35) [29,30], or the inclusion of guest molecules in the solvent [29,30]. We focus on the porous honeycomb network for our studies.

Using *ab initio* methods from quantum chemistry, we optimize the molecular (geometric and electronic) structure of the TSB3,5-C6 and TSB3,5-C10 molecules, and calculate the optimal geometry parameters required for the force field specification to be used in our moleculad dynamics (MD) studies. To start the MD simularions, we consider the experimental results for the epitaxial relationship between the TSB35 overlayer and the HOPG substrate, which includes the lattice spacing parameter and domain angle [14]: we find the optimal honeycomb, or hexagonal, lattice geometry for the TSB3,5-C6 and TSB3,5-C10 monolayers. We then construct and analyze the structure and functionality for each of the TSB3,5-C6 and TSB3,5-C10 monolayers, and their ability to act as a molecular sieve through extensive MD simulations. The introduction of guest molecules into the host (honeycomb) monolayer structure [12,13,22] gives insight to the mechanism of selective adsorption and host structure stabilization [29,30]. Specifically, we observe stabilization in both the TSB3,5-C6 and the TSB3,5-C10 monolayers when guest molecules (benzene and coronene, respectively) are introduced to the monolayer.

Chapter 2

Ab Initio Model of the System

I. Introduction to Electronic Structure Calculations

The main purpose of *ab initio* electronic structure calculations is to use approximate methods and wavefunctions to calculate, from fundamental quantum mechanics, the electronic structure and energy of a molecule, and how this electronic structure depends on the molecular geometry. From this "potential energy surface" (PES) that depends on the position of each of the atomic nuclei, the optimal molecular structure may be calculated by finding the energy minimum. Additionally, internal "elastic constants" related to variations of the structure from the minimum may also be calculated (we will later refer to these as "bonded interactions"), as well as atomic "partial charges" and local polarizabilities that are important for the "non-bonded" intra- and inter-molecular potentials [31].

There are several methods available to calculate the geometric and/or electronic structure of a molecular system, based on prior knowledge of the system: Molecular Mechanics (MM), Semi-Empirical, and *Ab Initio* methods. Note that there are also various hybrids of these methods (e.g., semi-*ab initio*) [31]. Molecular Mechanics uses a classical physics framework, which relies on empirical parameters, and does not calculate any electronic properties, but does well in calculating mechanical properties for systems of thousands of atoms. Semi-Empirical methods use a quantum mechanical framework, which also relies on empirical parameters, but can calculate saddle points, and works well for systems of hundreds of atoms. *Ab Initio* methods use a quantum mechanical framework, and do not rely on any empirical parameters. These methods are used when high accuracy is important with a small system (tens of atoms), and it may be used to explore *new* chemistry and physics, since it requires no experimental or previous computational data [31]. The progression is illustrated below in **Fig. 2.1**.

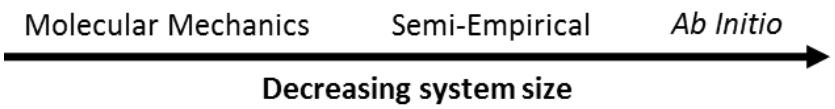


Figure 2.1 *Scheme of a few different methods of calculating properties of molecular systems. Accuracy increases as more sophisticated methods are employed (only possible for smaller system sizes).*

In our work we used *ab initio* methods for their fundamental value and accuracy (but this required some compromises that will be discussed below). The desired outcome of these calculations is the total electronic wavefunction and the associated energy, which requires the computation of atomic and molecular orbitals (AO, MO), by solving Schrödinger's equation

$$\widehat{H}\Psi = E\Psi, \tag{1}$$

where $\Psi(\mathbf{R}_n, \mathbf{r}_e)$ is the quantum wavefunction ($\mathbf{R}_n, \mathbf{r}_e$, represent all nuclear and electron coordinates, respectively), $\widehat{H} = \widehat{K} + \widehat{U}$ is the Hamiltonian operator for the system, $\widehat{K}$ is the kinetic energy, $\widehat{U}$ is the potential energy, comprised of the Coulomb interaction (nuclei-nuclei, electron-nuclei, and electron-electron), and E is the system energy. Given the large discrepancy between the masses of electrons and nuclei, the latter are treated classically (i.e., fixed coordinates $\mathbf{R}_n$, which are then treated as a paramter) and quantum mechanics is only considered for the electrons [31]. For a fixed set $\mathbf{R}_n$, **Eq. (1)** is solved for $\Psi(\mathbf{r}_e)$, $E(\mathbf{R}_n)$, the latter becoming the PES for molecular geometry minimization, etc.

For all but the simplest of systems, **Eq. (1)** must be solved numerically and the total electronic wavefunction approximated. Our calculations were performed using the *Gaussian09* code [32]. An outline of the procedures for an (approximate, numerical) electronic structure calculation can be seen in **Fig. 2.2** [31].

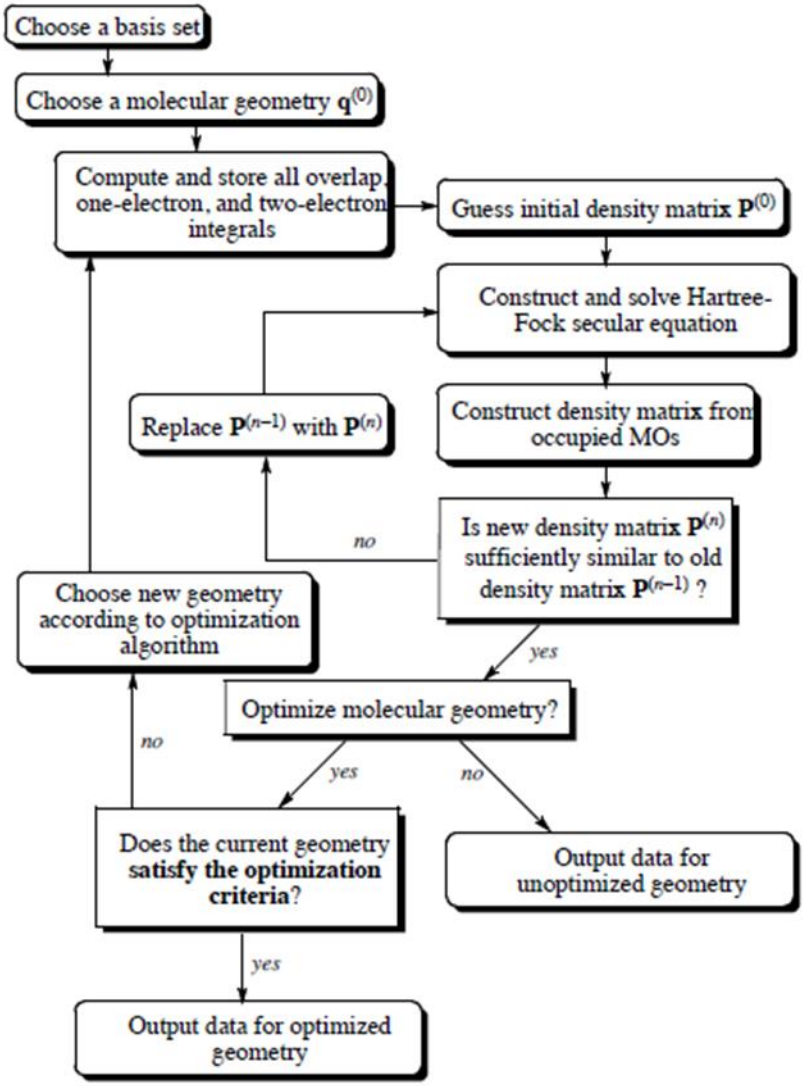


Figure 2.2 *Algorithmic scheme of an electronic structure calculation implemented to optimize the electronic wavefunction of the electrons contained in a molecule, and find the optimal molecular geometry. After Ref. [31].*

We begin with the (restricted) Hartee-Fock (HF, or RHF for closed-shell molecules) theory [31], and then progress to the 2nd order Møller-Plesset perturbation theory (MP2) [33], which accounts for ca. 94.0% of the electron exchange and correlation energy (E_{XC}) and is considered the "silver standard" in *ab initio* calculations. MP2 computational cost scales as N^5, which is a modest increase over HF's N^4 (here N is the number of degrees of freedom of the electronic wavefunction under consideration, see discussion on "basis sets" below). It is possible to further increase the accuracy of the perturbative

calculations, but at a very substantial cost, e.g., 4th order MP, or MP4 captures typically 99.5% of E_{XC}, but with a significantly higher computational cost ~ N^7 [33,34]. Non-perturbative methods such as the "gold standard" coupled-cluster (CC) are able to capture up to ~99.9% of E_{XC}, but it scaling up to ~ N^{10} prohibitive prohibitively costly for systems containing any more than a few atoms [35,36].

As with any approximate method, one has to balance the price/benefit budget; hence one may choose a more sophisticated theory but then has to settle for a more primitive basis set, or vice-versa [31]. In our work we decided to stop at the "silver standard" MP2, but explore larger and more precise electronic basis sets. In our calculations we employed the standard Pople basis sets [33], namely we used the following combination of theory and sets: HF/3-21G, MP2/3-21G, MP2/6-31G, and MP2/6-31G(*d,p*). The notation before and after the "/" indicates the level of theory and the basis set, respectively. For the Pople basis sets [31,37], the notation after the slash reads, "X-YZ G". The "X" represents the number of primitive Gaussians comprising each core AO basis function, the "Y" and "Z" denote that the valence AO are composed of two basis functions each, where the first one is a linear combination of Y primitive Gaussians, and the other is a linear combination of Z primitive Gaussians. Lastly, the notation "G(*d,p*)", also referred to as "G**", indicates that *d*-type and *p*-type polarization functions are used on heavy atoms and hydrogen atoms, respectively. Pople's 6-31G(*d,p*) basis set allows for a very good description of smaller atoms in the first rows of the periodic table (H-He, Li-Ne), since it includes polarization terms for the *s* and *p* orbitals [31].

The molecular orbitals are constructed as a linear combination of the separate, linearly independent atomic orbitals (**Eq. 2**), where the basis set determine how many functions are used and how the AO interact with each other through choices of split valence, polarization, and/or diffuse functions [31,32]:

$$\Psi \equiv \Psi_{total\ electronic} = \sum_{i=1}^{N} a_i\, \varphi_i\ , \qquad (2)$$

where N is the number of basis functions in the basis set $\{\varphi_i\}_{i=1}^{N}$, and the coefficients $\{a_i\}_{i=1}^{N}$ are the degrees of freedom used to minimize the molecular energy to find the electronic ground state wavefunction. The basis functions or AO are Slater-type orbitals (STO) φ_i^{STO}, which are constructed as linear combinations of Gaussian-type orbitals (GTO) φ_i^{GTO}. Note that this makes the total electronic wavefunction a sum of sums, and the total number of coefficients (variables) is what governs the scaling cost of the computation as described above. The STO replicates correct behavior at distances close to the nucleus (forms a cusp at the origin) and far from it (STO ~ e^{-r}, where r is the radial distance from the nucleus), whereas GTOs give good results for electronic behavior at intermediate distances from the nucleus (GTO ~ e^{-r^2}). **Figure 2.3** shows an example of the approximation for the 1s orbital using minimal basis sets STO-nG, where n is the number of primitive Gaussians comprising the core *and* valence orbitals. Recall that the Pople basis sets allow a different number of primitive Gaussians to be used for core and valence orbitals.

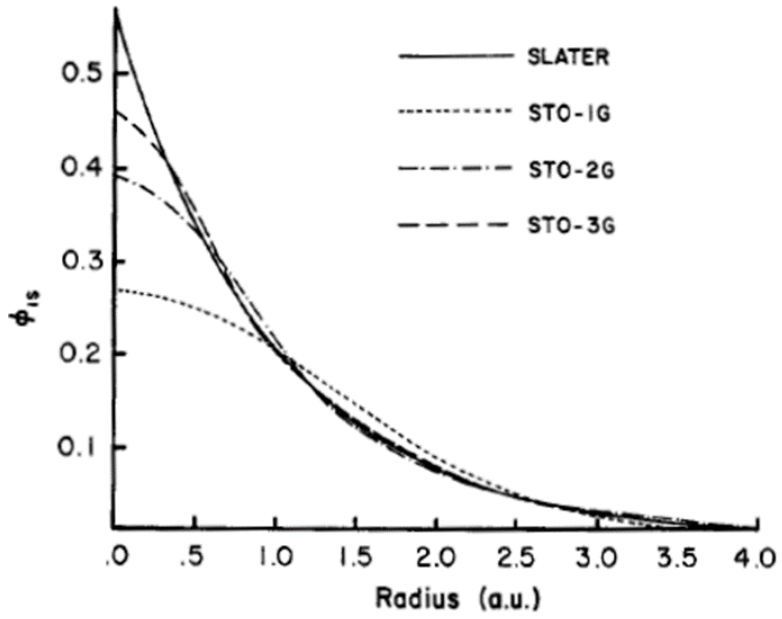


Figure 2.3 *An example of minimal basis sets (dashed lines) approaching the form of the Slater-type orbital (solid line) by creating linear combinations of primitive Gaussians, which are easier to calculate and replicate electronic behavior well at intermediate distances from the nucleus [31].*

Low levels of theory (e.g., Hartree-Fock) will be quick, but the resulting wavefunction and energy may not be accurate, while higher levels of theory (e.g., 2^{nd} order Møller-Plesset or higher) may take much more computational time, but will usually be more accurate. As a note of caution, even the higher levels of theory can give poor results, if the input/guess is too far from the true wavefunction. Thus, it is good practice to have a good initial guess, by beginning with a low level of theory, and progressively work up in complexity, recursively using the results as the next input/guess (Gaussian checkpoint files), and then to check that the molecular energy is decreasing as the method and basis set are increased in complexity. This also is advantageous in terms of minimizing the total computational cost.

II. *Ab Initio* Results: Construction of the Molecule

The initial planar configuration of the TSB35 molecule studied is presented in **Fig. 2.4**, namely TSB3,5-C6 and TSB3,5-C10. The overall features of the geometry are known from experiment [14]. The construction and optimization of these molecules was made complicated by the non-trivial geometry and large size of the molecule. *Ab initio* methods are well-suited for systems of tens of atoms, but each TSB3,5-C6 molecule has 168 atoms, and each TSB3,5-C10 has 240 atoms. An attempt to avoid saddle points and oscillations in the energy, indicated by negative eigenvalues (imaginary vibrational frequencies) in the Hessian matrix [31], was to fix certain parameters (e.g., bond lengths) of the molecule, and optimize the remainder. Thus, we decided to split the full molecule into fragments, which consist of tens of atoms each, and optimize them separately. This gives us a computationally efficient and accurate method for building and studying the geometric and electronic behavior of the TSB35 molecule.

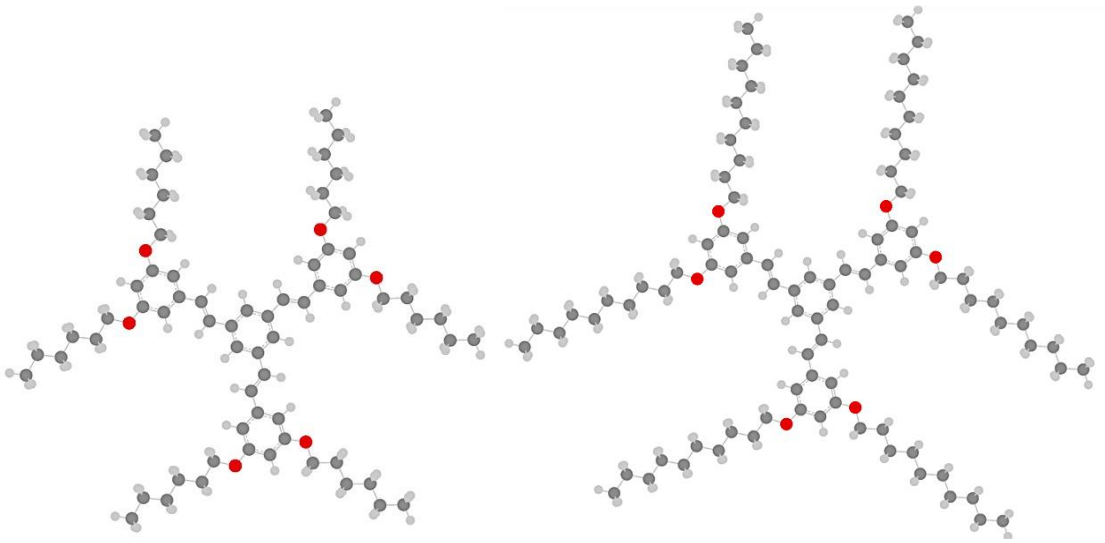


Figure 2.4 *The initial, planar configuration of the TSB3,5-C6 (left) and TSB3,5-C10 (right) molecules with optimized electronic structure. Carbon atoms are colored in dark gray, hydrogen atoms are colored in light gray, and oxygen atoms are colored in red.*

To build the structures shown in **Fig. 2.4**, the geometric and electronic structure was be optimized using *ab initio* methods in Gaussian09 [32]. We implemented a computationally efficient "fragmentation" process, where we split the full TSB35 molecule into its smaller molecular constituents, namely E-stilbene (stilbene), 1,3-diethoxy-5-methylbenzene (DEMB), and the corresponding alkane (hexane, decane), see **Figure 2.5**. Each fragment is progressively optimized from low-level theory/basis set to high-level theory/basis set (HF/3-21G, MP2/3-21G, MP2/6-31G, and MP2/6-31G(*d,p*)); the longest running time for an optimization was a few hours in a 24-CPU Intel$^{(R)}$ Xeon$^{(R)}$ E5-2620 2.0 GHz computer. The fragmentation also allows us to isolate certain portions of the full molecule, and extensively study its electronic and geometric behavior; which may be important for further *ab initio* calculations on calculating parameters such as force constants for the molecule, as it is not yet an extensively studied molecule, and the literature for certain parts of the TSB35 molecule is non-existent (e.g. oxygen link to the peripheral alkoxy chains).

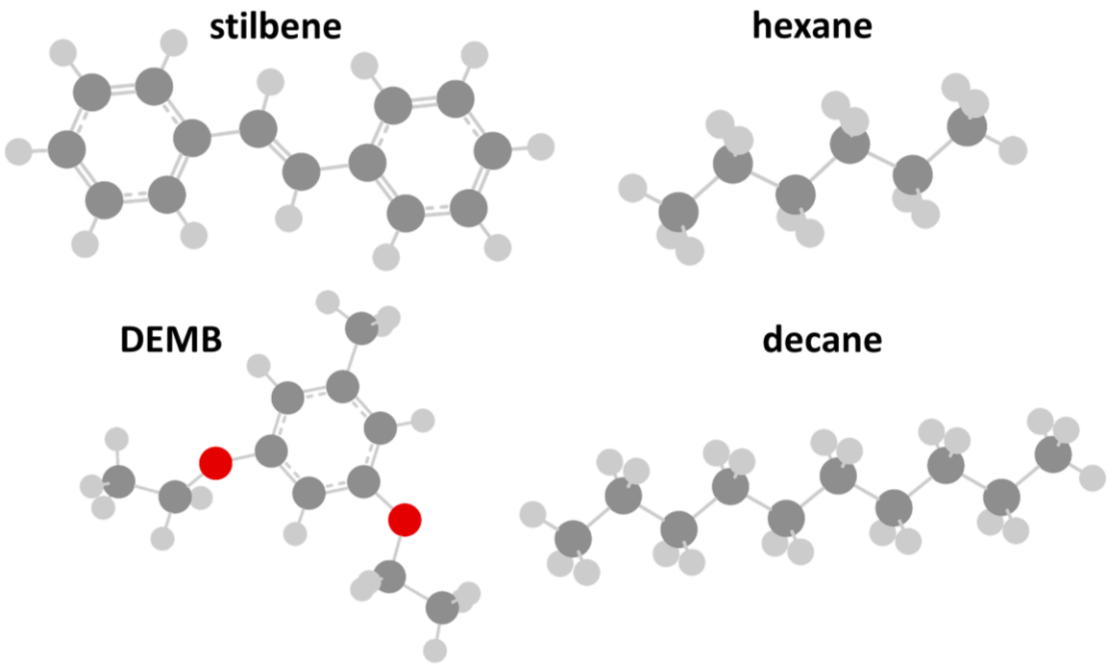


Figure 2.5 *Molecular fragments chosen and optimized for computational efficiency.*

Table 2.1 shows the molecular energy for each fragment decreasing as the level of theory/basis set is increased in complexity and size. Note that this comparison of energies across levels of theory and basis sets is not particularly rigorous, but the monotonically decreasing values is a quick check that the calculation is making a better estimate of the global minimum of the true molecular energy. An example plot of the decreasing energy of stilbene is shown in **Fig. 2.6**.

Table 2.1 *Molecular fragment energies for each level of method/basis set*[a] *.*

	Molecular energy (a.u.)[b]			
	Stilbene	DEMB	Hexane	Decane
HF/3-21G	-534.143843	-572.395477	-234.070668	-389.347093
MP2/3-21G	-535.381925	-573.628290	-234.634200	-390.283518
MP2/6-31G	-538.184831	-576.583402	-235.833290	-392.278816
MP2/6-31G(*d*,*p*)	-539.071534	-577.538320	-236.302770	-393.052004

[a] Note: Although energies are not known to 9 digits, often small energy differences *between* configurations are required; it is common practice to retain all digits out of *ab initio* calculations.

[b] Energies are in Hartree or a.u. (1 Hartree = 27.2116 eV = 2,625.6 kJ/mol).

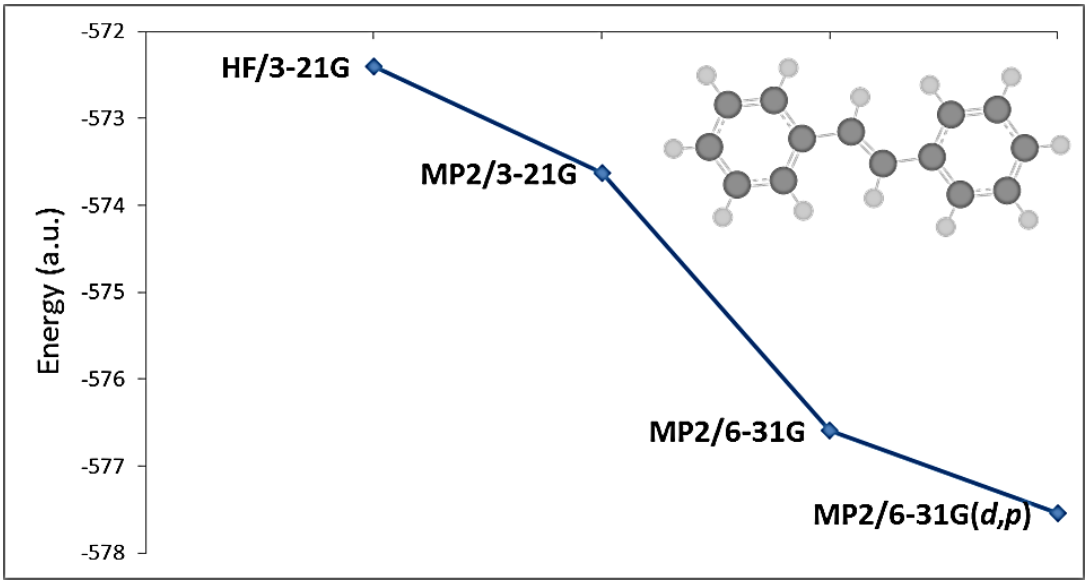


Figure 2.6 *Decreasing energy of stilbene, approaching the true minimum energy as level of theory and basis set are increased in complexity and size. Similar trends are seen for each molecular fragment. See* ***Table 2.1****.*

In piecing together the full molecule, it was important to capture the appropriate geometric and electronic behavior of each fragment and retain self-consistency for the entire TSB35 molecule in our model. To this end, it is important to compare and/or contrast similar atomic features between different fragments, such as aromatic carbon-carbon bonds of stilbene and DEMB, or aliphatic carbon-carbon bonds of hexane and DEMB, and then carefully assign Mulliken charges and equilibrium bond lengths, bond angles, and dihedral angles to the full assembled molecule.

An interesting observation during optimization was that the straight-chain alkanes, hexane and decane, would sometimes get "trapped" in a *gauche* conformation (*local* minimum) vs. the known *global* minimum for the *trans* conformation The cause was that the gradient descent algorithm was making steps in the nuclear configurations that were too large, thus sometimes missing the global minimum of the PES. The default value is 0.3 Bohr or radians. Reducing it to 0.1 Bohr or radians allowed the gradient descent to find the proper global minimum.

Another observation in the optimization of the stilbene molecule showed that the dihedral angle across the double bond preferred an angle of about 21.5° degrees from planarity. We desire a planar TSB35 configuration, since we will simulate an adsorbate-adsorbent system, where the adsorption potential would dominate the dihedral potential. Although there is no *a priori* reason why the molecule *in vacuo* would be planar, we find it more convenient to manually reset the dihedral to 180.0° degrees so that the molecule is planar in our initial geometry for setup, such that we will begin our simulations with the fully adsorbed TSB35 monolayer. The mechanism of the self-assembly process is of interest to explore, but occurs on much longer timescales than what we may feasibly simulate (tens of ns, in our case we found that 6 to 12 ns simulation sufficed for equilibrium MD simulations of pre-assembled structures).

With the geometric and electronic structure parameters taken from these optimizations, the full molecule is assembled by comparing and contrasting the various (local) electronic environments of the full molecule with the corresponding fragment, and then assign atom types/names according to the CGenFF database of CHARMM27 [38,39] (the force fields we use for MD simulations). The CGenFF/CHARMM27 is considered reasonable force field for use with generic organic molecules, and its naming convention includes a significant amount of information about the chemical/electronic environment of each atom. We assigned/matched the CGenFF atom types based on our *ab initio* calculations, supported by the fact that the local electronic environment for each carbon atom is not equivalent, but based on the strong locality of the covalent bond the atom types may be identified from their *local* structure (i.e., nearest bonds). Although the CGenFF is very complete it is lengthy, making it cumbersome to include in figures/tables/etc. For brevity, in figures and tables we used a short-hand notation, with a translation table provided in **Table 2.2**.

In most force fields, many different atoms are grouped in a single atom type for simplicity (e.g., all carbon atoms in an aromatic structure, even if the immediate chemical structure is not strictly identical). In our calculations we were compelled by significant variations in the local geometry and chemical environment to diversify the atom naming scheme CGenFF by including additional atom sub-types. **Figure. 2.7** illustrates the

adopted atom typing scheme (note that this figure is the central portion of the full TSB3,5-C6 molecule, **Fig. 2.4**). The aromatic carbons have black labels, "A", "B", "C", or "D". The alkene carbon atoms have golden labels, "A", "B", "C", or "D". Through trial-and-error, this self-consistent convention is used in order to appropriately split the carbon atoms into the sub-types, illustrated in **Fig. 2.7**, based on their connectivity and resulting electronic/geometric environments, which must be properly represented in the file structure for the molecular dynamics simulations. Note that the labeling scheme for the central ring is different from that of the peripheral rings. Also, note that this scheme does not require that atoms of the same type have the same Mulliken charge, since Mulliken charge is assigned to each atom individually in the topology (.TOP or .PSF) file (see **Appendix A**).

Table 2.2 *Translation of CGenFF Atom Type Labels.*

CGenFF Label	Short Label	Meaning
CG2R61	CA	Aromatic carbon
HGR61	HA	Aromatic hydrogen
CG2DC1	CH1	Double-bonded aliphatic carbon
HGA4	H1	Double-bonded aliphatic hydrogen
OG301	O	Oxygen
CG321	CH2	Single-bonded aliphatic carbon
HGA2	H2	Single-bonded aliphatic hydrogen
CG331	CH3	Single-bonded aliphatic terminal carbon
HGA3	H3	Single-bonded aliphatic terminal hydrogen
CG2R61	CG	Aromatic carbon (graphitic)

Table 2.3 and **Table 2.4** list all of the bond lengths and bond angles in the TSB35 molecule. According to **Fig. 2.7**, there are four sub-types (A, B, C, and D) for aromatic carbons and for double-bonded aliphatic carbons. To denote this in the following tables, we use the "Short Label" convention defined in **Table 2.2**, and follow it with the sub-type (A, B, C, or D) by hypenation, "-A, -B, -C, or -D". The use of parentheses denotes that several of the sub-types share that particular parameter. For example, the notation "CA-(C,D) : CA-(C,B) : O" implies that there are two angles share the same equilibrium bond angle of 124.2°: "CA-C : CA-C : O" and "CA-D : CA-B : O".

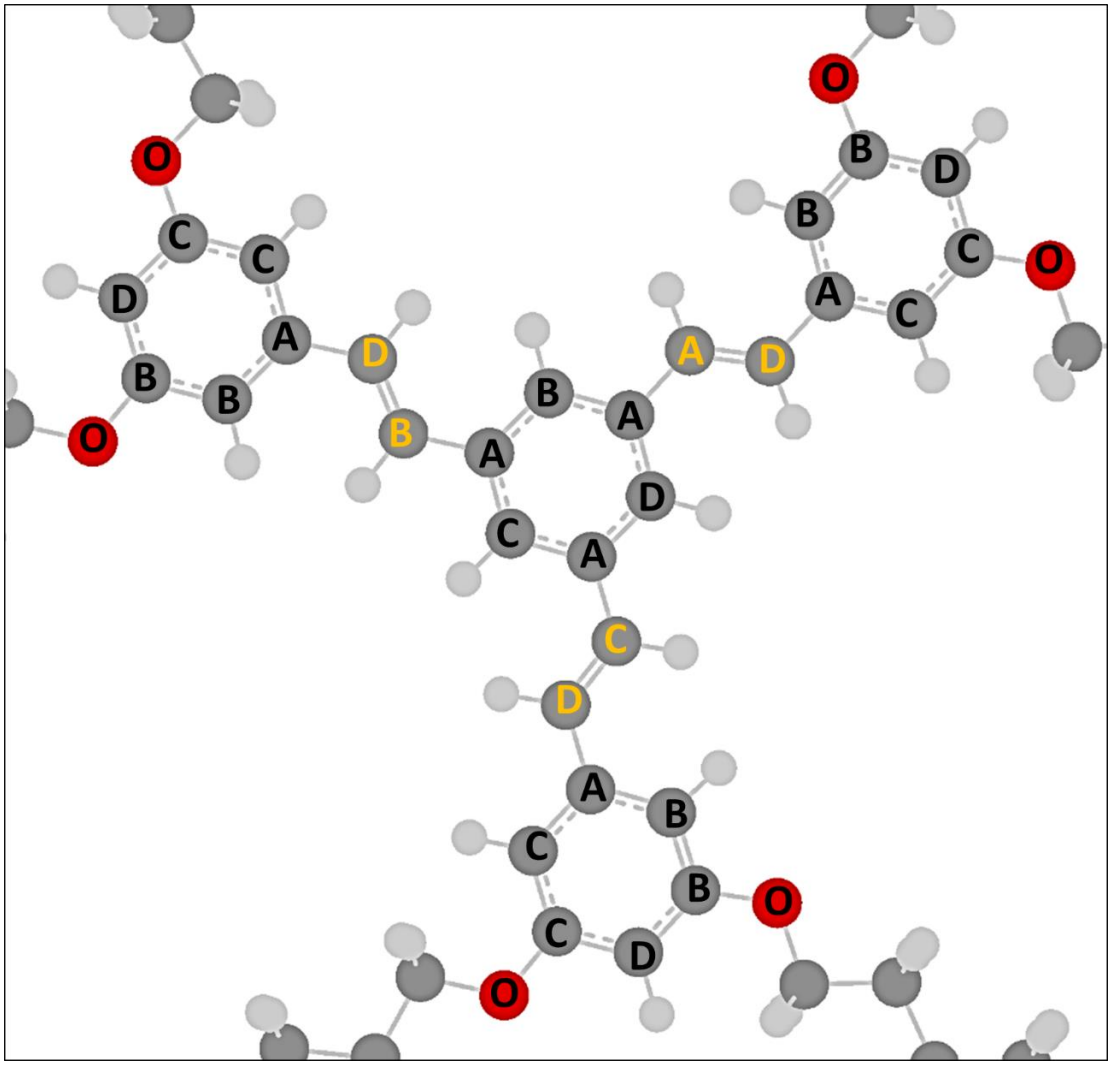


Figure 2.7 *Carbon atom type naming scheme for diversifying atom types based on connectivity and calculated electronic structure. Aromatic carbons are labeled in black, and aliphatic carbons are labeled in orange. See* ***Table 2.2****.*

In the regions where there are several of same type of bond or angle, the aromatic carbon-carbon bonds, “CA : CA”, or carbon-carbon-carbon angles, “CA : CA :CA”, for example, the equilibrium bond length or angle listed under MP2/6-31g(d,p) is the average of all of those bond lengths in that particular region. This is done as long as the calcaulted values are sufficiently close, typically within 0.1 Å for bond lengths, and 1° bond angles. For bond lengths and bond angles significantly outside of this tolerance, we used the atom sub-types, “A”, “B”, “C”, and “D”, to account for this.

Table 2.3 *Comparison of calculated bond lengths to CGenFF database [38, 39].*

Bond Length Types	Taken From Fragment	Equilibrium Bond Length (Å)	
		CGenFF	MP2/6-31g(d,p)
Central ring to alkene link			
CA : CA	Stilbene	1.38	1.40
CA : HA	Stilbene	1.08	1.08
CA : CH1-(A,B,C)	Stilbene	1.49	1.46
Alkene link to peripheral ring			
CH1-(A,B,C) : CH1-D	Stilbene	1.34	1.35
CH1-(A,B,C,D) : H1	Stilbene	1.10	1.09
CH1-D : CA-A	Stilbene	1.49	1.46
Peripheral ring to oxygen			
CA : CA	DEMB	1.38	1.40
CA : HA	DEMB	1.08	1.08
CA : O	DEMB	1.26	1.37
Oxygen to alkyl chain			
O : CH2	DEMB	1.26	1.43
CH2 : CH2	Alkane	1.53	1.52
CH2 : H2	Alkane	1.11	1.09
CH2 : CH3	Alkane	1.53	1.52
CH3 : H3	Alkane	1.11	1.09

Creating the different atom types was particularly important when measuring bond angles in two regions of the TSB35 molecule. The first region is under **Table 2.4**, "**Peripheral ring to oxygen**", and contains the angles "CH1-D : CA-A : CA-B" and "CH1-D : CA-A : CA-C". These listings are of the same base atom type, "CH1 : CA : CA", but the hyphenated label (sub-type) is different. The CGenFF literature defines the equilibrium bond angle to be 123.5° for both of them, but our calculated values are significantly different (122.4° and 117.2°). This calculated difference, corresponds to two different "directions" with which one may leave or enter the peripheral ring, where steric hindrance causes these two angles to be unequal. See **Figure 2.9** for illustration of the corresponding bond angles (near the bottom left of the figure).

Table 2.4 *Comparison of calculated bond angles to CGenFF database [38, 39].*

Bond Angle Types	Taken From Fragment	Equilibrium Bond Angle (°) CGenFF	MP2/6-31g(d,p)
Central ring to alkene link			
CA : CA : CA	Stilbene	120.0	120.0
CA : CA : HA	Stilbene	120.0	120.0
CA-(B,C,D) : CA-A : CH1-(A,B,C)	Stilbene	123.5	122.4
CA-(D,B,C) : CA : CH1-(A,B,C)	Stilbene	123.5	117.2
CA-A : CH1-(A,B,C) : H1	Stilbene	116.0	115.9
CA-A : CH1-(A,B,C) : CH1-D	Stilbene	123.5	125.2
Alkene link to peripheral ring			
CH1-(A,B,C) : CH1-D : H1	Stilbene	119.0	118.9
CH1-(A,B,C) : CH1-D : CA-A	Stilbene	123.5	125.2
H1 : CH1-(A,B,C) : CH1-D	Stilbene	119.0	118.9
H1 : CH1-D : CA-A	Stilbene	119.5	115.9
Peripheral ring to oxygen			
CH1-D : CA-A : CA-B	Stilbene	123.5	122.4
CH1-D : CA-A : CA-C	Stilbene	123.5	117.2
CA : CA : CA	DEMB	120.0	120.0
CA : CA : HA	DEMB	120.0	120.0
CA-(C,D) : CA-(C,B) : O	DEMB	120.0	124.2
CA-(B,D) : CA-(B,C) : O	DEMB	120.0	117.8
CA-(B,C) : O : CH2	DEMB	108.0	117.4
Oxygen to alkyl chain			
O : CH2 : CH2	DEMB	109.6	106.8
O : CH2 : H2	DEMB	109.5	110.0
CH2 : CH2 : CH2	Alkane	113.6	113.4
CH2 : CH2 : H2	Alkane	110.0	109.4
H2 : CH2 : H2	Alkane	109.0	106.4
CH2 : CH2 : CH3	Alkane	115.0	112.9
H2 : CH2 : CH3	Alkane	110.1	109.6
CH2 : CH3 : H3	Alkane	110.1	111.0
H3 : CH3 : H3	Alkane	107.5	107.8

The second region, also under **Table 2.4**, "**Peripheral ring to oxygen**", contains the angles "CA-(C,D) : CA-(C,B) : O" and "CA-(B,D) : CA-(B,C) : O". These also have the same base atom types, but the parenthesized sub-types distinguish the significantly different calculated bond angles (124.2° and 117.8°) from the given CGenFF literature value of 120.0°.

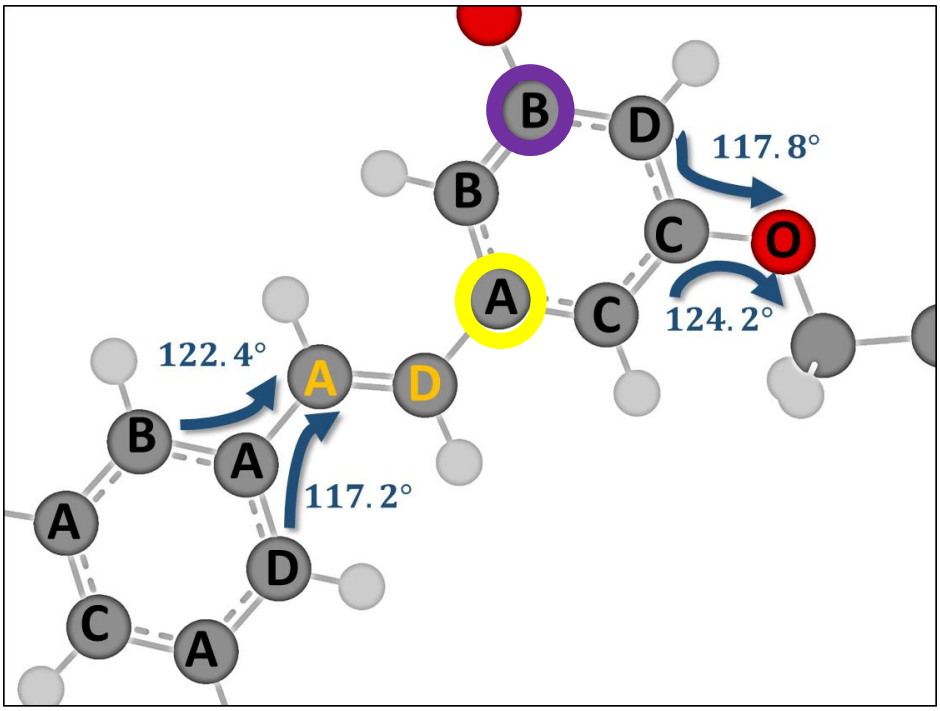


Figure 2.8 *Motivation for creating several carbon atom types, based on calculated geometric parameters of the molecule. Steric hindrance causes the angles to not be symmetric about the vertex carbon atom, where the angles are indicated.*

In piecing together the molecular fragments to build the full TSB35 molecule, we took care to retain the electronic and geometric information that we calculated in Gaussian09 by meticulously comparing and contrasting bond lengths and bond angles. This was done between each fragment that contained electronically or geometrically similar environments, and also with the CGenFF database in order to maximize the self-consistency of our model. **Appendix B** lists the remaining bond length, bond angle, and dihedral angle parameters, such as force constants and Lennard-Jones parameters. Note that all dihedral angles are set to 0.0° or 180.0°, so that we follow the adsorbed configuration of the TSB35 molecule [14].

We also carefully assign Mulliken partial charges in the full TSB35 molecule. To illustrate this, see **Fig. 2.9**, where the atom pairs in the black boxes are assigned the same equilibrium bond lengths and bond angles, but the calculated Mulliken charge may differ significantly. In the top panel, the electronic environment is very similar, and the assigned Mulliken charge can come from either the stilbene of the DEMB. In the bottom

panel, the electronic environment is very different, and the assigned Mulliken charges will be chosen from only one of the fragments. For the peripheral ring, the Mulliken charge assignment for the aromatic carbon atom that is adjacent to the oxygen atom (CA-B in the purple circle in **Fig. 2.9**) clearly must come from the DEMB fragment. An aromatic carbon closer to the center (CA-A in the yellow circle in **Fig. 2.9**) may have its Mulliken charge assigned from either stilbene or DEMB (results differ by less than 1%).

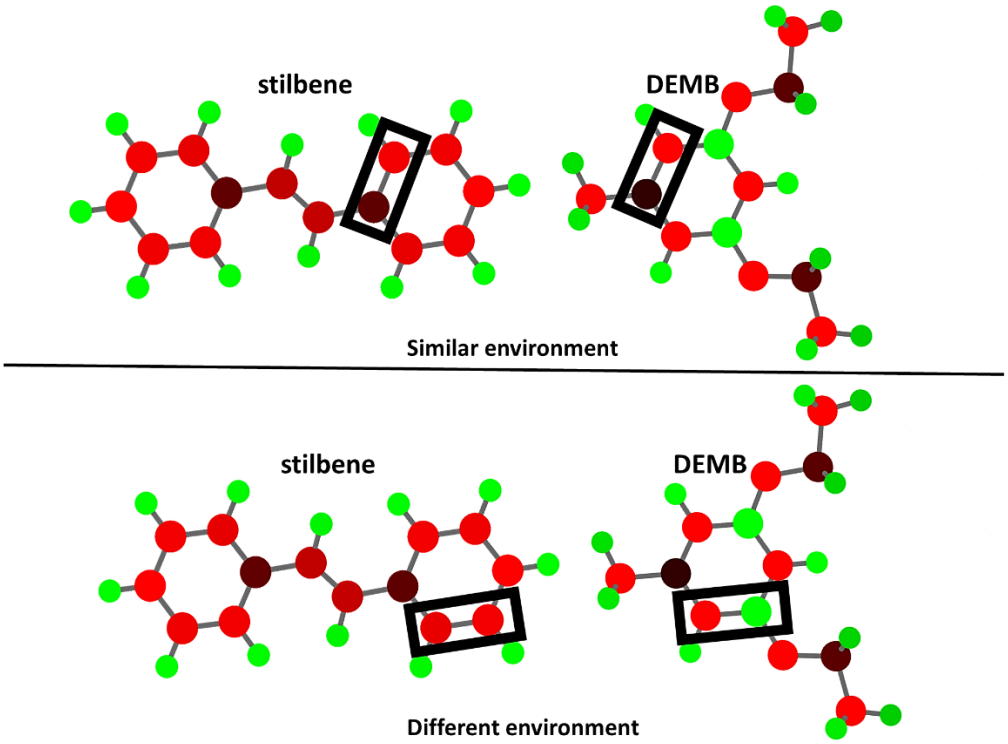


Figure 2.9 *(Top) A pair of aromatic carbons in the black boxes that have similiar electronic environments. (Bottom) A pair of aromatic carbons in the black boxes that have significantly different electronic environments, prompting us to take much care in building the full TSB35 molecule and to differentiate atom types when assigning Mulliken charge. Charges were taken from the structures that more closely resemble the actual TSB35 molecule. See text.*

The atom sub-type naming scheme and the Mulliken charge assignment scheme do not conflict, since atom typing and bonding order is extrapolated from the TOP and PDB files, and the Mulliken charges are assigned to atoms individually in the TOP file (**Appendix A**). Although it is possible to have each individual atom have an individual atom type this becomes overly cumbersome, and may result in "overfitting" system pa-

rameters from experimental observations. The computational physicist should strive to use the simplest and least biased model that is consistent with the system under consideration. In the end, we utilize ten different Mulliken charge assignments for the carbon atoms, and five different Mulliken charge assignments for the hydrogen atoms in the TSB35 molecule. All charge assignments are listed in **Table 2.5**, and the numeric labels in the table correspond with **Fig. 2.10**.

Table 2.5 *Calculated Mulliken charges.*

Atom type	Label (see figure)	$q\ (esu)$
Aromatic carbon:	1	0.00
	2	-0.15
	3	-0.22
	4	0.45
	5	-0.30
Aromatic hydrogen:	6	0.15
	7	0.16
Aliphatic carbon:	8	-0.13
	9	0.10
	10	-0.20
	11	-0.21
	12	-0.33
Aliphatic hydrogen:	13	0.14
	14	0.11
Oxygen:	15	-0.69
Graphitic carbon:	-	0.00

The kind of procedure shown in **Fig. 2.9** was followed for each distinct atom grouping (e.g. alkene link, ether link, terminal alkane, et cetera) to ensure that we retain accurate electronic and geometric behavior in the full TSB35 molecule. The central ring and alkene links are taken from stilbene, the peripheral rings and oxygen links (including the aliphatic carbon adjacent to oxygen) are taken from DEMB, and the rest of the alkoxy chain is from hexane (or decane for TSB3,5-C10). **Figure 2.11** shows the full TSB3,5-C6 molecule pieced together, and this is the structure that is represented in the CGenFF parameters, which are all contained in the PARAMS file for the simulation. See **Chapter 3**,

and **Appendix A** for an example excerpt of the PARAMS file, and **Appendix B** for all of the included values utilized in our simulations.

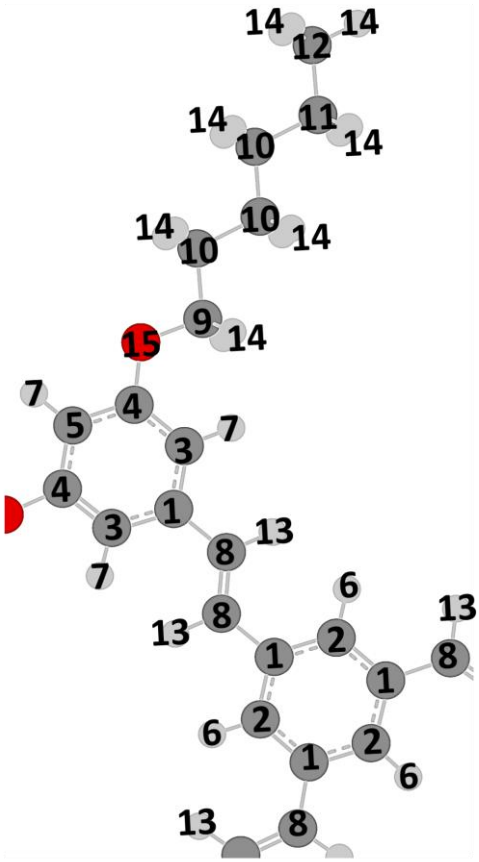


Figure 2.10 *Numerical labels indicating the ten different Mulliken charge assignments for carbon atoms, and five different assignments for hydrogen atoms, based on purely electronic considerations.*

Note that the chirality along the periphery of TSB35 molecule (blues arrows on the oxygen atoms in **Fig. 2.11**) required that we manually changed one of the dihedrals across the oxygen atom in the DEMB fragment. The optimization of DEMB minimized steric hindrance by bringing the molecule to the configuration shown in **Fig. 2.5** and **Fig. 2.9**. The desired configuration of the TSB35 molecule, according to experiment [12,13,14], has the chirality of the peripheral alkoxy chains as shown in **Fig. 2.4** and **Fig. 2.11** (blue arrows).

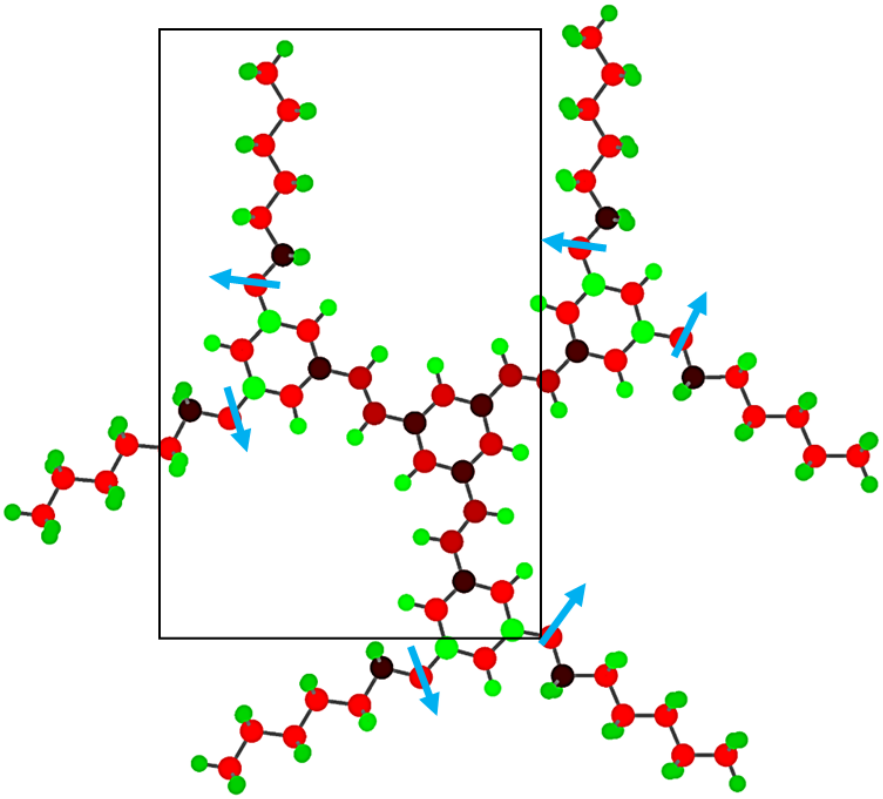


Figure 2.11 *The full TSB3,5-C6 molecule pieced together from the optimized fragments, colored according to assigned Mulliken charge (green = positive, black = neutral, red = negative). See* ***Table 2.5*** *with* ***Figure 2.11*** *for values. The black box corresponds to the region framed in* ***Fig. 2.11****, and the blue arrows indicate the chirality of the molecule.*

Chapter 3

Molecular Dynamics Simulation

I. Introduction to Molecular Dynamics Simulations

The purpose of Molecular Dynamics (MD) simulation is to create computational models to replicate complex systems that cannot be solved analytically ("laboratory conditions"), produce trajectories (e.g., solve the time-dependence of the system), and collect data for further processing (e.g., equilibrium properties such as average energy and heat capacity, or non-equilibrium properties such as diffusion constants, etc.).

In our work we utilize classical MD, where we assign follow the positions and velocities of the atomic nuclei ("atoms") of a fixed number of molecules, and solve Newton's equations of motion:

$$\frac{d^2\boldsymbol{r_i}}{dt^2} = \frac{\boldsymbol{F_i}(\boldsymbol{r_i})}{m_i} = -\frac{\mathbf{1}}{m_i}\boldsymbol{\nabla}_{\boldsymbol{r_i}} U(\boldsymbol{r_i}), \tag{3}$$

where $\boldsymbol{r_i}$ is the Cartesian position vector of the ith atom of mass m_i, $\boldsymbol{F}_i(\boldsymbol{r}_i)$ is its force acting on that atom, and $U(\boldsymbol{r}_i)$ the associated potential energy, including both the bonded and nonbonded input simulation parameters. The potential energy $U(\boldsymbol{r}_i)$ may be the result of *ab initio* calculations of the PES (**Chapter 2**), or fittings to experimental results ("empirical force fields"), or a combination of the two ("semi-empirical methods"). **Figure 3.1** shows a generic workflow of an MD simulation.

Normally the intramolecular and intermolecular forces present in the system, are calculated at each time step (since the calculation of the forces is usually the most time consuming part of a MD simulation, in some MD schemes, forces are calculated less frequently; in our simulations we recalculate all forces at each time step). We employ the CHARMM27 force field and CGenFF database [38,39]. The potential energy surface $U(\boldsymbol{r}_i)$ is, to a very good approximation separable into bonded and nonbonded terms:

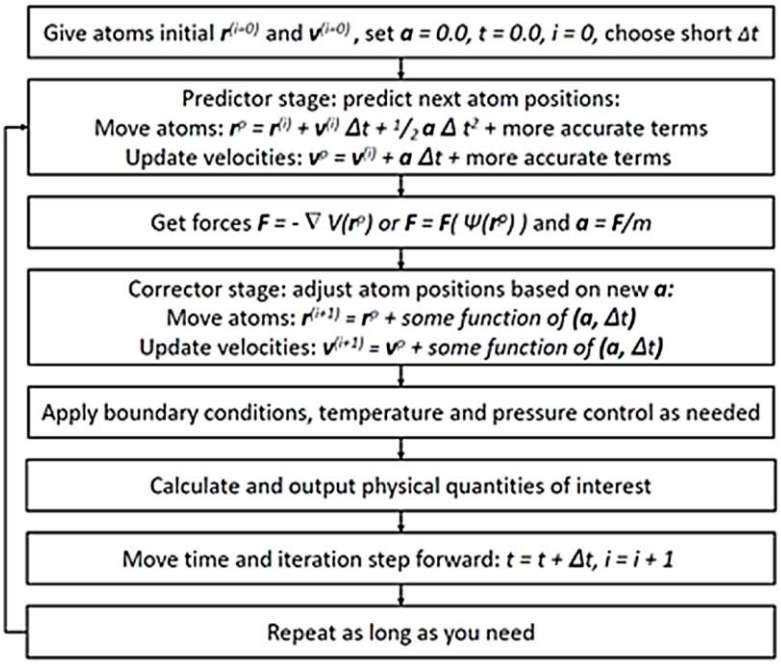


Figure 3.1 Algorithmic scheme of a molecular dynamics simulation.

$$U = U_{bonded} + U_{nonbonded} = (U_{bond} + U_{angle} + U_{UB} + U_{dihedral}) + (U_{LJ} + U_{Coulomb}). \quad (4)$$

The bonded internal molecular degrees of freedom considered are the two-body stretch (bond), the three-body bend (angle and Urey-Bradley), and the four-body torsion (dihedral), with their potential energy term listed below, respectively [38]:

$$U_{bond} = k_b(b - b_0)^2, \quad (5)$$

$$U_{angle} = k_\theta(\theta - \theta_0)^2, \quad (5)$$

$$U_{Urey-Bradley} = k_{UB}(s - s_0)^2, \quad (7)$$

$$U_{dihedral} = k_\chi(1 + \cos(n\chi - \delta)). \quad (8)$$

In our simulations we obtain the geometric parameters b_0, θ_0, s_0, and δ directly from the *ab initio* calculations (**Chapter 2**); values for the multiple sets of parameters are

listed **Appendix B**. The force constants k_b $(kcal\,mol^{-1}\text{Å}^{-2})$, $k_\theta(kcal\,mol^{-1}rad^{-2})$, k_{UB} $(kcal\,mol^{-1}\text{Å}^{-2}$, k_χ $(kcal/mol)$, and the dihedral multiplicity n (integer greater than or equal to 1) are taken from the CGenFF database [38,39], and are also listed in **Appendix B**. Urey-Bradley terms are not defined for all possible trios of atoms in the molecule; they are included for the peripheral alkyl chains (w/o oxygen) by legacy in the force field.

Non-bonded interactions included are of the Lennard-Jones 12-6 form, potential and the Coulomb potential and are included for all intermolecular forces, and for intramolecular forces (with scaled 1-4 atom pair interaction [38,39]):

$$U_{LJ} = \epsilon_{ij}\left[\left(\frac{r_{min,ij}}{r_{ij}}\right)^{12} - 2\left(\frac{r_{min,ij}}{r_{ij}}\right)^{6}\right], \tag{9}$$

$$U_{Coulomb} = \frac{kq_iq_j}{r_{ij}}. \tag{10}$$

The Lennard-Jones potential energy parameters ϵ_{ij} and $r_{min,ij}$ are taken from the CGenFF database [39], and are listed in the **Appendix B**. Mulliken charges q_i are from **Table 2.5**. Lorentz-Berthelot mixing rules are employed for U_{LJ}:

$$\epsilon_{ij} = \sqrt{\epsilon_i\epsilon_j}\,, \tag{11}$$

$$r_{min,ij} = (r_{min,i} + r_{min,j})/2. \tag{12}$$

Each atom type, and sub-type, are assigned force field parameters, and the molecular dynamics simulation reads all of this information from the input files. See **Appendices A**, **C** for details on building the input files, and how they are all related to each other.

II. Computational Cell: PBC and Commensurability

An example of the initial, pre-minimization configuration of the TSB35 monolayer, built from 32 optimized TSB3,5-C6 molecules adsorbed onto bulk graphite (comprised of six graphene layers in an A-B-A-B-A-B configuration), using the Groszek-type model for adsorption [40], and following other literature on molecular dynamics simulation of adsorption [41,42,43,44], is presented in **Fig. 3.2**. Each TSB3,5-C6 molecule contains 168 atoms, and each layer of graphene contains 5911 atoms. In general, a periodic adlayer (TSB35) may or may not be commensurate to the substrate. However, because we employ periodic boundary conditions (PBC) in our simulations (to avoid problems with edges and better simulate a macroscopic sample), this requires the PBC cell to be an *integer multiple of both* the adlayer substrate lattices, which in turn forces (perhaps artificially) the two lattices to be commensurate.

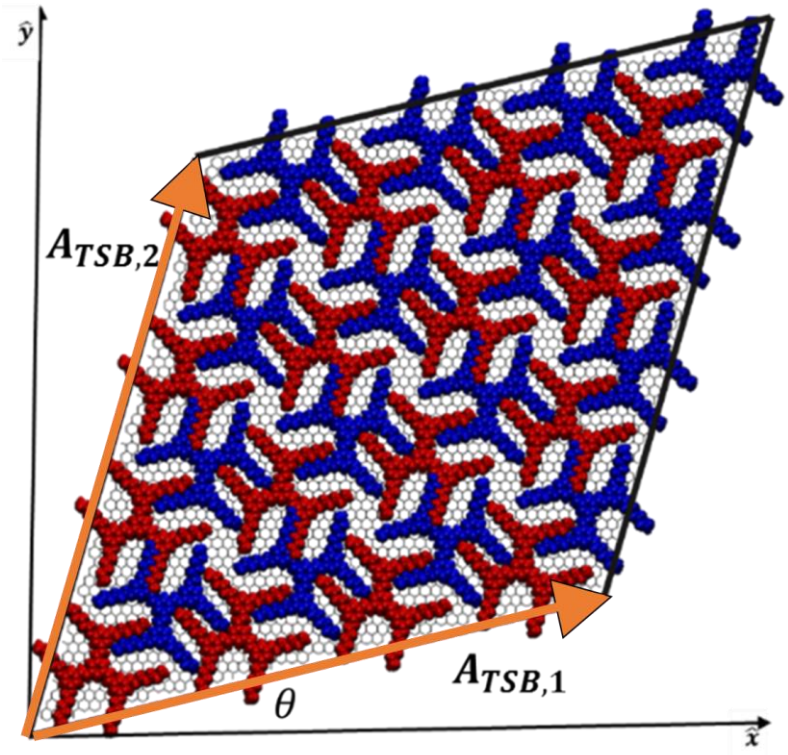


Figure 3.2 *The computational cell (4x4 unit cells) for the TSB3,5-C6 monolayer, pre-minimization and equilibration, showing the periodic boundary condition vectors (orange),* $A_{TSB,1}$ *and* $A_{TSB,2}$*, and the domain angle θ formed between both the PBC and adlayer lattices with the substrate lattice. Blue TSB35 are* 60.0° *rotated from the red TSB35 molecules, which allows the honeycomb lattice to form.*

Note that the relative orientation and displacement of the red and blue TSB35 molecules allows for the formation of the honeycomb TSB structure. The computational cell ($\boldsymbol{A}_{TSB,1}$, $\boldsymbol{A}_{TS,B,2}$) was constructed using the TSB lattice vectors: $\boldsymbol{A}_{TSB,i} = 4 * \boldsymbol{a}_{TSB,i}$. The domain angle θ, taken from the literature [14], in **Fig. 3.2** measures the angle between the horizontal axis of graphene (x-axis in **Fig. 3.2**), and the TSB35 super-lattice vector $\boldsymbol{a_{TSB,1}}$. To relate the information of the domain angle to the TSB35 lattice vectors, we write them in terms of the graphene lattice vectors $\boldsymbol{a}_1 = 1.42\ \text{Å}\ \sqrt{3}\ \hat{\boldsymbol{x}}$, $\boldsymbol{a_2} = 1.42\ \text{Å}\ \sqrt{3}\ (\frac{\hat{x}+\sqrt{3}\hat{y}}{2})$:

$$\boldsymbol{a_{TSB,1}} = n_1\boldsymbol{a_1} + n_2\boldsymbol{a_2}\ , \tag{13}$$

$$\boldsymbol{a_{TSB,2}} = n_3\boldsymbol{a_1} + n_4\boldsymbol{a_2}\ , \tag{14}$$

$$\begin{pmatrix}\boldsymbol{a}_{TSB,1}\\ \boldsymbol{a}_{TSB,2}\end{pmatrix} = \begin{pmatrix} n_1 & n_2 \\ -n_2 & n_1+n_2\end{pmatrix}\begin{pmatrix}\boldsymbol{a}_1\\ \boldsymbol{a}_2\end{pmatrix}. \tag{15}$$

The coefficients in the expression for $\boldsymbol{a_{TSB,2}}$ in **Eqn. 15** are written in terms of n_1 and n_2, which are rational numbers (commensurability condition) $n_i = p_i/q_i$ with $p_i, q_i \in \mathbb{Z}$ (and for the scenario of **Fig. 3.2**, $q_i = 4$ as $\boldsymbol{A}_{TSB,i} = 4 * \boldsymbol{a}_{TSB,i}$). The last equation (**Eq. 15**) includes conditions ensuring that the two lattice vectors form a triangular lattice (angle of 60.0° with respect to each other). The coefficients were analytically solved for using the dot product of $\boldsymbol{a_{TSB,1}}$ and $\boldsymbol{a_{TSB,2}}$ with $\boldsymbol{a}_1$, which yields two equations in two unknowns, namely n_1 and n_2:

$$\boldsymbol{a}_{TSB,1} \cdot \boldsymbol{a}_1 = a_{TSB}a\cos(\theta), \tag{16}$$

$$\boldsymbol{a}_{TSB,2} \cdot \boldsymbol{a}_1 = a_{TSB}a\cos(\theta + 60°), \tag{17}$$

that are solved for n_i subject to the commensurability requirement. **Table 2.6** shows calculated integer coefficients of each alkoxy chain length (n = 6, 8, 10, 12, 14), and the subsequently calculated values for the lattice spacing parameter and domain angle. The calculated values are within the defined error in the "Experimental Vales" column [14]. With the magnitude and direction of the TSB35 lattice vector, we calculate the super-

lattice vector dimensions, listed in **Table 2.7** for both of the TSB3,5-C6 and TSB3,5-C10 monolayers.

Table 3.1 *Lattice parameter and domain angle with calculated integer coefficients for each alkoxy chain length n.*

	Experimental Values[a]		Calculated Values		Calculated Integer Coefficients			
n	a_{TSB} (Å)	θ (°)	a_{TSB} (Å)	θ (°)	n_1	n_2	n_3	n_4
6	31 ± 2	11.8 ± 0.2	31.4	11.7	11	3	-3	14
8	34 ± 2	7.1 ± 0.2	34.7	7.1	13	2	-2	15
10	38 ± 2	3.2 ± 0.2	38.2	3.2	15	1	-1	16
12	41 ± 2	0.0 ± 0.4	41.8	0.0	17	0	0	17
14	45 ± 2	2.7 ± 0.0	45.6	2.7	17	2	-2	19

[a] Ref. [14]

Table 3.2 *Super-lattice vector dimensions for n = 6 and n = 10.*

Super-lattice vector	TSB3,5-C6	TSB3,5-C10
$\boldsymbol{A_{TSB,1}}$	(122.98 Å, 25.56 Å)	(152.49 Å, 8.52 Å)
$\boldsymbol{A_{TSB,2}}$	(39.35 Å, 119.28 Å)	(68.87 Å, 136.32 Å)

The program used to build the computational cells for the TSB3,5-C6 and TSB3,5-C10 monolayer systems, written in C++, is outlined below. See **Appendix C** for more details on compilation and implementation of the scripts, as well as full examples for creating the coordinate (PDB) and topology (TOP) files.

1. Input nuclear coordinates and Mulliken charges of a single TSB35 molecule (results from *ab initio* calculation, **Chapter 2**).
2. Create class and assign membership for each atom.
3. Define TSB35 lattice vectors.
4. Appropriately rotate and translate the molecule such that the center of mass lies above the first hexagonal cell of the graphite substrate.
5. Loop through the desired number of molecules in each direction of the TSB35 lattice vectors.
6. Write positions and other format-dependent information to the coordinate file.
7. Rotate molecule (instance of class) by 60 degrees.
8. Translate rotated molecule to the coordinates defined by the triangular lattice

basis vector.

9. Write positions and other format-dependent information to the coordinate file.

To write the coordinates for the graphite substrate, we have to first calculate the bounding box of the computational cell, as defined by the TSB35 super-lattice vectors, and account for the *A-B stacking* for alternating layers of graphene:

1. Calculate nuclear coordinates;
    a. if the coordinates lie within the bounding box, then increase counter,
    b. if not, continue.
2. Create class and assign membership for each atom per layer of graphene that lies within the bounding box.
3. Instantiate a flag to indicate when a potential graphite atom is very close to the bounding box, and may become missing.
4. Add any missing atoms.
5. Loop through number of atoms per layer.
6. Write positions and other format-dependent information to the coordinate file for the odd-numbered layers.
7. Shift entire layer for the *A-B stacking.*
8. Write positions and other format-dependent information to the coordinate file for the even-numbered layers.

Steps 1 through 4 require the script to find all of the carbon atoms for the graphene layers that lie within a bounding box with non-trivial geometry, as seen in **Fig. 3.2**. The subroutine for this is shown in **Listing 3.1**. Note that the *double* variables defined within the *for* loops and the following conditionals form the rhomboidal computational cell with the domain angle relative to the horizontal axis. The program can flag any atoms that may become missing due to proximity of the bounding box edges and rounding errors.

```
int countGrapheneAtoms (struct vect a1, struct vect a2, struct vect a3, struct vect b1, struct vect b2, struct vect
b3)
{
struct vect B1, B2, B12, C1, C2, C12;
// Define TSB super-lattice vectors
B1 = 4.0*b1;
B2 = 4.0*b2;
B12 = B1 + B2;
// Define boundaries of TSB super-lattice vectors (purely for calculation)
C1 = B1 + 2.0*b3;
C2 = B2 + 2.0*b3;
C12 = B12 + 2.0*b3;
double slopeC1 = (C1.y - 2.0*b3.y) / (C1.x - 2.0*b3.x);
double slopeC2 = (C2.y - 2.0*b3.y) / (C2.x - 2.0*b3.x);
double yInterceptC1 = C1.y - slopeC1*C1.x;
double yInterceptC2 = C2.y - slopeC2*C2.x;
double yIntercept1C12 = C12.y - slopeC1*C12.x;
double yIntercept2C12 = C12.y - slopeC2*C12.x;

// Initialize graphene basis vectors, where "1" and "2" denote basis atoms of the graphene unit cell
struct vect positionG1 = 2.0*b3;
struct vect positionG2 = 2.0*b3;
// Initialize a sufficiently large grid to scan for atoms inside the boundaries
int maxUnitCells1 = 500;
int maxUnitCells2 = 500;
int nAtomsPerLayer = 0;

// Calculate atomic positions, counting nAtomsPerLayer
for (int i=-maxUnitCells1; i<maxUnitCells1; i++)
{
        for (int j=-maxUnitCells2; j<maxUnitCells2; j++)
        {
                positionG1 = i*a1 + j*a2 + a3;
                positionG2 = i*a1 + j*a2 + 2.0*a3;

                double xLowerLimitG1 = (1.0/slopeC2)*( positionG1.y - yInterceptC2 );
                double yLowerLimitG1 = slopeC1*positionG1.x + yInterceptC1;
                double xUpperLimitG1 = (1.0/slopeC2)*( positionG1.y - yIntercept2C12);
                double yUpperLimitG1 = slopeC1*positionG1.x + yIntercept1C12;
                double xLowerLimitG2 = (1.0/slopeC2)*( positionG2.y - yInterceptC2 );
                double yLowerLimitG2 = slopeC1*positionG2.x + yInterceptC1;
                double xUpperLimitG2 = (1.0/slopeC2)*( positionG2.y - yIntercept2C12);
                double yUpperLimitG2 = slopeC1*positionG2.x + yIntercept1C12;

                if ( positionG1.x >= xLowerLimitG1 && positionG1.x <= xUpperLimitG1 )
                {       if ( positionG1.y >= yLowerLimitG1 && positionG1.y <= yUpperLimitG1 )
                        {       nAtomsPerLayer++;}}

                if ( positionG2.x >= xLowerLimitG2 && positionG2.x <= xUpperLimitG2 )
                {       if ( positionG2.y >= yLowerLimitG2 && positionG2.y <= yUpperLimitG2 )
                        {       nAtomsPerLayer++;}}
        }
}

// Flag for possible missing atoms, that are very close to the bonding box
// nAtomsPerLayer++;

return nAtomsPerLayer;}
```

***Listing 3.1**. Subroutine that creates atom layers.*

Now, we need to give the simulation information about the Mulliken charges and connectivity of the atom. See **Appendix A** for an example excerpt of a TOP file.

1. Input nuclear coordinates and Mulliken charges of a single TSB35 molecule.
2. Create class and assign membership for each TSB35 atom.
3. Determine bonding order for a single TSB35 molecule.
4. Define TSB35 lattice vectors.
5. Create class and assign membership for each atom per layer of graphene that lies within the bounding box.
6. Determine bonding order for a layer of graphene.
7. Write Mulliken charges, bonding order, and other format-dependent information to the topology file.

To begin a simulation, all the files listed in **Fig. 3.3** are needed (this is a chart of the file structure for NAMD2 [46]). This includes TOP, PDB, PARAMS, and CONF files. Thus far, we have the PARAMS file from the *ab initio* calculations, and the TOP and PDB files from the C++ scripts described above. These are all read into *psfgen* [47], which creates simulation-ready PDB and PSF files, extrapolating the split PDB and TOP files to define the entire system.

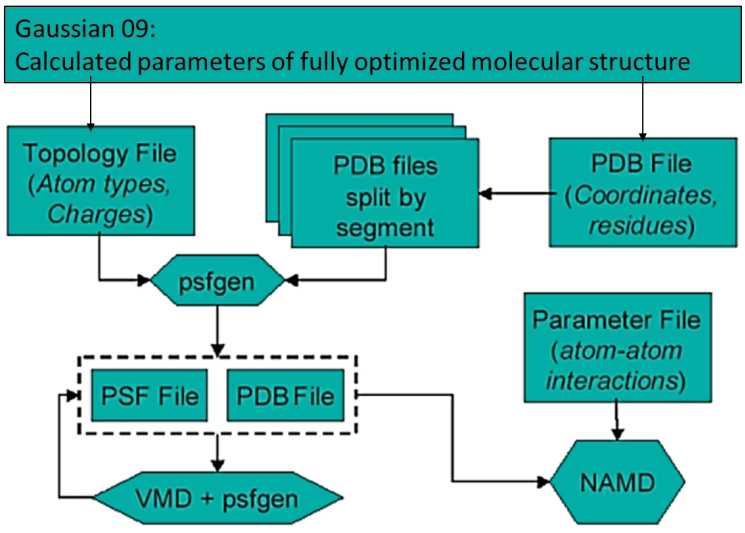


Figure 3.3 *Diagram showing the file and software dependencies required for setting up a NAMD2 simulation, after Ref. [46].*

III. Simulation Setup and Details

The last component required to start a NAMD2 [46] simulation is the CONF file. The CONF file is used to connect all of the other setup files, and contains required and optional parameters to run the molecular dynamics simulation. See **Appendix** C for examples of CONF files, and **Table 2.8** for the options chosen for our simulations.

Table 3.3 *NAMD2 configuration file options.*

timestep	1.0	cutoff	10.0	temperature	300
stepspercycle	4	switchdist	8.0	rescaleFreq	10
wrapAll	On	pairlistdist	12.0	outputEnergies	1000
wrapNearest	On			DCDfreq	1000
fixedAtoms	Graphite	exclude	Scaled1-4	minimize	20000
rigidBonds	all	1-4scaling	0.4	run	12 x 10^6

- timestep is in femtoseconds (10^{-15}s).
- stepspercycle is the interval at which neighbors are assigned.
- wrapAll and wrapNearest apply periodic boundary conditions.
- fixedAtoms looks for a PDB defining which atoms are fixed in simulation.
- rigidBonds fixes the carbon-hydrogen bond lengths.
- cutoff is the distance at which nonbonded energies are not calculated.
- switchdist is the distance where the smoothing function if applied.
- pairlistdist defines the size of the neighborhood for stepspercycle.
- exclude turns on 1-4scaling.
- 1-4scaling is a multiplicative factor to scale 1-4 interactions.
- temperature is the temperature (in Kelvin) of the simulation.
- rescaleFreq is the interval at which velocities rescale to maintain temperature.
- outputEnergies is the interval at which the ouput LOG file is written.
- DCDfreq is the interval at which the DCD trajectory file is written.
- minimize is the number to minimization steps.
- run is the number of equilibration steps.

The 1-4 scaling in the intramolecular forces (bonded interactions) excludes all 1-2 and 1-3 atom pairs from non-bonded interaction calculation, and then the 1-4 atom pairs are scaled by a user-defined scaling factor, which we set to 0.4 in our simulations [41]. The scaling is done to avoid over-estimation of the potential energy, since the dihedral potential energy will be present in 1-4 interactions. All 1-5 and higher atom pairs have both non-bonded potential energies (Lennard-Jones and Coulomb) fully accounted for in the TSB35 monolayer, up to the cutoff distance for pair separation. We use a cutoff dis-

tance r_{cutoff}, defined in simulation, for non-bonded interactions, such that any energy contribution due to a pair separation of $r_{ij} > r_{cutoff}$ is not calculated. See **Fig. 3.4** for reference.

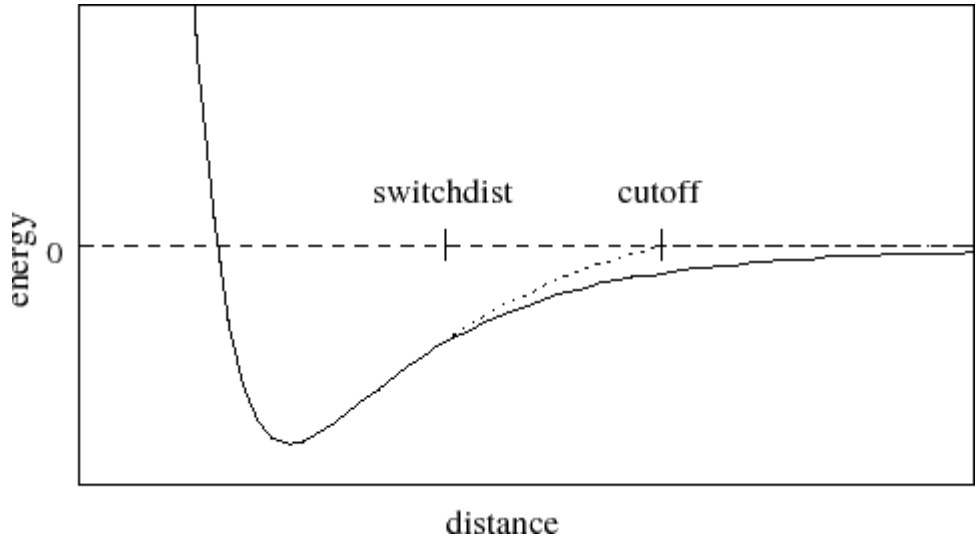


Figure 3.4 *Plot of interaction potential with switchdist and the smoothing function (dashed line) taking the nonbonded interactions to the cutoff distance.*

In our simulations we have not employed the particle mesh Ewald (PME) summation to do full electrostatic calculation in our system. NAMD2 [46] accounts for this by forcing the electrostatic potential energy to reach zero at the cutoff distance by means of an overall shift to the electrostatic potential. We choose not to use PME because our adsorbed monolayer system is effectively two-dimensional (2D), since there is zero Mulliken charge assigned to the substrate, and using PME may over-estimate the electrostatic potential energy of the system, since it is a three-dimensional calculation. The use of PME, furthermore, slows the MD simulations in a non-trivial way.

To further speed-up the calculations, we assumed the carbon atoms in the graphene layers to be fixed. In the integrator, we used a timestep of one femtosecond (1 fs) with fixed carbon-hydrogen bond lengths (internal constraint enforced by the RATTLE algorithm [48]), and ran six to twelve nanoseconds of equilibration 6-12 x 10^6 steps). The *NVT canonical* ensemble was approximated by periodically rescaling the velocities so that the average kinetic energy per degree of freedom was $k_B T/2$. We exclude any damp-

ing terms from the equations of motion, and instead use rescaling of the velocities to control the temperature (see **Table 3.3**). Prior to equilibration, minimization is used to eliminate "bad contacts" between nuclear coordinates. Choice of the number of minimization steps is trial-and-error, ranging from 1000 to 200,000 steps. Periodic boundary conditions (PBC) are implemented in each of the x, y, and z directions. The PBC in the x and y directions are defined by the super-lattice vectors from **Table 3.2**, and the z direction is taken to be large enough so that the systems do not interact in this direction (100 Å).

Chapter 3

Results and Discussion

I. Optimization of Lattice Size

We built the computational cell beginning with the experimental values for the lattice parameter and domain angle [14]. When this configuration was simulated, we observed significant disorder, and this configuration proved to be unfavorable, as there was significant compression in the monolayer and a very distorted the pore structure. **Figure 4.1** shows the compressed monolayer at 300 K after 12 ns of equilibration. The mechanism responsible lies in the peripheral alkoxy chains, where the dihedral angles cause the molecule to partially desorb from the substrate, in order to relieve compression due to a poor initial geometry for our model. The dihedral angle distribution for this section of the TSB35 molecule becomes centered around $\pm 60°$, rather than $\pm 180°$, and forms what we have called “dihedral bridges” or “gauche defect bridges” (see discussion below, and **Figs. 4.5**, **4.6**, and **4.7**).

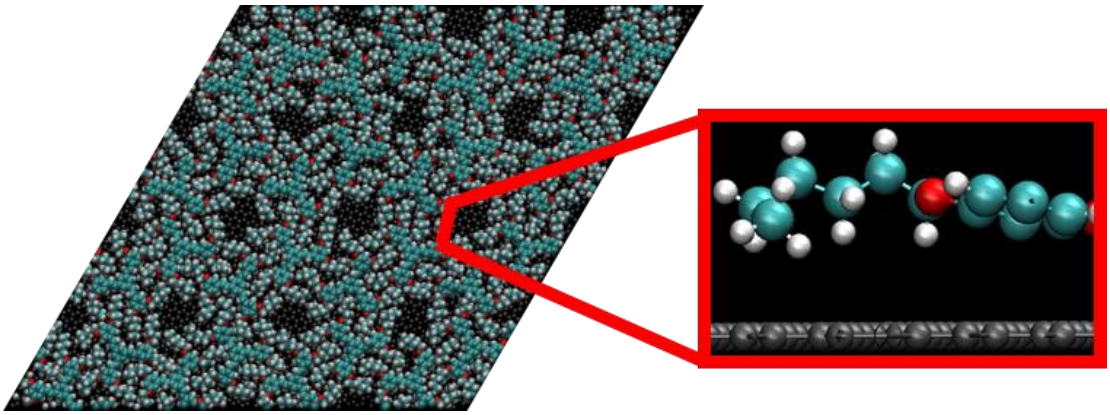


Figure 4.1 *Equilibrated TSB3,5-C6 monolayer, generated from the experimental lattice spacing parameter and domain angle, at 300 K in a compressed state, with persistent "gauche defect bridges".*

This prompted us to explore different initial lattice sizes (or TSB densities). Due to the required commensurability, the TSB35 lattice vector coefficients (n_1,n_2) satisfy (see **Chapter 3**) $n_i = p_i/q_i$, with $p_i \in \mathbb{Z}$ and $q_i = 4$. We, therefore, considered slight expansions of the TSB lattice (and corresponding expansion of the computational cell) by increasing (n_1,n_2) by quarter integers. Although in principle any quarter-integer values for (n_1,n_2) are acceptable, we considered the only the combinations (n_1,n_2) = {(11.00,3.00), (11.25,3.25), (11.50,3.50), (11.75,3.75), (12.00, 4.00)}, see **Tables 3.1** and **3.2**. The combination (n_1,n_2) = (11.25,3.25) was observed to still be overly compressed. The combination (n_1,n_2) = (11.50,3.50) "2 quarters addition" was observed to show an almost complete lattice relaxation, with a high degree of both intra- and intermolecular order, see **Fig. 4.2**. We refer to this system as being in "dynamic equilibrium", and the absence of gauche defects allows the pore structure to be highly ordered. **Table 4.1** lists the TSB35 lattice vector coefficients for the original and optimal lattice sizes, along with the calculated lattice spacing parameter and domain angle. The expansion of the TSB35 lattice vectors introduces some deviation with respect to the experimental best values for a_{TSB} and θ (see also **Table 3.1**).

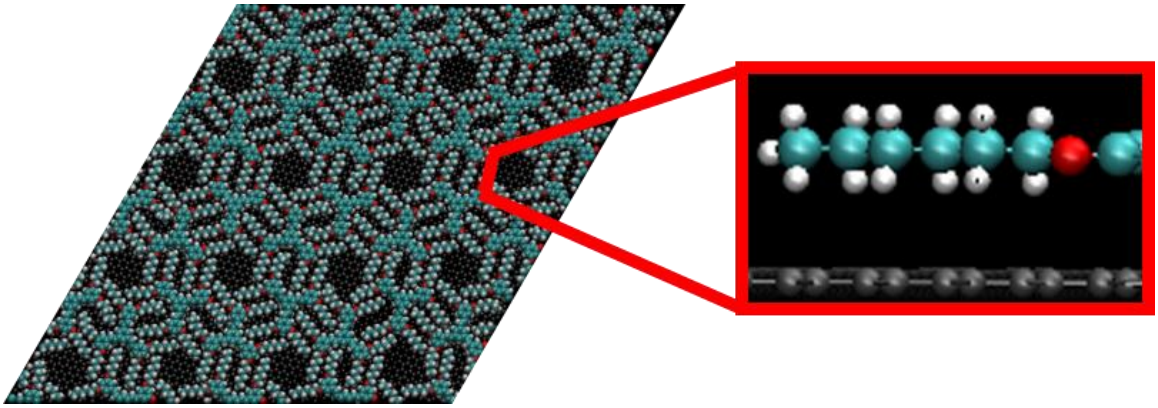


Figure 4.2 *TSB3,5-C6 monolayer at 300 K after 12 ns of equilibration with optimally expanded lattice parameters, which relieves compression in the monolayer.*

Table 4.1 *Originally calculated (compressed) lattice vector coefficients and the corresponding lattice parameter values, compared with the optimally calculated (expanded) values.*

	TSB3,5-C6		TSB3,5-C10	
	Original	Optimal	Original	Optimal
Lattice vector coeff. n_1	11.0	11.5	15.0	15.5
Lattice vector coeff. n_2	3.0	3.5	1.0	1.5
Lattice spacing a_{TSB} (Å)	31.4	33.4 (6.4%)	38.2	40.1 (5.0%)
Domain angle θ (°)	11.7	12.9 (10.3%)	3.2	4.6 (43.8%)

Table 4.2 *Originally calculated lattice vector dimensions and the optimally calculated lattice vector dimensions.*

	TSB3,5-C6		TSB3,5-C10	
	Original	Expanded	Original	Expanded
$\boldsymbol{A_{TSB,1}}$	(122.98 Å, 25.56 Å)	(130.94 Å, 27.12 Å)	(152.49 Å, 8.52 Å)	(159.87 Å, 12.78 Å)
$\boldsymbol{A_{TSB,2}}$	(39.35 Å, 119.28 Å)	(41.99 Å, 126.96 Å)	(68.87 Å, 136.32 Å)	(68.87 Å, 144.84 Å)

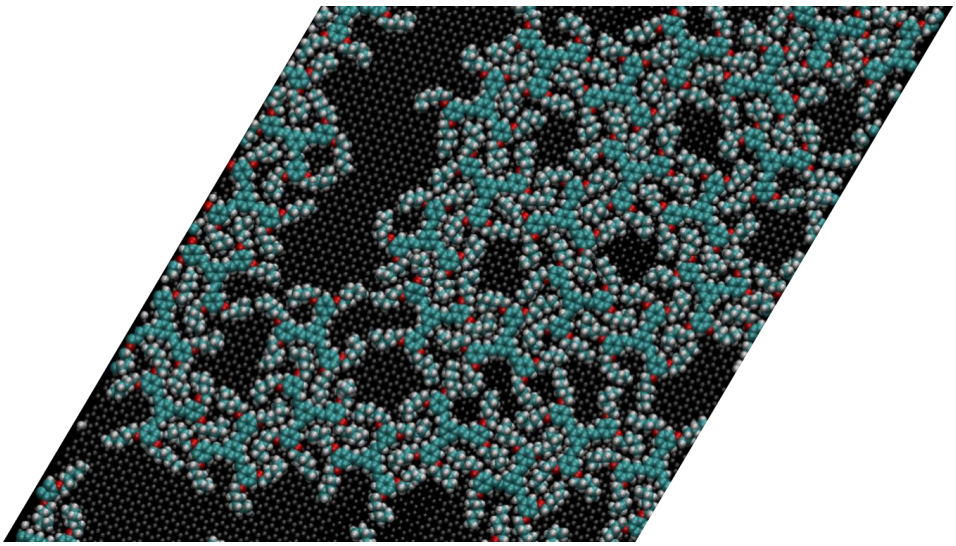


Figure 4.3 *TSB3,5-C6 monolayer at 300 K after 12 ns equilibration with four-quarter integer additons to the lattice vector coefficients. Observe the formation of high- and low-density domains. This implies that the lattice parameters chosen are larger than necessary.*

After finding the "dynamic equilibrium" of the monolayer, we added three and four quarter-integers to the coefficients, and created a low density configuration for the

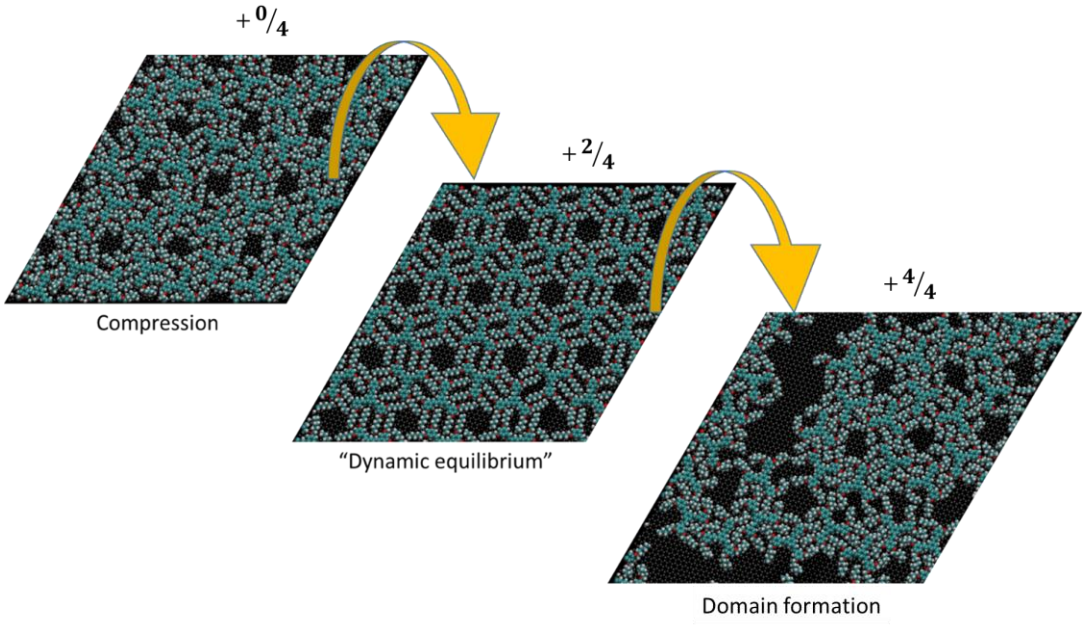
monolayer, shown below in **Fig. 4.3**. This caused domain formation in the monolayer, where there is coexistence of low- and high-density regions, such that the high-density region forms compressed pores, An interesting observation is that the dihedral angles lying on the edge of the high-density domains do not adsorb very strongly to the substrate, as it seems they should be able to do at 300 K. There are still numerous gauche defects present in the peripheral alkoxy chains on the edges of the high-density domain, where the lack of steric hindrance facilitates the appearance of these molecular defects.

A scheme of the progression through zero-, two-, and four-quarter integer additions is shown in **Fig. 4.4**. Lateral views for each one are included to show the departure from the monolayer structure.

The same procedure was followed for the TSB3,5-C10 molecule, and the same observations were made: the two quarter-integer addition to the TSB3,5-C10 lattice vector coefficients yielded the optimal lattice size and formed a reliable and uniform pore structure. At present, we cannot ascertain whether the systematic deviation from experimental values is due to the reduced system size or a small error in the optimal structure and interaction parameters of the TSB35 molecules.

Using the energy contributions of the dihedral and van der Waals potential energies, we verified that the optimal lattice expansion occurs at “two quarter” additions. See **Fig. 4.5** for the average dihedral and van der Waals potential energies of the TSB3,5-C6 monolayer interpolated as a function of the number of quarter-integers added to the coefficients of the lattice vectors.

The dihedral distribution may also be used to characterize the stability of the monolayer. In **Fig. 4.6**, the relevant dihedrals of the peripheral alkoxy chains are labeled for ease of viewing the following dihedral distributions in **Fig. 4.7**. These six dihedrals in the alkoxy chain are the majority held responsible for disorder in the TSB35 monolayer.



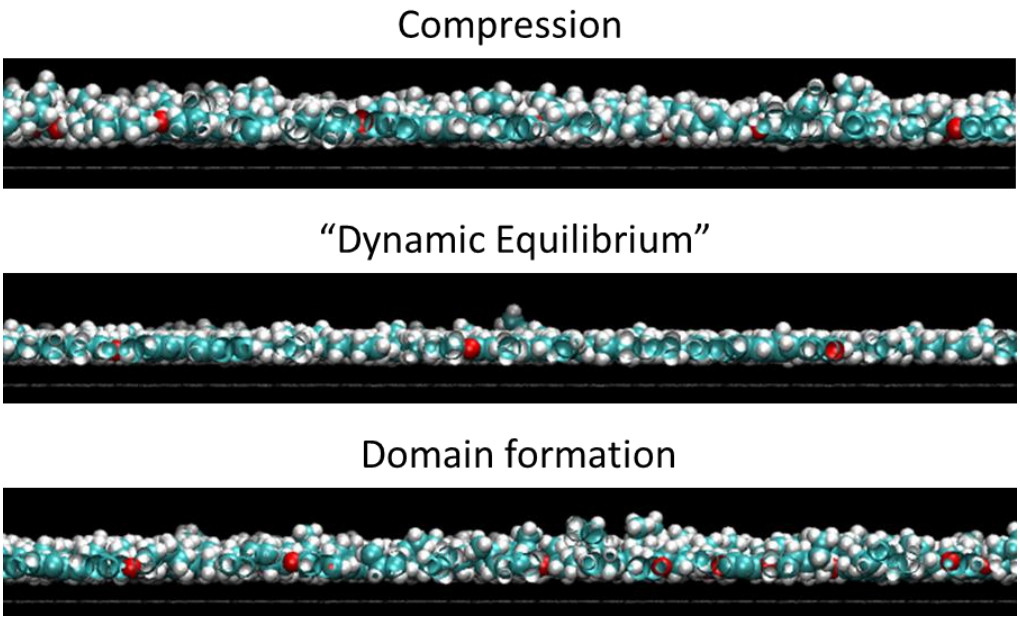


Figure 4.4 *Adding quarter-integers to the TSB35 lattice vector coefficients in order to relieve compression in the TSB3,5-C6 monolayer uncovers the formation of low- and high-density regions when the monolayer is expanded too far. All snapshots of the computational cell are shown after 12 ns of equilibration.*

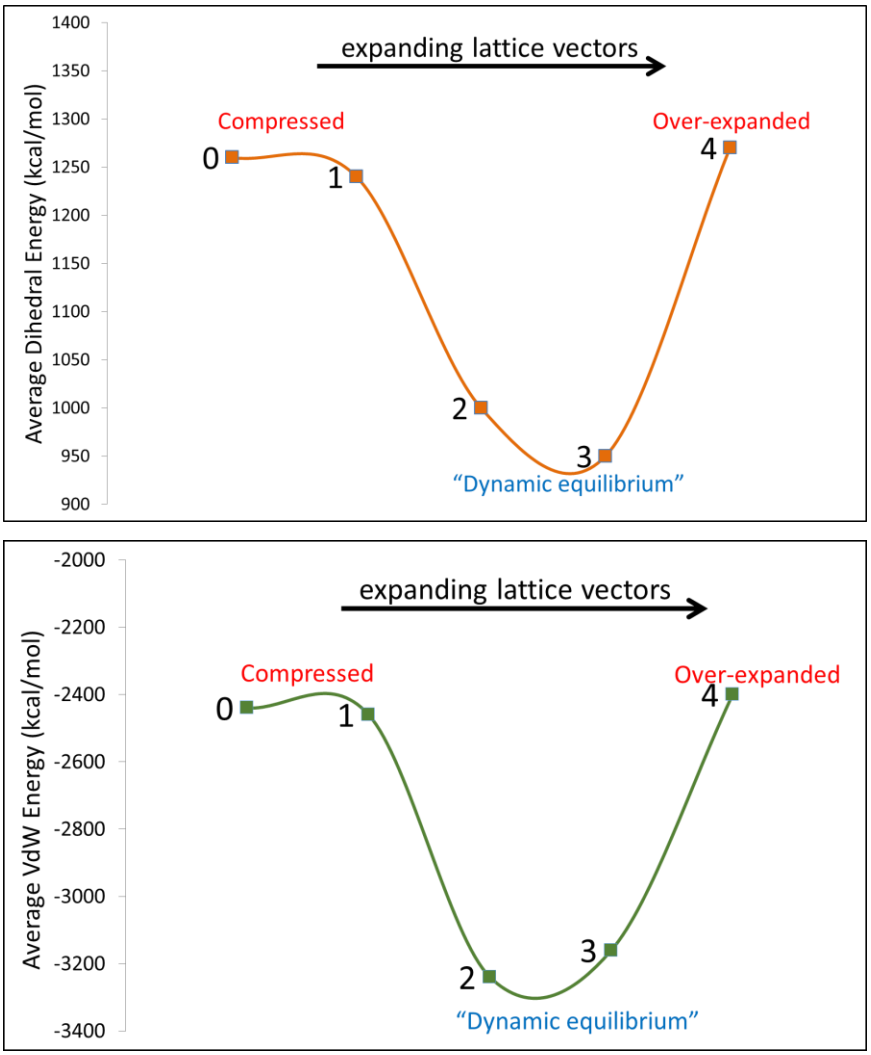


Figure 4.5 *Dihedral and van der Waals energy at each of the quarter-integer additions to the coefficients of the TSB35 lattice vectors, showing that two quarter-integers yields the most stable density for the monolayer.*

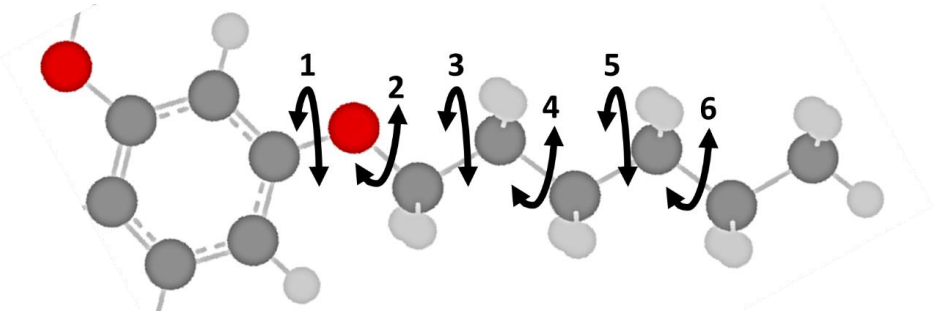


Figure 4.6 *Relevant dihedrals numerically labeled along the peripheral alkoxy chain. These dihedral angles are measured and used to quantify order in the monolayer.*

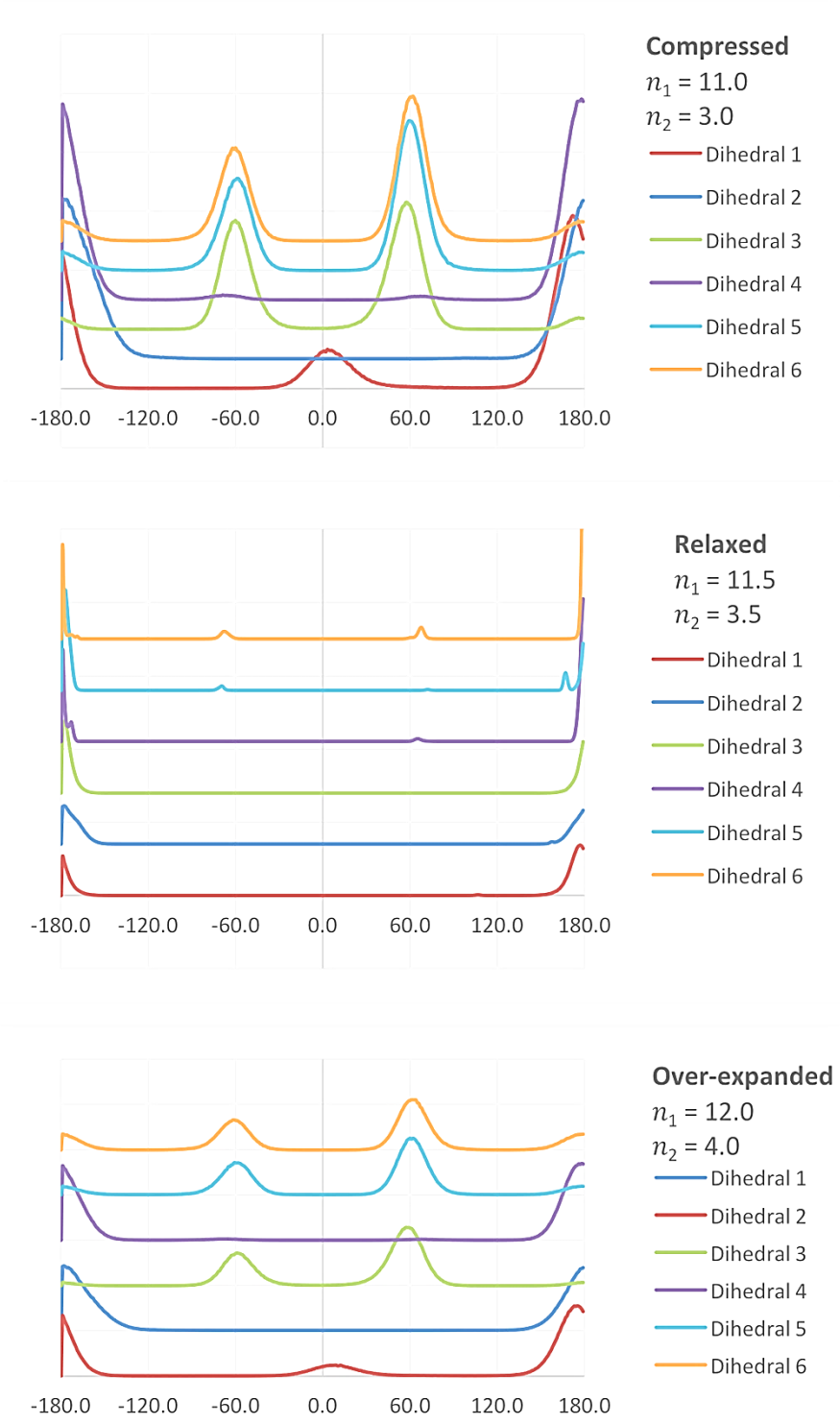


Figure 4.7 *Dihedral distributions show how the two quarter-integer additions receives the name of "Relaxed" monolayer, since the spreading pressure is zero. The "Compressed" monolayer's spreading pressure causes many gauche defects to form, and the "Over-expanded" monolayer exhibits formation of low- and high-density regions.*

In the “compressed” monolayer, which used the experimental values for the lattice spacing parameter and domain angle, there are many gauche defects present, especially in dihedrals 3, 5, and 6, since they are responsible for the “gauche defect bridges”. The “Over-expanded” monolayer also shows a significant count of gauche defects, and it is the “Relaxed” or optimal lattice size that produces the most highly ordered monolayer. The measurement of dihedral angles for the peripheral alkoxy chain atom selection was done using the TclTk plugin for VMD (Visual Molecular Dynamics) [47], and the *measure dihed* command. See **Appendix D** for the Tcl script used.

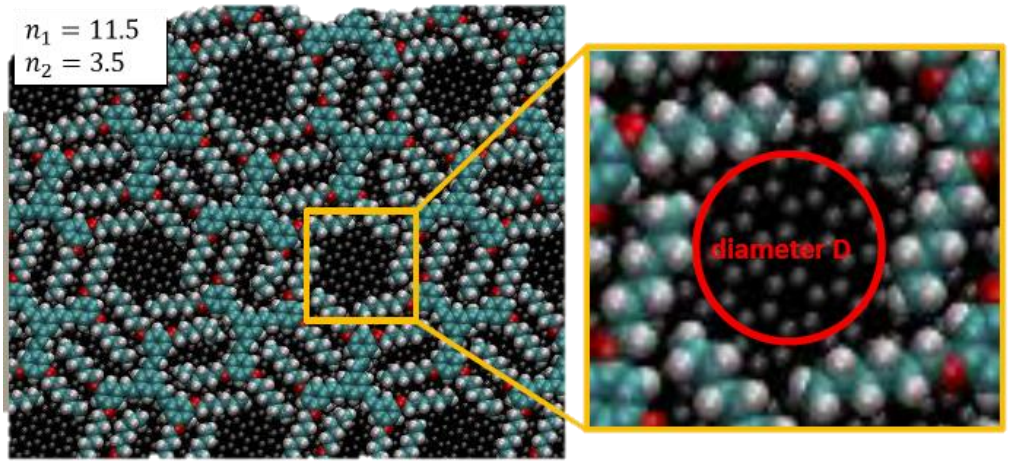


Figure 4.8 *Schematic of pore diameter D in the optimally sized lattice. Note that the representation used is a scaled van der Waals radii, so that the actual radii of atoms is larger, making the pores seem larger than they actually are.*

We give one more quantitative description of order in the monolayer at the optimal lattice size. The pore size is calculated as we increase temperature by use of *PSDSolv*, a pore size distribution (PSD) calculation algorithm [49]. This program employs the nonlinear optimization scheme *SolvOpt* to build a probability distribution of pore diameters. The pore size, or more specifically, the pore diameter, illustrated in **Fig. 4.8**, is important for determining whether the TSB35 monolayer may function as the molecular “sieve” that we wish to explore. The algorithm uses a test particle to measure largest possible diameters of pores throughout the system. We chose to use the argon atom (van der Waals radius 1.88 Å) as the test particle, since it is a common material used

in adsorption measurements, and for the determination of PSD's in porous media. The reported pore diameters are that of the largest inscribed circle for each pore that the algorithm finds, and it excludes the probe van der Waals diameter. Pore area was also calculated using a Hoshen-Kopelman inspired algorithm [50], written in Fortran, and it would be interesting to use this to calculate percent coverage of the substrate, as it is also reported in the literature [14].

At around room temperature (T = 300 K), we find that the distribution is tightly centered about 5.6 Å, and we observe more significant probabilities for larger pores as the temperature is increased. There is also a preference for smaller diameter pores forming as temperature increases, as kinetic energy dominates the dynamics. The peaks forming at about 2.2 Å are coming from the small spaces in between the alkoxy chains, and are inconsequential to the adsorbent properties of the TSB3,5-C6 monolayer (at higher temperatures, we observe the occasional benzene molecule being momentarily adsorbed in these smaller pores between alkoxy chains in a "vertical orientation", but the benzene admolecules lifetime in these pores is minimal). There is also the occasional formation formation of larger pores of up to 10 Å in diameter at higher temperatures, but these open and close constantly and are unlikely places for long-term adsorption of selected molecules. The distributions are taken over the last 6 ns of 12 ns of equilibration. The PSD for several lattice sizes (zero-, two-, and four-quarter integer additions from left to right) are shown in **Fig. 4.9**. There are twelve PSDs calculated for each lattice size. Each thin, colored curve represents one frame every 0.5 ns between 6 and 12 ns of equilibration. The thick, black curve in each plot is the average of these twelve curves. The optimal lattice size exhibits a stable pore diameter peak at about $D \approx 5.6$ Å, while the original lattice size shows favors smaller pores at around $D \approx 2.2$ Å and the last plot shows great disorder, and significant probabilities for small and large pores of highly fluctuating diameters.

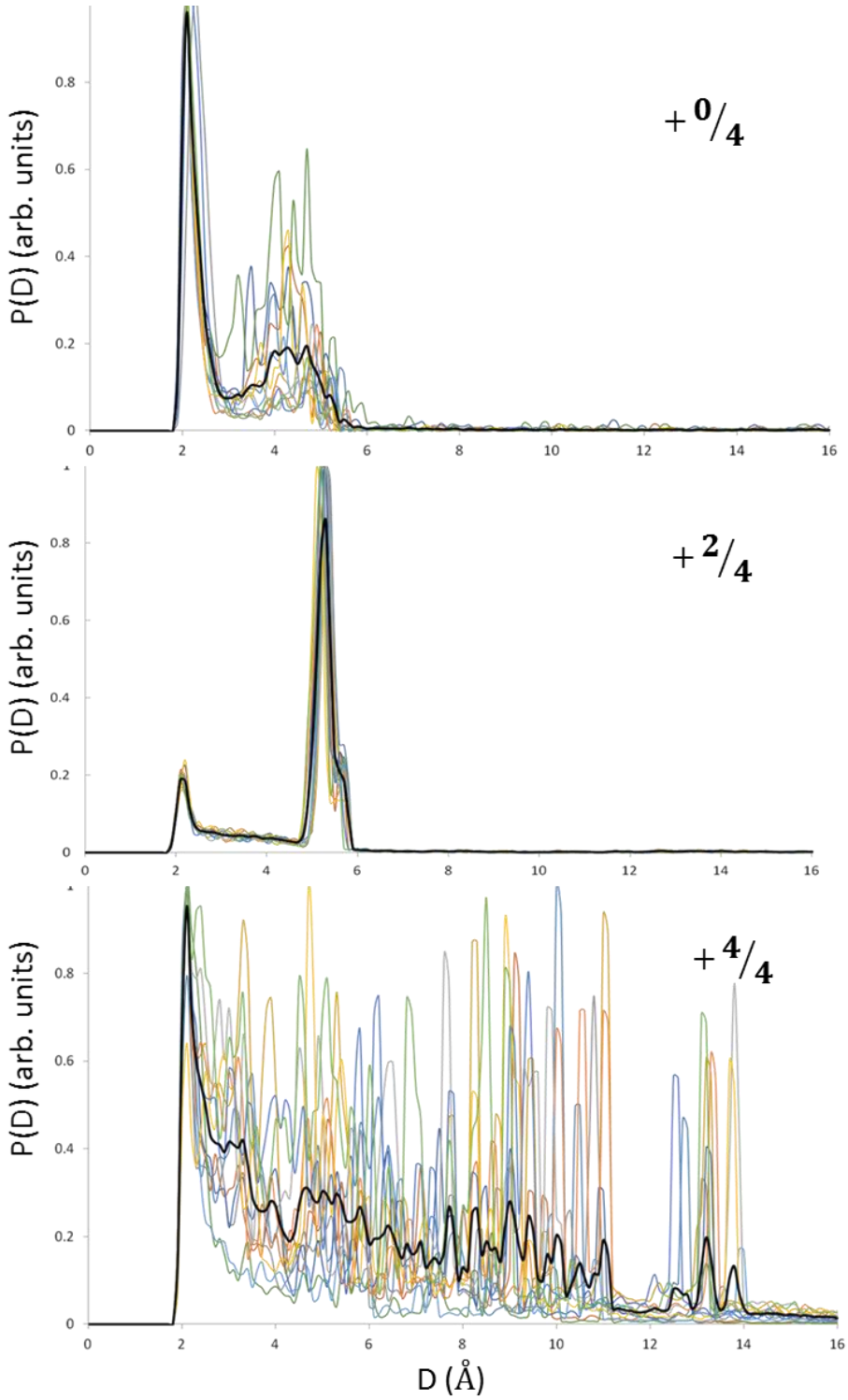


Figure 4.9 *PSDs for select lattice sizes, showing that two-quarter integer additions is most favorable. The distributions show the fluctuations by the thin, colored curves, and the average of these curves is shown in the thick, black curve.*

II. Thermal Effects on the Monolayer

With the optimal lattice size chosen, and a well-ordered system at room temperature (T = 300 K), we now increase the temperature of the simulation in order observe thermal effects on the monolayer. The literature reports liquid deposition of the TSB35 molecules at 333 K [14] and STM images being taken in the interval of T = 264 K to 305 K [12,13]. The simulations were each run for 12 ns of equilibration at temperatures T = 280, 300, 301, 302, 305, 310, 320, 340, 360, 380, 400, and 500 K. **Figure 4.10** depicts a series of select temperature snapshots (from the side and top), showing disorder induced by thermal fluctuations. Note that by 310 K, the pore structure is significantly distorted, but the overall pore structure remains throughout the equilibration. In simulation, the pores are observed to expand and contract rapidly, distorting the geometry of the pores, but the interactions of the peripheral alkoxy chains are strong enough to keep the monolayer and pore structure intact. This generic behavior remains until $T \approx 400$ K, where the potential for selective adsorptive properties are largely gone. The last temperature run at T = 500 K shows a destroyed monolayer, where this is no reliable size or shape to any of the pores. This general behavior is also observed from the side views shown in **Fig. 4.10**.

Increasing temperature introduces significant disorder to the monolayer, and effectively destroys the pore structure, due to the growing dominance of gauche defects in the peripheral alkoxy chains. This may be quantified through the total dihedral distributions and pore size distributions at each temperature.

We constructed distributions of the dihedral angles contained in the peripheral alkoxy chains, where we observe numerous gauche defects. The dihedrals considered are the six listed and labeled previously in **Figure 4.6**. The script used to measure the dihedral angles included the atom selection of just these peripheral alkoxy chains, and was done using the TclTk plugin for VMD (Visual Molecular Dynamics) [47], and the *measure dihed* command. See **Appendix D** for details and example of the script written in TclTk used to measure and compile the dihedral angles.

The distributions are taken over the last 6 ns of 12 ns of equilibration, and the temperature curves below in **Fig. 4.11** are the result of the average of the distribution of each dihedral angle in the peripheral alkoxy chains. Note that at T = 280 K, 300 K, and

301 K (blue, orange, and gray curves, respectively), the distribution is tightly around $\pm 180°$, i.e., flat molecules. At T = 302 K The distribution then begins picking up counts at $\pm 60°$ (yellow curve) and beyond. The number of gauche defects is also shown in **Fig. 4.11** as temperature rises to characterize the phase transition, and shows a drastic increase around T = 301 K. The counts are normalized to one for ease of reading.

The PSDs for various temperatures are shown in **Fig. 4.12**, and a select few (T = 300 K, 400 K, 500 K) PSDs are shown separately in **Fig. 4.13** to demonstrate the fluctuations at higher temperatures.

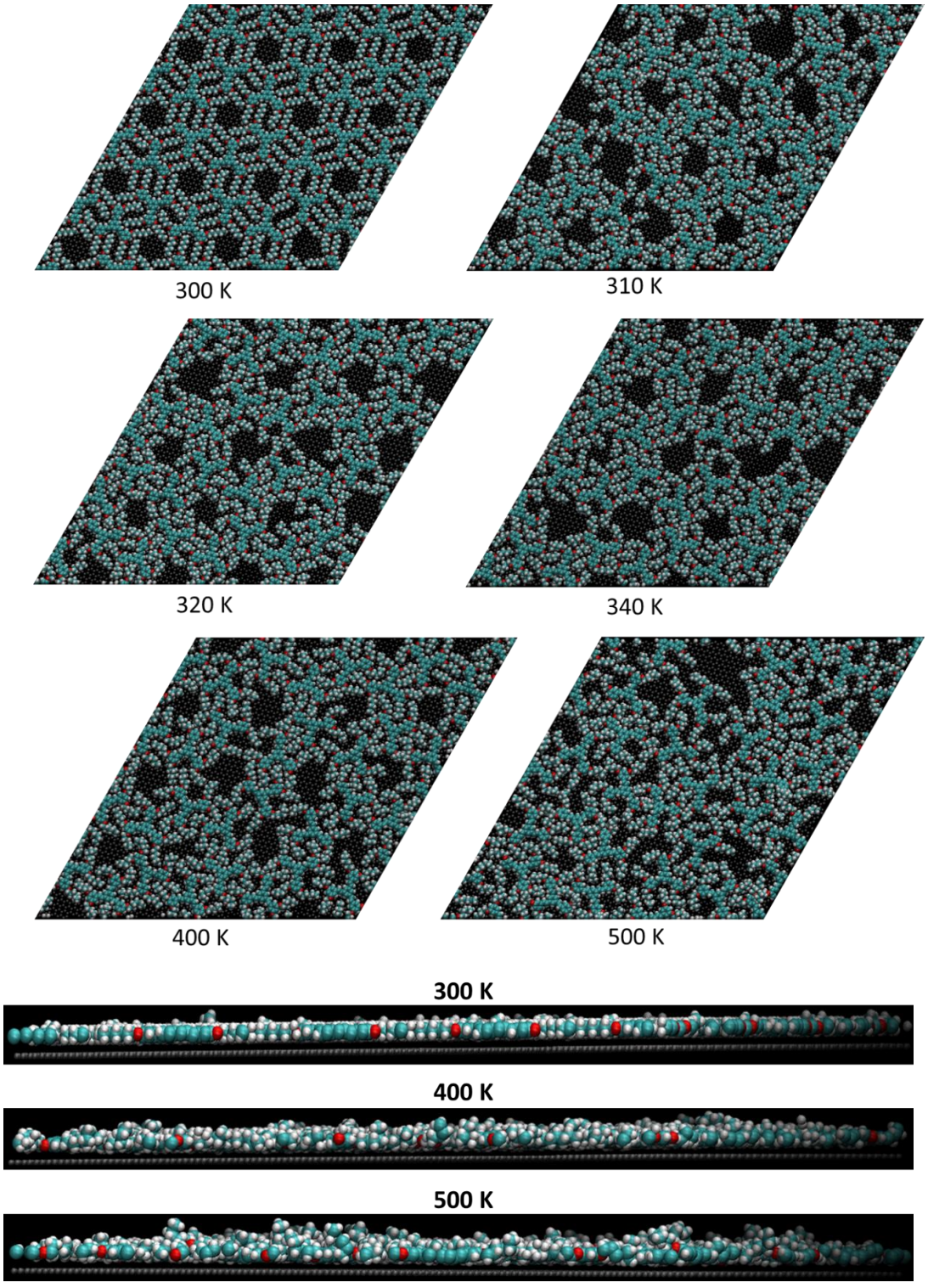
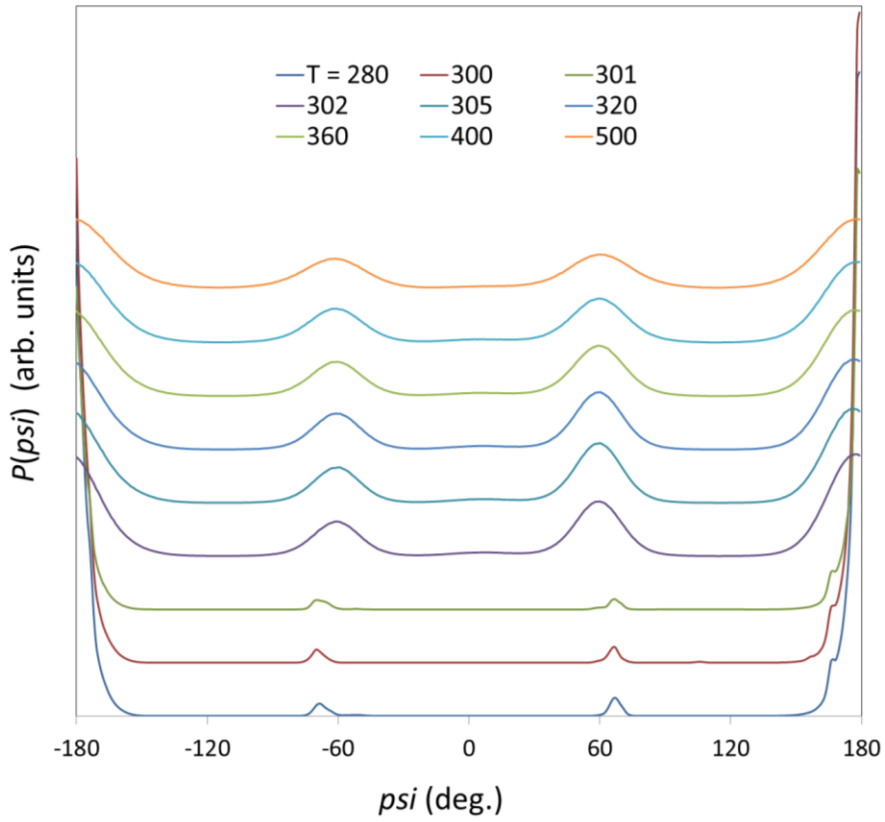


Figure 4.10 *Series of side snapshots of a few select temperatures, and top snapshots of a few more temperatures for the TSB3,5-C6 monolayer with optimal lattice size. Each snapshot is taken at the end of 12 ns of equilibration.*



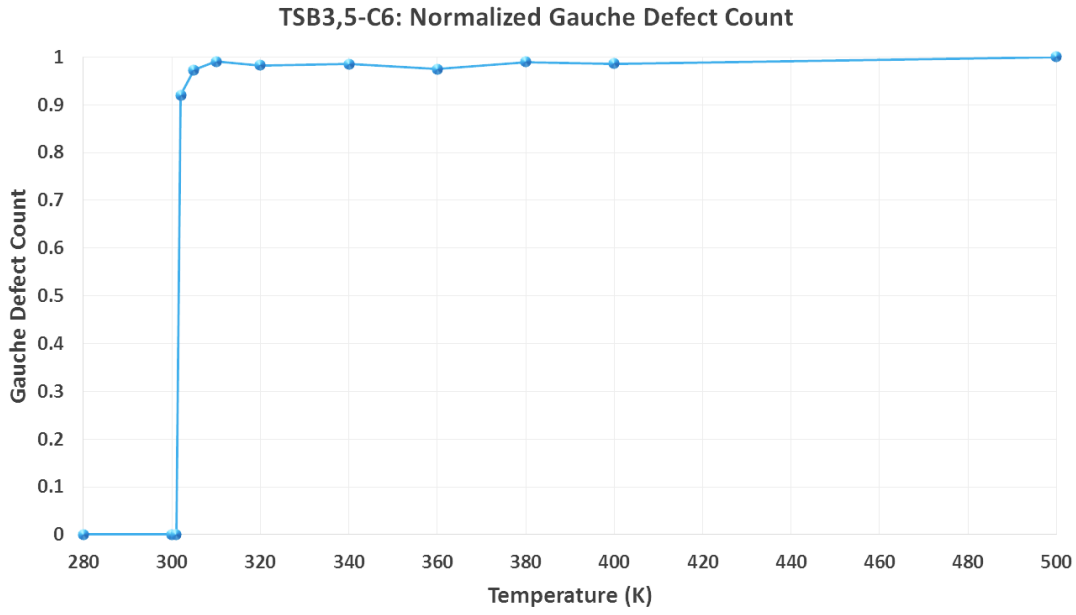


Figure 4.11 *(Top) Dihedral distribution for peripheral alkoxy chains. With increasing temperature, the number of gauche defects increases (counts for angles (bins) between -120 and 120 degrees). (Bottom) The number of gauche defects has a sharp increase in between 301 K and 302 K.*

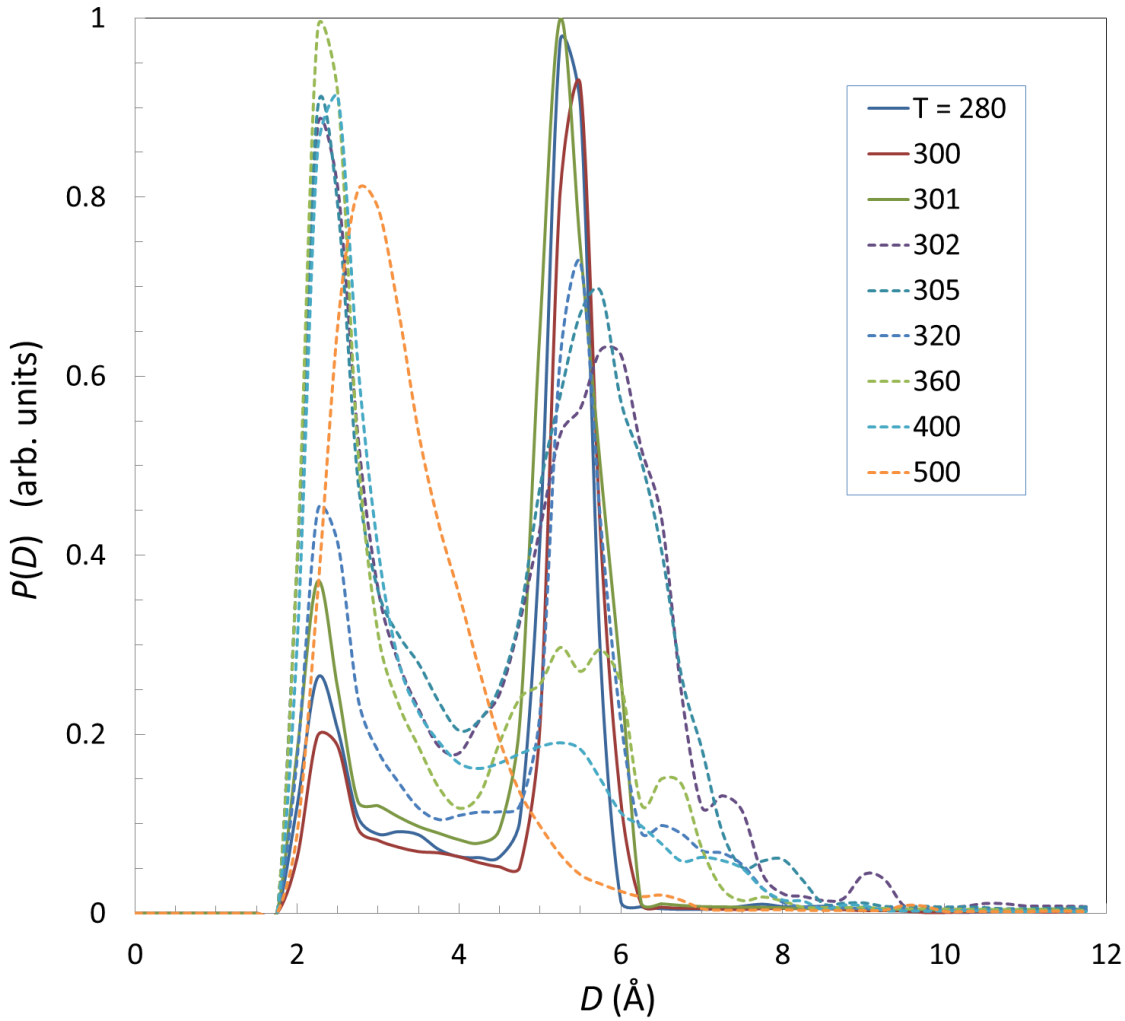


Figure 4.12 *PSD of optimal lattice size for various temperatures collected. The dashed curves represent the less well-ordered systems (greater than 301 K).*

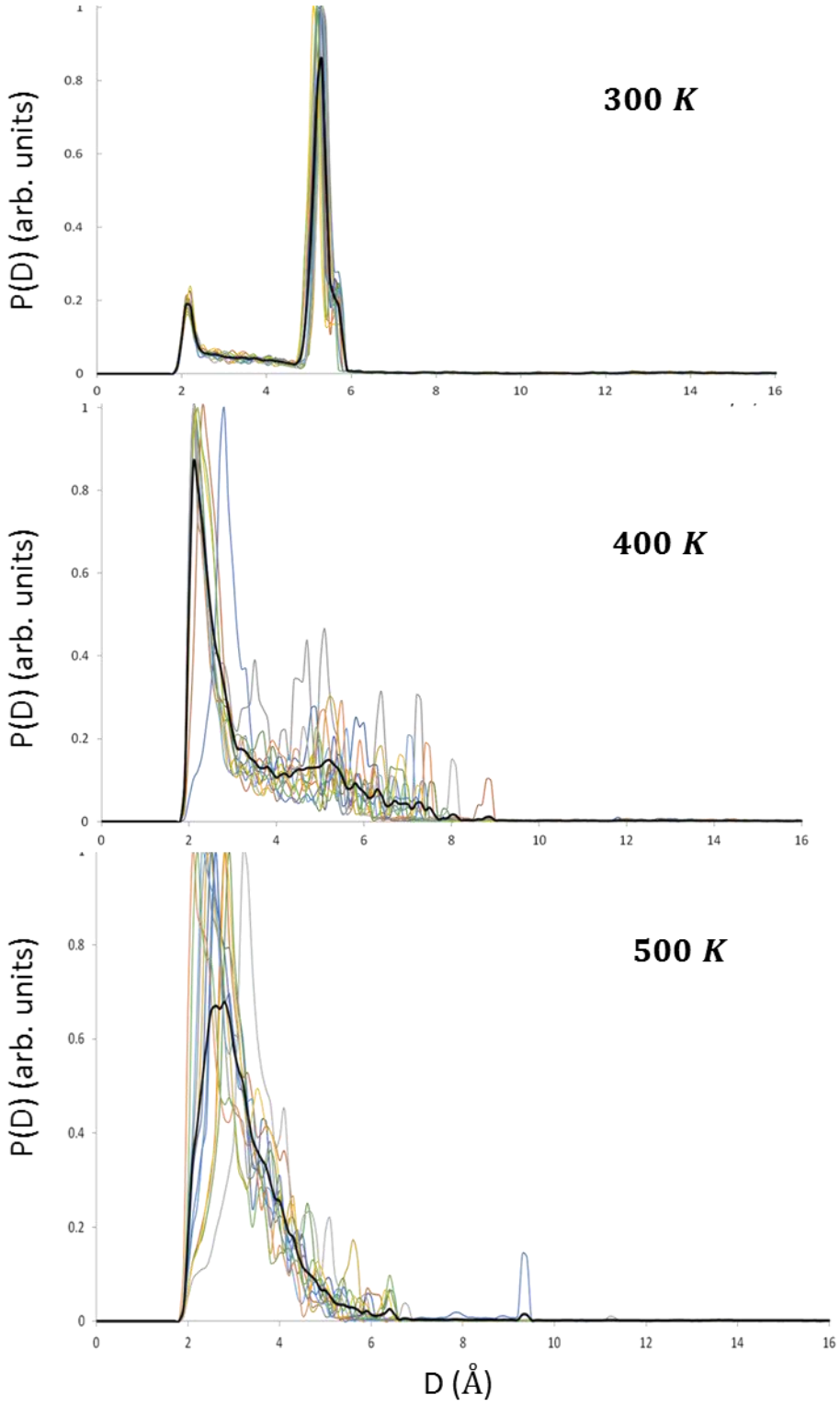


Figure 4.13 *Pore size distribution for TSB3,5-C6 monolayer. With increasing temperature, the distribution becomes less tight around 5.6 Angstroms, and picks up many counts for smaller pores, due to the alkoxy chains increasing kinetic energy, while also forming larger pores*

III. Adsorption of Guest Molecules

Following characterization of the pore structure, we begin to explore the functionality of the adsorption of the monolayer. The addition of guest molecules requires that the TOP and PDB files contain the proper information of the geometric and electronic structure of the guest molecules. See **Appendix C** for details of the setup and example scripts including guest molecules in the simulation.

Our first choice of guest is the benzene molecule, as it has a simple geometry. See **Fig. 4.14** for the guest-host configuration after 12 ns of equilibration at 300 K. The benzene molecules are observed to migrate to the pore walls, maximizing their surface area adsorbed, and remain strongly adsorbed to the monolayer and the substrate.

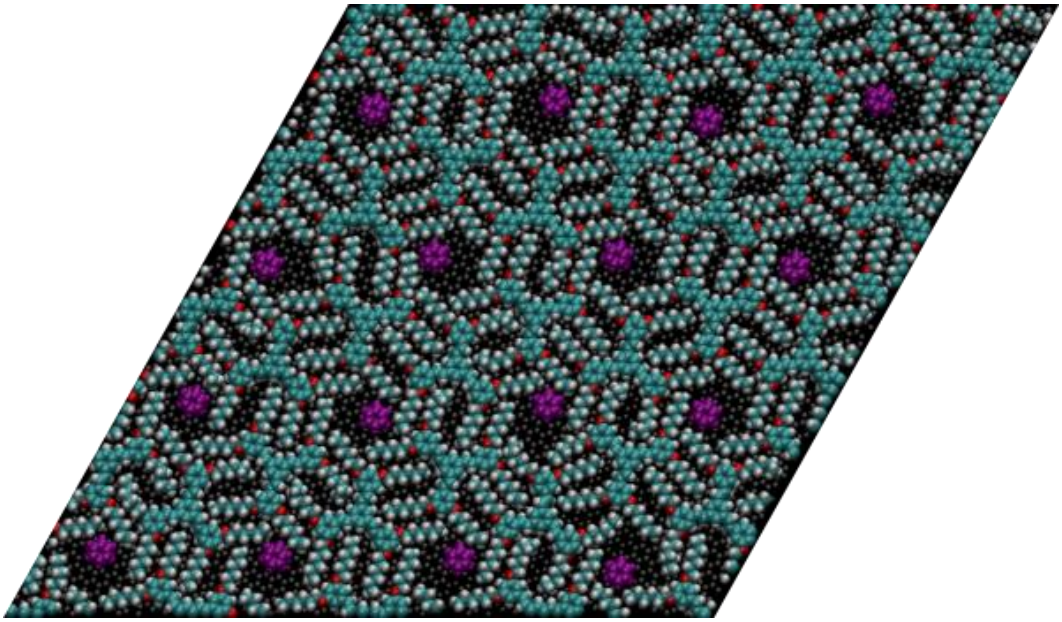


Figure 4.14 *The TSB3,5-C6 monolayer at 300 K with guest benzene molecules (purple) strongly adsorbed to the substrate and pore "walls" through 12 ns of equilibration.*

As we increase temperature up to 320 K, we begin to observe slight desorption and the occasional “pore hopping” of the guest molecules, and at even higher temperatures (greater than 340 K), we observe stronger desorption and more frequent “pore hop-

ping", occurring on femtosecond to picosecond timescales. **Fig. 4.15** is a top and side profile of the TSB3,5-C6 monolayer with benzene guest molecules at 320 K. Notice the few molecules above the monolayer, though they remain mostly adsorbed and partially inside the pores, as can be seen in the side profile.

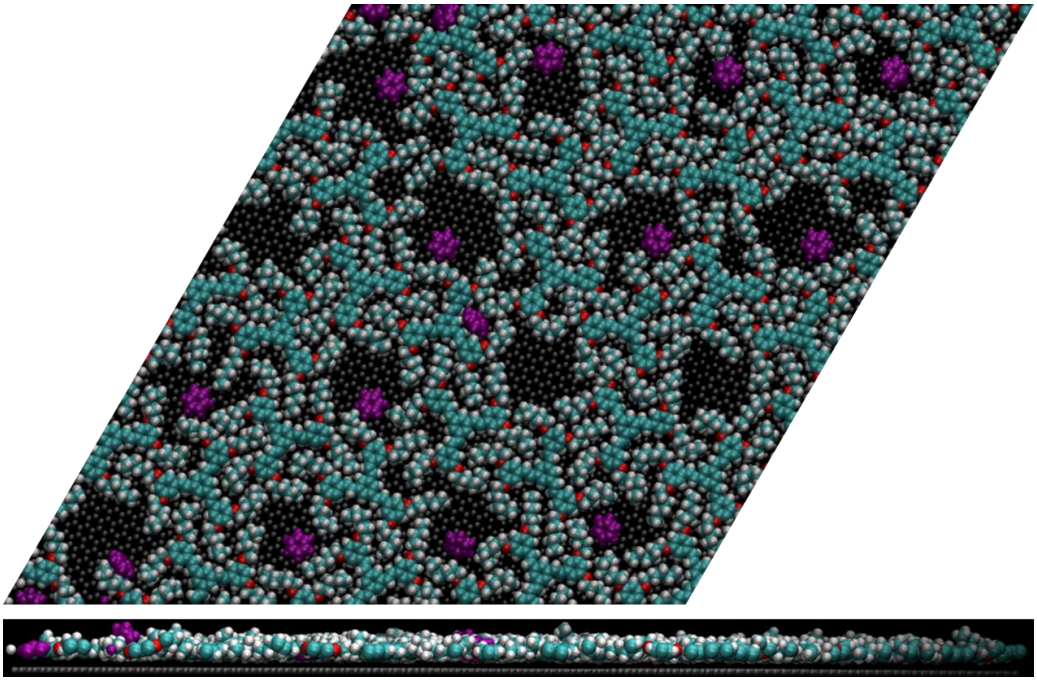


Figure 4.15 *Top and side profiles of the TSB3,5-C6 monolayer with benzene guest molecules (purple) at 320 K after 6 ns of equilibration.*

At a higher temperature T = 380 K, we observe that the benzene guest molecules will indeed desorb, see **Fig. 4.16**. There is also a few interesting cases where two, or sometimes more, benzene guest molecules hop into the same pore. It would be interesting to measure a residence time of the guest molecules per pore, and see how that changes as temperature is increased, but this was not performed in the present research effort.

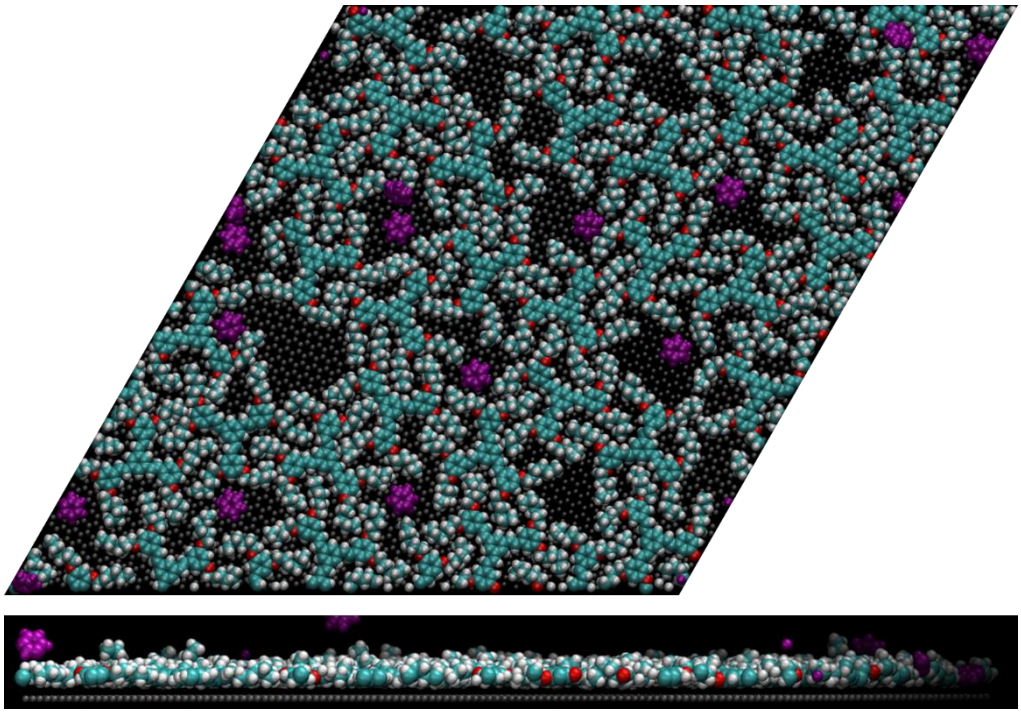
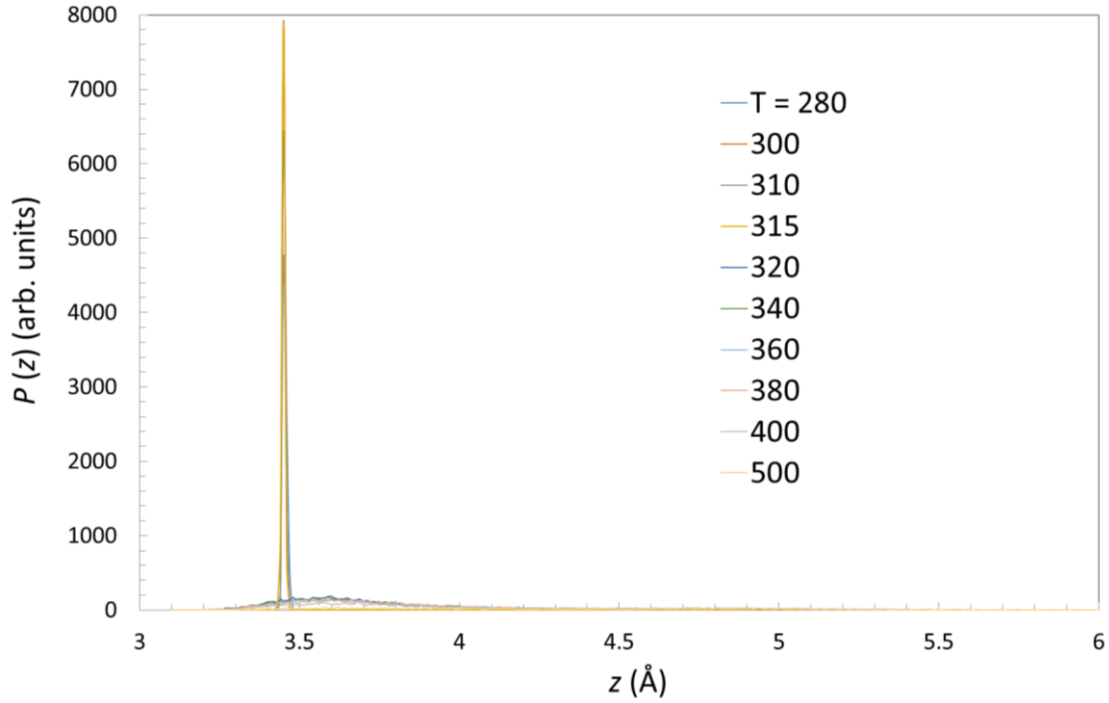
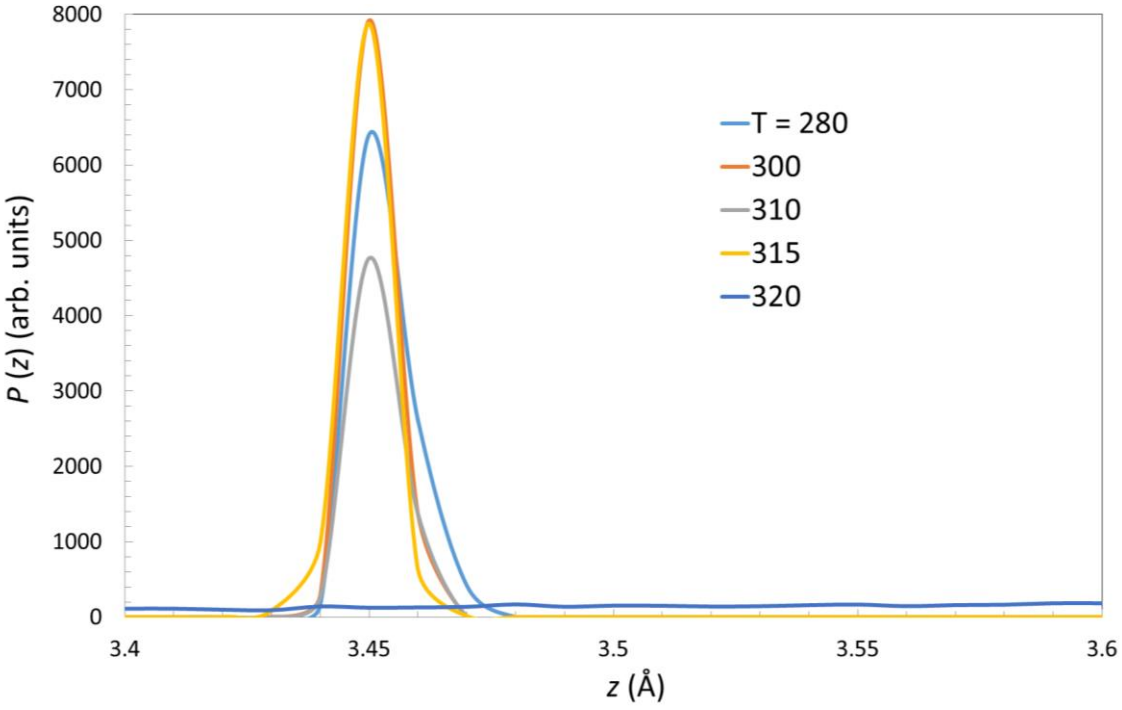


Figure 4.16 *Top and side profiles of the TSB3,5-C6 monolayer with benzene guest molecules (purple) at 380 K after 6 ns of equilibration. Note that the benzene is desorbing from the pore structure.*

To quantify the desorption we measured the *z*-component of the position of the center of mass of the guest molecules. **Figure 4.17** shows the distribution of the z-component of the center of mass of the guest benzene molecules. The top panel shows the full range of the distribution. The middle panel shows that the distributions at T = 280 K, 300 K, 310 K, and 315 K (lighter blue, orange, gray, yellow, respectively) have a very strong peak at $z \sim 3.4$ Å, indicating that the guest molecules are not leaving or moving around inside the pore. For the higher temperatures, beginning at 320 K (darker blue curve) the distribution becomes very widened and flattened out, and develops counts ranging up to 10 Å, as the benzene molecules begin hopping in and out of the pores, and are spending time free from strict adsorption in the pores. The bottom panel is rescaled to show the increased dynamics of the distribution. See **Appendix D** for details and example of the script written in TclTk used to measure and compile the center of mass for the guest molecules.


8000
7000
6000
5000
4000
3000
2000
1000
0
P (z) (arb. units)
3
3.5
4
4.5
5
5.5
6
z (Å)
T = 280
300
310
315
320
340
360
380
400
500


8000
7000
6000
5000
4000
3000
2000
1000
0
P (z) (arb. units)
3.4
3.45
3.5
3.55
3.6
z (Å)
T = 280
300
310
315
320

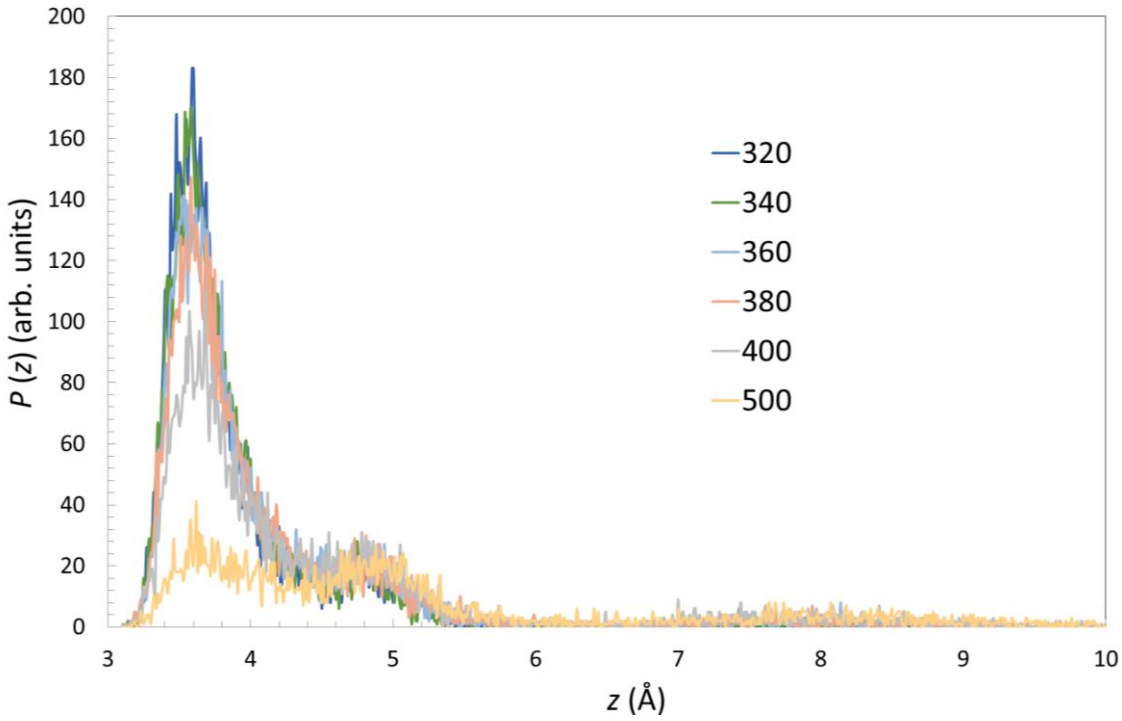


Figure 4.17 *The distribution of the z-component of the center of mass of the guest benzene molecules. (Top) Full range of distribution. (Middle) Shortened range to show detail of highly ordered temperatures. (Bottom) Rescaled to show detail of increased dynamics at higher temperatures.*

The stabilizing effects of guest molecules are also of interest, and we observe an extended interval of temperature where the monolayer-guest system remains well-ordered and strongly adsorbed to the HOPG substrate. In the TSB3,5-C6/benzene and TSB3,5-C10/coronene systems, we observe an upward shift in the phase transition, from well-ordered to disordered, of 10-15 K in the average potential energies. **Figure 4.18**, is an illustration of the stabilizing effect of the benzene guest molecule to the TSB3,5-C6 monolayer. It will be very interesting to explore this effect further, and pinpoint the "re-generation temperature" of the system, where the guest molecules are thermally removed, and order in the monolayer is restored.

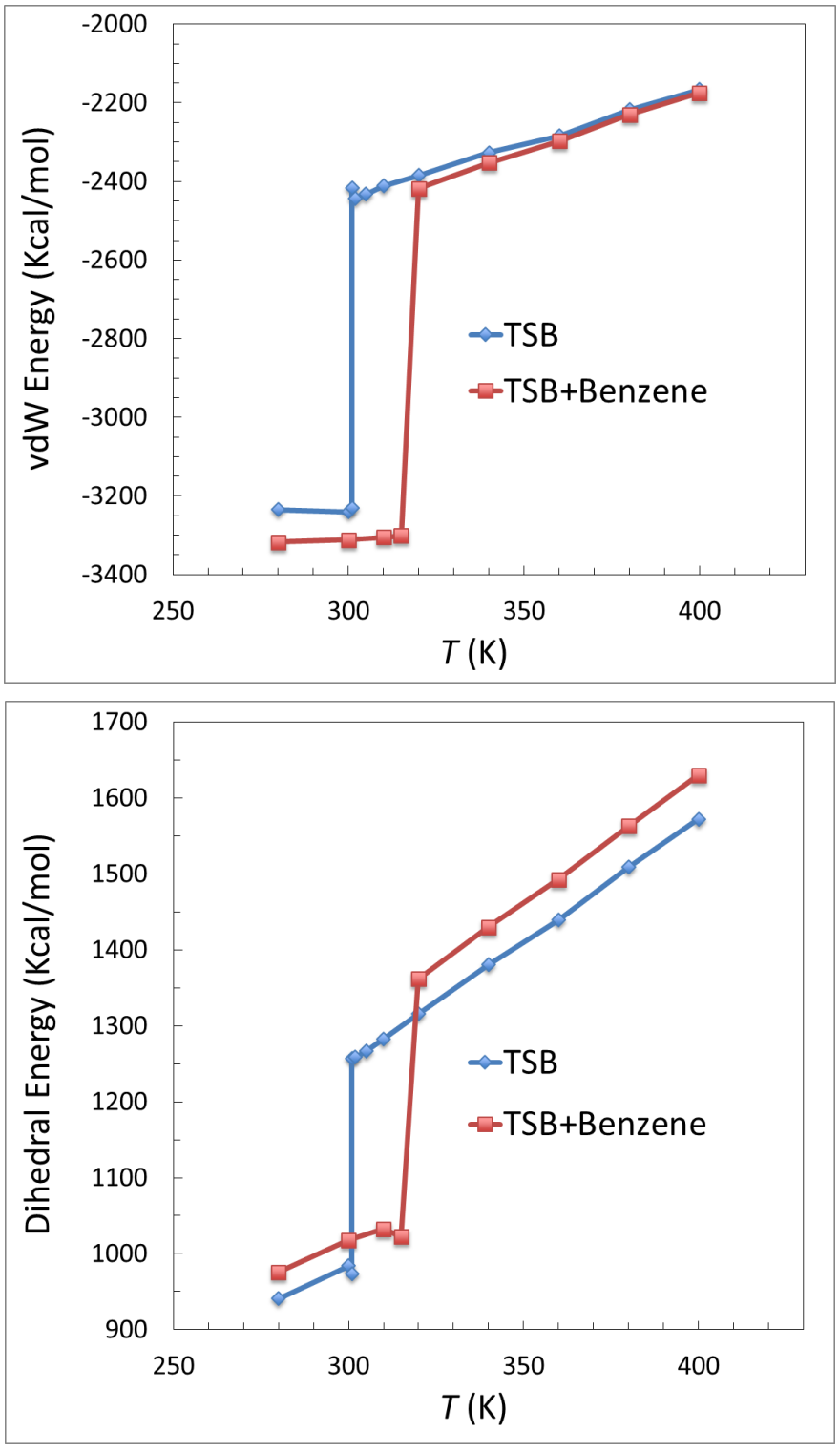


Figure 4.18 *Stabilizing effects of the addition of guest benzene molecules to to the TSB3,5-C6 monolayer. The transition (ordered → disordered) is shifted upwards by approximately 10 K in the van der Waals and dihedral potential energies.*

We also introduced pyrene (polycyclic aromatic hydrocarbon consisting of four fused benzene rings) as a guest molecule to the pores, see **Fig. 4.19**. We observe that the alkoxy chains tend to be attracted to the pyrene molecules in order to increase the adsorbed surface area for each guest pyrene molecule, significantly distorting the pore geometry, and creating regions of lower and higher density. We have not yet extensively studied these effects on the monolayer and its pore structure, as we may need to explore different initial geometries or lattice sizes (adding or subtracting quarter-integers to or from the TSB35 lattice vector coefficients) to give pyrene the substrate surface area necessary for it to adsorb to the substrate, without attracting the alkoxy chains.

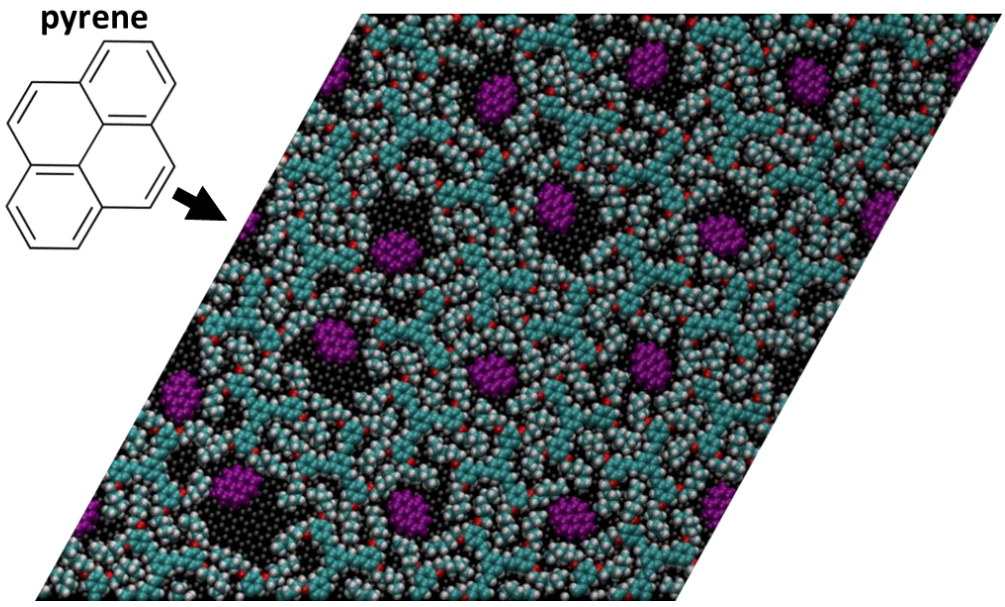


Figure 4.19 *The TSB3,5-C6 monolayer at 300 K with guest pyrene molecules (purple) distorting the pore structure after 6 ns of equilibration, due to poor initial geometry.*

The TSB3,5-C10 monolayer was also explored as a molecular sieve for the coronene guest molecules. The relative size of the coronene molecule to the pores of the TSB3,5-C10 monolayer is similar to that of the TSB3,5-C6/benzene system, and thus exhibits similar behavior. The coronene molecules are observed to adsorb to the "walls" of the pores, while remaining strongly adsorbed and planar, as shown in **Fig. 4.20**.

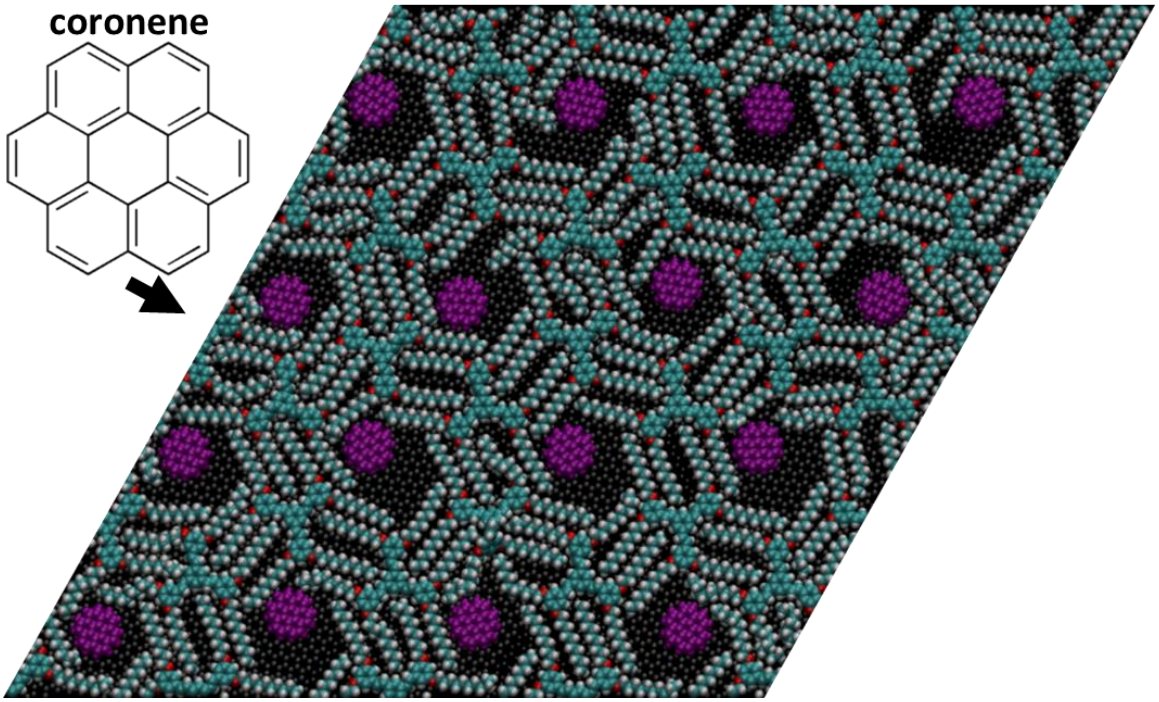


Figure 4.20 *The TSB3,5-C10 monolayer with guest coronene molecules (purple) at 300 K after minimization and 6 ns of equilibration. Note that this system exhibits behavior quite similar to the TSB3,5-C6 monolayer with benzene, as the guests adsorb to the pore walls and the substrate, resulting in a highly ordered lattice and strong adsorption.*

Chapter 5

Summary and Conclusions

Using fully quantum mechanical, electronic structure calculations, we have built an optimized computational model for the TSB35 family of molecules, and found accurate equilibrium values for geometric structure parameters b_0, θ_0, s_0, and δ. The computational model for the TSB35 monolayer adsorbed onto HOPG has also been built and analyzed using fully atomistic MD, yielding information on the functionality of the TSB3,5-C6 and TSB3,5-C10 monolayers as molecular sieves with tunable size and shape, at around room temperature. The experimental values for the lattice spacing parameter and the domain angle, used in construction of the computational cell, yielded a high-density, compressed configuration, and did not exhibit the desired pore structure. To relieve this, we appropriately expanded the TSB35 lattice vectors to find the density that correctly reproduced the well-ordered, stable pore structure observed in experiment. Our system is extremely stable at 300 K or below, and exhibits a phase transition from order to disorder at about 301-302 K. For the size and shape of the pores in the TSB3,5-C6 monolayer, we calculated a pore diameter of about 5.6 Å, which compares reasonably well to the literature value [14] of 6.2 Å, once differences in the "probe molecule" are taken into account. We see that our calculations make a good approximation of the system.

The host monolayer may accommodate various types of guest organic molecules due to its periodic, well-ordered pore structure. We used benzene as a guest molecule for the TSB3,5-C6 monolayer, and coronene for the TSB3,5-C10 monolayer, observing favorable conditions for adsorption in each case. The addition of pyrene to the TSB3,5-C6 monolayer destroyed the pore structure, as the pyrene molecules were too large for the pore at the fixed density of the system (though it is possible that more extensive simulations with a wider variety of computational cell sizes may find a structure that is able to accommodate pyrene as well). The transition temperature, at which the monolayer becomes disordered, is increased significantly by the addition of guest molecules. Quantita-

tively, we observed an increase in the phase transition temperature of about 10 K for the TSB3,5-C6/benzene system. Further exploration of this effect is of much theoretical and experimental interest.

Potential current and future work includes the detailed study of the adsorption of larger guest molecules, such as the pyrene molecule in the TSB3,5-C6 monolayer, and understanding how the pore structure is modified and (de)stabilized by the presence of guest molecules. The calculation of adsorption isotherms would give a nice quantitative understanding of the dynamics of the spreading pressure, as the concentration of TSB35 molecules changes due to the initial geometry (e.g. compressed monolayer to over-expanded monolayer). It may also be of interest to calculate pair correlation functions for the guest molecules as a measure of the probability that the guests will not desorb from a pore. It would be interesting to attempt to get a glance of the self-assembly process. This may be achieved by beginning a simulation away from, but sufficiently close to, the desired configuration for the monolayer, and measuring how some appropriate quantity changes (e.g. VdW potential energy between alkoxy chain atom selections) as the monolayer approaches the desired configuration.

With a detailed understanding of the TSB3,5-C6 and TSB3,5-C10 monolayers, the other TSB3,5-Cn ($n = 8, 12, 14$) may be subject to similar analyses with fully atomistic MD, and may allow trends to be identified in the relationship between pore geometry and admissible guest molecules.

Appendix A

Below is an excerpt from the topology (TOP) file for a single TSB3,5-C6 molecule. Recall **Table 2.2** for the renaming convention used here.

***Table A.1**. Excerpt from a TOP file, showing how Mulliken charge is assigned.*

Atom ID	Segment Name	Segment ID	Residue Name	Atom Name	Atom Type	Mulliken charge	Atomic Mass
28	TSBA	1	TSB	CR31	CA-A	0	12.011
29	TSBA	1	TSB	CR32	CA-B	-0.22	12.011
30	TSBA	1	TSB	CR33	CA-B	0.45	12.011
31	TSBA	1	TSB	CR34	CA-D	-0.3	12.011
32	TSBA	1	TSB	CR35	CA-C	0.45	12.011
33	TSBA	1	TSB	CR36	CA-C	-0.22	12.011
34	TSBA	1	TSB	HR32	HA	0.15	1.008
35	TSBA	1	TSB	HR34	HA	0.15	1.008
36	TSBA	1	TSB	HR36	HA	0.15	1.008
37	TSBA	1	TSB	CL11	CH1-A	-0.13	12.011
38	TSBA	1	TSB	CL12	CH1-D	-0.13	12.011
39	TSBA	1	TSB	HL11	H1	0.14	1.008
40	TSBA	1	TSB	HL12	H1	0.14	1.008

Note is that the “Atom Type” column is what communicates with the PARAMS file, in order to choose bonded/nonbonded (bond, angle, dihedral, Lennard-Jones, etc.) parameters for the simulation. The Mulliken charge is defined directly in the PSF and is associated with the “Atom Name” to retain independence from the PARAMS file. An example of a collection of lines in the PARAMS file:

***Table A.2**. Excerpt from a PARAMS file, showing how atom types are read in order to assign bonded and nonbonded parameters*

Atom Type 1	Atom Type 2	Elastic Force Constant	Equilibrium Bond Length
CA-A	CA-B	305	1.397
CA-A	CA-C	305	1.397
CA-A	CA-D	305	1.397
HA	CA-B	340	1.081
HA	CA-C	340	1.081
HA	CA-D	340	1.081
CH1-A	CA-A	365	1.4635
CH1-B	CA-A	365	1.4635
CH1-C	CA-A	365	1.4635
CH1-A	CH1-D	440	1.35
CH1-B	CH1-D	440	1.35
CH1-C	CH1-D	440	1.35
H1	CH1-A	360.5	1.0853
H1	CH1-B	360.5	1.0853
H1	CH1-C	360.5	1.0853

The "Atom Name" then communicates with the PDB file to assign nuclear coordinates to each atom. An excerpt from the PDB file for a single TSB3,5-C6 molecule:

***Table A.3.** Excerpt from a PDB file, showing how coordinates are assigned.*

Record	Atom ID	Atom Name	Res. Name	Res. ID	X Coord.	Y Coord.	Z Coord.	Beta	Occupancy	Seg. Name	Element
ATOM	28	CR31	TSB	1	6.162	2.448	3.133	1	0	TSBA	C
ATOM	29	CR32	TSB	1	6.164	3.841	3.175	1	0	TSBA	C
ATOM	30	CR33	TSB	1	7.292	4.584	3.192	1	0	TSBA	C
ATOM	31	CR34	TSB	1	8.54	3.954	3.172	1	0	TSBA	C
ATOM	32	CR35	TSB	1	8.578	2.556	3.128	1	0	TSBA	C
ATOM	33	CR36	TSB	1	7.401	1.808	3.114	1	0	TSBA	C
ATOM	34	HR32	TSB	1	5.241	4.405	3.188	1	0	TSBA	H
ATOM	35	HR34	TSB	1	9.449	4.529	3.178	1	0	TSBA	H
ATOM	36	HR36	TSB	1	7.477	0.73	3.075	1	0	TSBA	H
ATOM	37	CL11	TSB	1	-1.268	2.057	3.191	1	0	TSBA	C
ATOM	38	CL12	TSB	1	-1.455	3.392	3.178	1	0	TSBA	C
ATOM	39	HL11	TSB	1	-2.137	1.406	3.217	1	0	TSBA	H
ATOM	40	HL12	TSB	1	-0.587	4.043	3.152	1	0	TSBA	H

In the supplementary information, there are copies of the full TOP and PDB files for a single TSB3,5-C6 molecule and a single benzene guest molecule, and a copy of the full PARAMS file that we built.

Appendix B

Refer to **Tables 2.2**, **2.3**, **2.4**, and **2.5** for reference to calculated equilibrium values of bond length, bond angle, and dihedral angle, and the Mulliken charge assignments.

Table B.1. *Bond length potential parameters.*

Bond types	k_b $(kcal\ mol^{-1}Å^{-2})$	b_0 (Å)
CA-CA	305.0	1.40
CA-HA	340.0	1.08
CA-CH1	365.0	1.46
CH1-CH1	440.0	1.35
CH1-H1	360.5	1.09
CA-O	230.0	1.37
O-CH2	360.0	1.43
CH2-CH2	222.5	1.52
CH2-H2	309.0	1.09
CH2-CH3	222.5	1.52
CH3-H3	322.0	1.09
CG-CG	305.0	1.40

Table B.2. *Bond angle potential parameters.*

Angle types	k_θ $(kcal\ mol^{-1}rad^{-2})$	θ_0 (°)	k_{UB} $(kcal\ mol^{-1}Å^{-2})$	s_0(Å)
CA-CA-CA	40.0	120.0	35.0	2.42
CA-CA-HA	30.0	120.0	22.0	2.15
CA-CA-CH1 **	36.0	122.4	-	-
CA-CA-CH1 **	36.0	117.2	-	-
CA-CH1-CH1	48.0	125.2	-	-
CA-CH1-H1	32.0	115.9	-	-
CH1-CH1-H1	42.0	118.9	-	-
CA-CA-O **	110.0	124.2	-	-
CA-CA-O **	110.0	117.8	-	-
CA-O-CH2	65.0	117.4	-	-
O-CH2-CH2	45.0	106.8	-	-
O-CH2-H2	45.9	110.0	-	-
CH2-CH2-CH2	58.4	113.4	11.2	2.55
CH2-CH2-H2	26.5	109.4	22.5	2.15
H2-CH2-H2	35.5	106.4	5.4	1.77
CH2-CH2-CH3	58.0	112.9	8.0	2.55
H2-CH2-CH3	34.6	109.6	22.5	2.15
H3-CH3-CH2	34.6	111.0	22.5	2.17
H3-CH3-H3	35.5	107.8	5.4	1.74
CG-CG-CG	40.0	120.0	35.0	2.42

***Recall the different "directions" described in **Chapter 2 Section II**, and **Figure 2.9**.*

Table B.3. *Dihedral angle potential parameters.*

Angle types	$k_\chi\ (kcal\ mol^{-1})$	$\delta\ (°)$	$n\ (multiplicity)$
CA-CA-CA-CA	3.10	180.0	2
CA-CA-CA-HA	4.20	180.0	2
HA-CA-CA-CH1	2.40	180.0	2
CA-CA-CA-CH1	3.10	180.0	2
CA-CA-CH1-CH1	0.75, 0.19	180.0, 0.0	2, 4
CA-CA-CH1-H1	0.60	180.0	2
CA-CH1-CH1-H1	5.20	180.0	2
H1-CH1-CH1-H1	5.20	180.0	2
CA-CH1-CH1-CA	5.20	180.0	2
CA-CA-CA-O	3.10	180.0	2
HA-CA-CA-O	2.40	180.0	2
CA-CA-O-CH2	1.62, 0.19	180.0, 180.0	2, 4
CA-O-CH2-CH2	0.24, 0.29, 0.02	0.0, 0.0, 0.0	1, 2, 3
CA-O-CH2-H2	0.10	0.0	3
O-CH2-CH2-CH2	0.19	0.0	3
O-CH2-CH2-H2	0.19	0.0	3
CH2-CH2-CH2-CH2	0.15	0.0	1
CH2-CH2-CH2-CH3	0.15	0.0	1
H2-CH2-CH2-CH2	0.19	0.0	3
H2-CH2-CH2-H2	0.22	0.0	3
H2-CH2-CH3-H3	0.16	0.0	3
CH2-CH2-CH3-H3	0.16	0.0	3
CG-CG-CG-CG	3.10	180.0	2

Table B.4. *Nonbonded potential parameters.*

Atom type	Label (see figure)	$\epsilon\ (kcal/mol)$	σ (Å)	$q\ (esu)$
Aromatic carbon:	1	-0.070	1.99	0.00
	2	-0.070	1.99	-0.15
	3	-0.070	1.99	-0.22
	4	-0.070	1.99	0.45
	5	-0.070	1.99	-0.30
Aromatic hydrogen:	6	-0.030	1.36	0.15
	7	-0.030	1.36	0.16
Aliphatic carbon:	8	-0.068	2.09	-0.13
	9	-0.056	2.01	0.10
	10	-0.056	2.01	-0.20
	11	-0.056	2.01	-0.21
	12	-0.078	2.05	-0.33
Aliphatic hydrogen:	13	-0.031	1.25	0.14
	14	-0.035	1.34	0.11
Oxygen:	15	-0.100	1.65	-0.69
Graphitic carbon:	-	-0.070	1.99	0.00

Appendix C

C.1 File Structure and Work Flow

The NAMD2 environment requires a complex set of interlinked files and formats. **Figure C.1** depicts a scheme [46] of the file (PDB, TOP, PSF, PARAMS, CONF) and software (Gaussian09, PSFGEN, NAMD2) dependencies to get from the *ab initio* results of the electronic structure calculation to the fully atomistic molecular dynamics simulation.

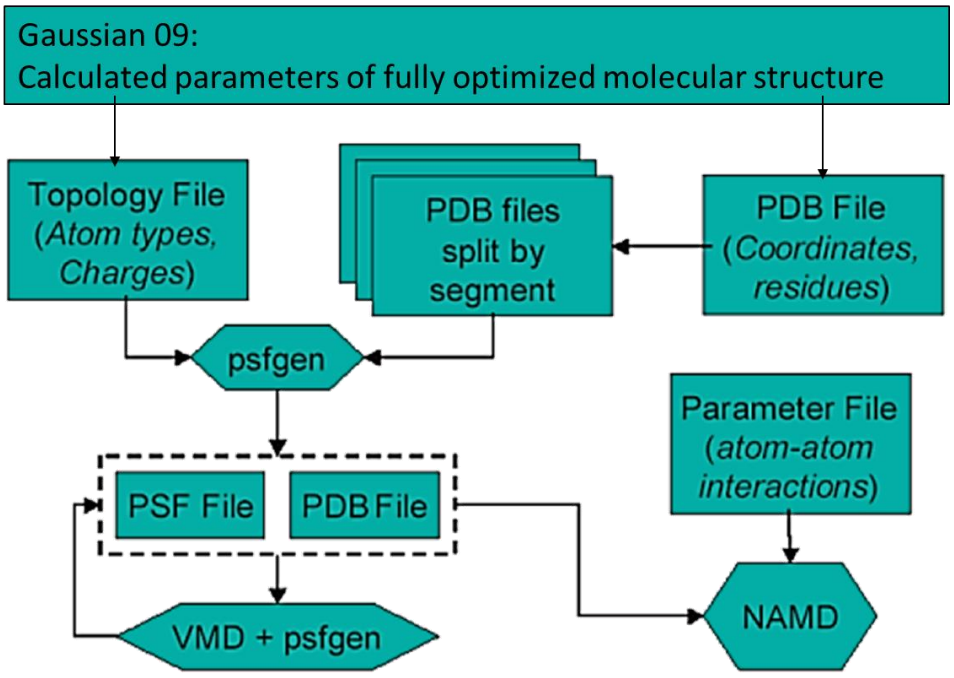


Figure C.1 *Diagram showing the file and software dependencies required for setting up a NAMD2 simulation, after Ref. [46].*

C.2 Building the .PDB and .TOP Files

The VMD visualization and NAMD2 simulation require at the basic level, the .PDB file, which supplies information on atomic coordinates and some interaction information, and the .TOP file, which supplies information on the bonding order and Mulliken charges. These two files communicate through the software to match atom labeling. Through C++ scripts, I have put significant effort into the automation and modularization the process of generating all of the necessary scripts, and strongly recommend the provided NAMD and VMD Tutorials online for complimentary and supplementary information.

The first step is to create the .PDB files, with or without a guest molecule. Choose the appropriate script, compile (`g++ -o prog filename.cpp`), and run the script from the command line:

- **01-a_TSB_PDB-writer.cpp**
    - `./prog nCarbon nQuarter`
- **01-b_TSB+guest_PDB-writer.cpp**
    - `./prog nCarbon nQuarter guest guestSegname nAtomsperGuest`

where “nCarbon” is the number of carbon atoms in the alkoxy chain(e.g. 6), “nQuarter” is the number of quarter integers used in the lattice vector expansion (e.g. 2), “guest” is the name of the chosen guest molecule (e.g. benzene), “guestSegname” is the chosen 3 or 4 character guest segment name (e.g. BENZ), and “nAtomsPerGuest” is the total number of atoms per guest molecule (e.g. 12).

Next, create the .PDB file for the graphite layers:

- **02_graphite_PDB-writer.cpp**
    - `./prog nCarbon nQuarter`

C.2.1 Example C++ script: 01-b_TSB+guest_PDB-writer.cpp

```
#include <iostream>
#include <iomanip>
#include <math.h>
#include <stdlib.h>
#include <fstream>
#include <ctime>
#include <complex>
#include <cstdio>
#include <string>
#include "../MASTER-FILES/AtomParams.h"
using namespace std;

//////////////////////////////////////////////////////////////////
// Subroutines to transform nuclear coordinates: Translate,
// xAxisRotation, yAxisRotation, zAxisRotation
//
// e.g.
// void zAxisRotation(AtomParams **Atom,double rotationAngle,int nAtoms)
// {
//  struct vect tempPosition, position;
//
//  for (int i=0; i<nAtoms; i++)
//  {
//    position = Atom[i]->getPosition();
//
//    tempPosition.x = position.x*cos(rotationAngle) -
// position.y*sin(rotationAngle);
//    tempPosition.y = position.x*sin(rotationAngle) +
// position.y*cos(rotationAngle);
//    tempPosition.z = position.z;
//
//    Atom[i]->setPosition( tempPosition );
//  }
// }
//////////////////////////////////////////////////////////////////

int main(int argc, char *argv[])
{
    ////////////////
    // USER INPUT //
    ////////////////
  if (argc != 6) {
    cout << "Must have 5 command line arguments (nCarbon, nQuarter,
guest, guestSegname, nAtomsPerGuest)!" << endl;
  }

  int nCarbon = atoi(argv[1]);
  int nQuarter = atoi(argv[2]);
  string guest = argv[3];
  string guestSegname = argv[4];
  int nAtomsPerGuest = atoi(argv[5]);

  float n1, n2;
  double latticeSpacing, domainAngle;
  if (nCarbon == 6) {

    n1 = 11.0 + float(nQuarter)*0.25;
```

```
  n2 = 3.0 + float(nQuarter)*0.25;

  if (nQuarter == 0) {
    latticeSpacing = 31.401;
    domainAngle = 11.742;
  } else if (nQuarter == 1) {
    latticeSpacing = 32.414;
    domainAngle = 12.331;
  } else if (nQuarter == 2) {
    latticeSpacing = 33.430;
    domainAngle = 12.885;
  } else if (nQuarter == 3) {
    latticeSpacing = 34.450;
    domainAngle = 13.407;
  } else if (nQuarter == 4) {
    latticeSpacing = 35.472;
    domainAngle = 13.898;
  } else {}

} else if (nCarbon == 10) {

  n1 = 15.0 + float(nQuarter)*0.25;
  n2 = 1.0 + float(nQuarter)*0.25;

  if (nQuarter == 0) {
    latticeSpacing = 38.182;
    domainAngle = 3.198;
  } else if (nQuarter == 1) {
    latticeSpacing = 39.135;
    domainAngle = 3.901;
  } else if (nQuarter == 2) {
    latticeSpacing = 40.095;
    domainAngle = 4.571;
  } else if (nQuarter == 3) {
    latticeSpacing = 41.059;
    domainAngle = 5.209;
  } else if (nQuarter == 4) {
    latticeSpacing = 42.028;
    domainAngle = 5.818;
  } else {}

} else {}

int nAtomsPerTSB = 54+6*(3*nCarbon+1);
int nAtomsTotal = nAtomsPerTSB + nAtomsPerGuest;

  // Create Instance of the class AtomParams
AtomParams *Atom[nAtomsTotal];

  ////////////////////////////////////////////
  // Read TSB input files (from Gaussian09) //
  ////////////////////////////////////////////
stringstream inStream1;
inStream1 << "../cpp-input/tsb35-c" << nCarbon << "_cpp-input.dat";
ifstream in1(inStream1.str().c_str());

cout << endl;
cout << "TSB input file to be read: " << inStream1.str() << endl;

string tempAtomName, tempAtomType, tempElementType;
struct vect tempPosition;
double tempCharge;
```

```
  for (int i=0; i<nAtomsPerTSB; i++)
  {
    in1 >> tempAtomName >> tempAtomType >> tempPosition.x >> tempPosi-
tion.y >> tempPosition.z >> tempElementType >> tempCharge;
    Atom[i] = new AtomParams(tempAtomName, tempAtomType, tempPosition,
tempElementType, tempCharge);
  }

  cout << "TSB input file read and stored in class AtomParams/Atom." <<
endl;
  cout << endl;

    // Create TSB output PDB files
  stringstream outStream1;
  outStream1 << "tsb35-c" << nCarbon << "_0" << nQuarter << "-
quarter_TSBA-PDB.pdb";
  ofstream OUT1(outStream1.str().c_str());
  OUT1 << setiosflags(ios::fixed) << setprecision(3);

  stringstream outStream2;
  outStream2 << "tsb35-c" << nCarbon << "_0" << nQuarter << "-
quarter_TSBB-PDB.pdb";
  ofstream OUT2(outStream2.str().c_str());
  OUT2 << setiosflags(ios::fixed) << setprecision(3);

  cout << "TSB output files created: " << outStream1.str() << endl;
  cout << "and " << outStream2.str() << endl;
  cout << endl;

      ////////////////////////////////////////////
      // Read guest input files (from Gaussian09) //
      ////////////////////////////////////////////
    stringstream inStream2;
    inStream2 << "../cpp-input/" << guest << "_cpp-input.dat";
    ifstream in2(inStream2.str().c_str());

    cout << "Guest input file to be read: " << inStream2.str() << endl;

    for (int i=nAtomsPerTSB; i<nAtomsTotal; i++)
    {
      in2 >> tempAtomName >> tempAtomType >> tempPosition.x >> tempPosi-
tion.y >> tempPosition.z >> tempElementType >> tempCharge;
      Atom[i] = new AtomParams(tempAtomName, tempAtomType, tempPosition,
tempElementType, tempCharge);
    }

    cout << "Guest input file read and stored in class AtomParams/Atom."
<< endl;

      // Create guest output PDB file
    stringstream outStream3;
    outStream3 << "tsb35-c" << nCarbon << "_0" << nQuarter << "-quarter_"
<< guestSegname << "-PDB.pdb";
    ofstream OUT3(outStream3.str().c_str());
    OUT3 << setiosflags(ios::fixed) << setprecision(3);

    cout << "guest output file created: " << outStream3.str() << endl;


    ////////////////////////////
    // Build lattice vectors //
    ////////////////////////////
  struct vect a1, a2, a3, b1, b2, b3;
```

```
  double a = sqrt(3.0)*1.42;
  double triAngle = 60.0*(PI/180.0);

    // Define graphene lattice vectors
  a1.x = a;
  a1.y = 0.0;
  a1.z = 0.0;
  a2.x = a*cos(triAngle);
  a2.y = a*sin(triAngle);
  a2.z = 0.0;
  a3 = (1.0/3.0)*(a1+a2);

    // Define TSB lattice vectors
  b1 = n1*a1 + n2*a2;
  b2 = (-1.0)*n2*a1 + (n1+n2)*a2;
  b3 = (-1.0/3.0)*(b1+b2);

  double Bmag = 4.0*sqrt(b1.x*b1.x + b1.y*b1.y + b1.z*b1.z);

    // Calculate center of mass for the TSB molecule
  double mass;
  double totalMass=0.0;

  struct vect position;
  struct vect CoM;
  CoM.x=0.0;
  CoM.y=0.0;
  CoM.z=0.0;

  for (int i=0; i<nAtomsPerTSB; i++)
  {
    mass = Atom[i]->getMass();

    position = Atom[i]->getPosition();
    Atom[i]->setPosition(position);

    CoM.x += mass * position.x;
    CoM.y += mass * position.y;
    CoM.z += mass * position.z;

    totalMass += mass;
  }

  CoM = (1.0/totalMass)*CoM;

  Translate(Atom, (-1.0)*CoM, nAtomsPerTSB);
  double rotationAngle = 60.0*(PI/180.0);
  zAxisRotation(Atom, rotationAngle, nAtomsPerTSB);


////////////////////////////////////////////////////////////////////////

  struct vect hexagonCenter;
  hexagonCenter.x = 0.5*a;
  hexagonCenter.y = 0.5*a/sqrt(3.0);
  hexagonCenter.z = 3.366;

    // translate to a reasonable distance above graphene sheet, centered
on the first graphene carbon
  Translate(Atom, hexagonCenter, nAtomsPerTSB);

    //define number of TSB molecule "rows" in directions of b1 and b2
  int nMolecules1 = 4;
```

```
  int nMolecules2 = 4;

  struct vect comTSBA, comTSBB;

  int moleculeCount = 0;
    // Write TSBA molecules
  OUT1 << "CRYST1  " << Bmag << "  " << Bmag << "  100.000  " << do-
mainAngle << "  " << domainAngle << "  " << domainAngle << "          P 1
1" << endl;
  for (int i=0; i<nMolecules1; i++)
  {
    for (int j=0; j<nMolecules2; j++)
    {
      for (int k=0; k<nAtomsPerTSB; k++)
      {
       position = Atom[k]->getPosition();
       position = position + i*b1 + j*b2;
       Atom[k]->setPosition(position);

       OUT1 << "ATOM " << setw(6) << k + moleculeCount*nAtomsPerTSB + 1;
       OUT1 << " " << setw(4) << Atom[k]->getAtomName() << " TSB   " <<
setw(3) << moleculeCount+1 << "     ";
       OUT1 << setw(7) << position.x << " " << setw(7) << position.y << "
" << setw(7) << position.z << "  ";
       OUT1 << "0.00  0.00      " << setw(4) << "TSBA" << endl;

       if (i==0 && j==0)
       {
          mass = Atom[k]->getMass();
          position = Atom[k]->getPosition();
          Atom[k]->setPosition(position);

          comTSBA.x += mass * position.x;
          comTSBA.y += mass * position.y;
          comTSBA.z += mass * position.z;

          totalMass += mass;
       }

       position = Atom[k]->getPosition();
       position = position - i*b1 - j*b2;
       Atom[k]->setPosition(position);
      }
      moleculeCount++;
    }
  }

  comTSBA = (1.0/totalMass)*comTSBA;

    // Place molecule on origin by subtracting: (r_i - r_com)
  Translate(Atom, (-1.0)*CoM, nAtomsPerTSB);
    // Rotate by 60 degrees, interdigtating the rest of the TSB molecules
  zAxisRotation(Atom, rotationAngle, nAtomsPerTSB);
    // Replace molecule at (r_i + r_com)
  Translate(Atom, CoM, nAtomsPerTSB);

  moleculeCount = 0;
    // Write rotated molecules, using the TSB basis vector b3
  OUT2 << "CRYST1  " << Bmag << "  " << Bmag << "  100.000  " << do-
mainAngle << "  " << domainAngle << "  " << domainAngle << "          P 1
1" << endl;
  for (int i=0; i<nMolecules1; i++)
  {
```

```
    for (int j=0; j<nMolecules2; j++)
    {
      for (int k=0; k<nAtomsPerTSB; k++)
      {
       position = Atom[k]->getPosition();
       position = position + i*b1 + j*b2 + b3;
       Atom[k]->setPosition(position);

       OUT2 << "ATOM " << setw(6) << k + moleculeCount*nAtomsPerTSB + 1;
       OUT2 << " " << setw(4) << Atom[k]->getAtomName() << " TSB   " <<
setw(3) << moleculeCount+1 << "     ";
       OUT2 << setw(7) << position.x << " " << setw(7) << position.y << "
" << setw(7) << position.z << "  ";
       OUT2 << "0.00  0.00      " << setw(4) << "TSBB" << endl;

       if (i==1 && j==1)
       {
         mass = Atom[k]->getMass();
          position = Atom[k]->getPosition();
          Atom[k]->setPosition(position);

          comTSBA.x += mass * position.x;
          comTSBA.y += mass * position.y;
          comTSBA.z += mass * position.z;

          totalMass += mass;
       }

       position = Atom[k]->getPosition();
       position = position - i*b1 - j*b2 - b3;
       Atom[k]->setPosition(position);
      }
      moleculeCount++;
    }
  }

  comTSBB = (1.0/totalMass)*comTSBB;

  struct vect comGuest = comTSBA + comTSBB;

  moleculeCount = 0;
    // Write guest molecules to PDB file
  OUT3 << "CRYST1  " << Bmag << "   " << Bmag << "  100.000  " << do-
mainAngle << "  " << domainAngle << "  " << domainAngle << "          P 1
1" << endl;
  for (int i=0; i<nMolecules1; i++)
  {
    for (int j=0; j<nMolecules2; j++)
    {
      for (int k=nAtomsPerTSB; k<nAtomsTotal; k++)
      {
       position = Atom[k]->getPosition();
       position = position + i*b1 + j*b2 + comGuest;
       Atom[k]->setPosition(position);

       OUT3 << "ATOM " << setw(6) << k - nAtomsPerTSB + mole-
culeCount*nAtomsPerGuest + 1;
       OUT3 << " " << setw(4) << Atom[k]->getAtomName() << " COR   " <<
setw(3) << moleculeCount+1 << "     ";
       OUT3 << setw(7) << position.x << " " << setw(7) << position.y << "
" << setw(7) << position.z << "  ";
       OUT3 << "0.00  0.00      " << setw(4) << "COR " << endl;
```

```
        position = Atom[k]->getPosition();
        position = position - i*b1 - j*b2 - comGuest;
        Atom[k]->setPosition(position);
       }
       moleculeCount++;
     }
   }

   return 0;
}
```

After this, create the .TOP files, choosing the appropriate script:

- **03-a_TSB+graphite_TOP-writer.cpp**
  - `./prog nCarbon nQuarter`
- **03-b_TSB+guest+graphite_TOP-writer.cpp**
  - `./prog nCarbon nQuarter guest guestSegname nAtomsperGuest`

C.2.2 Example C++ script: 03-b_TSB+guest+graphite_TOP-writer.cpp

```
#include <iostream>
#include <iomanip>
#include <math.h>
#include <stdlib.h>
#include <fstream>
#include <ctime>
#include <complex>
#include <cstdio>
#include <string>
#include "../MASTER-FILES/AtomParams.h"
using namespace std;

  // Subroutine to count number of atoms per graphene layer, given the
boundaries defined by the super-lattice vectors
int countGrapheneAtoms (struct vect a1, struct vect a2, struct vect a3,
struct vect b1, struct vect b2, struct vect b3)
{
  struct vect B1, B2, B12, C1, C2, C12;

    // Define TSB super-lattice vectors
  B1 = 4.0*b1;
  B2 = 4.0*b2;
  B12 = B1 + B2;
```

```
    // Define boundaries of TSB super-lattice vectors
  C1 = B1 + 2.0*b3;
  C2 = B2 + 2.0*b3;
  C12 = B12 + 2.0*b3;

  double slopeC1 = (C1.y - 2.0*b3.y) / (C1.x - 2.0*b3.x);
  double slopeC2 = (C2.y - 2.0*b3.y) / (C2.x - 2.0*b3.x);
  double yInterceptC1 = C1.y - slopeC1*C1.x;
  double yInterceptC2 = C2.y - slopeC2*C2.x;

  double yIntercept1C12 = C12.y - slopeC1*C12.x;
  double yIntercept2C12 = C12.y - slopeC2*C12.x;

    // Initialize graphene basis vectors, where "1" and "2" denote basis
atoms of the graphene unit cell
  struct vect positionG1 = 2.0*b3;
  struct vect positionG2 = 2.0*b3;

    // Initialize a sufficiently large grid to scan for atoms inside the
boundaries
  int maxUnitCells1 = 500;
  int maxUnitCells2 = 500;
  int nAtomsPerLayer = 0;

    // Calculate atomic positions, counting nAtomsPerLayer
  for (int i=-maxUnitCells1; i<maxUnitCells1; i++)
  {
    for (int j=-maxUnitCells2; j<maxUnitCells2; j++)
    {
      positionG1 = i*a1 + j*a2 + a3;
      positionG2 = i*a1 + j*a2 + 2.0*a3;

      double xLowerLimitG1 = (1.0/slopeC2)*(positionG1.y - yInterceptC2);
      double yLowerLimitG1 = slopeC1*positionG1.x + yInterceptC1;

      double xUpperLimitG1 = (1.0/slopeC2)*(positionG1.y-yIntercept2C12);
      double yUpperLimitG1 = slopeC1*positionG1.x + yIntercept1C12;

      double xLowerLimitG2 = (1.0/slopeC2)*(positionG2.y - yInterceptC2);
      double yLowerLimitG2 = slopeC1*positionG2.x + yInterceptC1;

      double xUpperLimitG2 = (1.0/slopeC2)*(positionG2.y-yIntercept2C12);
      double yUpperLimitG2 = slopeC1*positionG2.x + yIntercept1C12;

      if (positionG1.x >= xLowerLimitG1 && positionG1.x <= xUpperLimitG1)
      {
        if (positionG1.y >= yLowerLimitG1 && positionG1.y<=yUpperLimitG1)
        {
          nAtomsPerLayer++;
         }
      }

      if (positionG2.x >= xLowerLimitG2 && positionG2.x <= xUpperLimitG2)
      {
        if (positionG2.y>=yLowerLimitG2 && positionG2.y<=yUpperLimitG2)
        {
          nAtomsPerLayer++;
        }
      }
    }
  }

// flag here for atoms very close to the boundary and may be missing
```

```
//nAtomsPerLayer++;

  return nAtomsPerLayer;
}

  // Subroutine to populate the members of class Graphene
void assignGrapheneParams(AtomParams **Graphene, struct vect a1, struct
vect a2, struct vect a3, struct vect b1, struct vect b2, struct vect b3)
{
  struct vect B1, B2, B12, C1, C2, C12;

    // Define TSB super-lattice vectors
  B1 = 4.0*b1;
  B2 = 4.0*b2;
  B12 = B1 + B2;

    // Define boundaries of TSB super-lattice vectors (purely for calcu-
lation)
  C1 = B1 + 2.0*b3;
  C2 = B2 + 2.0*b3;
  C12 = B12 + 2.0*b3;

  double slopeC1 = (C1.y - 2.0*b3.y) / (C1.x - 2.0*b3.x);
  double slopeC2 = (C2.y - 2.0*b3.y) / (C2.x - 2.0*b3.x);
  double yInterceptC1 = C1.y - slopeC1*C1.x;
  double yInterceptC2 = C2.y - slopeC2*C2.x;

  double yIntercept1C12 = C12.y - slopeC1*C12.x;
  double yIntercept2C12 = C12.y - slopeC2*C12.x;

    // Initialize graphene basis vectors, where "1" and "2" denote basis
atoms of the graphene unit cell
  struct vect positionG1 = 2.0*b3;
  struct vect positionG2 = 2.0*b3;

    // Initialize a sufficiently large grid to scan for atoms inside the
boundaries
  int maxUnitCells1 = 500;
  int maxUnitCells2 = 500;

    //assign appropriate graphene atom positions and atom indices to
class AtomParams
  int atomCount = 0;
  for (int i=-maxUnitCells1; i<maxUnitCells1; i++)
  {
    for (int j=-maxUnitCells2; j<maxUnitCells2; j++)
    {
      positionG1 = i*a1 + j*a2 + a3;
      positionG2 = i*a1 + j*a2 + 2.0*a3;

      double xLowerLimitG1 = (1.0/slopeC2)*(positionG1.y - yInterceptC2);
      double yLowerLimitG1 = slopeC1*positionG1.x + yInterceptC1;

      double xUpperLimitG1 = (1.0/slopeC2)*(positionG1.y-yIntercept2C12);
      double yUpperLimitG1 = slopeC1*positionG1.x + yIntercept1C12;

      double xLowerLimitG2 = (1.0/slopeC2)*(positionG2.y - yInterceptC2);
      double yLowerLimitG2 = slopeC1*positionG2.x + yInterceptC1;

      double xUpperLimitG2 = (1.0/slopeC2)*(positionG2.y-yIntercept2C12);
      double yUpperLimitG2 = slopeC1*positionG2.x + yIntercept1C12;

      if (positionG1.x >= xLowerLimitG1 && positionG1.x <= xUpperLimitG1)
```

```
      {
        if (positionG1.y >= yLowerLimitG1 && positionG1.y<=yUpperLimitG1)
        {
          Graphene[atomCount] = new AtomParams(positionG1, atomCount+1);
          atomCount++;
         }
      }

      if (positionG2.x >= xLowerLimitG2 && positionG2.x <= xUpperLimitG2)
      {
        if (positionG2.y >= yLowerLimitG2 && positionG2.y<=yUpperLimitG2)
        {
          Graphene[atomCount] = new AtomParams(positionG2, atomCount+1);
          atomCount++;
        }
      }
    }
  }

//account for missing atoms here
// struct vect missingAtomPosition;
//missingAtomPosition.x = ;
//missingAtomPosition.y = ;
//missingAtomPosition.z = ;
//
//Graphene[atomCount] = new AtomParams(missingAtomPosition, atomCount+1);
//atomCount++;
}

  // Subroutine that writes to topology file for entire system
void writeTOP(AtomParams **Atom, AtomParams **Graphene, int nCarbon, int
nQuarter, string guest, string guestSegName, int nAtomsPerTSB, int
nAtomsPerGuest, int nAtomsPerLayer)
{
    // Create output TOP file
  stringstream outStream;
  outStream << "tsb35-c" << nCarbon << "_0" << nQuarter << "-
quarter_with-" << guest << "_TOP.top";
  ofstream OUT(outStream.str().c_str());
  OUT << setiosflags(ios::fixed) << setprecision(3);

  cout << "TOP file to generate: " << outStream.str() << endl;

    // topology file header
  OUT << "27  1" << endl;
  OUT << endl;
  OUT << "MASS     1 HGA2     1.00800 H ! nonpolar, aliphatic H" << endl;
  OUT << "MASS     2 HGA3     1.00800 H ! nonpolar, aliphatic H" << endl;
  OUT << "MASS     3 HGR61   1.00800 H ! aromatic H" << endl;
  OUT << "MASS     4 HGA4     1.00800 H ! alkene H" << endl;
  OUT << "MASS     5 CG2R61A 12.01100 C ! aromatic C" << endl;
  OUT << "MASS     6 CG2R61B 12.01100 C ! aromatic C” << endl;
  OUT << "MASS     7 CG2R61C 12.01100 C ! aromatic C" << endl;
  OUT << "MASS     8 CG2R61D 12.01100 C ! aromatic C" << endl;
  OUT << "MASS     9 CG321   12.01100 C ! aliphatic CH2" << endl;
  OUT << "MASS    10 CG331   12.01100 C ! aliphatic CH3" << endl;
  OUT << "MASS    11 CG2DC1A 12.01100 C ! alkene C” << endl;
  OUT << "MASS    12 CG2DC1B 12.01100 C ! alkene C” << endl;
  OUT << "MASS    13 CG2DC1C 12.01100 C ! alkene C” << endl;
  OUT << "MASS    14 CG2DC1D 12.01100 C ! alkene C” << endl;
  OUT << "MASS    15 OG301 15.99940 O ! ether O " << endl;
  OUT << "MASS    16 CG2R61 12.01100 C ! aromatic C (graphite)" << endl;
  OUT << endl;
```

```
  OUT << "AUTO ANGLES DIHE" << endl;
  OUT << endl;

    // TSB molecule topology entry
  OUT << "RESI TSB          0.00" << endl;
  OUT << endl;

  string atomName, atomType;
  double charge;
  for (int i=0; i<nAtomsPerTSB; i++)
  {
    atomName = Atom[i]->getAtomName();
    atomType = Atom[i]->getAtomType();
    charge = Atom[i]->getCharge();

    OUT << "ATOM " << left << setw(5) << atomName << right << setw(4) <<
atomType << right << setw(8) << charge << endl;
  }
  OUT << endl;

  struct bondList *temp;
  for (int i=0; i<nAtomsPerTSB; i++)
  {
    atomName = Atom[i]->getAtomName();
    temp = Atom[i]->getBond();

    if (temp != NULL)
      OUT << "BOND ";
    while (temp !=NULL)
    {
      OUT << left << setw(4) << atomName << "  " << temp->atomName << "";
      temp = temp->next;
    }
    if (Atom[i]->getBond() != NULL)
      OUT << endl;
  }
  OUT << endl;

    // Guest molecule topology entry
  OUT << "RESI " << guestSegName << "          0.00" << endl;
  OUT << endl;

  int nAtomsTotal = nAtomsPerTSB + nAtomsPerGuest;
  for (int i=nAtomsPerTSB; i<nAtomsTotal; i++)
  {
    atomName = Atom[i]->getAtomName();
    atomType = Atom[i]->getAtomType();
    charge = Atom[i]->getCharge();

    OUT << "ATOM " << left << setw(5) << atomName << right << setw(4) <<
atomType << right << setw(8) << charge << endl;
  }
  OUT << endl;

  for (int i=0; i<nAtomsPerGuest; i++)
  {
    atomName = Atom[i]->getAtomName();
    temp = Atom[i]->getBond();

    if (temp != NULL)
      OUT << "BOND ";
    while (temp !=NULL)
    {
```

```
      OUT << left << setw(4) << atomName << "  " << temp->atomName << "";
      temp = temp->next;
    }
    if (Atom[i]->getBond() != NULL)
      OUT << endl;
  }
  OUT << endl;

    // Graphene layer topology entry
  OUT << "RESI GRAP         0.00" << endl;
  OUT << endl;

  for (int i=0; i<nAtomsPerLayer; i++)
  {
   stringstream ss;

  // Assign naming scheme for graphitic carbon atoms

    atomName=ss.str();

    OUT << "ATOM " << left << setw(5) << atomName << right << setw(4) <<
"CG2R61" << right << setw(8) << 0.000 << endl;
  }
  OUT << endl;

  for (int i=0; i<nAtomsPerLayer; i++)
  {
    stringstream ss;

    // Assign naming scheme for graphitic carbon atoms

    atomName=ss.str();

    temp = Graphene[i]->getBond();

    if (temp != NULL)
      OUT << "BOND ";
    while (temp !=NULL)
    {
      OUT << left << setw(4) << atomName << "  " << temp->atomName << "
";
      temp = temp->next;
    }
    if (Graphene[i]->getBond() != NULL)
      OUT << endl;
  }
  OUT << endl;
    //
}



int main(int argc, char *argv[])
{
    ////////////////
    // USER INPUT //
    ////////////////
  if (argc != 6) {
    cout << "Must have 5 command line arguments (nCarbon, nQuarter,
guest, guestSegName, nAtomsPerGuest)!" << endl;
  }

  int nCarbon = atoi(argv[1]);
```

```
int nQuarter = atoi(argv[2]);
string guest = argv[3];
string guestSegName = argv[4];
int nAtomsPerGuest = atoi(argv[5]);

float n1, n2;
double latticeSpacing, domainAngle;
if (nCarbon == 6) {

  n1 = 11.0 + float(nQuarter)*0.25;
  n2 = 3.0 + float(nQuarter)*0.25;

  if (nQuarter == 0) {
    latticeSpacing = 31.401;
    domainAngle = 11.742;
  } else if (nQuarter == 1) {
    latticeSpacing = 32.414;
    domainAngle = 12.331;
  } else if (nQuarter == 2) {
    latticeSpacing = 33.430;
    domainAngle = 12.885;
  } else if (nQuarter == 3) {
    latticeSpacing = 34.450;
    domainAngle = 13.407;
  } else if (nQuarter == 4) {
    latticeSpacing = 35.472;
    domainAngle = 13.898;
  } else {}

} else if (nCarbon == 10) {

  n1 = 15.0 + float(nQuarter)*0.25;
  n2 = 1.0 + float(nQuarter)*0.25;

  if (nQuarter == 0) {
    latticeSpacing = 38.182;
    domainAngle = 3.198;
  } else if (nQuarter == 1) {
    latticeSpacing = 39.135;
    domainAngle = 3.901;
  } else if (nQuarter == 2) {
    latticeSpacing = 40.095;
    domainAngle = 4.571;
  } else if (nQuarter == 3) {
    latticeSpacing = 41.059;
    domainAngle = 5.209;
  } else if (nQuarter == 4) {
    latticeSpacing = 42.028;
    domainAngle = 5.818;
  } else {}

} else {}

int nAtomsPerTSB = 54+6*(3*nCarbon+1);
int nAtomsTotal = nAtomsPerTSB + nAtomsPerGuest;

  // Create Instance of the class AtomParams
AtomParams *Atom[nAtomsTotal];

  ////////////////////////////////////////////
  // Read TSB input files (from Gaussian09) //
  ////////////////////////////////////////////
stringstream inStream1;
```

```
  inStream1 << "../cpp-input/tsb35-c" << nCarbon << "_cpp-input.dat";
  ifstream in1(inStream1.str().c_str());

  cout << endl;
  cout << "TSB input file to be read: " << inStream1.str() << endl;

  string tempAtomName, tempAtomType, tempElementType;
  struct vect tempPosition;
  double tempCharge;

  for (int i=0; i<nAtomsPerTSB; i++)
  {
    in1 >> tempAtomName >> tempAtomType >> tempPosition.x >> tempPosi-
tion.y >> tempPosition.z >> tempElementType >> tempCharge;
    Atom[i] = new AtomParams(tempAtomName, tempAtomType, tempPosition,
tempElementType, tempCharge);
  }

  cout << "TSB input file read and stored in class AtomParams/Atom." <<
endl;

    ////////////////////////////////////////////////
    // Determine bonding order for the TSB molecule //
    ////////////////////////////////////////////////
  struct vect position1, position2, distance;
  for (int i=0; i<nAtomsPerTSB; i++)
  {
    for (int j=i+1; j<nAtomsPerTSB; j++)
    {
      position1 = Atom[i]->getPosition();
      position2 = Atom[j]->getPosition();
      distance = position2 - position1;

      if (position1.Norm(distance) < 1.6)
      {
       Atom[i]->addBond(Atom[j]->getAtomName());
      }
    }
  }

  cout << "Bonding order for TSB done." << endl;
  cout << endl;

    ////////////////////////////////////////////
    // Read guest input files (from Gaussian09) //
    ////////////////////////////////////////////
  stringstream inStream2;
  inStream2 << "../cpp-input/" << guest << "_cpp-input.dat";
  ifstream in2(inStream2.str().c_str());

  cout << "Guest input file to be read: " << inStream2.str() << endl;

  for (int i=nAtomsPerTSB; i<nAtomsTotal; i++)
  {
    in2 >> tempAtomName >> tempAtomType >> tempPosition.x >> tempPosi-
tion.y >> tempPosition.z >> tempElementType >> tempCharge;
    Atom[i] = new AtomParams(tempAtomName, tempAtomType, tempPosition,
tempElementType, tempCharge);
  }

  cout << "Guest input file read and stored in class AtomParams/Atom." <<
endl;
```

```
    ////////////////////////////////////////////////
    // Determine bonding order for the guest molecule //
    ////////////////////////////////////////////////
  for (int i=nAtomsPerTSB; i<nAtomsPerGuest; i++)
  {
    for (int j=i+1; j<nAtomsTotal; j++)
    {
      position1 = Atom[i]->getPosition();
      position2 = Atom[j]->getPosition();
      distance = position2 - position1;

      if (position1.Norm(distance) < 1.6)
      {
       Atom[i]->addBond(Atom[j]->getAtomName());
      }
    }
  }

  cout << "Bonding order for " << guest << " done." << endl;
  cout << endl;

    ////////////////////////////
    // Build lattice vectors //
    ////////////////////////////
  struct vect a1, a2, a3, b1, b2, b3;
  double a = sqrt(3.0)*1.42;
  double triAngle = 60.0*(PI/180.0);

    // Define graphene lattice vectors
  a1.x = a;
  a1.y = 0.0;
  a1.z = 0.0;
  a2.x = a*cos(triAngle);
  a2.y = a*sin(triAngle);
  a2.z = 0.0;
  a3 = (1.0/3.0)*(a1+a2);

    // Define TSB lattice vectors
  b1 = n1*a1 + n2*a2;
  b2 = (-1.0)*n2*a1 + (n1+n2)*a2;
  b3 = (-1.0/3.0)*(b1+b2);

    // Use boundaries defined by TSB lattice vectors to calculate atom
per graphene layer
  int nAtomsPerLayer = countGrapheneAtoms(a1, a2, a3, b1, b2, b3);
    // Create class for graphene atoms
  AtomParams *Graphene[nAtomsPerLayer];

  cout << "Graphene parameters read and stored in class
AtomParams/Graphene." << endl;

    // Call to function to go back through grid and assign positions and
atom names to the graphene atoms
  assignGrapheneParams(Graphene, a1, a2, a3, b1, b2, b3);

    ////////////////////////////////////////////////
    // Determine bonding order for graphene layer //
    ////////////////////////////////////////////////
  struct vect positionG1, positionG2;
  for (int i=0; i<nAtomsPerLayer; i++)
  {
    stringstream ss;
```

```
    // Assign naming scheme for graphitic carbon atoms

    string atomName=ss.str();

    for (int j=i+1; j<nAtomsPerLayer; j++)
    {
      positionG1 = Graphene[i]->getPosition();
      positionG2 = Graphene[j]->getPosition();
      distance = Graphene[j]->getPosition() - Graphene[i]->getPosition();

      if (positionG1.Norm(distance) < 1.6)
      {
       stringstream ss;

       // Assign naming scheme for graphitic carbon atoms

       string nextAtomName=ss.str();
      Graphene[i]->addBond(nextAtomName);
      }
    }
  }

  cout << "Bonding order for graphene done." << endl;
  cout << endl;

  writeTOP(Atom, Graphene, nCarbon, nQuarter, guest, guestSegName,
nAtomsPerTSB, nAtomsPerGuest, nAtomsPerLayer);

  return 0;
}
```

C.3 PSFGEN: Building Simulation-Ready .PDB and .PSF Files

With separate .PDB and .TOP files for each segment, they must all be combined using PSFGEN in order to be fed into the simulation. PSFGEN is run in the VMD TclTk Console with the command "`source pgnfile`". The scripts available are:

- **a_TSB+graphite.pgn**
- **b_TSB+guest+graphite.pgn**

What this script does is combine all of the separate segments into one .PDB and one .PSF. The .PSF file contains the same information as the .TOP file, but extrapolated for the entire system.

C.3.1 Example Tcl Script: b_TSB+guest+graphite.pgn

```
package require psfgen

set nCARBON [lindex $argv 0]
set nQUARTERS [lindex $argv 1]
set GUEST [lindex $argv 2]
set GUESTSEGNAME [lindex $argv 3]

set QUARTERS 0${nQUARTERS}-quarter
set TSBDIR ../../02_cpp-writer/${QUARTERS}/tsb35-
c${nCARBON}_${QUARTERS}_TSB
set GUESTDIR ../../02_cpp-writer/${QUARTERS}/tsb35-
c${nCARBON}_${QUARTERS}_${GUESTSEGNAME}
set GRAPDIR ../../02_cpp-writer/${QUARTERS}/tsb35-
c${nCARBON}_${QUARTERS}_LAY

topology ../../02_cpp-writer/${QUARTERS}/tsb35-
c${nCARBON}_${QUARTERS}_with-${GUEST}_TOP.top

pdb ${TSBDIR}A-PDB.pdb
segment TSBA {pdb ${TSBDIR}A-PDB.pdb}
coordpdb ${TSBDIR}A-PDB.pdb TSBA

pdb ${TSBDIR}B-PDB.pdb
segment TSBB {pdb ${TSBDIR}B-PDB.pdb}
coordpdb ${TSBDIR}B-PDB.pdb TSBB

pdb ${GUESTDIR}-PDB.pdb
segment ${GUESTSEGNAME} {pdb ${GUESTDIR}-PDB.pdb}
coordpdb ${GUESTDIR}-PDB.pdb ${GUESTSEGNAME}

pdb ${GRAPDIR}1-PDB.pdb
segment LAY1 {pdb ${GRAPDIR}1-PDB.pdb}
coordpdb ${GRAPDIR}1-PDB.pdb LAY1

pdb ${GRAPDIR}2-PDB.pdb
segment LAY2 {pdb ${GRAPDIR}2-PDB.pdb}
coordpdb ${GRAPDIR}2-PDB.pdb LAY2

pdb ${GRAPDIR}3-PDB.pdb
segment LAY3 {pdb ${GRAPDIR}3-PDB.pdb}
coordpdb ${GRAPDIR}3-PDB.pdb LAY3

pdb ${GRAPDIR}4-PDB.pdb
segment LAY4 {pdb ${GRAPDIR}4-PDB.pdb}
coordpdb ${GRAPDIR}4-PDB.pdb LAY4

pdb ${GRAPDIR}5-PDB.pdb
segment LAY5 {pdb ${GRAPDIR}5-PDB.pdb}
coordpdb ${GRAPDIR}5-PDB.pdb LAY5

pdb ${GRAPDIR}6-PDB.pdb
segment LAY6 {pdb ${GRAPDIR}6-PDB.pdb}
coordpdb ${GRAPDIR}6-PDB.pdb LAY6

writepdb ../../04_NAMD/${QUARTERS}/tsb35-
c${nCARBON}_${QUARTERS}_with-${GUEST}_simPDB.pdb
```

```
writepsf ../../04_NAMD/${QUARTERS}/tsb35-
c${nCARBON}_${QUARTERS}_with-${GUEST}_simPSF.psf

writepdb tsb35-c${nCARBON}_${QUARTERS}_with-${GUEST}_simPDB.pdb
writepsf tsb35-c${nCARBON}_${QUARTERS}_with-${GUEST}_simPSF.psf

# Generate fixedAtoms file

mol load pdb tsb35-c${nCARBON}_${QUARTERS}_with-
${GUEST}_simPDB.pdb
mol addfile tsb35-c${nCARBON}_${QUARTERS}_with-${GUEST}_simPSF.psf

set all [atomselect top all]

set overlayer [atomselect top "segname TSBA TSBB ${GUESTSEGNAME}"]
set graphite [atomselect top "segname LAY1 LAY2 LAY3 LAY4 LAY5
LAY6"]

$all set beta 0
$graphite set beta 1

$all writepdb tsb35-c${nCARBON}_${QUARTERS}_with-
${GUEST}_fixedPDB.pdb
$all writepdb ../../04_NAMD/${QUARTERS}/tsb35-
c${nCARBON}_${QUARTERS}_with-${GUEST}_fixedPDB.pdb
```

C.4 NAMD2 Configuration Files

With the system-wide .PDB and .PSF ready, we lastly define the bonded and non-bonded potential parameters through the .PARAMS file.

- **tsb35-cn_charmm.params**

The .PARAMS file speaks with the .PDB and .PSF files to define interactions in the simulation. The NAMD2 configuration file now ties all of these files together, and defines any additional parameters for the system's environment (e.g. cutoff, PME, rigidBonds) to execute the simulation.

- **00-a_NAMD-starter_TSB+graphite_with-min.conf**
- **00-b_NAMD-starter_TSB+guest+graphite_with-min.conf**
- **01-a_NAMD-continue_TSB+graphite_new-temp.conf**
- **01-b_NAMD-continue_TSB+guest+graphite_new-temp.conf**

- **02-a_NAMD-continue_TSB+graphite_same-temp.conf**
- **02-b_NAMD-continue_TSB+guest+graphite_same-temp.conf**

The “00” label indicates that the system is not minimized yet. The “01” label indicates that the system has been minimized and equilibrated, but we now wish to change the temperature. The “02” label indicates that we are continuing an equilibration, without changing the temperature.

C.4.1 Example NAMD2 CONF File:

00-b_NAMD-starter_TSB+guest+graphite_with-min.conf

```
############################
# USER INPUT, FILL BY HAND
############################
set nCARBONS  ;# 6,8,10,12,14
set nQUARTERS ;# 0,1,2,3,4
set TIME  ;# total time at end of this simulation(ie: 06)

set MOLECULE tsb35-c${nCARBONS}
set GUEST
set LATTICE ${MOLECULE}_0${nQUARTERS}-quarter_with-${GUEST}

set temperature  ;# in Kelvin

set minSteps ;# number of steps in minimization
set runSteps ;# number of steps in this simulation
############################

timestep 1.0
firsttimestep 0
stepspercycle 4

cutoff 10.0
switching on
switchdist 8.0
pairlistdist 12.0
margin 1.0

coordinates  ../../${LATTICE}_simPDB.pdb
structure ../../${LATTICE}_simPSF.psf
parameters ../../../CHARMM-parameters/${MOLECULE}_charmm.params
paraTypeCharmm on

if {1} {
```

```
if {$nQUARTERS == 0} {
  cellBasisVector1 152.490   8.520   0.000
  cellBasisVector2 68.866   136.320   0.000
  cellBasisVector3 0.000   0.000   100.000
  cellOrigin
} elseif {$nQUARTERS == 1} {
  cellBasisVector1 156.179   10.650   0.000
  cellBasisVector2 68.866   140.580   0.000
  cellBasisVector3 0.000   0.000   100.000
  cellOrigin
} elseif {$nQUARTERS == 2} {
  cellBasisVector1 159.868  12.780    0.000
  cellBasisVector2 68.866   144.840   0.000
  cellBasisVector3 0.000   0.000   100.000
  cellOrigin
} elseif {$nQUARTERS == 3} {
  cellBasisVector1 163.558 14.910    0.000
  cellBasisVector2 68.866   149.100   0.000
  cellBasisVector3 0.000   0.000   100.000
  cellOrigin
} elseif {$nQUARTERS ==4} {
  cellBasisVector1 167.247  17.040   0.000
  cellBasisVector2 68.866   153.360   0.000
  cellBasisVector3 0.000   0.000   100.000
  cellOrigin 79.912  57.515  0.000
}
} else {
  puts "Cell basis vectors not set."
{
}

wrapAll on
wrapNearest on

exclude scaled1-4
1-4scaling 0.4

temperature $temperature
rescaleFreq 10
rescaleTemp $temperature

CoMmotion no
rigidBonds all

fixedAtoms on
fixedAtomsFile ../../${LATTICE}_fixedPDB.pdb
fixedAtomsCol B

outputname ${LATTICE}_${temperature}K_${TIME}ns_OUTPUT
binaryoutput no
outputEnergies 1000

restartname  ${LATTICE}_${temperature}K_${TIME}ns_RESTART
restartfreq 5000

DCDfile ${LATTICE}_${temperature}K_${TIME}ns_DCD.dcd
DCDfreq 1000
```

```
minimize $minSteps
run $runSteps
```

C.4.2 Example NAMD2 Configuration File:

01-b_NAMD-continue_TSB+guest+graphite_new-temp.conf

```
############################
# USER INPUT, FILL BY HAND
############################
set nCARBONS   ;# 6,8,10,12,14
set nQUARTERS   ;# 0,1,2,3,4
set TIME   ;# total time at end of this simulation (ie: 06)

set MOLECULE tsb35-c${nCARBONS}
set GUEST
set LATTICE ${MOLECULE}_0${nQUARTERS}-quarter_with-${GUEST}

set temperature  ;# new temperature (in Kelvin)

set RESTARTTIME ;# total time of previous simulation (ie: 12)
set RESTARTTEMP ;# previous temperature

set runSteps  ;# numsteps for this simulation (ie: 6000000)
############################

timestep 1.0
firsttimestep 0
stepspercycle 4

cutoff 10.0
switching on
switchdist 8.0
pairlistdist 12.0
margin 1.0

coordinates  ../../${LATTICE}_simPDB.pdb
structure ../../${LATTICE}_simPSF.psf
parameters ../../../CHARMM-parameters/${MOLECULE}_charmm.params
```

```
paraTypeCharmm on

wrapAll on
wrapNearest on

exclude scaled1-4
1-4scaling 0.4

temperature $temperature
rescaleFreq 10
rescaleTemp $temperature

CoMmotion yes
rigidBonds all

fixedAtoms on
fixedAtomsFile  ../../${LATTICE}_fixedPDB.pdb
fixedAtomsCol B

outputname ${LATTICE}_${temperature}K_${TIME}ns_OUTPUT
binaryoutput no
outputEnergies 1000

restartname ${LATTICE}_${temperature}K_${TIME}ns_RESTART
restartfreq 5000

DCDfile ${LATTICE}_${temperature}K_${TIME}ns_DCD.dcd
DCDfreq 1000

if {1} {
set inputname
../../${RESTARTTEMP}K/${RESTARTTIME}ns/${LATTICE}_${RESTARTTEMP}K
_${RESTARTTIME}ns_RESTART
binCoordinates ${inputname}.coor
extendedSystem ${inputname}.xsc
#binVelocities ${inputname}.vel   ;# DO NOT USE temperature OR
reinitvels WITH THIS
}

reinitvels $temperature
run $runSteps
```

Appendix D

D.1 Dihedral Distribution

The dihedral distribution is built using the *measure dihed* command in TclTk, and the atom selection are based on the "atom names". For example, "set sel(6) [atomselect top "name CA3 CB3 CC3 CD3 CE3 CF3"]" selects the third carbon atom from the oxygen atom in each alkoxy chain in the monolayer. Next, the atom index of each atom is gotten and the dihedrals are measured. This is all output to file, where final analysis was done in Mathematica and Excel. The example script below is for the TSB3,5-C6 monolayer with optimal lattice size at 300 K after 12 ns of equilibration.

```
############################
# USER INPUT, FILL BY HAND
############################
set nCARBON   6
set nQUARTER   2

set MOLECULE   tsb35-c${nCARBON}
set LATTICE   0${nQUARTER}-quarter
set TEMP   300
set TIME   12
############################

mol new
../../../../04_NAMD/${LATTICE}/${MOLECULE}_${LATTICE}_simPS
F.psf
mol addfile
../../../../04_NAMD/${LATTICE}/${TEMP}K/${TIME}ns/${MOLECUL
E}_${LATTICE}_${TEMP}K_${TIME}ns_DCD.dcd waitfor all

set nFrames [molinfo top get numframes]
set frameSkip 1

set sel(1) [atomselect top "name CR12 CR14 CR22 CR24 CR32
CR34"]
set sel(2) [atomselect top "name CR13 CR15 CR23 CR25 CR33
CR35"]
set sel(3) [atomselect top "name O13 O15 O23 O25 O33 O35"]
set sel(4) [atomselect top "name CA1 CB1 CC1 CD1 CE1 CF1"]
set sel(5) [atomselect top "name CA2 CB2 CC2 CD2 CE2 CF2"]
```

```
set sel(6) [atomselect top "name CA3 CB3 CC3 CD3 CE3 CF3"]
set sel(7) [atomselect top "name CA4 CB4 CC4 CD4 CE4 CF4"]
set sel(8) [atomselect top "name CA5 CB5 CC5 CD5 CE5 CF5"]
set sel(9) [atomselect top "name CA6 CB6 CC6 CD6 CE6 CF6"]

set nTypes   9
set nChains    6

for {set i 1} {$i < [expr $nTypes+1]} {incr i} {
  set lst($i) [$sel($i) get index]
}

for {set i 1} {$i < [expr $nChains+1]} {incr i} {
  set outfile($i) [open "dihed${i}.dat" w]
}

for {set j 0} {$j < [llength $lst(1)]} {incr j} {
 for {set f 0} {$f < $nFrames} {incr f $frameSkip} {
  for {set i 1} {$i < [expr $nChains+1]} {incr i} {
   set dihed($i) [measure dihed [list [lindex $lst($i) $j]
[lindex $lst([expr $i+1]) $j] [lindex $lst([expr $i+2]) $j]
[lindex $lst([expr $i+3]) $j]] frame $f]
  }

 for {set i 1} {$i < [expr $nChains+1]} {incr i} {
  puts $outfile($i) $dihed($i)
 }
 }
}

for {set i 1} {$i < [expr $nChains+1]} {incr i} {
  close $outfile($i)
}
```

D.2 Guest Molecule Center of Mass Distribution

The guest molecule center of mass distribution is built using the *measure center* command in TclTk. The example script below is for the TSB3,5-C6 monolayer with optimal lattice size and benzene guest molecules at 300 K after 12 ns of equilibration.

```
############################
# USER INPUT, FILL BY HAND
############################
set nCARBON   6
set nQUARTER   2

set MOLECULE   tsb35-c${nCARBON}
set GUEST   benzene
set LATTICE   0${nQUARTER}-quarter
set TEMP   300
set TIME   12
############################

mol new ../../04_NAMD/${MOLECULE}_${LATTICE}_with-
${GUEST}_simPSF.psf
mol addfile
../../04_NAMD/${TEMP}K/${TIME}ns/${MOLECULE}_${LATTICE}_wit
h-${GUEST}_${TEMP}K_${TIME}ns_DCD.dcd waitfor all

set nFrames [molinfo top get numframes]
set eqStart 6000
set frameSkip 10

set benzene [atomselect top "segname BENZ" frame 0]
set nAtomsPerBENZ 12
set nBENZ [expr [$benzene num]/$nAtomsPerBENZ]

set outfile [open "zcom-list.dat" w]

for {set f $eqStart} {$f < $nFrames} {incr f $frameSkip} {
  for {set i 1} {$i < [expr $nBENZ+1]} {incr i} {
    set sel [atomselect top "segname BENZ and resid $i"
frame $f]

    set com [measure center $sel weight mass]
    set zcom [lindex $com 2]

    puts $outfile $zcom
  }
}

close $outfile
```

References

1. Majumder, M., Chopra, N., Andrews, R. & Hinds, B. J., Nanoscale hydrodynamics: enhanced flow in carbon nanotubes. *Nature* **438,** 44 (2005).
2. Férey, G., Mellot-Draznieks, C., Serre, C., Millange, F., Dutour, J., Surblé, S. & Margiolaki, I., A Chromium Terephthalate-Based Solid with Unusually Large Pore Volumes and Surface Area. *Science* **309,** 2040–2042 (2005).
3. Love, J.C., Estroff, L., Kriebel, J., Nuzzo, R., Whitesides, G., Self-Assembled Monolayers of Thiolates on Metals as a Form of Nanotechnology. *Chem. Rev.* **105**, 1103-1169 (2005).
4. Bellec, A., Arrigoni, C., Douillard, L., Fiorini-Debuisschert, C., Mathevet, F., Kreher, D., Attias A.-J., Charra, F., Formation of Hydroxyl-Functionalized Stilbenoid Molecular Sieves at the Liquid/Solid Interface on Top of a 1-decanol Monolayer. *Nanotechnology* **25**, 435604 (2014).
5. Tahara, K., Abraham, M., Igawa, K., Katayama, K., Oppel, I., Tobe, Y., Porous Molecular Networks Formed by the Self-Assembly of Positively-Charged Trigonal Building Blocks at the Liquid/Solid Interface. *Chem. Commun.* **50**, 7683 (2014).
6. Fiorini, C., Charra, F., Self-Assembly: Mastering Photonic Processes at Nanoscale. *Opto-Electron. Rev.* **18**(4), 376-383 (2010).
7. Mali, K., Schwab, M., Feng, X., Müllen, K., De Feyter, S., Structural Polymorphism in Self-Assembled Networkds of a Triphenylene Based Macrocycle. *Phys. Chem. Chem. Phys.* **15**, 12495 (2013).
8. Ha, N.T.N., Gopakumar, T.G., Hietschold, M., Polymorphism Driven by Concentration at the Solid-Liquid Interface. *J. Phys. Chem. C* **115**, 21743-21749 (2011).
9. Wang, X., Jiao, T., Zhang, Z., Chen, T., Liu, M., Wan, L., Wang, D., Structural Motif Modulation in 2D Supramolecular Assemblies of Molecular Dipolar Unit Tethered by Alkylene Spacer. *J. Phys. Chem. C* **117**, 16392-16396 (2013).
10. Tamaki, Y., Muto, K., Miyamura, K., Odd-Even Effect in the Surface Structure

of Alkyloxy-Substituted Anthraquinone on HOPG Observed by Scanning Tunneling Microscope. *Bull. Chem. Soc. Jpn.* **86**(3), 354-362 (2013).

11. Barth, J. V., Costantini, G. & Kern, K., Engineering atomic and molecular nanostructures at surfaces. *Nature* **437,** 671–679 (2005).
12. Schull, G., Douillard, L., Fiorini-Debuisschert, C., Charra, F., Mathevet, F., Kreher, D. & Attias, A.-J., Selectivity of Single-Molecule Dynamics in 2D Molecular Sieves. *Adv. Mater.* **18,** 2954–2957 (2006).
13. Schull, G., Ness, H., Douillard, L., Fiorini-Debuisschert, C., Charra, F., Mathevet, F, Kreher, D. & Attias A.-J., Single Atom Substitution for Marking and Motion Tracking of Individual Molecules by Scanning Tunneling Microscopy. *J. Phys. Chem. C* **112,** 14058–14063 (2008).
14. Arrigoni, C., Schull, G., Bléger, D., Douillard, L., Fiorini-Debuisschert, C., Mathevet, F., Kreher, D., Attias, A.-J. & Charra, F., Structure and Epitaxial Registry on Graphite of a Series of Nanoporous Self-Assembled Molecular Monolayers. *J. Phys. Chem. Lett.* **1,** 190–194 (2010).
15. Barth, J. V., Molecular architectonic on metal surfaces. *Annu Rev Phys Chem* **58,** 375–407 (2007).
16. Kikkawa, Y., Kihara, H., Takahashi, M., Kanesato, M., Balaban, T.S., Lehn, J., Two-Dimensional Structures of Anthracene Derivatives: Photodimerization and Host-Guest Chemistry. *J. Phys. Chem. B* **114**, 16718-16722 (2010).
17. Schmidt-Mende, L., Fechtenkötter, A., Müller, K., Moons, E., Friend, R.H. & MacKenzie, J.D., Self-Organized Discotic Liquid Crystals for High-Efficiency Organic Photovoltaics. *Science* **293,** 1119–1122 (2001).
18. Subramanian, V., Chang, P.C., Lee, J.B., Molesa, S.E. & Volkman, S.K., Printed organic transistors for ultra-low-cost RFID applications. *IEEE Transactions on Components and Packaging Technologies* **28,** 742–747 (2005).
19. Theobald, J. A., Oxtoby, N.S., Philips, M.A., Champness, N.R. & Beton, P.H., Controlling molecular deposition and layer structure with supramolecular surface assemblies. *Nature* **424,** 1029–1031 (2003).
20. Griessl, S. J. H., Lackinger, M., Jamitzky, F., Markert, T., Hietschold, M. & Heckl, W.M., Incorporation and Manipulation of Coronene in an Organic Tem-

plate Structure. *Langmuir* **20,** 9403–9407 (2004).

21. Stepanow, S., Lingenfelder, M., Dmitriev, A., Spillmann, H., Delvigne, E., Lin, N., Deng, X., Cai, C., Barth, J.V. & Kern, K., Steering molecular organization and host–guest interactions using two-dimensional nanoporous coordination systems. *Nat Mater* **3,** 229–233 (2004).
22. Gimzewski, J. K., Joachim, C., Schlitter, R.R., Langlais, V., Tang, H. & Johannsen, I., Rotation of a Single Molecule within a Supramolecular Bearing. *Science* **281,** 531–533 (1998).
23. Uda, M., Momotake, A., Arai, T., 1,3,5-Tristyrylbenzene Dendrimers: A Novel Model System to Explore Oxygen Quenching in a Highly Organized Environment. *Org. Biomol. Chem.* **1**, 1635-1637 (2003).
24. Xu, S., Zeng, Q., Lu, J., Wang, C., Wan, L., Bai, C., The Two-Dimensional Self-Assembled *n*-Alkoxy-Substituted Stilbenoid Compounds and Triphenylenes Studied by Scanning Tunneling Microscopy. *Surface Science* **538**, L451-L459 (2003).
25. Meier, H., Karpouk, E., Lehmann, M., Schollmeyer, D., Enkelmann, V., Guest-Host Systems of 1,3,5-Tristyrylbenzene. *Z. Naturforsch.* **58b**, 775-781 (2003).
26. Balandina, T., Tahara, K., Sändig, N., Blunt, M.O., Adisoejoso, J., Lei, S., Zerbetto, F., Tobe, Y., De Feyet, S., Role of Substrate in Directing the Self-Assembly of Multicomponent Supramolecular Networks at the Liquid-Solid Interface. *ACS Nano* **6**, 8381-8389 (2012).
27. Szabelski, P., Rzysko, W., Panczyk, T., Ghijsens, E., Tahara, K., Tobe, Y., De Feyter, S., Self-assembly of molecular tripods in two dimensions: structure and thermodynamics from computer simulations. *RSC Adv.* **3**, 25159-25165 (2013).
28. Ghijsens, E., Adisoejoso, J., Van Gorp, H., Destoop, I., Noguchi, A., Ivasenko, O., Tahara, K., Van der Auweraer, M., Tobe, Y., De Feyter, S., On the stability of surface-confined nanoporous molecular networks. *J. Chem. Phys.* **142**, 101932 (2015).
29. Lei, S., Tahara, K., Adisoejoso, J., Balandina, T., Tobe, Y., De Feyter, S., Towards two-dimensional nanoporous networks: crystal engineering at the solid-liquid interface. *Cryst. Eng. Comm.* **12**, 3369-3381 (2010).

30. Furukawa, S., Tahara, K., De Schryver, F.C., Van der Auweraer, M., Tobe, Y., De Feyter, S., Structural Transformation of a Two-Dimensional Molecular Network in Respone to Selective Guest Inclusion. *Angew. Chem. Int. Ed.* **46**, 2831-2834 (2007).
31. See Cramer, C.; *Essentials of Computational Chemistry: Theories and Models*, 2nd Ed., (Wiley, 2004).
32. Gaussian 09, Revision **C.01**, M. J. Frisch et al., Gaussian, Inc., Wallingford CT, 2009.
33. Møller, C., Plesset, M. S., Note on an Approximation Treatment for Many-Electron Systems. *Phys. Rev.* **46**, 618-622 (1934).
34. Bartlett, R.J., Many-Body Perturbation Theory and Coupled Cluster Theory for Electron Correlation in Molecules. *Ann. Rev. Phys. Chem.* **32**, 359 (1981).
35. Cizek, J., Calculation of Wavefunction Components in Ursell-Type Expansions Using Quantum- Field Theoretical Methods. *J. Chem. Phys.* **45**, 4256 (1966).
36. Crawford, T.D. and Schaefer, H.F., in Reviews in Computational Chemistry, vol. 14, Ed. by Lipkowitz, K.B. and Boyd, D.B. (Wiley, 1996).
37. Hehre, W.J., Stewart, R.F., and Pople, J.A., Self-Consistent Molecular-Orbital Methods. I. Use of Gaussian Expansions of Slater-Type Atomic Orbitals, *J. Chem. Phys.* **51**, 2567 (1969).
38. Vanommeslaeghe, K. & MacKerell, A. D., Automation of the CHARMM General Force Field (CGenFF) I: Bond Perception and Atom Typing. *J. Chem. Inf. Model.* **52,** 3144–3154 (2012).
39. Vanommeslaeghe, K., Raman, E. P. & MacKerell, A. D., Automation of the CHARMM General Force Field (CGenFF) II: Assignment of Bonded Parameters and Partial Atomic Charges. *J. Chem. Inf. Model.* **52,** 3155–3168 (2012).
40. Groszek, A.J., Selective Adsorption at Graphite/Hydrocarbon Interfaces. *Proc. Roy. Soc. Lond. A.* **314**, 473-498 (1970).
41. Firlej, L., Kuchta, B., Roth, M. W. & Wexler, C., Molecular simulations of intermediate and long alkanes adsorbed on graphite: tuning of non-bond interactions. *J Mol Model* **17,** 811–816 (2011).
42. Roth, M.-W., Kaspar, M., Wexler, C., Firlej, L. & Kuchta, B., Molecular dynam-

ics simulations of submonolayer hexane and pentane films on graphite. *Molecular Simulation* **36,** 326–333 (2010).

43. Wexler, C., Firlej, L., Kuchta, B. & Roth, M. W., Melting of hexane monolayers adsorbed on graphite: the role of domains and defect formation. *Langmuir* **25,** 6596–6598 (2009).
44. Roth, M. W., Pint, C. L. & Wexler, C., Phase transitions in hexane monolayers physisorbed onto graphite. *Phys. Rev. B* **71,** 155427 (2005).
45. Mitani, H. & Niizeki, K., An analysis of two-dimensional modulated structures on a triangular lattice in terms of a complex quadratic field. *J. Phys. C: Solid State Phys.* **20,** 1017 (1987).
46. Phillips, J. C., Braun, R., Wang, W., Gumbart, J., Tajkhorshid, E., Villa, E., Chipot, C., Skeel, R.D., Kalé, L. & Schulten, K., Scalable Molecular Dynamics with NAMD. *J. Comput Chem* **26,** 1781–1802 (2005).
47. Humphrey, W., Dalke, A. & Schulten, K., VMD: visual molecular dynamics. *J Mol Graph* **14,** 33–38, 27–28 (1996).
48. Andersen, H. Rattle: A "velocity" version of the shake algorithm for molecular dynamics calculations. *J. Comp. Phys.***52**, 24-34 (1983).
49. Bhattacharya, S. & Gubbins, K. E., Fast Method for Computing Pore Size Distributions of Model Materials. *Langmuir* **22,** 7726–7731 (2006).
50. Hoshen, J. & Kopelman, R., Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm. *Phys. Rev. B* **14,** 3438–3445 (1976).

VITA

Alexander St. John was born in the unincorporated town of Salem, Wisconsin. His curiosity and scientific mindset began when his grandfather, George St. John, would walk him through the yard, asking and encouraging questions, identifying the things that he saw around him, and lending him interesting books over the years. Alex began his higher education at UW-Parkside in Kenosha, WI, at the age of 18, when he enrolled in the chemistry program. Four semesters into college, he began taking physics courses, and was quickly won over by the scientific and mathematical rigor used in physics to tackle deep questions that arise from the seemingly most simple of issues. Throughout the physics curriculum, under the direction of Dr. Jeffrey Schmidt and Dr. David Bruning, Alex enjoyed theoretical physics, statistical mechanics, and exactly solvable problems.

Entering graduate school, Alex began with theoretical work on Coulomb drag with Dr. Giovanni Vignale, and soon became interested in the power of computational methods, and the leaps and bounds that technology has been making to solve many problems across many disciplines, utilizing the marriage of theoretical and experimental methods. Alex then began working with Dr. Carlos Wexler on many-body problems using computational methods, such as electronic structure calculation and molecular dynamics simulation.

Alex will now enjoy the development and implementation of deep machine learning algorithms for hyperspectral image recognition and statistical analysis in precision UAV agriculture as Chief Analytics Officer of Aerial Agriculture Inc.