8-2014

Protein Nano-Object Integrator: Generating Atomic-Style Objects for Use in Molecular Biophysics

Nicholas Smith
Clemson University, nsmith@g.clemson.edu

PROTEIN NANO-OBJECT INTEGRATOR: GENERATING ATOMIC-STYLE OBJECTS FOR USE IN MOLECULAR BIOPHYSICS

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Physics and Astronomy

---

by
Nicholas David Fenimore Smith
August 2014

Accepted by:
Dr. Emil Alexov, Committee Chair
Dr. Feng Ding
Dr. Hugo Sanabria

ABSTRACT

As researchers obtain access to greater and greater amounts of computational power, focus has shifted towards modeling macroscopic objects while still maintaining atomic-level details. The Protein Nano-Object Integrator (ProNOI) presented here has been designed to provide a streamlined solution for creating and designing macro-scale objects with atomic-level details to be used in molecular simulations and tools. To accomplish this, two different interfaces were developed: a Protein Data Bank (PDB), PDB-focused interface for generating regularly-shaped three-dimensional atomic objects and a 2D image-based interface for tracing images with irregularly shaped objects and then extracting three-dimensional models from these images. Each interface is dependent upon the C++ backend utility for generating the objects and ensures that the output is consistent across each program. The objects are exported in a standard PDB format which allows for the visualization and manipulation of the objects via standard tools available in Molecular Computational Biophysics.

DEDICATION

I would like to take this time to thank both my parents, David and Ruth Smith, and my research advisor, Dr. Emil Alexov, for their extraordinary support of my path through the graduate program here at Clemson University. They have both supported every decision I have made and have helped me succeed in spite of the trials I have faced. I am truly grateful for their care and support.

TABLE OF CONTENTS

TABLE OF FIGURES

INTRODUCTION

In the field of Computational Biophysics, much emphasis has been placed on developing and designing methods and software for modeling and visualizing macromolecules and nano-objects. Two general types of presenting macromolecular structures exist; the first being atomic-style objects which are comprised of a number of atoms with specific properties and are arranged in a specific manner in three dimensional (3D) space. The other method is called continuum modeling which focuses on continuous structures with a well-defined shape and macroscopic characteristics but without any atomic details. For Molecular Biophysics, atomic-style objects are preferred as they are natively supported by the tools used to process and simulate molecular structures (Cornell, Cieplak et al. 1995) (Mackerell 2004) (Phillips, Braun et al. 2005). Of particular interest is the DelPhi program, which is an application that calculates the electrostatic potential by solving the Poisson-Boltzmann Equation (PBE) (Li, Li et al. 2012). Electrostatics play a profound role in molecular biology since biological macromolecules are made of thousands or even millions of atoms, with different sizes and partial charges situated at short distances within the macromolecule. The importance of electrostatic interactions and energies is illustrated by the fact that many biological phenomena are predominantly electrostatic in origin such as the salt-dependence of binding (Talley, Ng et al. 2008) and folding (Tan and Chen 2011), pH-dependence (Alexov 2004), and pKa shifts in proteins (Alexov, Mehler et al. 2011) and RNAs (Tang, Alexov et al. 2007). The main obstacle in modeling electrostatics in biological systems is the presence of water (Baker and McCammon 2003). DelPhi is based on the continuum approach and solves the PBE via a Finite-Difference algorithm. In parallel, the DelPhi web server was developed which was aimed at providing easy access to electrostatic calculators

for biological systems that did not require existing computational infrastructure on the user's end (Sarkar, Witham et al. 2013). Many other electrostatic computational software packages exist, such as APBS (Baker, Sept et al. 2001), AMBER (Cornell, Cieplak et al. 1995), CHARMM (Mackerell 2004), and MIBPB (Chen, Chen et al. 2011), which all require an atomic-style presentation of the objects being modeled. In addition, all existing visualization packages, such as PyMol (Schrödinger 2010), Chimera (Pettersen, Goddard et al. 2004), and Jmol (Herraez 2006), also need atomic structures of the objects to be visualized. As the availability of significant computational power becomes available to researchers, scientists have been able to model larger and larger objects, including nano-objects, with atomic level details (Li, Petukh et al. 2013) (Baker, Sept et al. 2001) (Sarikaya, Tamerler et al. 2003) (Stone, Phillips et al. 2007). However, experimentally, it is very unlikely that atomic representations of large objects will be available anytime soon. So then, there is a need for converting these nano-objects into atomic-style representations for use in molecular simulations and visualization. This is what this research was intended to accomplish. It was done in two steps: first, by developing an atomic-style presentation for simple geometrical figures and second, by extending the approach to include irregularly shaped objects.

The first endeavor developed by the lab was the Protein Nano-Object Integrator known as ProNOI (Smith, Campbell et al. 2012). This program generated atomic structures in the form of Protein Data Bank (PDB) files for a handful of geometrical objects such as a cone, a parallelepiped, a sphere, and a cylinder. These objects could be used to represent a variety of scenarios occurring in molecular biophysics: the cone a tip of an atomic force microscope, the parallelepiped as a plate of a dielectric material such as glass or a membrane, the sphere as a

virus or spherical nanoparticle, and the cylinder as a subcellular microtubule or as a generalization of a long piece of DNA. The base program was written in C++, an object-oriented language, and integrated into both the DelPhi 2.0 webserver a(Smith, Witham et al. 2012) and its own standalone Java interface(Smith, Campbell et al. 2012). By developing a program that was able to generate and model such hybrid systems, this provided researchers with better tools for analyzing much more complex systems.

The second stage of ProNOI development was the incorporation of tubes and spheroids with the addition of a 2D image viewer for tracing purposes. These two new objects can be used to represent more realistic macroscopic objects within molecular and subcellular environments. This provides researchers with a tool that can generate scenes with objects that are close approximations of things such as cellular organelles, and it can even generate approximations of highly irregular objects such as the endoplasmic reticulum or the neuron by using splines and other geometric principles. In keeping with design principles, this new version of ProNOI, termed Image-based ProNOI, generates these objects in the same format as its previous version and seamlessly extends the current functions of the program.

DelPhi was also updated in the midst of this project. Previously, the code base was written in FORTRAN, a computing language that is remarkably fast but somewhat outdated as it does not embrace many of the modern programming paradigms and features available to researchers today. This prompted the lab to begin developing a C++ version of DelPhi which incorporated a number of unique features such as a scalable interface for managing and marshalling data, a modular architecture that would allow future developers to extend its functionality, and native support for ProNOI object generation integration. Once this development is finished, DelPhi will

begin relying on the C++ portion of ProNOI for modelling geometric objects and other macroscopic spaces for some of the more unique problems it is attempting to tackle.

SIGNIFICANCE

Currently in the field of computational biophysics, there is no visualization and modeling package able to handle objects without explicitly defined atomic details. This means that researchers are unable to visualize and simulate environments that may not have an atomic structure, such as a dielectric plate or the tip of an atomic force microscope, without significant efforts. This project is aimed at providing a streamlined solution for creating and designing macro-scale objects with atomic-level details. These objects would then be exported in a format that is easily recognizable by standard molecular viewers and visualization packages such as Jmol (Herraez 2006), VMD (Humphrey, Dalke et al. 1996), or Chimera (Pettersen, Goddard et al. 2004). This allows researchers to first design the macro-scale scene using these new tools and then place the biological macro-molecules or other predefined objects into this scene via tools that are commonly available in standard Biophysics software packages. It is important to note that the variability of the dimensions and the makeup of these common objects prevents a uniform the standardization of their structure, which, in turn, limits their availability to the research community. Since the project aims to provide researchers with tools for manipulating and scaling these objects according to their own specifications, standardization of the macro-objects' structure is not an issue as both the macroscopic dimensions and atomic-level properties can be redefined and rescaled using the tools developed here with relative ease.

The capabilities of typical Computational Biophysics tools, while varied and robust, have lacked functions which can handle scalable atomic objects. Most molecular viewers such as Jmol, VMD, Chimera, provide tools for translating and rotating proteins and other biomolecules in a scene. Other functions include protonating the residues of a biomolecule, creating an explicit water shell or box, sequence alignment and other homology modeling techniques, and direct downloads from protein databases for direct insertion of PDB structures into a scene (Dolinsky, Czodrowski et al. 2007) (Case, Cheatham et al. 2005) (Phillips, Braun et al. 2005; Biasini, Bienert et al. 2014). However, each of these viewers has not been able to provide scalable atomic structures for nano-structure simulations and would benefit greatly from this development.

While developing the PDB-based ProNOI, it was critical that, whenever the objects were altered, the differences between the original macro-object and the altered macro-object be visible to the user. This would allow the user to see how this change will affect the overall scene and to better utilize the sliders and property boxes to precisely manipulate the macro-object to meet their exact specifications. This entire process is done in real-time by presenting the user with a wireframe graphic representation which is later converted to a full atomic model once the user has finalized the changes to the scene.

Another point to note is that, while the primary focus of this work is rendering integrated biological and nano-scale objects, this methodology can be applied to a number of other disciplines. The primary physical problem addressed by the targeted programs is obtaining an electrostatic potential distribution throughout the space of the scene by solving the Poisson-Boltzmann, however, programs that are equipped to handle similar problems such as molecular diffusion, heat transfer, and other types of simulations that require atomic-level details could

benefit from these tools as well. By simply converting the output style of the objects to a more suitable format for the associated problem, new types of geometries and environments could be explored without requiring a massive effort on the part of the research community.

As DelPhi is converted into a C++ application, the C++ ProNOI object generation utilities have been integrated into the architecture of DelPhi and been made available for use. This integration of the ProNOI library into DelPhi will allow it to tackle a variety of new problems and pave the way for future developments. As the C++ conversion is finalized and tested, DelPhi will begin to branch out into new disciplines and begin applying its methods of grid-based finite difference to a new collection of simulations. By enabling DelPhi to internally handle environments comprised of biological macromolecules and nano-objects, this gives the program a significant edge over existing applications and programs.

METHODS

GENERATION OF ATOMIC-STYLE REGULAR GEOMETRIC OBJECTS (PDB-BASED PRONOI)

Here, the algorithms for generating an atomic-style presentation of the four regular geometrical shapes (parallelepiped, sphere, cone and cylinder) are presented and described. The variables required to construct each shape have been featured as the (a) subfigure of figures 1-4.

PARALLELEPIPED

The initial dimensions of the parallelepiped are built upon user's input which provides the coordinates on the reference corner and three adjacent vertices, as seen on Figure 1a. It is necessary for the parallelepiped to be filled with pseudo-atoms such that the distance between the pseudo-atoms be a constant scalar parameter, which can be varied to achieve the desired degree of resolution. As this would limit the number of points being constructed on the path of the line, it was found that a seam would form at the ends of the vectors and result in non-uniform objects. In order to correct this, the maximum number of points k_v allowed on the line $\vec{v}$ is calculated using integer division on the given resolution p. This is translated into a new spacing between the points, represented by p_v , that evenly distributed the points along the line by the following equations:

$$k_v = |\vec{v}|/p \quad \textbf{(1.1)}$$

$$p_v = |\vec{v}|/k_v \quad \textbf{(1.2)}$$

Each vector of the box is then normalized and multiplied by the new spacing to create an incremental vector. This is then used to create an array of points along the path of the original vector, and is also done for the original three vectors of the box. A different incremental vector is added to a copy of the array of points to extend the points over a single side of the box. This is repeated for each side of the box with several if-statements to eliminate duplicate points on the edges and corners. Once the arrays that define the shape are complete, a method for printing the points out in the proper PDB format was designed as can be seen in Figure 1a.

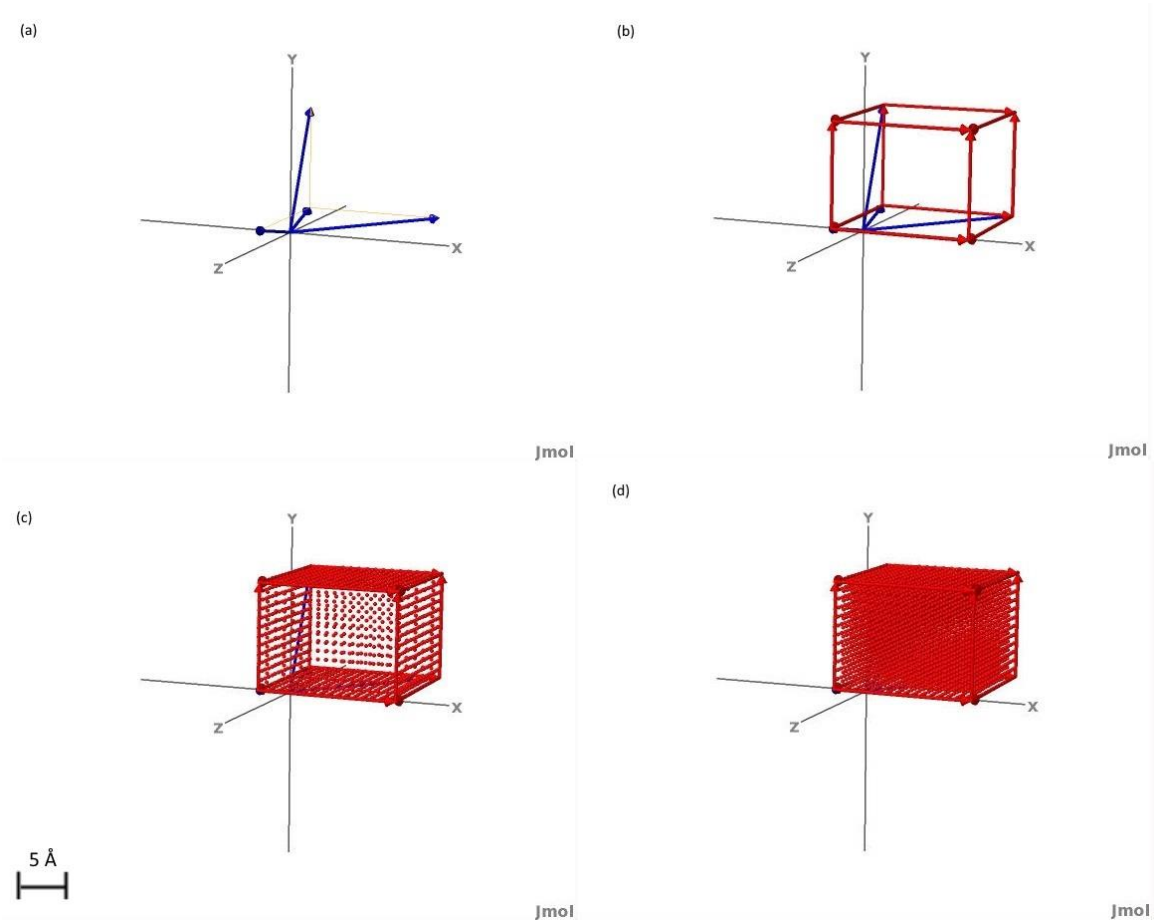

Figure 1 (a) The input parameters of the parallelepiped with the three base vectors and the origin (b) The generated sketch of the parallelepiped before point generation (c) The surface rendering of the parallelepiped (c) The solid rendering of the parallelepiped. These figures were generated with precision 1.0, origin vector (2,2,2), and vectors (2,2,10), (2,8,2), (10,2,2).

SPHERE

The user inputs the coordinates of the center of the sphere and its radius as in Figure 2a, which are then used for the pseudo-atomic filling of the resulting sphere. By translating to arc-length, the calculations that involve creating cyclic polygons to fit a maximum number of points inside a circle in the sphere were relaxed. The final process generated a sphere using a completely spherical coordinate system. First, the change in theta, $\Delta\theta$, is calculated by the following formula:

$$k_r = \pi r/p \quad \textbf{(2.1)}$$

$$\Delta\theta = \pi r/k_r \quad \textbf{(2.2)}$$

where k_r is the number of points allowed on the circle by the precision p and the radius of the circle r. Then, starting with the top of the sphere and by incrementing θ, a circle is generated for each value of θ between zero and π which then uses the above equation to generate an incremental value for φ using $2\pi r$ instead of πr. These incremental values are then looped over to generate an array of points that extended over the surface of the sphere. This process results in a uniform distribution of points across the surface of the shape with no singularities.

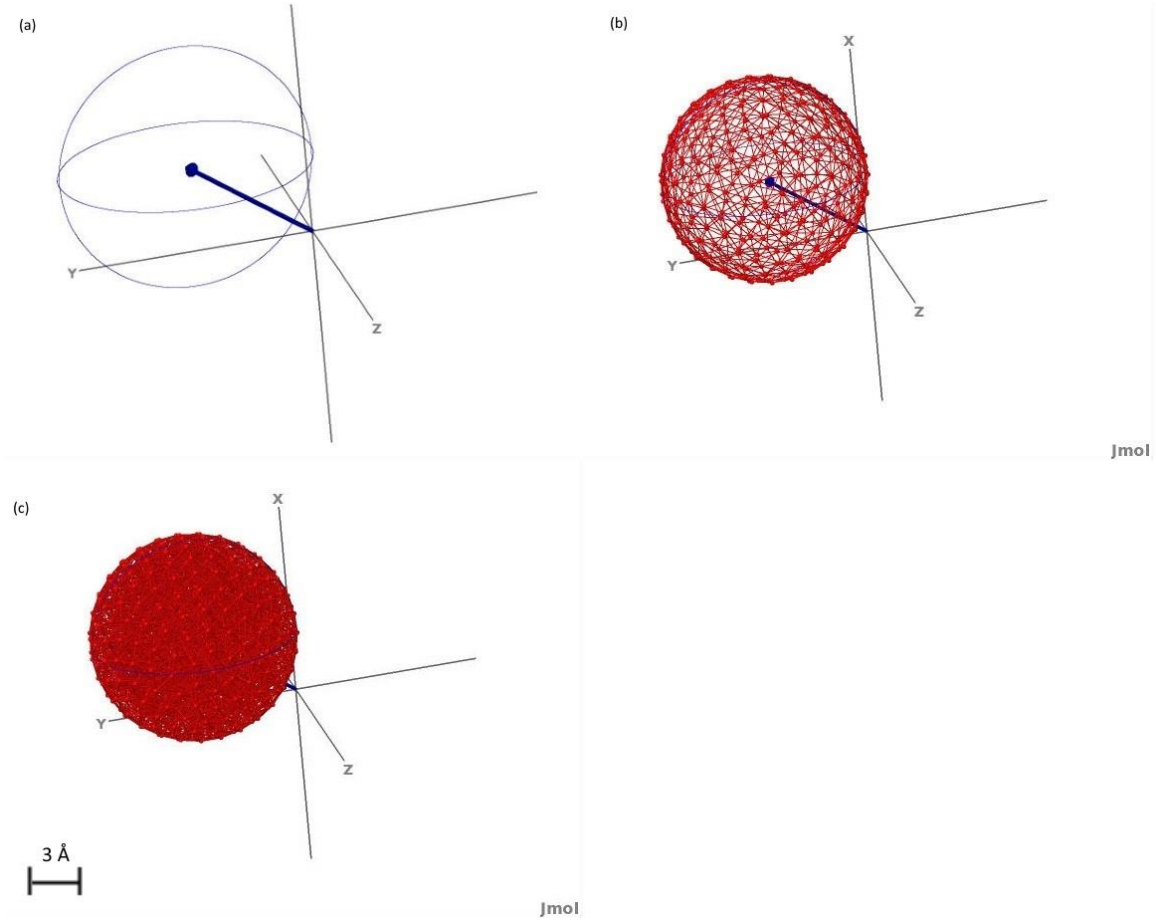

Figure 2 (a) The initial sketch of the sphere with the radius and origin (b) The surface rendering of the sphere (c) The solid rendering of the sphere. These figures were generated with precision 1.0, origin vector (6,6,6), and radius 5.0.

CYLINDER

Generally, the cylinder is constructed by generating circles along a vector for the body. A quaternion rotation algorithm handles rotation of desired shapes/sizes. The cylinder is constructed first by calculating the direction vector that pointed from the first origin to the second. This is then translated into spherical coordinates and copied; this clone is incremented by π/2 in the θ direction and its radius is set to the given radius of the cylinder. By cross multiplying with the original vector, a third normalized vector is created, and its radius set to the given radius. These latter two vectors form the plane normal to the direction vector and thereby normal to the cylinder itself, and the third vector is later used as the axis of rotation for the quaternion. The rotational quaternion *q* is formed using the following formula:

$$\cos(\theta) = \hat{z} \cdot \widehat{dir} \tag{3.1}$$

$$q = \cos(\theta/2) + \vec{v}\sin(\pi - \theta/2) \tag{3.2}$$

where $\hat{z}$ is the unit vector in the z direction, $\widehat{dir}$ is the unit vector pointing towards the direction vector, $\vec{v}$ is the axis of rotation (the third aforementioned vector), and θ is the angle between the z-axis and the direction vector. This quaternion rotates a given vector about the axis of rotation by θ. The rotation equation for rotating a vector $\vec{v}$ is:

$$\vec{v}' = q\ \vec{v}\ q^* \tag{4}$$

where q^* is the complex conjugate of the quaternion *q* and $\overrightarrow{v'}$ is the new rotated vector from the original vector. The cylinder is then built by generating circles of points centered along z-axis with spacing equal to the given resolution of the object between them. The top and bottom of the cylinder are generated by creating circles of varying radii. The radius for each new circle is calculated by transforming a vector that lay in the x-y plane with a magnitude of the given radius

into an incremental vector with a length equal to the given precision and then looping over this vector to create an ever-increasing radius up to the edge of the outer cylindrical ring. Once the set of circles are constructed, they are added to by the direction vector to create the top surface of the cylinder. Each of these points are then inserted into the above equation to find the new point in the cylinder's actual direction and written out to the resulting PDB file as rendered in Figure 3b.

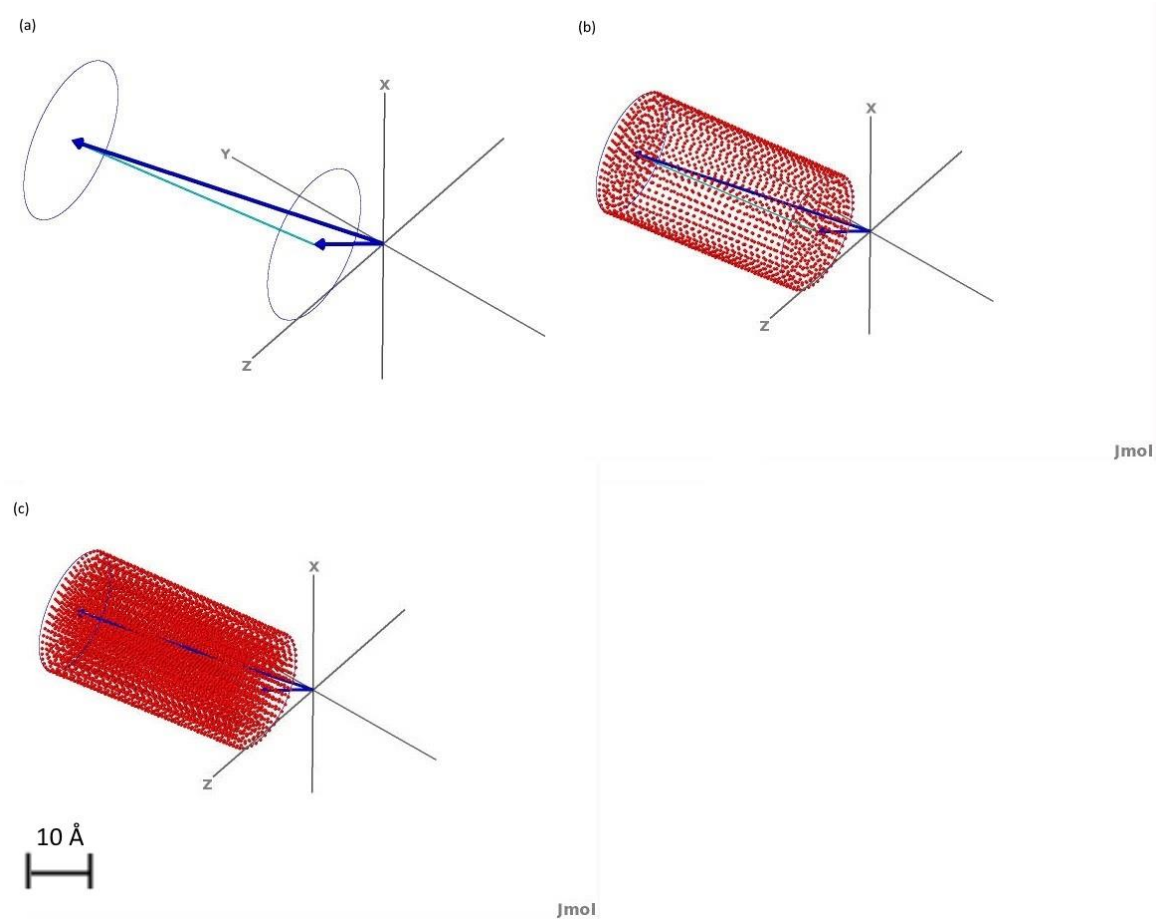

Figure 3 (a) The initial sketch of the cylinder with the radius, direction, and origin (b) The surface rendering of the cylinder (c) The solid rendering of the cylinder. These figures were generated with precision 1.0, origin vector (1,2,3), direction vector (9,10,11), and radius 4.0.

CONE

The user specifies the coordinates of the two origins and the opening angle in degrees which is exemplified in Figure 4a. The radius *r* of the cone is calculated using the following formula:

$$r = \left|\overrightarrow{dir}\right| \tan\theta \quad \textbf{(5)}$$

where $\widehat{dir}$ is the direction vector and ϑ is the opening angle. The direction and right angle vectors are calculated using the same steps as the cylinder as well as the rotational quaternion and its associated parameters. The difference lay in the construction of the outer rings of the cone. The radii of these rings are found by multiplying the sine of the opening angle by a new precision which is then subtracted incrementally from the radius of the bottom level. The new precision is determined by using the outer edge of the cone to redistribute the allowed number of points, similar to previous objects. At each level, the circle of points is formed in the x-y plane using the radius and a z-axis increment of cosine of the opening angle multiplied by the new precision, and rotated using the quaternion methods developed for the cylinder (Figure 4b).

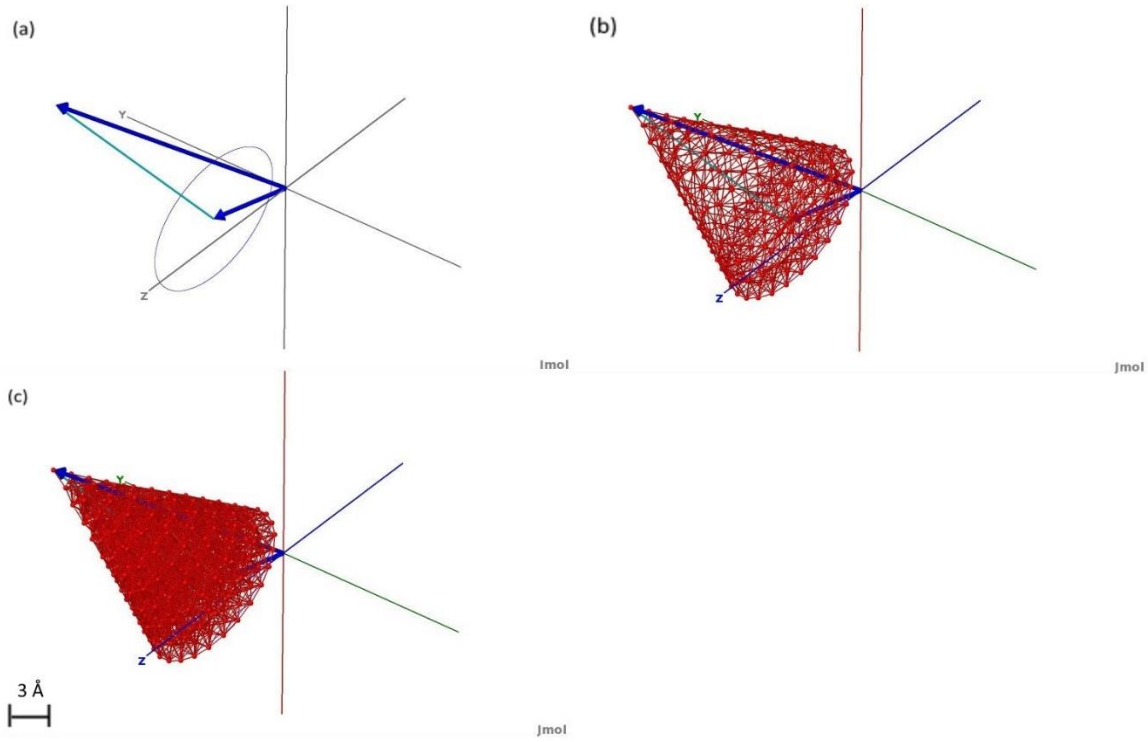

Figure 4 (a) The initial sketch of the cone with the opening angle, direction, and origin (b) The surface rendering of the cone (c) The solid rendering of the cone. These figures were generated with precision 1.0, origin vector (-1,2,3), direction vector (6,7,8), and opening angle of 23.7°.

GENERATION OF IRREGULAR GEOMETRIC OBJECTS (IMAGE-BASED PRONOI)

In the latest development, several new objects were designed in order to accurately represent the several new objects that did not conform to uniform shapes. These two new objects are a spheroid and a tube. The spheroid shape is drawn by rotating an ellipse about one of its axes and the tube can be thought of as a curve with the cross section of a circle. The target environment for these objects was an image, and so, the shapes were designed to be placed in the x-y plane. However, by using modern molecular viewers, it is trivial to move and rotate the output to any desired orientation or z-offset. These two shapes would form the basis of development for the Image-based ProNOI and were later incorporated into an expansion of the C++ object generation utilities.

SPHEROID

The spheroid is generated using only a handful of variables. Since the basic shape can be reduced to an ellipse, the only inputs required are the A-axis, the B-axis, the origin vector, and the angle of rotation in the x-y plane as can be seen from Figure 5a. It should be noted that the A-axis and B-axis parameters correspond to the semi major and semi minor axes of the ellipse, but, depending on the user's input, the semi major axis could be either the a-axis or the b-axis. One point of interest to note is that while the x-y plane cross section is an ellipse, the x-z plane cross section is a circle. So, rather than slicing the shape via the x-y plane by directly incrementing z, the circular cross section was divided into evenly spaced angle increments that were precision length apart using the same method that was developed for the PDB-based ProNOI (eq. 2). Each angle generated from this division of the circle was then used as the basis for an ellipse in the x-y plane. This new ellipse has its own A and B axes (termed A' and B') which can be derived from the general form of a spheroid:

$$\frac{x^2 + z^2}{A^2} + \frac{y^2}{B^2} = 1 \tag{6.1}$$

$$\frac{x^2}{A^2 - z^2} + \frac{y^2}{(1 - \frac{z^2}{A^2})B^2} = 1 \tag{6.2}$$

$$A' = \sqrt{A^2 - z^2} \tag{6.3}$$

$$B' = B\sqrt{1 - \frac{z^2}{A^2}} \tag{6.4}$$

$$A' = B'\frac{A}{B} \tag{6.5}$$

$$B' = B\ sin\theta \tag{6.6}$$

And since the spheroid's cross section in the x-z plane is a circle, the B' axis in Equation 6.6 can be easily identified from right angle methods. Once this ellipse's dimensions have been calculated, the problem then arises as to how to evenly space pseudo-atoms along the perimeter. In order to calculate the elliptical arc length and design a method similar to the circular point generation, it would require computing and approximating elliptical integrals of the second kind and involve several other integration tools and techniques not currently implemented in the ProNOI package. However, it is important to note that the desired outcome is not precise arc length but precise absolute distance between the pseudo-atoms placed on the ellipse. So, by using the equations for the parametric form of an ellipse:

$$\vec{p}(t) = \begin{cases} x(t) = A\sin(t) \\ y(t) = B\cos(\mathrm{t}) \end{cases} \tag{7}$$

And by approximating the location of a neighboring point by varying *t* and using the derivatives of the above equations, the location of the next point can be approximated and placed by using the iterative method shown here:

$$\vec{m}(t_i) = \begin{cases} x'(t_i) = Acos(t_i) \\ y'(t_i) = -Bsin(t_i) \end{cases} \tag{8.1}$$

$$t_{i+1} = t_i + \Delta t = t_i + \frac{precision - \|\vec{p}(t_i) - \vec{p}(t_0)\|}{\|\vec{m}(t_i)\|} \quad \textbf{(8.2)}$$

By looping through this process until Δt approaches zero (or less that 0.001 for PDB files), the next point can be spaced approximately precision length apart from the previous point $p(t_0)$. Once this next point is found, the process repeats until t approaches 2π and comes back to the start of the ellipse. This can then be repeated for each ellipse generated by the circle-plane slicing method, thereby generating a spheroid where no two pseudo-atoms are closer than the precision value specified as seen in Figure 5b. Now, this method only builds a hollow spheroid, but if both the A-axis and B-axis are decremented uniformly – such that the minimum distance between successive spheroids remains at precision – then a solid spheroid can be generated via multiple hollow spheroids as can be seen in Figure 5c.

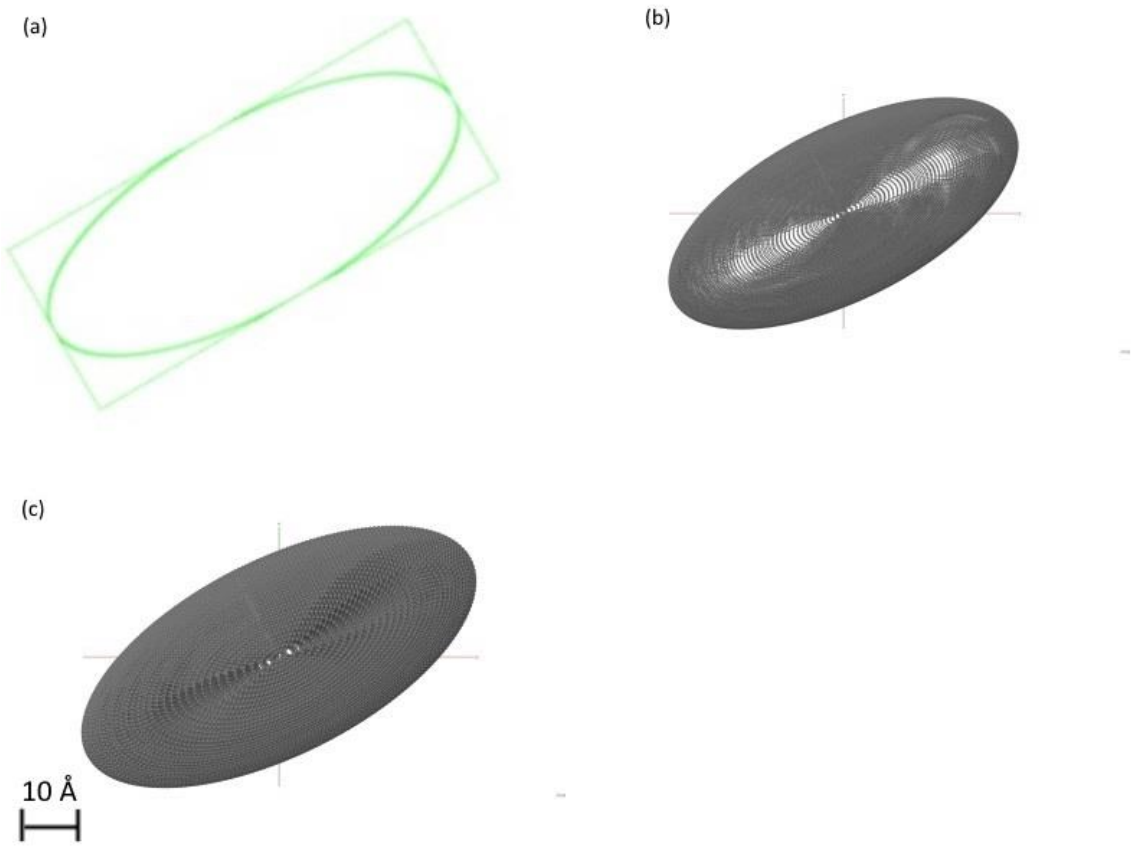

Figure 5 (a) initial trace of the spheroid as a 2D ellipse in the interface program with a bounding box for reference (b) The surface rendering of the spheroid (c) The solid rendering of the spheroid. These figures were generated with precision 1.0, origin vector (126,111,0), A and B axes as 88 and 39.5, and rotational angle as 27°.

TUBE

The second new shape generated by the Image-based ProNOI is the tube. This object can be reduced to a set of control points with a thickness, a bias, and a tension parameter to specify the behavior and appearance of the tube as shown in Figure 6a. The tube generation was accomplished through the use of Cubic Hermite splines to interpolate between the control points, circular rings to provide a surface for the shell of the tube, and rails of points to prevent torsion along the length of the tube (Catmull 1974).

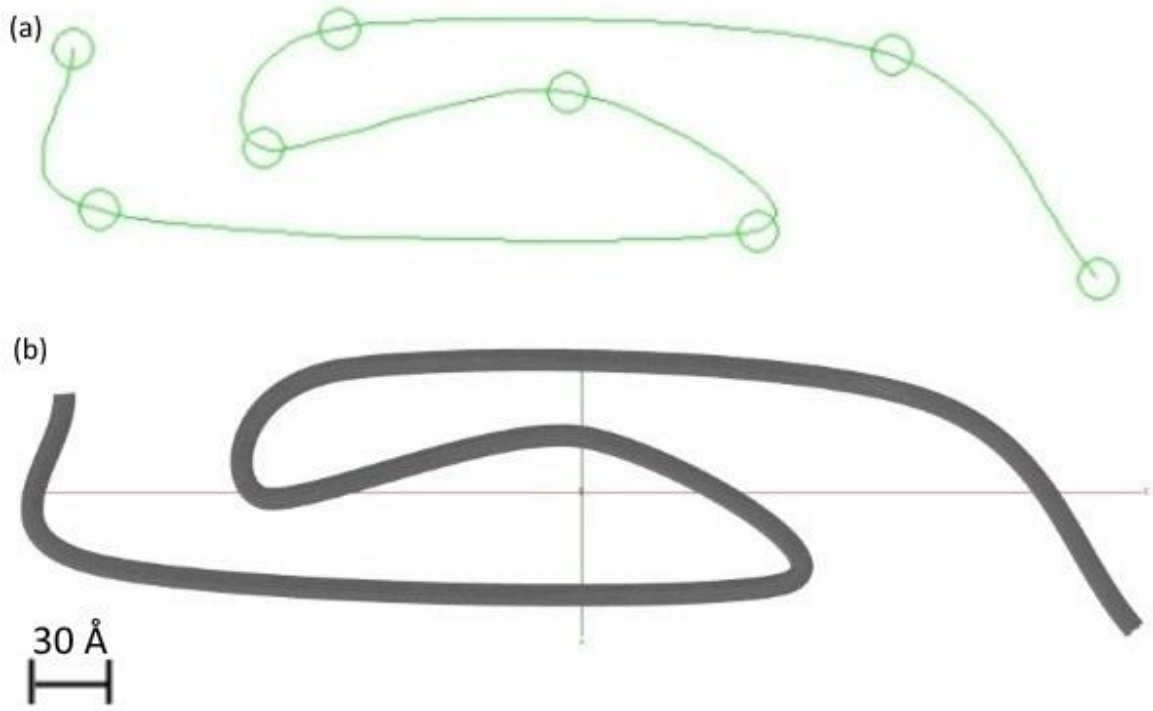

Figure 6 (a) The first visualization step inside the interface, each circle is a click from the user added as a control point (b) The final rendering of the tube in Jmol. These figures were generated with precision 1.0 Å, tube width of 10 Å, bias and tension of 0, and a collection of 8 points.

The cubic Hermite splines used for the curve interpolation use a basis of four piecewise polynomial functions to interpolate points between two predefined control points and their neighbors while maintaining the continuity of the curves and their first derivatives with their neighboring curves at the endpoints of the intervals. The four polynomial basis functions are:

$$h_{00}(t) = 2t^3 - 3t^2 + 1 \quad \textbf{(9.1)}$$

$$h_{10}(t) = t^3 - 2t^2 + t \quad \textbf{(9.2)}$$

$$h_{01}(t) = -2t^3 + 3t^2 \quad \textbf{(9.3)}$$

$$h_{11}(t) = t^3 - t^2 \quad \textbf{(9.4)}$$

They each use a parameter *t* that is defined to be on [0, 1] for interpolating successive points on the curve. These functions are then combined by utilizing two inner control points, between

which the spline will be generated, and two outer control points, necessary for determining the derivatives at the two inner points. This results in the following equation:

$$\vec{p}(t) = h_{00}(t)\overrightarrow{p_1} + h_{10}(t)\overrightarrow{m_1} + h_{01}(t)\overrightarrow{p_2} + h_{00}(t)\overrightarrow{m_2} \quad \textbf{(10)}$$

where $p_{1,2}$ represents the two inner control points and $m_{0,1}$ represents the approximate derivatives at those points. The derivatives were found by utilizing the following equations:

$$(\overrightarrow{m_1})_x = \frac{(x_1 - x_0)(1 + bias)(1 - tension)}{2} + \frac{(x_2 - x_1)(1 - bias)(1 - tension)}{2} \quad \textbf{(11.1)}$$

$$(\overrightarrow{m_2})_x = \frac{(x_2 - x_1)(1 + bias)(1 - tension)}{2} + \frac{(x_3 - x_2)(1 - bias)(1 - tension)}{2} \quad \textbf{(11.2)}$$

where $x_{0,1,2,3}$ are the four control points with $x_{1,2}$ representing the x-component of the inner control points and the bias and tension parameters are user-specified and control the overall shape and curvature of the spline. By recomputing the derivatives for each spatial component, these parameters are then fed into the interpolation equation, and, by precisely varying *t*, a uniform distribution of points can be interpolated along the interval (p_1, p_2). Now, since the tubes are expected to be placed within a 2D image, the above equations have only been generalized to a 2D plane as seen in Figure 6b, however, it is trivial to extend this method to three dimensions if necessary.

Once the Image-based ProNOI has been given a set of ordered control points, it then begins to interpolate evenly spaced points along the spline curve to act as guide points for the tube. It was

noted in earlier designs of this program that, unless the points were somewhat close together, the tube collapsed and deformed in region with high curvature. The additional control point generation serves to eliminate this malformation by over-specifying the system. The method used to distribute these points along the curve is similar to the spheroid's ellipse point generation as it uses derivative prediction and a similar iterative method to place points along the curve. By taking the derivative with respect to *t* of the four cubic Hermite basis functions:

$$h_{00}'(t) = 6t^2 - 6t \quad \textbf{(12.1)}$$

$$h_{10}'(t) = 3t^2 - 4t + 1 \quad \textbf{(12.2)}$$

$$h_{01}'(t) = -6t^2 + 6t \quad \textbf{(12.3)}$$

$$h_{11}'(t) = 3t^2 - 2t \quad \textbf{(12.4)}$$

The slope at each interpolated point can be found and used to approximately predict where the next point will lie, given a value for the change in *t*. So, by using the iterative method shown previously and replacing *p* and *m* with the values defined by the spline functions, guide points can be evenly distributed along the curves.

Once the guide points have been generated, a ring of points is then generated around each guide point in order to act as the rails upon which the actual tube points will be placed. Each ring is designed in the same fashion as the circle point distribution for the spheroid and has a diameter equal to the user-specified thickness. However, it is critical that each rail remains at the same location on the tube in order to prevent twisting and contorting the final spline curve. Therefore, as the process loops over each guide point, it creates a new collection of points in a ring precision-length apart in the x-y plane and then rotates this ring in the x-y plane so that the top point of the ring lines up with the spline's tangential direction. The ring is then rotated out

of the x-y plane so that the direction vector normal to the ring is parallel to the spline's tangential direction. Each point on the ring is then added to its own separate rail array for use in generating the actual tube pseudo-atoms in this next step.

Upon generating the ring of rail points around the initial guide points, the process then takes each rail array and, by using the spline interpolation shown earlier, generates a curve composed of evenly spaced points. This method ensures that each point on the tube will remain precision length apart even in areas with sharp curvature.

In order to generate a solid tube, the process described above is repeated with a precisely shrinking thickness around the same control points. This variable thickness ensures that the tubes are precision length apart without altering the shape of the resulting tube as can be seen

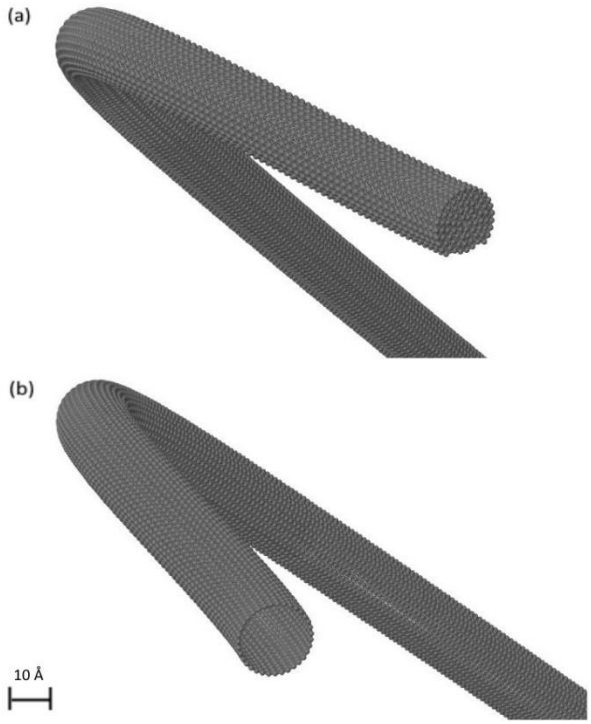

Figure 7 (a) Close-up of the solid tube generation (b) Close-up of the hollow tube generation. These figures were generated with precision 1.0 Å, tube width of 10 Å, bias and tension of 0, and a collection of 8 points.

in Figure 7a. The change in thickness is found by using the same line-point distribution method for the original PDB-based ProNOI methods and the change is bounded so that the last tube generated is a single line in the center of the solid tube.

VISUALIZATION METHODS

The PDB-based ProNOI was developed for modifying PDB files using a Java interface with an embedded Jmol Applet (see Figure 8). This allowed the program to bypass many of the typical visualization hurdles by using pre-developed tools in its design. This program handles the four regular objects: parallelepiped, sphere, cylinder, and cone, and allows for the insertion and modification of multiple objects within the loaded PDB. To optimize the object visualization process, each inserted object is first traced by utilizing the vector drawing methods within the Jmol applet controlled via sliders and text boxes assigned to each of the objects dimensional properties. This prevents excessive system calls to the C++ object generation tool and allows the user to easily align the atomic-style objects with the other objects in the scene without a large amount of computational overhead. Once properly placed, an object's atomic properties, such as atomic precision, dielectric constant, atomic radius, atom type, and object name, can then be altered to the user's specifications. Once the entire scene has been sketched out and configured, the objects can then be generated and the PDB updated to display the results of the C++ object generation utility immediately. If further modifications are needed, each object can be individually modified and tampered with and the scene regenerated. If this occurs, newly modified objects will be marked in the list to show that their properties have not been posted to the PDB. Once this has finished, the scene can be regenerated to display the new output of the

object generator.

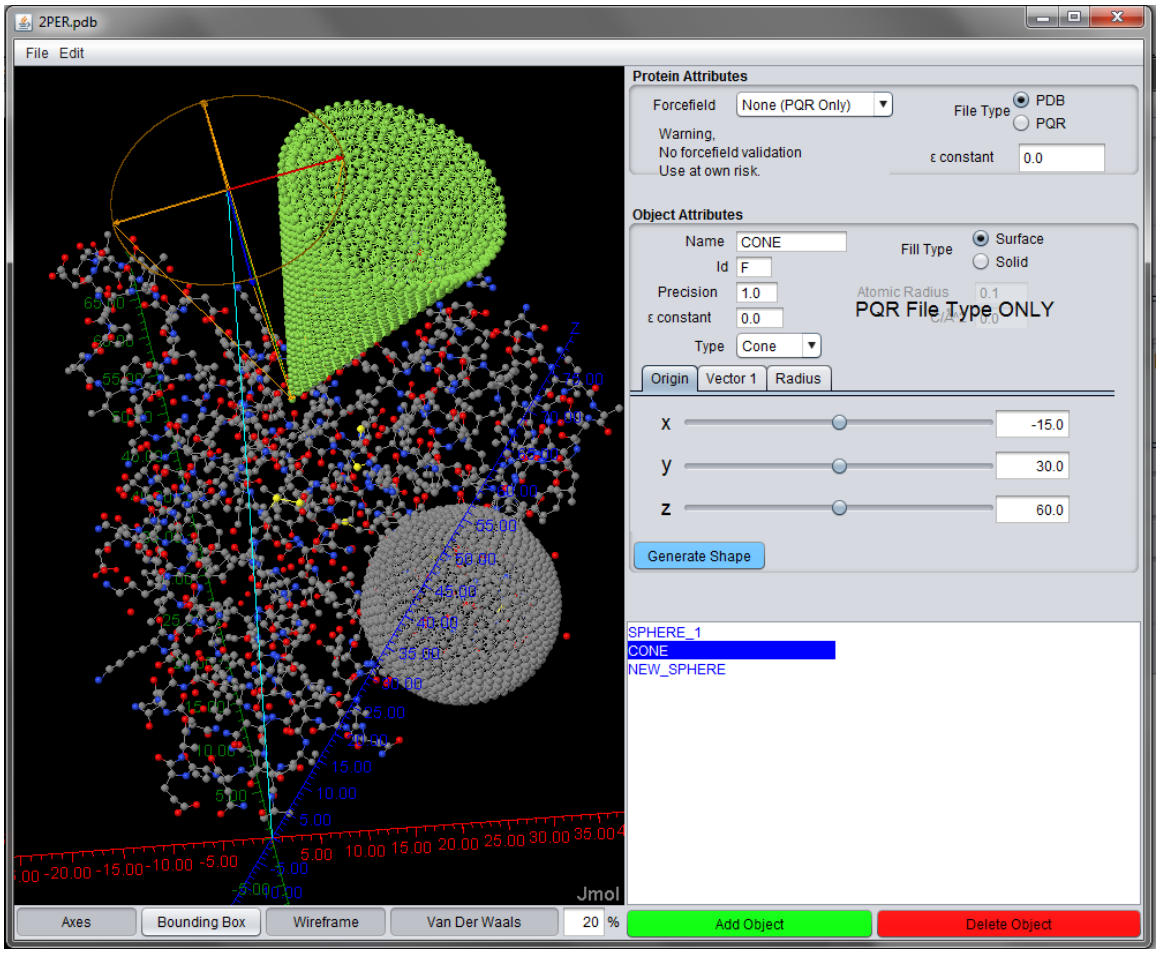

Figure 8 Screenshot of the PDB-based ProNOI with multiple objects rendered around a protein.

The Image-based ProNOI was developed for processing 2D images and converting them to 3D PDB models (see Figure 9). It uses the wxPython interface library for rendering the tools within the window and the manipulating the images. Several drawing methods have already been exposed in the library for tracing irregular objects such as spheroids and tube by using ellipses and splines, respectively. These simple drawing tools were later hooked up to sliders and text boxes in order to enable a higher degree of precision when tracing the objects. The designer also supports the creation and tracking of multiple objects within the image. Each ellipse can be

adjusted separately as the designer provides a fine degree of control over its shape, orientation, and location within the image. The splines are traced separately but can still be added and deleted freely. They have also been coupled with a simple undo button to help mitigate any mistakes the user might make.

It is important to note that the spline drawing function within the wxPython library does not represent a cubic Hermite spline. In order to better match the output from the object generation code, a spline interpolation method was developed that divided each interval into segments by slowly incrementing the parametric *t* parameter in the spline equations (eq. 9). Once these points were found, the interface then drew straight lines between the points on the interval, which proved to be an accurate approximation for the spline as long as the interval was divided into ten or more segments.

Once the objects within the image were traced and their properties assigned, the designer then sends this data to the object generation code to be processed. The data sent to the program, however, is scaled by a couple of parameters. The most important parameter is the image scale. As these images used in the program might not necessarily have all the same Angstroms per pixel, it was critical that the user be given a method for mapping the resolution of the image,

measured in pixels, onto the PDB environment, measured in Angstroms.

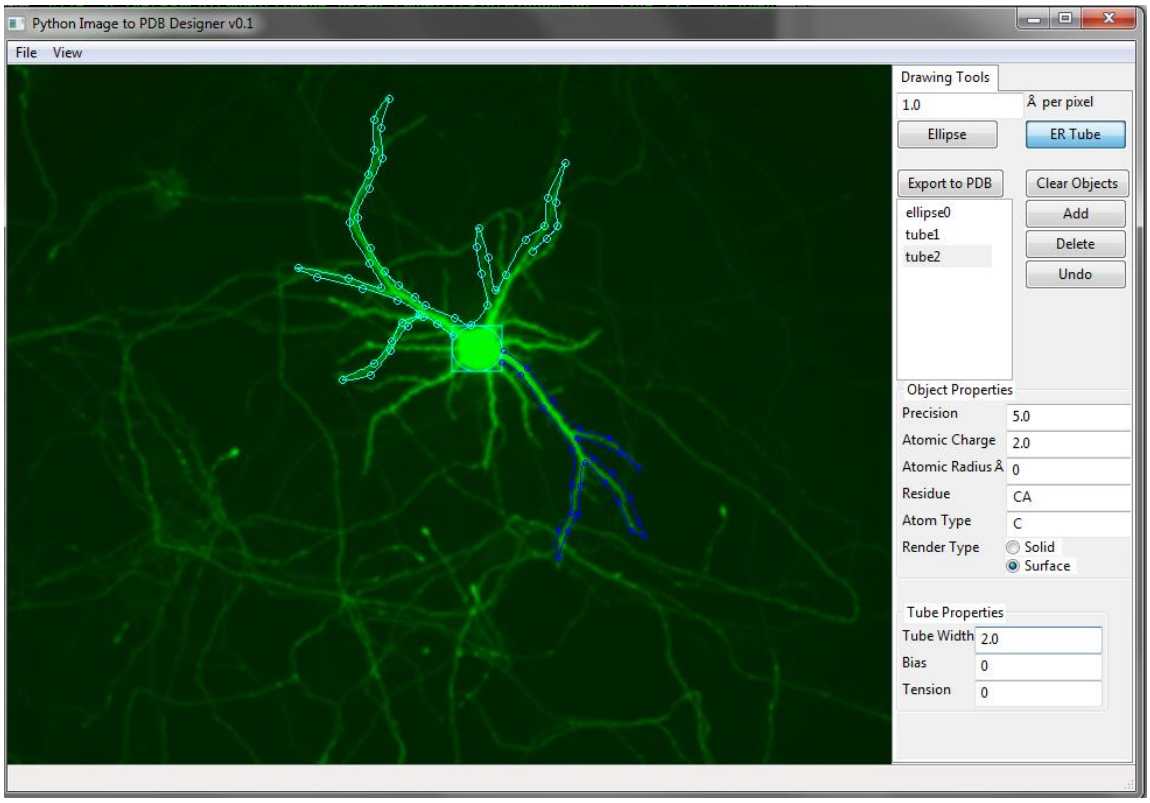

Figure 9 A screenshot of the Image-based ProNOI tracing a fluorescent neuron. Several tubes and an ellipse are present.

FUTURE DEVELOPMENTS

In future versions of the project, the lab plans on adding several useful features to the code depending on the success/reception of the initial release. The first major addition will be off-plane object generation for both tubes and ellipsoids. Currently, the tubes and ellipsoids that are traced in the image are centered in the x-y plane; later developments will add options for reorienting these shapes in 3D space and tools for accurately visualizing these changes. In addition, the PDB-based ProNOI will be updated to include these two new shapes.

Future versions of the Image-based ProNOI will attempt to automate the tracing portion of the interface by using tracing the outline of the objects in the image. This will then allow the program to fit either an ellipse or a tube into the outline to pass to the object generation code which will help prevent a significant portion of user error.

As DelPhi is further developed, the methods designed here will allow DelPhi to handle complex environments and situations resulting from the problems involving modeling biological macromolecules alongside cellular components and nano-objects. In addition, DelPhi is planning on expanding its toolset to be able to handle problems such as heat diffusion, dispersion, and other types of fundamental Molecular Biophysics problems. By allowing DelPhi to tackle a wider variety of environments, ProNOI will greatly enhance its utility and usability.

RESULTS

The results presented here demonstrate the level of similarity between the generated objects and some examples found in molecular biology, biochemistry, and nano-science. Here we focus on the geometrical presentation of the objects, rather on the physical characteristics. The reason for that is the complexity of the objects and the lack of analytical solution to compare with. It should be reiterated that we are aiming at modeling objects for which PDB structure does not exist and therefore comparison of electrostatic potential and energy calculated with DelPhi using the object and PDB structure is not available as well. However, in case of sphere and parallelepiped, where analytical solutions can be obtained, we provide comparison of electrostatic energy.

Each one of the objects was generated by ProNOI in less than a second on a low-end laptop. As the generation of the objects did not lead to any significant computational delay in the user interface and performed significantly faster than expected, no precise runtime measurements were taken. In addition, the issues complicating the rendering of some of these figures occurred when other visualization packages attempted to visualize the scenes generated by ProNOI. The number of atoms placed into the file was simply too large for the viewer to handle. This typically occurred at atom counts larger than 100,000. So, the current bottleneck in the process is the limits of the current visualization software.

SPHERE

The first image is simply for illustrating the similarities between the sphere rendering (Figure 10b) and the crystal structure of a virus (PDB ID: 3KIC) (Lerch, Xie et al. 2010) (Figure 10a). Very few viruses have PDB structure and modeling them as a sphere with a particular radius and charge distribution may be the only way of studying them computationally. While this representation may mask some of the finer details of the virus structure, simulations much larger than the virus itself will benefit greatly from this scaled-down version of the virus protein.

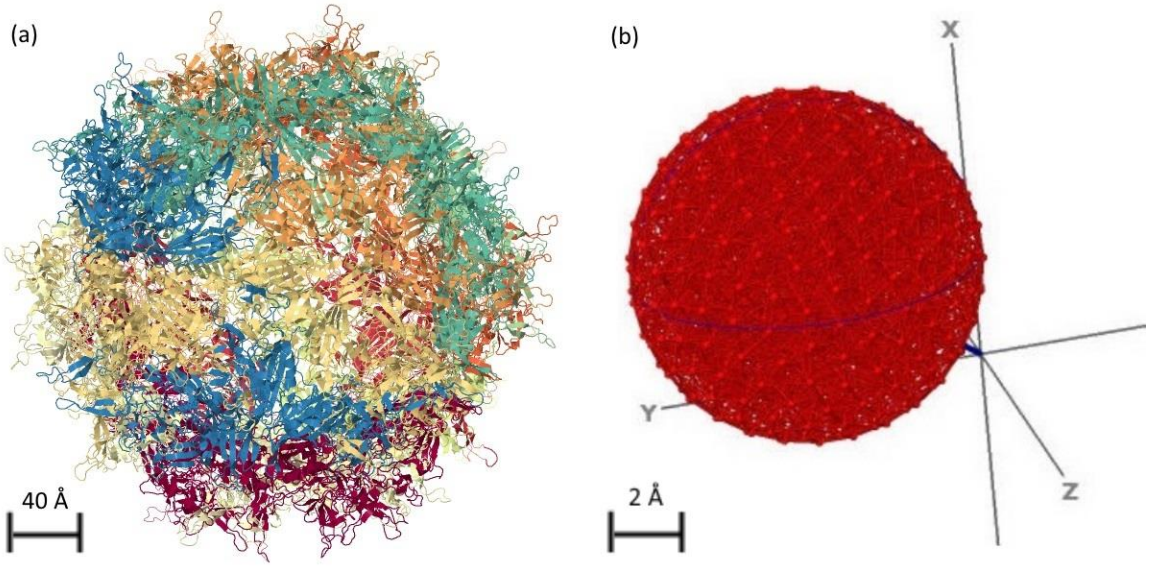

Figure 10 (a) adeno-associated virus serotype 3B (PDB ID 3KIC) (b) The sphere solid rendering shown previously.

As mentioned above, our focus is geometrical modeling. However, in the case of a sphere, an analytical solution for the electric potential of a continuous sphere does exist. This solution has been compared to the electric potential calculated through DelPhi using a ProNOI generated sphere with a radius of 2Å and a precision of 1Å (see Figure 11). As evidenced by the graph, the atomic-style object closely mirrors the continuous analytical solution at each point on the curve. This reaffirms the accuracy of the type of modeling chosen.

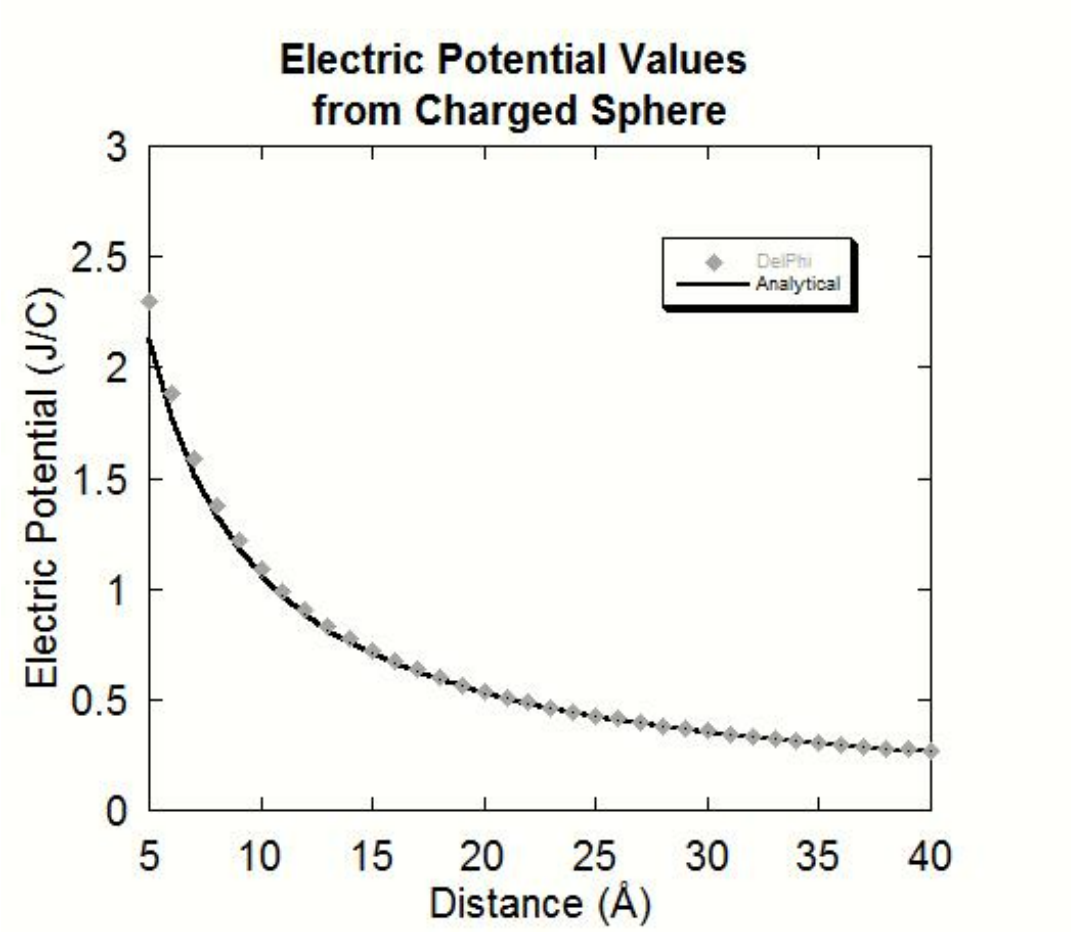

Figure 11 The analytical solution for the electric potential of a continuous sphere as compared to the results of the atomic-style sphere electric potential.

PARALLELEPIPED

In Figure 12, one of the featured POPE membranes from the ProBLM Webserver (Alexov, Kimmett et al. 2014) is shown next to the solid rendering of the parallelepiped from the PDB-based ProNOI. As stated above, in many cases the PDB file of membrane may not be available and the membrane should be modeled via a parallelepiped object. In the ProBLM Webserver, there is an option to use ProNOI to build a membrane with similar properties to the explicit membranes but with user-specified dimensions, allowing for greater control over the resulting PDB and subsequent simulations.

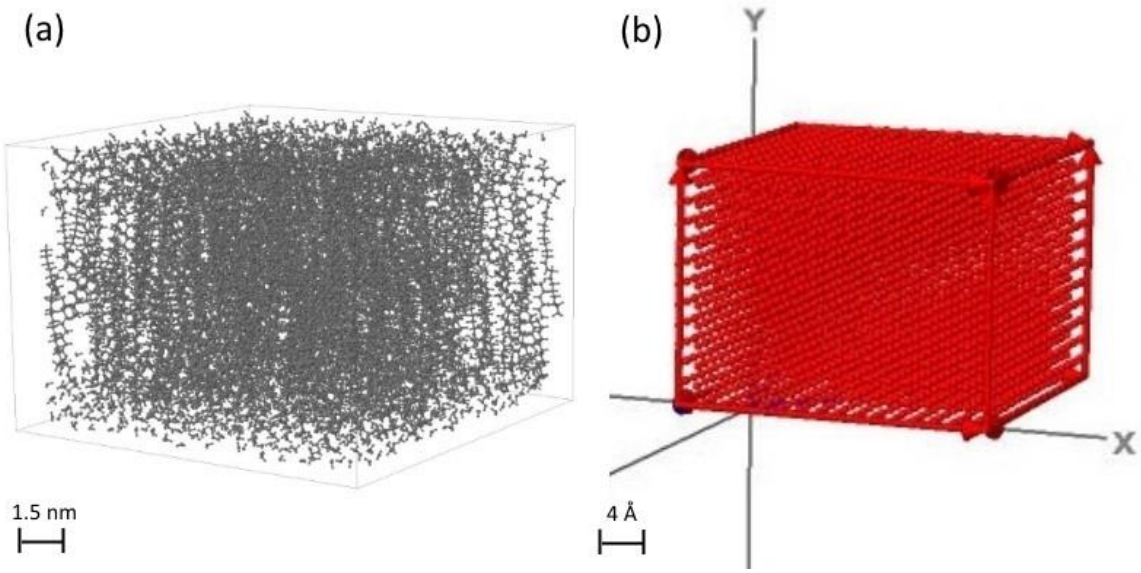

Figure 12 (a) *The POPE 75x75 membrane from the ProBLM Webserver (b) The solid rendering of the parallelepiped shown previously.*

In order to demonstrate the accuracy of this style of modeling, a series of experiments were performed to compare the solvation energies of a set of parallelepipeds as a spherical charge approached the objects with an older style of continuum parallelepiped modeling in DelPhi. The results of this simulation can be seen in Figure 13 and it is evident that the energies of the atomic-style objects closely mirror that of the continuum model.

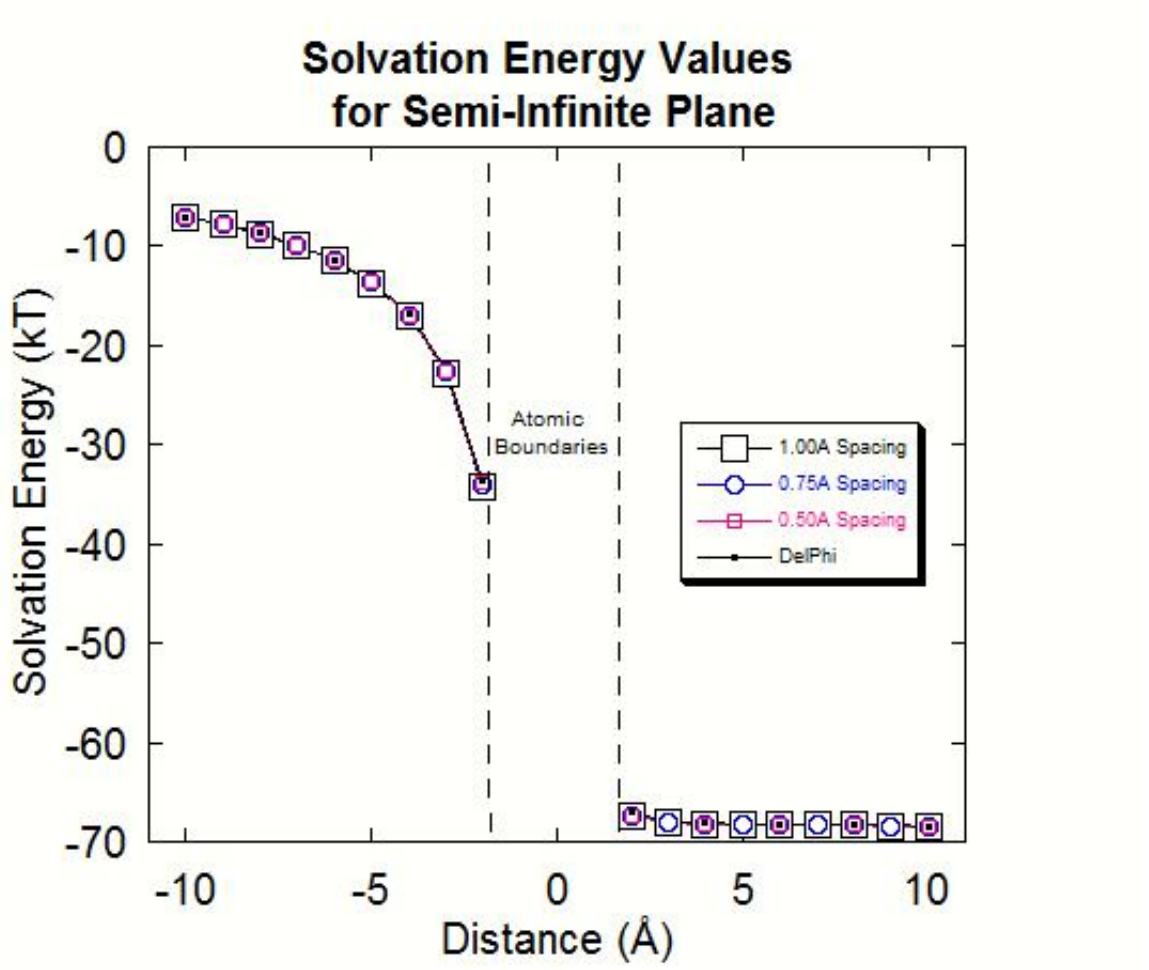

Figure 13 The chart comparing a set of atomic-style parallelepipeds with various values of precision generated by ProNOI with the continuous plane solution provided by DelPhi

CYLINDER

In Figure 14, the first ever photograph of DNA is shown next to the solid cylinder generated by the PDB-based ProNOI. It can be easily seen that this strand of DNA can be generalized to a long cylinder with specific dielectric properties and surface/volumetric charge.

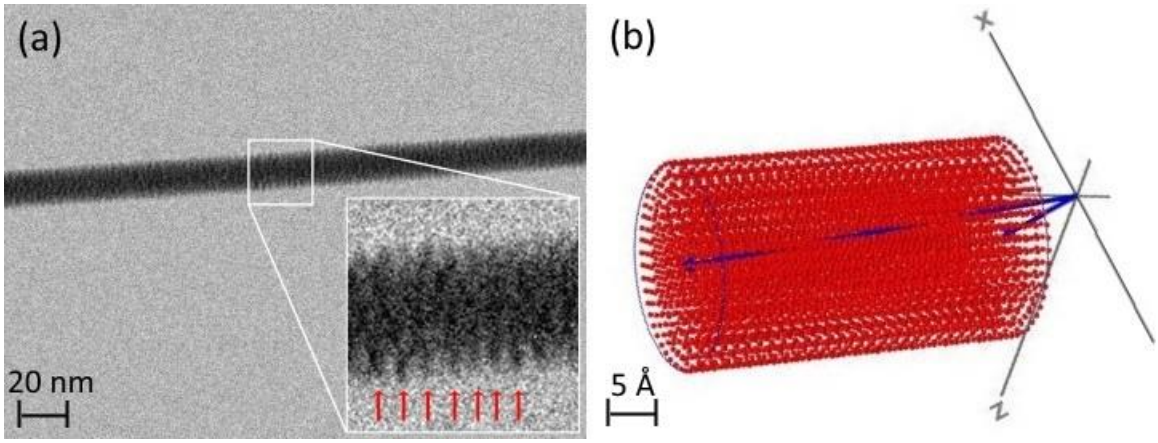

Figure 14 (a) A photograph of a strand of DNA adapted with permission from (Gentile, Moretti et al. 2012). Copyright 2014 American Chemical Society. (b) The cylinder solid rendering shown previously.

As before, an analytical solution exists for the electric potential of a large disc of charge, which can be compared to a wide cylinder generated by ProNOI. This experiment used an atomic-style cylinder with a radius of 5Å, height of 0.5Å, and a precision of 1.0Å that was charged with 1e per atom alongside a theoretical disk of the same radius. The results of the this simulation done in DelPhi can be seen in Figure 15 and clearly shows the atomic-style object results closely matches the analytical electric potential.

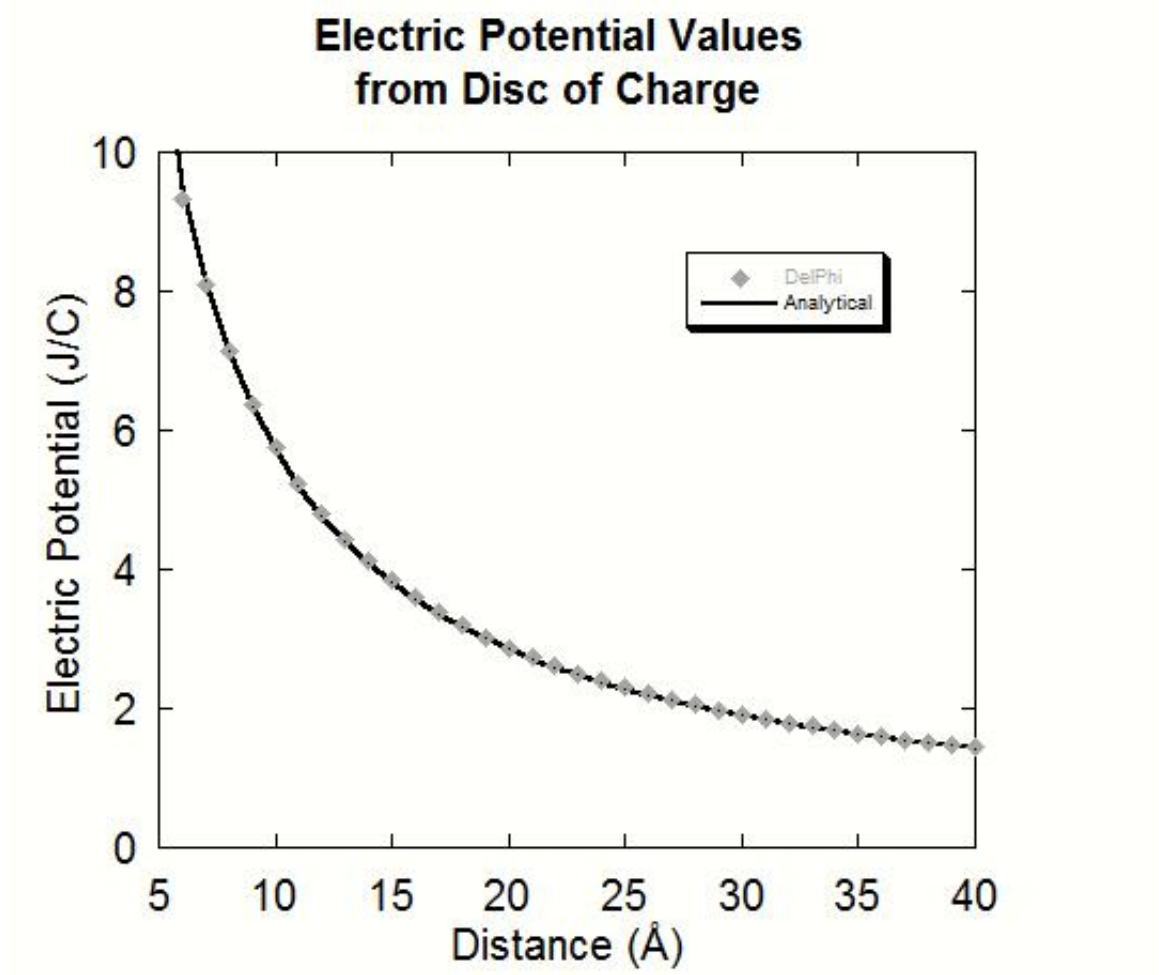

Figure 15 The comparison of the known electric potential a disk of charged material with the computational solution of the electric potential of a wide cylinder as provided by DelPhi.

CONE

In Figure 16, the tip of a used atomic force microscope cantilever can be seen alongside the solid cone rendering from the PDB-based ProNOI. The similarities between the two objects are clearly shown by the figure below.

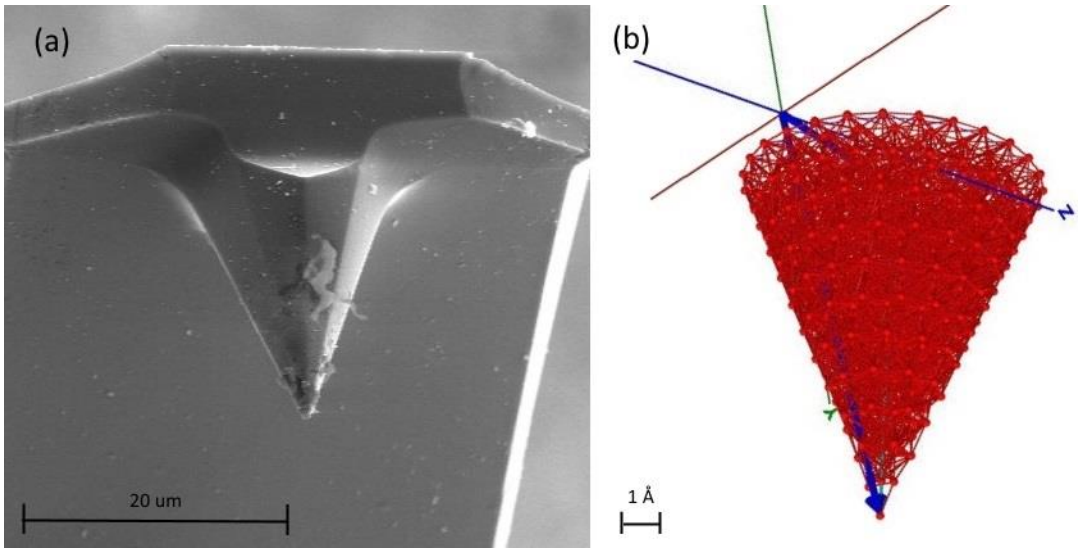

Figure 16 (a) View of cantilever in Atomic Force Microscope (magnification 1000x) adapted from http://commons.wikimedia.org/wiki/File:AFM_%28used%29_cantilever_in_Scanning_Electron_Microscope,_magnification_1000x.GIF (b) The cone solid rendering shown previously.

TUBE

In Figure 17, a strand of ssDNA is shown above the PDB of a tube designed to trace the length it. This figure partially demonstrates the capabilities of the Image-based ProNOI as this same method that was used to create the tube can be applied to much more complex images.

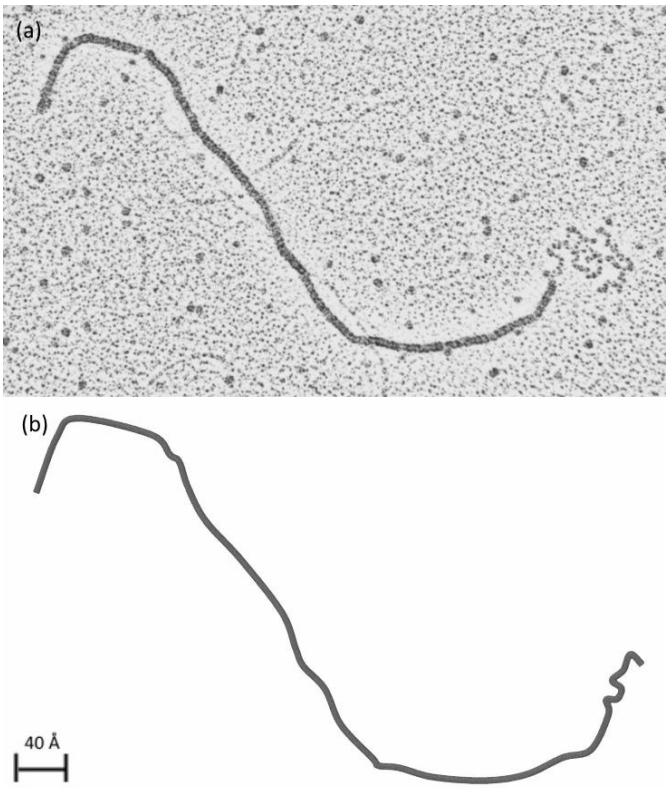

Figure 17 (a) A wiggled strand of single-stranded linear DNA adapted from the Electron Micrograph Library, Institute for Molecular Virology, University of Wisconsin - Madison (b) The tube rendering of the DNA generated via the Image-based ProNO shown previously.

SPHEROID

Figure 18 shows a mitochondrion modeled next to the spheroid generated from the Image-based ProNOI. While the structure may have internal components, these features are masked by the shell of the mitochondrion, which resembles the shape of a spheroid.

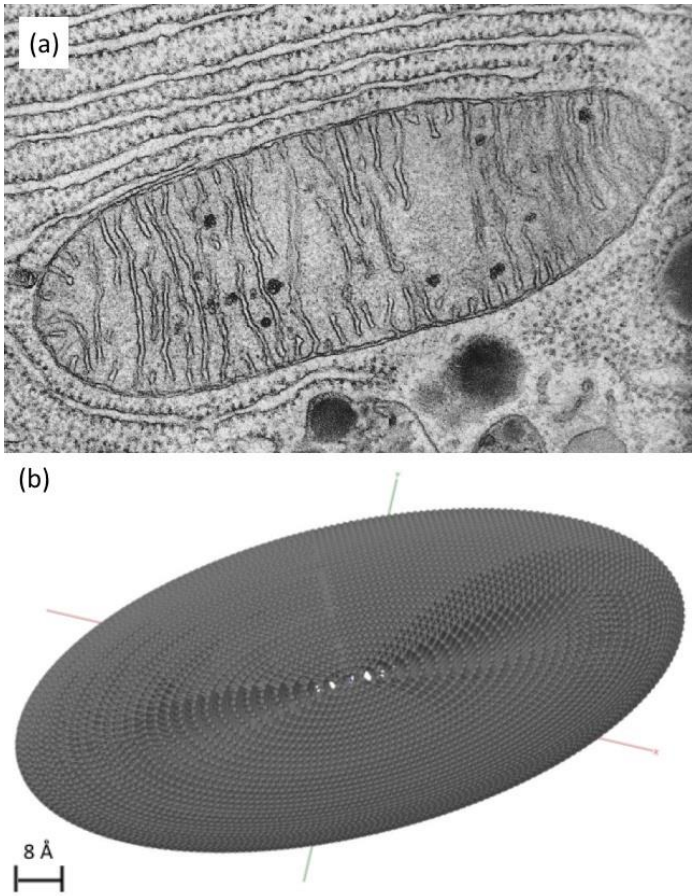

Figure 18 (a) A TEM photograph of a mitochondrion, a standard textbook image (b) The solid rendering of the spheroid from ProNOI shown previously.

HYBRIDIZED COMPLEX OBJECTS

The next figure, Figure 19, shows a neuron that is highlighted via fluorescent microscopy. The outline of this cell was traced using the Image-based ProNOI and exported to a PDB file shown in Figure 19b using a combination of both spheroids and tubes. This figure demonstrates that the shapes can be combined and rearranged into hybrid shapes consisting of multiple “basic” shapes for advanced object modeling, and since the program innately supports multiple objects

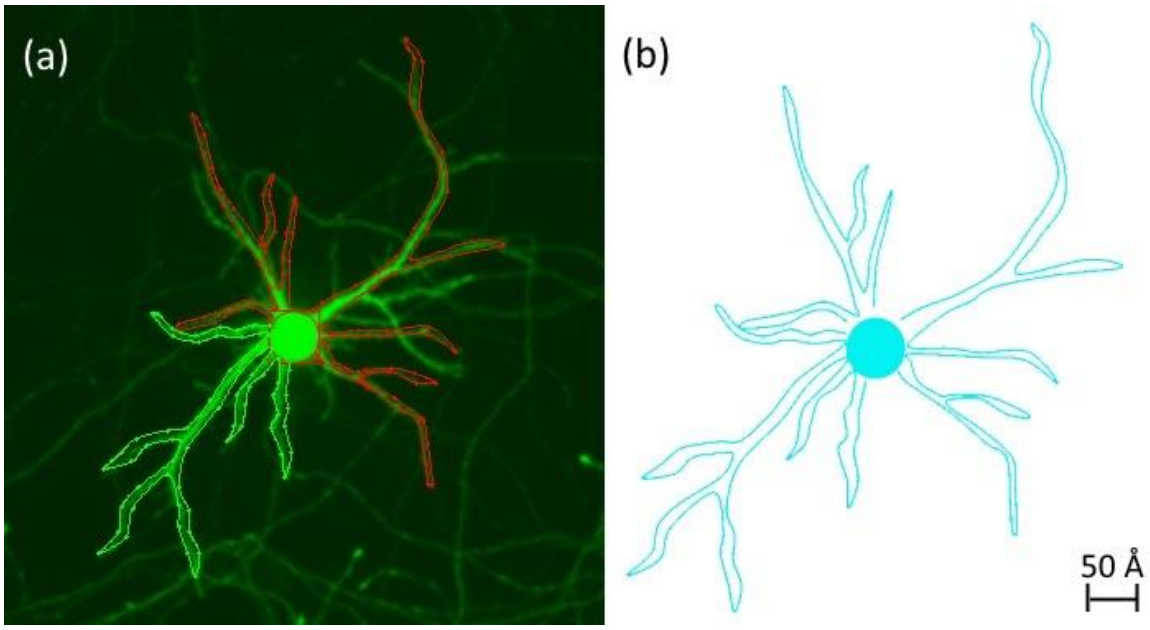

Figure 19 (a) The ProNOI-traced outline of a fluorescent neuron, a standard textbook image (b) The hybrid spheroid-tube PDB rendering generated from ProNOI rendered using multiple objects each with a precision of 1.0 Å and a bias and tension of 0

REFERENCES

Alexov, E. (2004). "Numerical calculations of the pH of maximal protein stability. The effect of the sequence composition and three-dimensional structure." Eur J Biochem **271**(1): 173-185.

Alexov, E., T. Kimmett, et al. (2014). "ProBLM Web Server: Protein and membrane placement and orientation package." Computational and Mathematical Methods in Medicine.

Alexov, E., E. L. Mehler, et al. (2011). "Progress in the prediction of pKa values in proteins." Proteins **79**(12): 3260-3275.

Baker, N. A. and J. A. McCammon (2003). "Electrostatic interactions." Methods Biochem Anal **44**: 427-440.

Baker, N. A., D. Sept, et al. (2001). "Electrostatics of nanosystems: application to microtubules and the ribosome." Proc Natl Acad Sci U S A **98**(18): 10037-10041.

Biasini, M., S. Bienert, et al. (2014). "SWISS-MODEL: modelling protein tertiary and quaternary structure using evolutionary information." Nucleic Acids Res.

Case, D. A., T. E. Cheatham, 3rd, et al. (2005). "The Amber biomolecular simulation programs." J Comput Chem **26**(16): 1668-1688.

Catmull, E. a. R. R. (1974). "A Class of Local Interpolating Splines." Computer Aided Geometric Design: 317-326.

Chen, D., Z. Chen, et al. (2011). "MIBPB: a software package for electrostatic analysis." J Comput Chem **32**(4): 756-770.

Cornell, W. D., P. Cieplak, et al. (1995). "A Second Generation Force Field for the Simulation of Proteins, Nucleic Acids, and Organic Molecules." Journal of the American Chemical Society **117**(19): 5179-5197.

Dolinsky, T. J., P. Czodrowski, et al. (2007). "PDB2PQR: expanding and upgrading automated preparation of biomolecular structures for molecular simulations." Nucleic Acids Res **35**(Web Server issue): W522-525.

Gentile, F., M. Moretti, et al. (2012). "Direct imaging of DNA fibers: the visage of double helix." Nano Lett **12**(12): 6453-6458.

Herraez, A. (2006). "Biomolecules in the computer: Jmol to the rescue." Biochem Mol Biol Educ **34**(4): 255-261.

Humphrey, W., A. Dalke, et al. (1996). "VMD: visual molecular dynamics." J Mol Graph **14**(1): 33-38, 27-38.

Lerch, T. F., Q. Xie, et al. (2010). "The structure of adeno-associated virus serotype 3B (AAV-3B): insights into receptor binding and immune evasion." Virology **403**(1): 26-36.

Li, C., M. Petukh, et al. (2013). "Continuous development of schemes for parallel computing of the electrostatics in biological systems: implementation in DelPhi." J Comput Chem **34**(22): 1949-1960.

Li, L., C. Li, et al. (2012). "DelPhi: a comprehensive suite for DelPhi software and associated resources." BMC Biophys **5**: 9.

Mackerell, A. D., Jr. (2004). "Empirical force fields for biological macromolecules: overview and issues." J Comput Chem **25**(13): 1584-1604.

Pettersen, E. F., T. D. Goddard, et al. (2004). "UCSF Chimera--a visualization system for exploratory research and analysis." J Comput Chem **25**(13): 1605-1612.

Phillips, J. C., R. Braun, et al. (2005). "Scalable molecular dynamics with NAMD." J Comput Chem **26**(16): 1781-1802.

Sarikaya, M., C. Tamerler, et al. (2003). "Molecular biomimetics: nanotechnology through biology." Nat Mater **2**(9): 577-585.

Sarkar, S., S. Witham, et al. (2013). "DelPhi Web Server: A comprehensive online suite for electrostatic calculations of biological macromolecules and their complexes." Commun Comput Phys **13**(1): 269-284.

Schrödinger, L. (2010). The PyMOL Molecular Graphics System, Version~1.3r1.

Smith, N., B. Campbell, et al. (2012). "Protein Nano-Object Integrator (ProNOI) for generating atomic style objects for molecular modeling." BMC Struct Biol **12**: 31.

Smith, N., S. Witham, et al. (2012). "DelPhi web server v2: incorporating atomic-style geometrical figures into the computational protocol." Bioinformatics **28**(12): 1655-1657.

Stone, J. E., J. C. Phillips, et al. (2007). "Accelerating molecular modeling applications with graphics processors." J Comput Chem **28**(16): 2618-2640.

Talley, K., C. Ng, et al. (2008). "On the electrostatic component of protein-protein binding free energy." PMC Biophys **1**(1): 2.

Tan, Z. J. and S. J. Chen (2011). "Salt contribution to RNA tertiary structure folding stability." Biophys J **101**(1): 176-187.

Tang, C. L., E. Alexov, et al. (2007). "Calculation of pKas in RNA: on the structural origins and functional roles of protonated nucleotides." J Mol Biol **366**(5): 1475-1496.