**MoleViewer: A Simple Protein Molecular Viewer and Alignment Tool in C#**

Mingzhao Liu

May 2016

**Abstract**

Understanding key differences between related proteins is critical to our understanding of biological pathways. In order to aid the process of determining how and where proteins differ from one another, this project implements MoleViewer, a molecular viewer capable of loading two proteins, performing a simple structural alignment, and determining and highlighting a residue of greatest deviation in C#. While the development environment was perhaps unsuited to a 3D intensive program, MoleViewer was able to overcome some of the shortcomings of the framework. MoleViewer is capable of viewing large protein structures and providing a simple and intuitive control scheme. The program implements an efficient iterative closest point algorithm for aligning two proteins. Root-Mean-Square Deviation is provided to determine the quality of the alignment.

**Introduction**

Our understanding of proteins is critical to determine the processes and functionality of genetic data. The expression of genes directly results in the production of associated protein products. Thus it is possible to determine differences in gene function by observing the differences in the structure of protein produced. Alterations in the structure of a protein may affect how it reacts chemically, which in turn changes the pathways or functions it is involved in, or how it interacts with those pathways or functions. These differences in structure are of great interest to structural biologists.

A protein is a three dimensional structure formed by one or more chains of amino acid residues. Amino acids are essential molecules for biological processes, containing a amine group and a carboxylic acid group, hence the name of amino acid. These amino acids are typically classified by physical or chemical properties, such as acidity and electrical charge. Amino acids are comprised of many atoms and atom groups, but the junction of the important amine group, carboxylic acid group, and the side chain unique to each amino acid occurs at an atom known as the alpha carbon. Tracing this alpha carbon along the protein chain will show a skeletal structure of the protein. The sequence of amino acids is known as the primary structure. The shape of the local segment of the protein is known as the secondary structure, as defined by hydrogen bonding. Secondaries structures can be found as helices, sheets, or turns. The geometric conformation of the entire continuous protein chain is known as the tertiary structure. If a protein is made up of many of these continuous chains, the conformation of multiple chains that interact together as a single unit is known as the quaternary structure. The tertiary and quaternary structures play an important role in allowing us to determining the function of proteins.

Proteins are assembled from component amino acids, one residue at a time, effectively creating a random coil of loose amino acid residues. They are referred to as residues when chained together because the chemical reaction to append residues removes the carboxylic acid group. The amino acids residues then interact with each other to fold the protein into an ordered shape. Proper folding is essential to proper function, as improper folding renders a protein unusable or even toxic.

To help determine protein function, protein structural alignments may be used to determine evolutionary similarities. Proteins that have similar structures may be related and may perform similar functions. Structural biology aims to determine the three dimensional structures of thousands of representative proteins and describe their functions. To achieve this, structural biologists have created sophisticated tools protein analysis. Advanced protein alignment algorithms are capable of taking into account not only the tertiary structure of the protein, but also incorporate sequence data from the primary structure.

The RCSB Protein Data Bank (PDB) files store a large amount of protein structure data. However, it impractical to manually compare protein structures for similarity. Molecular viewers are required to visualize and effectively alignment proteins. Programs that can parse PDB files and determine best fit between two proteins greatly reduce the amount of time needed for structural comparisons. A PDB file contains all of the biological information pertaining to a protein, including how the structural data was obtained. For this project, the only relevant part of a PDB file is the atomic record. Each line is a separate atom in the protein and is whitespace separated into columns.

The columns relevant to this project are as follows. Column 1 indicates that this the line is an atom record. Column 2 indicates the atom number. Column 3 indicates the element of the atom. Column 4 indicates the type of atom, notably if it is the alpha carbon. Column 6 indicates what kind of residue the atom belongs to. Column 7 indicates which chain the atom belongs to, as represented by a character. Column 9 is the number of the residue on this chain. Columns 11, 12, and 13 are the x, y, and z coordinates of the atom in angstroms. These coordinates are obtained from experiments where a protein is crystallized and measured using x-ray scattering. However, current experimental technique represents a severe shortcoming in structural biology. Proteins do not exist naturally as crystallized structures. In reality, the amino acid residues of a protein may rotate about their chemical bonds, and the exact locations of some atoms may vary. As a result, comparisons of protein structure are performed using the alpha carbon backbone, whose location does not vary.

Tools such as PyMOL are capable of rendering three dimensional models, providing protein alignment algorithms, numerous other analysis utilities, and Python extendable modules. It not only provides alignment tools, but also provides a measurement of how accurate the alignment might be. This measurement is the Root-Mean-Square deviation of atomic positions (RMSD), the square root of the average of the square of the distances between alpha carbon backbones. Emulating all of the features of such a program was beyond this project. As a result, I have endeavored to create a molecular viewer program that can read in the updated PDBx/mmCIF file format, render a model of the protein in three dimensions, with mouse driven camera controls, an alignment tool for aligning two proteins using their alpha carbon backbones, providing a measurement for the accuracy of alignment, and highlight deviations.

**Installation**

To install the program, download the program from the GitHub repository found at <https://github.com/mliu2/SnrMoleView>. To use the program, open the MoleViewer.exe which should be found in the /MoleViewer/MoleViewer/bin/x64/Release/ directory. MoleViewer requires a 64 bit Windows operating system and the Microsoft .NET Framework.

**User Manual**

In order to load in a protein model, click the load button beside the corresponding slot. A dialog box will prompt for the file, which should be a PDBx/mmCIF file from [www.rcsb.org](http://www.rcsb.org/). The protein model will be displayed in the gray area to the left. Use the mouse to move the camera. Left click and drag inside of the gray area to rotate the camera. Middle click and drag inside of the gray area to pan the camera. With the mouse cursor in the gray area, scroll the mouse wheel to zoom in or out. At the top left is a Help button. This will display another window that explains the camera controls, the alignment algorithm, as well as the color key for the protein model. To the right is an About button, which will give more information about the source code of the the project. Just bellow the Load buttons is an Align button. When two proteins are loaded, this will align the second protein to the first, highlight the residue on the first protein that is most distance from the nearest alpha carbon on the second protein, and provide the RMSD. The RMSD is shown in the Message Feed, the text display found in the bottom right of the screen. The Full Atom Model checkbox will toggle between displaying only the alpha carbons and displaying all of the atoms.

**Program Design**

The project can be broken into three major parts: the parser, the model, and the alignment algorithm. The data structures for the model that holds information about the protein are designed following the biological patterns from the bottom up. Atoms together make a Residue, Residues together make a Chain, Chains together make a Protein. However, Residues and Chains were dropped as they provided no real functionality beyond being yet another layer of containers. A Backend object holds the Proteins and manipulate them. The display holds a Backend, from which it obtains the information to display. Thus each layer can be separated without much difficulty. The parser is implemented into the underlying data structures. The model is implemented into the Backend and the display. The alignment algorithm is implemented into the Backend.

The first part of the project is the parser. The parser must be able to read a PDBx/mmCIF file and convert its atom records into usable data structures. The PDBx/mmCIF is described at <http://mmcif.wwpdb.org/docs/tutorials/content/atomic-description.html>. However, for the purpose of this project, the file must contain at least 1 atom record. Each atom record needs to be a line, whitespace separated into columns. Column 1 must indicate that it is an atom record by saying “ATOM”. Column 2 needs to indicate the atom number, an integer greater than 1. Column 3 needs to indicate the element of the atom; “C” for carbon, “N” for nitrogen, and “O” for oxygen. Column 4 needs to indicate the type of atom, although the program only cares if it is noted “CA” for alpha carbon. Column 6 needs to indicate what kind of residue the atom belongs to, although the program does not actually use this information. This information is recorded for possible addition features for the future. Column 7 needs to indicate which chain the atom belongs to, as represented by a character. Column 9 needs to indicate the number of the residue on this chain. Columns 11, 12, and 13 need to be parsable into double precision values and represent x, y, and z coordinates of the atom in angstroms. For example,

ATOM 1 C CA . MET A 1 1 ? 53.553 -7.050 35.606 1.00 47.40 ? ? ? ? ? ? 0 ACE A C 1

is a parsable line. The first column indicates that it is an atom record. The second column indicates that it is the first atom in the record. The third column is “C” indicating that it is carbon atom. The fourth column is “CA” indicating that it this is an alpha carbon. The sixth column is “MET” indicating that this residue is methionine. Column seven indicates that this atom is found on chain A. Column nine indicates that this is the first amino acid residue on chain A. Columns 11, 12, and 13 can be parsed into the coordinates 53.553, -7.050, 35.606. Example files are provided in the /MoleViewer/test files/ directory. The required implementing the Atom and Protein classes. The Atom class holds information extracted from each line of the atom record in a PDBx/mmCIF file. The class also provides accessors and mutators for the data members. The Protein class contains a list of Atoms and provides functions that aggregate data about the Atoms or manipulate them all at once. The actual parser function can be found in the Protein class, as it needs to be able to access the list of Atoms to populate.

The second part of the project is the modeling of the protein. This also includes the camera controls for the mouse. The display, MainWindow, holds a Viewport3D object that displays the models. The Viewport3D object holds a DirectionalLight and a PerspectiveCamera. The DirectionalLight illuminates models in the Viewport3D from a specified direction. The PerspectiveCamera directs the view into the Viewport3D. The Viewport3D object holds a ModelVisual3D object, which holds a Model3DGroup, an object that holds a collection of models to be displayed. These models are generated in the display using data from the Backend and underlying data structures. The display must also implement mouse-driven controls for the camera. To do this, the display implements event handlers for mouse buttons being held down, being released, and the movement of the mouse. When a relevant mouse button is clicked, the display will keep track of if it is being held down. If it is, and the mouse moves, the display will apply a transformation to the camera. The display also tracks the location of the mouse when it was clicked, as the amount of movement of the camera varies by the amount of movement of the mouse. The display needs to track the depression of the middle and left mouse buttons and the movement of the scroll wheel.

The third part of the project is to provide an alignment tool that will superimpose two proteins and report the accuracy using RMSD. Alignment is implemented into the Backend object, as it holds and can access both of the loaded proteins. The Backend can call the matrix conversion functions of each protein and input them into the alignment algorithm. The Backend can then perform the proper transformations on the proteins based on the results of the alignment. In order to report the RMSD, the Backend holds a list that keeps track of the the alpha carbon pairings as well as the distance between each alpha carbon once the alignment has been performed. The list holds objects of the ResNumPair class, which holds information about alpha carbon pairs and the distance between them and implements a comparison function to allow the list to be sortable. From this list, the Backend can calculate the RMSD and determine which alpha carbon pair has the greatest distance. All of this information is passed to the display to show to the user. The ResNumPair class holds the residue numbers of the two alpha carbons, the chain each alpha carbon is found on,and the distance between the two atoms. Importantly, it is extends the IComparable interface, allowing a container of ResNumPairs to be sortable.

In summary, the implemented classes are as follows:

* Atom, which holds the physical information from individual atom records of a PDBx/mmCIF file.
* Protein, which holds a list of Atoms and contains functions that manipulate many atoms and aggregate data from many atoms, such as transformations and finding the number of alpha carbons It also parses files to populate each Atom in the list.
* Backend, which holds two Proteins and a list of ResNumPairs. It contains functions that require data from both proteins. It also provides the data that needs to be displayed to the user. The alignment algorithm and RMSD calculations are implemented here.
* ResNumPairs, which holds the residue numbers of alpha carbon pairs, their chain, and the distance between the two. It extends the IComparable interface to allow it to be sortable in a collection.
* MainWindow, which holds the Viewport3D display object, a ModelVisual3D to display, a Model3DGroup that holds the spherical models of atoms, and a Backend object from which it can get the data needed for the display.
* The HelpWindow and AboutWindow classes exist independently of the data flow and exist to provide information about the program. They contain no member variables.

**Methods and Materials**

This program was developed in C# using Windows Presentation Forms (WPF) in the .NET framework. Files are read line by line as strings. Relevant strings are parsed to create Atom objects that track atom type, number, and position. The reader only reads atoms that are of the protein chain. It does not process heteroatoms, atoms that are crystallized along with the protein and are important to the overall complex, but not actually part of the protein chains. The CIF parsing library for C++ was determined to be too complex for the task and thus was not used. The Protein class is a container class with a list of Atoms with functions that affect the entire list of Atoms. When a protein is loaded and parsed, its center will be transformed to the origin. This focuses the camera on the protein and enables a simpler camera implementation. A Backend object holds two Proteins and passes data to the display. It contains functions required by the display, but that also need to access data from the Proteins, acting as a intermediary. The display creates this Backend object when it is loaded.

Rendering of protein models is done through the WPF Viewport3D framework, which is the WPF native 3D rendering framework. Meshes in Viewport3D are created as a set of triangles with normal vectors to indicate where texture materials are to be applied. Sphere generation is achieved using the BeachBallSphere method implemented by Charles Petzold. The size of spheres was determined using the known covalently bonded radius of the element1.

Camera controls are implemented by applying transformations to the camera. Rotation of the camera would create the effect of rotating the camera's viewing vector like a first person camera, as opposed to rotating the object, as would be expected. Instead, rotation control is achieved by translating the object to the center, and determining a spherical shell around it, along which the camera would be transformed. This method of projecting the two dimensional mouse coordinates into a three dimensional trackball around the center is adapted from the 3DTools library for WPF. Zooming is achieved by translating the camera back along the view axis. Panning is achieved by translating the camera in the xy-plane.

Proteins are aligned using Iterative Closest Point (ICP) algorithm. ICP is a mesh alignment algorithm that is used to minimize the difference between two point clouds. One cloud is selected as the reference cloud, the one that is not moving. The other cloud is determined to the source cloud, which will be transformed. For each point in the source cloud, the algorithm determines the closest point in the reference cloud. The algorithm then determines the amount to translate and rotate the source cloud by using a mean squared deviation cost function in order to best align all source points to their paired reference points. The source cloud is transformed, and the source points are re-associated to the closest reference point and the algorithm iterates until the mean squared deviation is below a set value2. This project implements LIBICP, a C++ library for fitting two or three dimensional point clouds using singular value decomposition to determine the point-to-point transformation. Here, it used with a .NET wrapper for ease of use in C#. Once the transformation is determine, the coordinates of atoms in the protein are converted into a matrix, and Math.NET Numerics is used to apply the previously calculated transformation to the matrix. The matrix is then fed back to the protein object, which will set the coordinates of its atoms accordingly.

Root-Mean-Square Deviations (RMSD) calculations are performed after alignment to determine the quality of alignment. The program determines RMSD by finding the closest alpha carbon on the opposite protein. The distances for each alpha carbon are then summed, squared, averaged, and rooted to determine the deviation in angstroms. The distances for each alpha carbon are then sorted, and the alpha carbon on the first protein with the greatest distance from nearest alpha carbon on the opposite chain are recorded. The display will then highlight both the residue of the recorded alpha carbon and the residue of its paired alpha carbon.

**Testing**

Each part of the program was error tested independently. The parser was tested using correctly five formatted files from the Protein Data Bank. Each protein object was checked to contain the same number of atoms as the file, indicating that every atom record had been successfully parsed. For example, the protein file 1u19.cif had 5498 atoms parsed, matching with the 5498 atom records the 1u19 protein contains. The record of one was then changed to have the correct format, except that one of the fields contained unparsable data. The parser was able to detect this and threw an exception. Columns were then deleted from a correct atom record. Again, the parser was able to detect this and threw an exception.

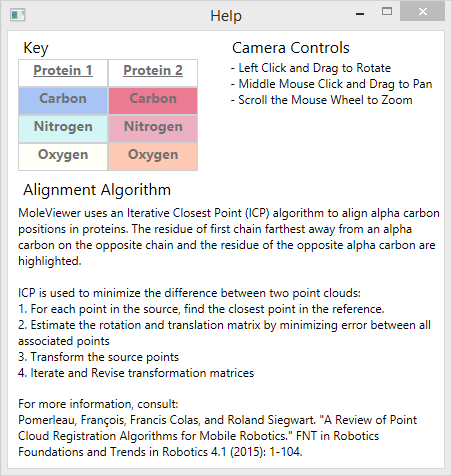
The model generation allowed the confirmation that the coordinates of the atom were being correctly parsed. The sphere generation could be visually inspected to ensure that the model generate was, in fact, a sphere. The location confirmation require the use of some proteins with familiar structures. They were loaded into the program and the resulting models were inspected side by side with the same protein loaded into PyMOL with comparable settings. For example, protein 1u19 is a contains distinctive helical structures.

The alignment algorithm was tested by taking a small protein structure and applying transformations to the original structure. The alignment algorithm was then run to alignment the two identical structures. The algorithm was able to totally superimpose them each time, recording an RMSD of 0.00 angstroms each time.

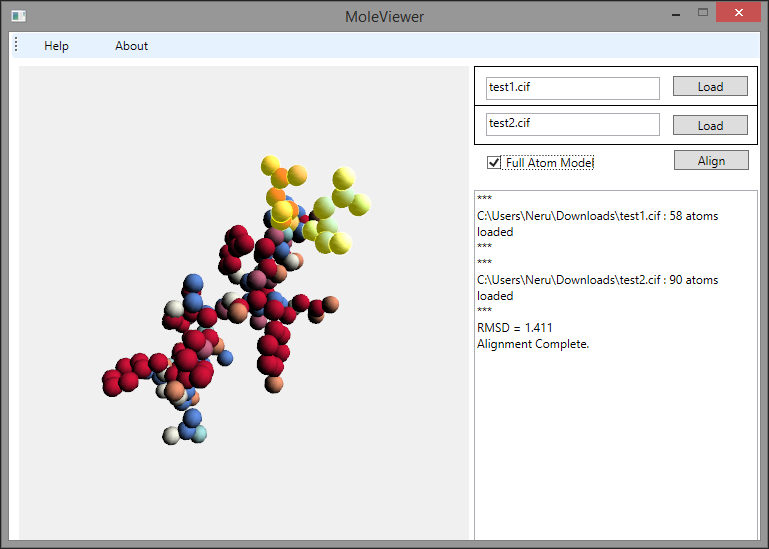
**Results**

The resulting program, MoleViewer, is a C# .NET Framework program compiled for Windows 64 bits operating systems. It is capable of rendering large protein molecules in both alpha carbon and full atom mode, provided it is operating on a 64 bit operating system. A 32 bit operating system cannot allocate sufficient memory to render a protein of about 20,000 atoms. However, there is a noticeable time delay required to load such a large protein. Furthermore, switching the display from alpha carbon to display all atoms involves an even larger delay of nearly a minute. If two large proteins are loaded, toggling to all atoms will take multiple minutes. If the proteins are too large, the program will inform the user that it has run out of memory.

In addition, MoleViewer has fully mouse driven camera controls. Left click and drag will rotate. Middle click and drag will pan. Scroll wheel will zoom. The camera control behave intuitively and smoothly. A help page shows the user the color scheme used for the atoms, camera controls, and an explanation of ICP alignment. There is an about page, with information on sources for code libraries that were used. In addition, there are hover-over tool tips that indicates the purpose of each user interface element.

 **Figure 1. MoleViewer Help Page.**The key for element coloration is provided, along with controls and a brief explanation of ICP.

MoleViewer features ICP alignment between two proteins. ICP requires a minimum of five points to align, so the program will not align anything with fewer than five alpha carbons. The program will also not align if there is only one protein loaded. The algorithm requires that its clouds be in similar conformations and close to each other. This means that if proteins are not in proximity, the alignment will be less useful. In addition, the alignment is purely structural, and does not take into account the residues being aligned. As a result, the result of alignment is more likely to vary with the initial position of the proteins. Once the alignment is completed, MoleViewer will calculate the RMSD and display it in the output feed. The distance from each alpha carbon on the first protein to the nearest alpha carbon on the other protein is recorded. The residue whose carbon has the greatest distance and residue of the nearest alpha carbon on the opposite protein are both highlighted in yellow.

**Figure 2. Sample alignment in MoleViewer.** Two small protein sequences (test1.cif and test2.cif) )are loaded in and aligned. The residue of greatest deviation is highlighted. RMSD is shown in the output feed

**Discussion**

MoleViewer succeeded in providing a molecular viewer capable of displaying two proteins in C# using the .NET Framework. The model display camera is entirely mouse controlled and works as expected. MoleViewer provides an algorithm for alignment as well as providing an indicator of how good the alignment is. However, it has failed to provide an efficient molecular viewer. Toggling between full atom mode and alpha carbon mode can take a few seconds in order to render the molecules. Displaying two proteins with more than 20,000 atoms in full atom mode failed. The program stopped and displayed that it had run out of available memory. Loading in proteins may take a few seconds if it is of significant size. This may be a result of the programming environment being unsuited to such a program. Windows Presentation Forms uses Viewport3D to display objects in three dimensions. However, the system does not provide native support for polygon generation. Each mesh must be generated by passing a set of three points and a normal vector to generate mesh triangles. Thus code from a preexisting sphere mesh generator was used for this project. This is indicative that 3D rendering is not the primary purpose of WPF. As such, generation of a large number of meshes may be less efficient in this framework than a dedicated 3D rendering system, such as OpenGL.

Camera movement functionality was adapted from the 3DTools library for WPF. 3DTools needed to be adapted, as it is nine years out of date, and also does not appear to provide any support for camera panning.

Of note also is the chosen alignment algorithm. As previously stated, iterative closest point is a mesh alignment algorithm, not a protein alignment algorithm. While the protein backbone can be treated like a mesh, ICP lacks the finesse and specificity of dedicated protein alignment algorithms. ICP does not take into account the residues or protein domains that are being aligned. For example, PyMOL first performs a sequence alignment to determine how to pair residues, followed by a structural superposition3. ICP is best used with some amount of position adjustment beforehand to get the best results2. However, writing an algorithm capable of dynamically determining how to position a protein mesh so that two proteins are close in conformation is beyond the scope of this project. As such, the ICP alignment implemented here may vary more with the initial position of both proteins than other algorithms. The implementation of this algorithm also proved troublesome. LIBICP requires inputs of *n x 3* matrices for alignment in 3D space. However, in order to apply the rotation matrix, the *3 x 3* rotation matrix needs to be multiplied by the matrix of coordinates, which now needs to be *3 x n*, necessitating two different matrix conversion functions for the Protein class. RMSD calculations could also be made more efficient, as I could not determine how to access the *k-d* tree of positions LIBICP creates for its calculations and so had to perform a second search through all of the alpha carbons.

The program provides highlighting post-alignment of the residue on the first protein whose alpha carbon is furthest away from the nearest alpha carbon on the other protein. It also highlights the residue whose alpha carbon it is closest to. It is unclear if this, or providing the highlighting based on the alpha carbons in the second protein is more helpful. Both are not provided as this was determined to be inefficient. This would require yet another full iteration to search through the alpha carbon combinations. Only one protein can be the reference in the alignment. Providing highlighting based on the reference protein would highlight what is important in the reference protein, while providing highlighting based on the second protein would highlight how it differs from the reference protein. In addition, this highlighting is completely unhelpful if both proteins are not of approximately equal size.

**Conclusion**

MoleViewer is a program capable of providing full atom and alpha carbon models of proteins data recorded in the PDBx/mmCIF file format. It features a simple alignment algorithm, RMSD reporting, and highlighting for the residue with greatest deviation. While it provides simple features and is inefficient at handling large molecules, it provides the basic functionality of a protein molecular viewer. Its simplicity allows it to have a simple and intuitive control scheme, at least less intimidating than the complex control panel of advanced molecular viewers such as PyMOL. It has also demonstrated that Windows Presentation Forms is not an ideal development environment for 3D rendering. Despite the shortcomings of the WPF environment, MoleViewer has thus far performed adequately as a molecular viewer and protein alignment tool.

Works Cited

1. Housecroft, Catherine E., Sharpe, Alan G. Inorganic Chemistry. Third ed. Pearson / Prentice Hall: Essex, UK. 2008. pp. 31, 1013, 1014

2. Pomerleau, François, Francis Colas, and Roland Siegwart. "A Review of Point Cloud Registration Algorithms for Mobile Robotics." FNT in Robotics Foundations and Trends in Robotics 4.1 (2015): 1-104.

3. The PyMOL Molecular Graphics System, Version 1.8 Schrödinger, LLC.